Incentives Build Robustness in BitTorrent

Bram Cohen
bram@bitconjurer.org

May 22, 2003

Abstract

The BitTorrent file distribution system uses tit-for-tat as a method of seeking pareto efficiency. It achieves a higher level of robustness and resource utilization than any currently known cooperative technique. We explain what BitTorrent does, and how economic methods are used to achieve that goal.

1 What BitTorrent Does

When a file is made available using HTTP, all upload cost is placed on the hosting machine. With BitTorrent, when multiple people are downloading the same file at the same time, they upload pieces of the file to each other. This redistributes the cost of upload to downloaders, (where it is often not even metered), thus making hosting a file with a potentially unlimited number of downloaders affordable.

Researchers have attempted to find practical techniqes to do this before[3]. It has not been previously deployed on a large scale because the logistical and robustness problems are quite difficult. Simply figuring out which peers have what parts of the file and where they should be sent is difficult to do without incurring a huge overhead. In addition, real deployments experience very high churn rates. Peers rarely connect for more than a few hours, and frequently for only a few minutes [4]. Finally, there is a general problem of fairness [1]. The total download rate across all downloaders must, of mathematical necessity, be equal to the total upload rate. The strategy for allocating upload which seems most likely to make peers happy with their download rates is to make each peer's download rate be proportional to their upload rate. In practice it's very difficult to keep peer download rates from sometimes dropping to zero by chance, much less make upload and download rates be correlated. We will explain how BitTorrent solves all of these problems well.

1.1 BitTorrent Interface

BitTorrent's interface is almost the simplest possible. Users launch it by clicking on a hyperlink to the file they wish to download, and are given a standard "Save As" dialog, followed by a download progress dialog which is mostly notable for having an upload rate in addition to a download rate. This extreme ease of use has contributed greatly to BitTorrent's adoption, and may even be more important than, although it certainly complements, the performance and cost redistribution features which are described in this paper.

1.2 Deployment

The decision to use BitTorrent is made by the publisher of a file. Downloaders use BitTorrent because it's the only way to get the file they want. Frequently downloaders cease uploading as soon as their download completes, although it is considered polite to leave the client uploading for a while after your download completes. The standard implementation continues to upload until the window is closed, which frequently results in uploads continuing until the user gets back to their machine.

In a typical deployment, the number of incomplete downloaders, (meaning ones which do not have the

whole file), increases very rapidly after the file is made available. It eventually peaks and then falls off at a roughly exponential rate. The number of complete downloaders increases slowly, peaks some time after the number of incomplete downloaders does, then also falls off exponentially. The peak of incomplete downloaders passes as downloaders complete. The peak of incomplete downloaders passes as finished downloaders stop uploading. The exponential falloff of both reflects the rate of new downloaders joining after the initial rush is over.

file made available by host, downloaders use BT because they want the file.downloaders upload while downloading, then leave.

2 Technical Framework

2.1 Publishing Content

To start a BitTorrent deployment, a static file with the extension .torrent is put on an ordinary web server. The .torrent contains information about the file, its length, name, and hashing information, and the url of a tracker. Trackers are responsible for helping downloaders find each other. They speak a very simple protocol layered on top of HTTP in which a downloader sends information about what file it's downloading, what port it's listening on, and similar information, and the tracker responds with a list of contact information for peers which are downloading the same file. Downloaders then use this information to connect to each other. To make a file available, a 'downloader' which happens to have the complete file already, known as a seed, must be started. The bandwidth requirements of the tracker and web server are very low, while the seed must send out at least one complete copy of the original file.

2.2 Peer Distribution

All logistical problems of file downloading are handled in the interactions between peers. Some information about upload and download rates is sent to the tracker, but that's just for statistics gathering. The tracker's responsibilities are strictly limited to helping peers find each other.

Although trackers are the only way for peers to find each other, and the only point of coordination at all, the standard tracker algorithm is to return a random list of peers. Random graphs have very good robustness properties [2]. Many peer selection algorithms result in a power law graph, which can get segmented after only a small amount of churn. Note that all connections between peers can transfer in both directions.

In order to keep track of which peers have what, BitTorrent cuts files into pieces of fixed size, typically a quarter megabyte. Each downloader reports to all of its peers what pieces it has. To verify data integrity, the SHA1 hashes of all the pieces are included in the .torrent file, and peers don't report that they have a piece until they've checked the hash. Erasure codes have been suggested as a technique which might help with file distribution [3], but this much simpler approach has proven to be workable.

Peers continuously download pieces from all peers which they can. They of course cannot download from peers they aren't connected to, and sometimes peers don't have any pieces they want or won't currently let them download. Strategies for peers disallowing downloading from them, known as choking, will be discussed later. Other suggested approaches to file distribution have generally involved a tree structure, which doesn't utilize the upload capacity of leaves. Simply having peers announce what they have results in less than a tenth of a percent bandwidth overhead and reliably utilizes all available upload capacity.

2.3 Pipelining

When transferring data over TCP, like BitTorrent does, it is very important to always have several requests pending at once, to avoid a delay between pieces being sent, which is disastrous for transfer rates. BitTorrent facilitates this by breaking pieces further into sub-pieces over the wire, typically sixteen kilobytes in size, and always keeping some number, typically five, requests pipelined at once. Every time a sub-piece arrives a new request is sent. The amount of data to pipeline has been selected as a value which

can reliably saturate most connections.

2.4 Piece Selection

Selecting pieces to download in a good order is very important for good performance. A poor piece selection algorithm can result in having all the pieces which are currently on offer or, on the flip side, not having any pieces to upload to peers you wish to.

2.4.1 Strict Priority

BitTorrent's first policy for piece selection is that once a single sub-piece has been requested, the remaining sub-pieces from that particular piece are requested before sub-pieces from any other piece. This does a good job of getting complete pieces as quickly as possible.

2.4.2 Rarest First

When selecting which piece to start downloading next, peers generally download pieces which the fewest of their own peers have first, a technique we refer to as 'rarest first'. This technique does a good job of making sure that peers have pieces which all of their peers want, so uploading can be done when wanted. It also makes sure that pieces which are more common are left for later, so the likelihood that a peer which currently is offering upload will later not have anything of interest is reduced.

Information theory dictates that no downloaders can complete until every part of the file has been uploaded by the seed. For deployments with a single seed whose upload capacity is considerably less than that of many downloaders, performance is much better if different downloaders get different pieces from the seed, since redundant downloads waste the opportunity for the seed to get more information out. Rarest first does a good job of only downloading new pieces from the seed, since downloaders will be able to see that their other peers have pieces the seed has uploaded already.

For some deployments the original seed is eventually taken down for cost reasons, leaving only current downloaders to upload. This leads to a very significant risk of a particular piece no longer being available from any current downloaders. Rarest first again handles this well, by replicating the rarest pieces as quickly as possible thus reducing the risk of them getting completely lost as current peers stop uploading.

2.4.3 Random First Piece

An exception to rarest first is when downloading starts. At that time, the peer has nothing to upload, so it's important to get a complete piece as quickly as possible. Rare pieces are generally only present on one peer, so they would be downloaded slower than pieces which are present on multiple peers for which it's possible to download sub-pieces from different places. For this reason, pieces to download are selected at random until the first complete piece is assembled, and then the strategy changes to rarest first.

2.4.4 Endgame Mode

Sometimes a piece will be requested from a peer with very slow transfer rates. This isn't a problem in the middle of a download, but could potentially delay a download's finish. To keep that from happening, once all sub-pieces which a peer doesn't have are actively being requested it sends requests for all sub-pieces to all peers. Cancels are sent for sub-pieces which arrive to keep too much bandwidth from being wasted on redundant sends. In practice not much bandwidth is wasted this way, since the endgame period is very short, and the end of a file is always downloaded quickly.

3 Choking Algorithms

BitTorrent does no central resource allocation. Each peer is responsible for attempting to maximize its own download rate. Peers do this by downloading from whoever they can and deciding which peers to upload to via a variant of tit-for-tat. To cooperate, peers upload, and to not cooperate they 'choke' peers. Choking is a temporary refusal to upload; It stops uploading, but downloading can still happen

and the connection doesn't need to be renegotiated when choking stops.

The choking algorithm isn't technically part of the BitTorrent wire protocol, but is necessary for good performance. A good choking algorithm should utilize all available resources, provide reasonably consistent download rates for everyone, and be somewhat resistant to peers only downloading and not uploading.

3.1 Pareto Efficiency

Well known economic theories show that systems which are pareto efficient, meaning that no two counterparties can make an exchange and both be happier, tend to have all of the above properties. In computer science terms, seeking pareto efficiency is a local optimization algorithm in which pairs of counterparties see if they can improve their lot together, and such algorithms tend to lead to global optima. Specifically, if two peers are both getting poor reciprocation for some of the upload they are providing, they can often start uploading to each other instead and both get a better download rate than they had before.

BitTorrent's choking algorithms attempt to achieve pareto efficiency using a more fleshed out version of tit-for-tat than that used to play prisoner's dilemma. Peers reciprocate uploading to peers which upload to them, with the goal of at any time of having several connections which are actively transferring in both directions. Unutilized connections are also uploaded to on a trial basis to see if better transfer rates could be found using them.

3.2 BitTorrent's Choking Algorithm

On a technical level, each BitTorrent peer always unchokes a fixed number of other peers (default is four), so the issue becomes which peers to unchoke. This approach allows TCP's built-in congestion control to reliably saturate upload capacity.

Decisions as to which peers to unchoke are based strictly on current download rate. Calculating current download rate meaningfully is a surprisingly difficult problem; The current implementation essentially uses a rolling 20-second average. Former choking algorithms used information about long-term net transfer amounts, but that performed poorly because the value of bandwidth shifts rapidly over time as resources go away and become available.

To avoid situations in which resources are wasted by rapidly choking and unchoking peers, BitTorrent peers recalculate who they want to choke once every ten seconds, and then leave the situation as is until the next ten second period is up. Ten seconds is a long enough period of time for TCP to ramp up new transfers to their full capacity.

3.3 Optimistic Unchoking

Simply uploading to the peers which provide the best download rate would suffer from having no method of discovering if currently unused connections are better than the ones being used. To fix this, at all times a BitTorrent peer has a single 'optimistic unchoke', which is unchoked regardless of the current download rate from it. Which peer is the optimistic unchoke is rotated every third rechoke period (30 seconds). 30 seconds is enough time for the upload to get to full capacity, the download to reciprocate, and the download to get to full capacity. The analogy with tit-for-tat here is quite remarkable; Optimistic unchokes correspond very strongly to always cooperating on the first move in prisoner's dilemma.

3.4 Anti-snubbing

Occasionally a BitTorrent peer will be choked by all peers which it was formerly downloading from. In such cases it will usually continue to get poor download rates until the optimistic unchoke finds better peers. To mitigate this problem, when over a minute goes by without getting a single piece from a particular peer, BitTorrent assumes it is 'snubbed' by that peer and doesn't upload to it except as an optimistic unchoke. This frequently results in more than one concurrent optimistic unchoke, (an exception to the exactly one optimistic unchoke rule mentioned above), which causes download rates to recover much more quickly when they falter.

3.5 Upload Only

Once a peer is done downloading, it no longer has useful download rates to decide which peers to upload to. The current implementation then switches to preferring peers which it has better upload rates to, which does a decent job of utilizing all available upload capacity and preferring peers which noone else happens to be uploading to at the moment.

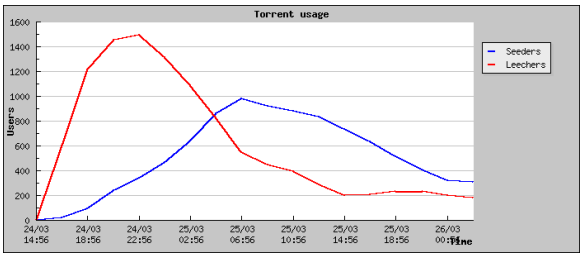


Figure 1: The number of complete downloaders ('seeders') and incomplete downloaders ('leechers') of a large deployment of an over 400 megabyte file over time. There must have been at least 1000 successful downloads, since at one time there were that many complete downloaders. The actual number of downloads completed during this period was probably several times that.

4 Real World Experience

BitTorrent not only is already implemented, but is already widely deployed. It routinely serves files hundreds of megabytes in size to hundreds of concurrent downloaders. The largest known deployments have had over a thousand simultaneous downloaders. The current scaling bottleneck (which hasn't actually been reached) appears to be the bandwidth overhead of the tracker. Currently that's about a thousandth the total amount of bandwidth used, and some minor protocol extensions will probably get it down to a ten thousandth.

References

[1] E. Adar and B. A. Huberman. Free riding on gnutella. *First Monday*, 5(10), 2000.

[2] A.-L. Barabási. *Linked: The New Science of Networks*. Perseus Publishing, 2002.

[3] M. Castro, P. Druschel, A.-M. Kermarrec, A. Nandi, A. Rowstron, and A. Singh. Splitstream: High-bandwidth content distribution in cooperative environments. In *Proceedings of IPTPS03*, Berkeley, USA, Feb. 2003.

[4] P. Maymounkov and D. Mazieres. Kademlia: A peer-to-peer information system based on the xor metric. In *Proceedings of IPTPS02*, Cambridge, USA, Mar. 2002.

The Road Ahead for Networking: A Survey on ICN-IP Coexistence Solutions

Mauro Conti, *Senior Member, IEEE*, Ankit Gangwal, Muhammad Hassan, Chhagan Lal, and Eleonora Losiouk

***Abstract*—In recent years, the usage model of the Internet has changed, pushing researchers towards the design of the Information-Centric Networking (ICN) paradigm as a possible replacement of the existing architecture. Even though both Academia and Industry have investigated the feasibility and effectiveness of ICN, achieving the complete replacement of the Internet Protocol (IP) is a challenging task: (i) the process involves multiple parties, such as Internet Service Providers (ISPs), that need to coordinate among each other; (ii) it requires an indefinite amount of time to update hardware and software of network components; and (iii) it is a high risk goal that might introduce unexpected complications. Thus, the process of replacing the current Internet will inevitably lead towards a period of coexistence between the old and the new architectures. Given the urgency of the problem, this transition phase will happen very soon and people should address it in a smooth way. Some research groups have already addressed the coexistence by designing their own architectures, but none of those is the final solution to move towards the future Internet considering the unaltered state of the networking. To design such architecture, the research community needs now a comprehensive overview of the existing solutions that have so far addressed the coexistence. The purpose of this paper is to reach this goal by providing the first comprehensive survey and classification of the coexistence architectures according to their features (i.e., deployment approach, deployment scenarios, addressed coexistence requirements and additional architecture or technology used) and evaluation parameters (i.e., challenges emerging during the deployment and the runtime behaviour of an architecture). We believe that this paper will finally fill the gap required for moving towards the design of the final coexistence architecture.**

***Index Terms*—Coexistence solutions, future Internet architectures, information-centric networking, Internet protocol, secure transition.**

Manuscript received October 15, 2019; revised March 11, 2020; accepted April 19, 2020. Date of publication May 15, 2020; date of current version August 21, 2020. This work was supported by the European Commission through the Horizon 2020 Programme, as part of LOCARD Project under Grant 832735. The work of Mauro Conti was supported by the Marie Curie Fellowship funded by the European Commission under Agreement PCIG11-GA-2012-321980. The work of Ankit Gangwal was supported by the CaRiPaRo Fellowship funded by Fondazione Cassa di Risparmio di Padova e Rovigo. *(Corresponding author: Eleonora Losiouk.)*

Mauro Conti, Ankit Gangwal, Muhammad Hassan, and Eleonora Losiouk are with the Department of Mathematics, University of Padua, 35121 Padua, Italy (e-mail: conti@math.unipd.it; gangwal@math.unipd.it; hassan@math.unipd.it; elosiouk@math.unipd.it).

Chhagan Lal is with the Department of Validation Intelligence for Autonomous Software Systems, Simula Research Laboratory, 1364 Fornebu, Norway (e-mail: chhagan@simula.no).

Digital Object Identifier 10.1109/COMST.2020.2994526

I. Introduction

THE CURRENT Internet architecture was designed for a small research community over three decades ago with the purpose of interconnecting multiple heterogeneous networks. At that time, nobody foresaw the popularity and longevity that the Internet architecture started gaining in late '80s and early '90s and that led towards the connection of over 3 billion of mobile and desktop devices. Today, people exploit networking devices for a variety of purposes, that go from simple Web browsing to video conferencing and content distribution, with the expectation of being always connected, regardless of their time and place. The misalignment between the original design and the current usage highlighted the limitations of the IP-based architecture and motivated researchers to explore new solutions to overcome them. Among those limitations, the primary concern is the performance of the current Internet, which has to cope with the huge number of connected devices all over the world and with the new pattern of use of the network. According to this study [1], currently there are around 23 billions of connected devices in the world, each one identified by a unique IP address and consuming the network bandwidth. With such a huge number of devices, the first issue is the availability of unique IP addresses to be assigned. Even though researchers originally chose to allocate 32 bits to compose an IP address through the IPv4 protocol, they had to introduce the IPv6 protocol to extend the number of allocated bits from 32 to 128. Network Address Translation (NAT) [2] is also another solution addressing the same problem, and it allows to assign the same public address to a set of devices belonging to the same private network. Thus, when using the private network each device has its own IP address, chosen within a range of private IP addresses, but, for an entry external to the network, all the devices have the same public IP address. To enable the communication between the private network and the Internet a firewall is responsible for intercepting a request, forwarding it to the Internet with the public IP and redirecting the incoming response to the appropriate device.

Another problem is given by the type of network traffic: most of it is made of HyperText Transfer Protocol (HTTP) requests, which means that users have changed the way they use Internet from a low-bandwidth interactive and store-and-forward approach towards a Web and content dominated traffic. To support this, Cisco Visual Networking Index [3] shows that in recent years video traffic delivery has suddenly become very popular on the Internet, with an Internet traffic

that will be 194 exabytes per month by 2021, and multimedia traffic up to 82%, from 70% in 2015. Furthermore, due to the technological advancements in hardware devices and an increasing deployment of pervasive computing application, it is indicated that the number of communicating devices (including smart devices) will be three-times more than the world's population [4]. Moreover, it has also been reported [5] that 86% of worldwide user traffic consists of only video data, which consists of Video on Demand (VoD), video streaming, Point to Point (P2P), and Television (TV).

Finally, from a security and privacy point of view the current Internet is not even able to guarantee some essential requirements, such as origin authentication, data integrity or data confidentiality, because of its lack of security by design. This is the motivation for the introduction of solutions, such as Internet Protocol Security (IPsec) suite [6] or Transport Layer Security (TLS) [7], that work on top of the current Internet and are aimed at overcoming its limitations.

For the above-mentioned reasons, researchers started designing new Internet architecture, such as Recursive INternetwork Architecture (RINA) [8] or ICN [9], that might replace the current one in the future. Among those, the most promising architectures adhere to the ICN paradigm: a new network communication model in which the traditional host-centric paradigm has been moved to the new information-centric networking. While in the current Internet two endpoints can start communicating only if they know the respective IP address, explicitly or by use of a Domain Name System (DNS), in ICN they can send requests specifying only content names, without being aware of contents location in the network. This decoupling between request sending and content transferring introduces several benefits: reduction of latency and network load due to in-network caching [10]–[13], inherent content integrity [14] and better support for mobility due to name-based routing [15], [16].

Due to its benefits and potential next-generation applications, ICN is gaining significant attention from both Industry and Academia, and several works surveyed and evaluated ICN architectures from different perspectives:

- *Architectural components* - Xylomenos *et al.* [17] summarize the core functionalities and discuss the differences, as well as the key weaknesses, of the existing ICN architectures. Ahlgren *et al.* [18] focus their discussion about ICN architectures on the undertaken design choices and features.
- *Network features* - Ioannou and Weber [12] focus on the ICN caching problem and provide an analysis of the existing caching policies, along with the forwarding mechanisms that complement these policies. Abdullahi *et al.* in [19] span their survey through some selected ICN projects by investigating the used caching approaches. In their survey, Zhang *et al.* [20] focus on the state-of-art techniques for reducing cache redundancy and improving the availability of cached content. Zhang *et al.* in [21] survey the caching mechanisms in detail, illustrate how they work, and analyze their possible benefits and drawbacks. Finally, Bari *et al.* in [22] analyze and compare the naming and routing mechanisms proposed by the existing ICN research projects.
- *Security and privacy attacks* - AbdAllah *et al.* in [23] survey the existing literature in the direction of security and privacy in ICN, and discuss some open questions. Tourani *et al.* in [24] provide a summary and a taxonomy of security attacks on ICN architectures, along with their impact on ICN features (e.g., naming, routing, and caching).
- *Software and tools* - In [25], Tortelli *et al.* portray the composition of open source software tools available for ICN, such as ndnSIM, ccnSim, CCNPL-Sim, and Icarus.
- *Application to mobile networks* - Fang *et al.* in [16] present a brief survey of mobile ICN and discuss on the research issues and challenges. Liu *et al.* in [26] highlight the advances in ICN and analyze its development trends, mainly focusing on Information-centric mobile.

However, none of those survey articles discuss the research issues and challenges affecting an ICN-IP coexistence scenario, as we aim to do in this paper. Only in [27], researchers from InterDigital Inc. and Huawei Technologies Co. Ltd. provided a comparison among the existing coexistence architectures, but they focused specifically on the different deployment approaches chosen by each solution.

ICN is also a suitable networking model for various emerging technologies, such as Internet of Things (IoT) [28], [29] and 5G [16], [30]. In the first scenario, ICN can help with establishing the connectivity among smart devices in an IoT environment, as well as in a smart city, in a smart e-health, and in a smart grid context. Also, the management of the huge amount of data generated by IoT devices (i.e., the IoT big data) is challenging in the existing IP architecture, while it is minimized by the in-network caching feature in ICN. This feature allows to reduce the traffic load on data producers by caching data on intermediate routers. Additionally, the receiver-driven communication in ICN allows IoT-receivers to ask for data without revealing their location information, thus being privacy supporting. Similarly, there are various advantages coming up from an ICN-based 5G architecture (i.e., 5G-ICN): (i) 5G-ICN provides a single protocol able to handle mobility and security, instead of using a diverse set of IP-based Third Generation Partnership Project (3GPP) protocols (such as in the case of existing mobile networks, e.g., Long Term Evolution (LTE), 3G, 4G), (ii) it provides a unifying platform with the same layer-3 Application Programming Interfaces (APIs) to integrate heterogeneous radios (e.g., Wifi, LTE, 3G) and wired interfaces in the same network, (iii) it converges services like computing, storage, and networking over a single platform, which enhances the flexibility of enabling virtualized service logic and caching functions anywhere in the network.

Motivation: The benefits of ICN can occur only in a full-ICN scenario, which implies a complete replacement of the current Internet. Despite its obvious need, this is a long and complex process, that requires the coordination among the different parties (i.e., ISPs), time, costs for updating hardware and software of the network components and ability

to face all the new possible challenges. Previous attempts to replace a widely used technology, protocol or architecture (e.g., IPv4/IPv6 protocol, 3G/4G technology, 4G/5G technology) have always faced a long period of coexistence between the old and the new solution. In the same way, the replacement of the current Internet will involve a transition phase during which IP and ICN architectures will coexist. More specifically, we envision that in a coexistence scenario there will be ICN and IP "islands" surrounded by an IP or an ICN "ocean", where an "island" will be a single device, a computer, an application or a server running either the ICN or the IP protocol, while an "ocean" will be a network containing components, that run different architectures.

Researchers working in this field have already addressed the coexistence of IP and ICN following two separate approaches. In the first, the research groups designed future Internet architectures facing the coexistence only during the deployment of their testbeds and without considering it as part of the initial design. On the contrary, in the second case, the design of the future Internet architectures specifically addressed the coexistence of IP and ICN.

All the existing networking solutions that consider the coexistence are affected by a strong limitation: the lack of a comprehensive approach in addressing the coexistence. The purpose of those solutions is to improve a network performance indicator, without considering all the issues that arise in a coexistence scenarios, especially those regarding the security and privacy of the end users. To design the first complete coexistence architecture, it is necessary first to have a comprehensive overview of strengths and weaknesses of the existing solutions.

Contribution: The purpose of this paper is to provide the first complete survey and classification of the existing coexistence solutions. Details of ICN and of its working methodology are out of scope for this paper, since there are already several surveys addressing this aim [16], [24], [31]. Overall, the contributions of this paper are as follows:

1) We define a set of relevant features necessary for comprehensively analyze a coexistence architecture.
2) We provide the first comprehensive classification of all the main coexistence solutions.
3) We discuss the open issues and challenges affecting the existing coexistence architectures, by providing possible insights to design a more reliable future Internet architecture.

Organization: The paper is organized as follows: in Section II, we introduce the ICN concept, by comparing it with the current IP architecture and by illustrating its main benefits; Section III describes all the criteria we identified and used for the analysis and classification of the coexistence architectures; in Section IV we deeply illustrate each coexistence architecture and provide the motivation for our classification; in Section V, we discuss the main strengths and limitations of the current coexistence architectures, providing insights for improving the design of the future Internet; finally, in Section VI we conclude the paper.

II. Background

The purpose of this section is to provide an overview of the ICN paradigm (Section II-A), a comparison of the main features of IP and ICN architectures (Section II-B), the benefits of ICN (Section II-C) and, finally, the emerging technologies (Section II-D).

A. ICN Overview

The ICN concept was first implemented in 2001 in the TRIAD project [32], by introducing a new *content layer* in the IP communication model. This layer provided several content-based features, among which: hierarchical content caching, content replication and content discovery, multicast-based content distribution, and name-based routing. Moreover, the layer supported end-to-end communication based on content name and Uniform Resource Locator (URL) by relying on IP addresses only to reduce the role of transient routing tags. Although TRIAD routing mechanism used content names instead of IP addresses, the Transmission Control Protocol (TCP) and the IP protocols were still the backbone of the proposed architecture. In 2006, UC Berkeley and ICSI proposed the Data-Oriented Networking Architecture (DONA) [33], which improved TRIAD by incorporating data authenticity and persistence as key objectives of the architecture, but still having a strong dependency on the underlying TCP/IP. In 2009, the Palo Alto Research Center (PARC) revealed the Content-Centric Networking (CCN) [34] project. Soon after, the National Science Foundation (NSF) introduced its "Future Internet Architecture" program, which paved the way for Named-Data Networking (NDN) [35] - a branch of the CCN project. Both CCN and NDN significantly moved the TRIAD and DONA projects forward, by introducing a new network layer to definitely replace the existing TCP and IP ones. Thus, CCN and NDN are considered two key projects due to the considerable attention they brought to the ICN paradigm from both Academia and Industry, influencing also the design of the ICN architecture [36].

B. Comparison Between IP-Based and ICN-Based Internet Architectures

Originally developed as part of the ARPANET project [37] during the 1960s, the current Internet is now often referred as TCP/IP architecture due to its most well-known protocols (i.e., TCP and IP). On the contrary, the ICN paradigm was first introduced in the TRIAD project [32] in 2001 and, then, followed by several architectures adhering to its new communication model. Since ICN is a paradigm, we will consider here the five main architectures to describe the technical features of the future Internet, while we will provide a comprehensive description of all the architectures addressing the ICN-IP coexistence in Section IV: (i) the DONA architecture [33], (ii) the CCN architecture [34], (iii) the NDN architecture [35], (iv) the Publish-Subscribe Internet Technologies (PURSUIT) architecture [38], and (v) the Network of Information (NetInf) architecture [39].

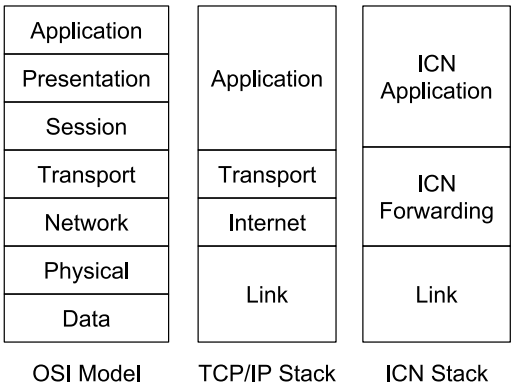


Fig. 1. Adaptation of the OSI seven layer model in the TCP/IP and ICN protocol stacks.

Protocol Stack: Both TCP/IP and ICN rely on a layered protocol stack, which is comparable to the Open Systems Interconnection (OSI) Reference Model [40], as shown in Fig. 1.

The TCP/IP stack includes the following four layers [41]:

- *Application* - it combines the functionality of the *Application*, *Presentation* and *Session* layers of the OSI model. It is responsible for sending and receiving data and it is specific for a particular type of application (e.g., DNS, HTTP).
- *Transport* - it targets the *Transport* layer of the OSI model and it is responsible for the end-to-end data transfer and data streams. Its most important protocols are TCP, which provides a reliable and connection-oriented service, and User Datagram Protocol (UDP), which offers an unreliable and connection-less service.
- *Internet* - equivalent to the *Network* layer of the OSI model, it provides addressing and routing functionalities to ensure the delivery of messages to their destination. IP is the most important protocol, but it does not provide flow control or error handling.
- *Link* - equivalent to the *Data* and *Physical* layers of the OSI model, it manages the interaction among physical network components and it works as an interface with the network hardware.

Since the ICN stack is an evolution of the TCP/IP one [42]–[44], each layer is described with respect to the corresponding one in the Internet stack. More specifically, the layers of the ICN stack are the following ones:

- *ICN Application* - the protocols of this layer address content names instead of hosts locations. For example, the URL inside an HTTP request is replaced with the complete name of a content.
- *ICN Forwarding* - for any ICN-compliant architecture this layer offers routing functionalities for ICN interest and data packets equivalent to the TCP/IP *Network* layer in such a way that source and destination IP addresses are removed from the network packets and only the addressed content name is declared. According to the specific architecture, this layer can also provide the features of the TCP/IP *Transport* layer. In that case, the Interest/Data messages replace the TCP/IP segment/acknowledgement (ACK) messages and the content requester becomes responsible for the message sending rate in place of the content source (producer or intermediaterouter).
- *Link* - to be ICN-compliant, this layer introduces a mapping between Media Access Control (MAC) addresses and content names.

Routing: The purpose of the routing functionality is to route network packets from the source node till the destination node on one way and, then, from the destination to the source on the other.

Each TCP/IP packet specifies both source and destination nodes by including their IP addresses. An IP address is the unique identifier of each network component and it contains both the address of the network and the address of the specific component within that network. In the current Internet, routers are the main responsible for the routing functionality. Equipped with at least two IP interfaces (i.e., an incoming and an outgoing one), each router receives IP packets in the incoming interface and checks whether there is a match, based on the longest prefix, in its Forwarding Information Base (FIB) internal data structure. The FIB contains a mapping between a network prefix and a router's outgoing interface, together with the next-hop IP address. If there is a match in the FIB for the incoming packet, this is forwarded through the outgoing interface towards the next node in the network.

In ICN, the routing functionality differs according to the specific design of each architecture, but they all have a common design choice: the packets sent by a requester contain only the full name of the content and no IP addresses, neither the content requester's one nor the content source's one. In NDN and CCN architectures, contents are expressed through hierarchical names and routers use a longest-prefix match approach to find a possible entry in their FIB, which returns the name-prefix/prefixes of the next node/nodes in the network. On the contrary, DONA exploits a flat naming scheme to point to the contents available in the network and a name-based routing to redirect the packets until they reach the content source. A different approach is used by PURSUIT, which relies on a publish/subscribe model. Publishers publish their contents in the network and subscribers ask for a specific content by using a flat name scheme, made of two components: the Rendezvous Identifier (RI) and the Scope Identifier (SI). The first element addresses the component responsible to find the match between publisher and subscriber for a specific content, while the second is used to identify the sub-network where the rendezvous is. Once the subscriber obtains the location of the publisher from the rendezvous node, it sends its packet to the Topology Manager (TM) of the network where the content publisher is. The TM, then, identifies the path from the publisher to the subscriber and adds a series of Forwarding Identifiers (FIs) to the header of the packets. After that, the Forwarding Nodes (FNs) forward the packets only by using the FIs, without any routing table. Finally, the NetInf architecture adheres to both the approaches: name resolution, based on the publish/subscribe paradigm, and name-based routing.

Name Resolution: In the TCP/IP architecture there is a dedicated network component responsible for the name resolution, which is the DNS. This is a distributed service, which translates domain names, expressed in hierarchical URLs, into the corresponding IP addresses. The Internet is organized into separate DNS zones, each one under the direct control of an authoritative DNS server, and everytime a network device sends a request to its local DNS server, this might reply with a value saved in its cache or, otherwise, forward the same request to a remote server.

In ICN, the name resolution differs according to the chosen forwarding approach. In case of name-based routing, the requester specifies a content by providing its full name, which is the same analyzed by the routers to find the next hop in the network. On the other hand, in the name resolution approach, used by PURSUIT or NetInf, there is always a dedicated node in the network, which is responsible for the mapping between publishers and subscribers.

Storing: In the TCP/IP architecture, routers do not have caching features, while in ICN, caching is fundamental and almost any node is able to cache contents and to serve the corresponding requests.

Traffic Management: In the current Internet, the traffic management, in terms of connection management, flow control and congestion control, is guaranteed by the TCP protocol. The establishment of a connection is regulated by the three-way handshake mechanism, through which the TCP protocol checks for the availability of the remote server, before exchanging any data with it. Only at the end of the handshake, the real communication starts, together with the data exchange, and it is regulated by the introduction of sequence numbers in the message blocks that enable the destination node to properly order all the received messages. The flow control is provided by the ACK messages received by the sender from the receiver every time a packet has been properly delivered. Thus, a sender never overflows the receiving host because the re-transmission of a packet is performed only after a timeout, which corresponds to either an ACK not received by the sender or three ACKs received. Finally, the congestion control refers to the prevention of the routers from becoming overflowed.

In ICN, some architectures, such as DONA, still rely on the existing transport protocols so that all the forwarding mechanisms and transport functionalities are guaranteed. However, other ICN solutions, such as NDN, do not provide the *Transport* layer functionalities and, instead, delegate them to the application itself or to the network packets. After a certain timeout, an application can transmit again a packet, which by design has a limited lifetime to prevent network congestion. Moreover, the availability of distributed caches, which means contents, all over the network should prevent losses due to congestion.

C. Benefits of ICN-Based Architectures

The following ones are the key ICN benefits, which better motivate why this architecture is a potential candidate for the future Internet.

1) Scalable and Cost-Efficient Content Distribution: In a future world where the mobile video traffic will be dominant (e.g., video data will consume more than 80% of the IP traffic, wireless mobile devices will generate two-third of the Internet traffic [45], Netflix and YouTube together amount nearly 50% of Internet traffic), the current network operators will face challenges in meeting the bandwidth requirements from end users. Thus, the inherent ICN support for caching at the network layer [34], together with the receiver-driven mechanism, the inherent support for mobility and the multicast routing, make ICN fit the new network use in a multimedia streaming context [46]–[51].

2) Mobility and Multihoming: ICN also meets the requirements of the 5G network, such as global Internet access and user mobility over dense and heterogeneous networks by adapting to multiple radio access technologies (e.g., Wi-Fi and LTE). As a matter of fact, ICN supports the mobility at the network layer by decoupling time and space between request resolution and content transfer [16]. In particular, two fundamental ICN features encourage seamless consumer mobility [15], [16]. The first is the receiver-driven communication model, where it is up to the consumer to request location-independent contents. The second is the connectionless request/response communication model between consumer and producer. Therefore, when a mobile consumer connects to a new Point of Attachment (PoA), the above two features allow the consumer to re-issue interests for the data that he has not received from the previous PoA. On the contrary, producer mobility is more challenging in ICN because of no distinction between routing locator and content identifier. Previous work have already proposed new solutions for an efficient management of producer mobility in ICN [15], [52].

3) Disruption Tolerance: Achieving an end-to-end communication through TCP/IP transport sessions in challenged networks is often difficult due to the sparse connectivity, high-speed mobility, and disruptions of such networks. Since the application protocol sessions are bound to transport sessions, the communication fails as soon as the transport session fails. In the current Internet, several applications do not require seamless communication with end-to-end paths [53]. As the primary objective is to access data objects, ICN is the perfect approach for Delay-Tolerant Networking (DTN) architectures [54], [55] due to the in-network caching with hop-by-hop transport functionality, which provides a store-and-forward mechanism and enables a better performance and reliability.

4) Security: Unlike the TCP/IP architecture, the ICN design comes with the security in mind. In particular, in ICN the security follows a data-centric model, which focuses on the importance of guaranteeing content integrity and source authentication. For a content-centric architecture, where contents can be located and provided in any point of the network, and not only by the original content producer, the above-mentioned features are particularly significant. To achieve this aim, ICN contents are always signed by the producer, thus allowing consumers to always verify content integrity and data-origin authentication [56].

D. Emerging Technologies

Before thinking of redesigning the whole Internet architecture, researchers and companies have provided several solutions, which work on top of the current Internet, to overcome some of its limitations. Among those, the most successful attempts are the following emerging architectures: Software-Defined Networking (SDN), Network Functions Virtualization (NFV), Content Delivery Network (CDN) and DTN.

1) Software-Defined Networking: SDN [57] is an emerging networking paradigm that separates network control logic (i.e., the control plane) from the underlying switches and routers that forward the traffic (i.e., the data plane). By separating the control and data planes, the network switching/routing devices become simple forwarding devices and the control logic is incorporated in a logically centralized controller. This separation primarily helps in simplifying network (re)configuration, policy enforcement, and evolution [58]. The control plane and the data plane communicate via a well-defined programming interface, i.e., the forwarding elements of the data plane request for instructions from the controller as well as the controller has direct control over the data plane elements using APIs. The most popular flavor of such APIs is OpenFlow [59]. An OpenFlow switch has one or more flow tables for handling packet-rules. When a rule matches with the incoming traffic, the OpenFlow switch performs certain actions (forwarding, modifying, dropping, etc.) on the traffic flow. The rules installed by the controller decide the role of an OpenFlow switch, i.e., it can behave as a switch, router, firewall, or middlebox (such as traffic-shaper, load-balancer).

2) Network Functions Virtualization: Diversity and dominance of proprietary appliances made service deployment, as well as testing, complex. NFV [60] was designed as a technology to leverage Information Technology (IT) virtualization by exporting network functions from the underlying dedicated hardware equipment to general software running on Commercial Off-The-Shelf (COTS) devices. Using NFV, the key network functions can be performed at various network locations, e.g., network nodes, data-centers, network edge, as required. NFV is different from SDN, and it only deals with the virtualization of network functions.

3) Content Delivery Network: The initial implementation of the Internet was designed to manage the traffic in a passive, end-to-end, and "best effort" approach [61]. With the explosion of user data and commercial content over the Internet, the "best effort" approach for traffic management became inefficient and unscalable. To handle this situation, CDN [61]–[63] was designed [45], [64], [65]. Nowadays, CDN appears as an integral and essential overlay network for the Internet [66]–[68] since it primarily aims to improve bandwidth availability, accessibility, and precise content delivery through content replication.

CDN architecture consists of several cache servers that are strategically located across the Internet. Typically, CDN holds a hierarchy of servers with multiple Points-of-Presence (PoP) that stores copies of identical content to satisfy user's demand from most appropriate/closest site [69]. It also has back-end servers for intra-CDN content distribution. CDN categorically distributes Web contents to the cache servers, which are positioned close to the users. As a result, CDN offers fast, efficient, and reliable Web services to the users.

There are two fundamental approaches for the deployment of CDN: (i) overlay model, where content is replicated to thousand of servers worldwide, and (ii) network model, where routing configurations recognize the application services and forward them based on the predefined policies.

Even though CDNs improve content delivery, their performance is limited by the underlaying ISPs. Usually, CDNs do not manage independent packet data services, rather they rely on the ISPs to make packet routing decisions. Moreover, both ISPs and CDN collectively provide end-to-end Quality of Experience (QoE)[1] for content delivery. Thus, coordination between ISPs and CDN providers causes a massive impact on the overall QoE [66].

4) Delay-Tolerant Networking: In the late 1990s, the widespread use of wireless protocols, together with an increasing interest in vehicular communication, encouraged researchers to design the Interplanetary Internet (IPN) architecture. This was the first attempt to address the need of long distance communications that were inevitably affected by packets loss/corruption and delays. DTN [70] was first introduced as an adaptation of the IPN for terrestrial networks [71]: it is an overlay architecture that operates above the protocol stack of *ad-hoc* wireless networks and enables gateway functionality to interconnect them. To provide communication among networks having excessive delays due to highly repetitive link disruptions, DTN adopts the "store-carry-forward" routing scheme [72]: the main idea of this scheme is to have multiple nodes distributed over the network, each one able to receive a copy of the same message and then send it back to the destination node. This way, the delivery performance is improved and the destination node can receive the message from any location inside the network.

III. Coexistence Architectures: Features and Evaluation Parameters

In order to classify the existing architectures, we identified the necessary features and evaluation parameters to have a complete overview of each coexistence solution. The former come with the design of a coexistence architecture, while the latter refer to the challenges introduced during its deployment in a real scenario. The features are as follows: *deployment approaches*, *deployment scenarios*, *addressed coexistence requirements* and *additional architecture or technology used*. On the other side, the evaluation parameters are: *traffic management*, *access control*, *scalability*, *dynamic network management* and *latency*. In the remaining part of this section, we will describe features (Section III-A) and evaluation parameters (Section III-B) used for analyzing each coexistence architecture.

A. Features

1) Deployment Approaches: The deployment of ICN into the TCP/IP architecture inevitably raises the following

[1]QoE is an all-inclusive model, which defines the quality perceived by a user when retrieving content or applications over the Internet.

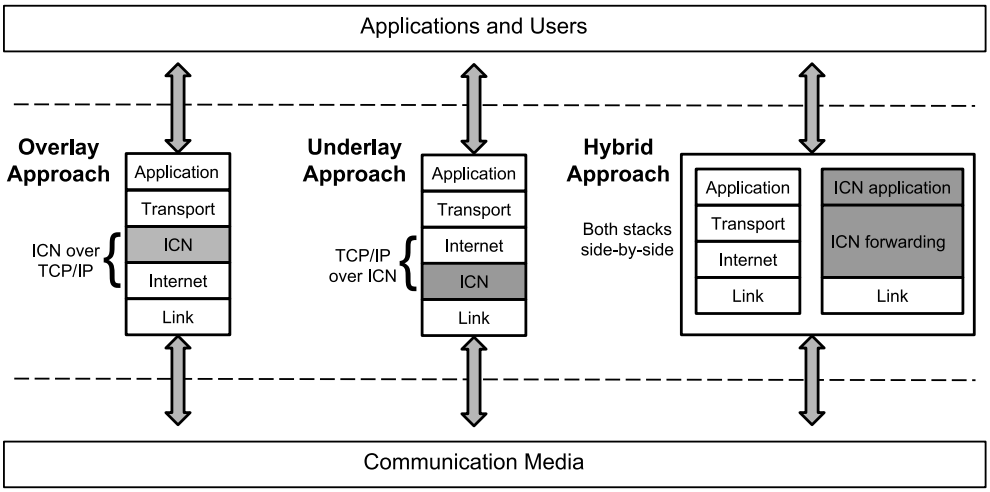


Fig. 2. Deployment approaches of ICN into the TCP/IP architecture.

question: *How to introduce the ICN protocol into the TCP/IP protocol?* To achieve this aim, researchers identified three possible approaches, shown in Fig. 2: *overlay* in case of ICN running on top of the IP protocol, *underlay* in case of ICN running under the IP protocol and *hybrid* in case of a coexistence of both IP and ICN protocols [27]. In the *overlay* deployment approach, the aim is to enable the communication among several ICN "islands" in an IP "ocean" and is achieved through a tunnel over the Internet protocol. On the contrary, the *underlay* solution involves the introduction of proxies and protocol conversion gateways near to either ICN or IP "islands" to properly deliver and receive outgoing and incoming requests. As an example, an HTTP request sent to an ICN "island" is intercepted by a gateway, which is responsible for translating it into an ICN Interest. Then, the resulting ICN data packet is translated again into an HTTP reply sent back to the requester. Finally, the *hybrid* approach claims the coexistence of both ICN and IP, by adopting dual stack nodes able to handle the semantics of both IP and ICN packets. Given the diversity of the two protocols, from a semantic and format point of view, a dual stack node can use various options to infer content names from an IP packet, such as performing deep packet inspection in the payload or looking into the content name in the IP option header.

2) Deployment Scenarios: The purpose of this feature is to analyze all the possible scenarios in which a coexistence architecture can be deployed among the others we identified and that are illustrated in Fig. 3.

Each deployment scenario involves two "islands", which run either the same networking architecture or two separate ones, surrounded by an ICN or an IP "ocean". The possible different deployment scenarios are as follows:

- *ICN-ICN communication in IP "ocean".*
- *ICN-IP communication in IP "ocean".*
- *ICN-IP communication in ICN "ocean".*
- *IP-IP communication in ICN "ocean".*
- *Border Island* - communication between different "islands" in separate "oceans".

3) Addressed Coexistence Requirements: In a coexistence scenario, the heterogeneity of the different networks might

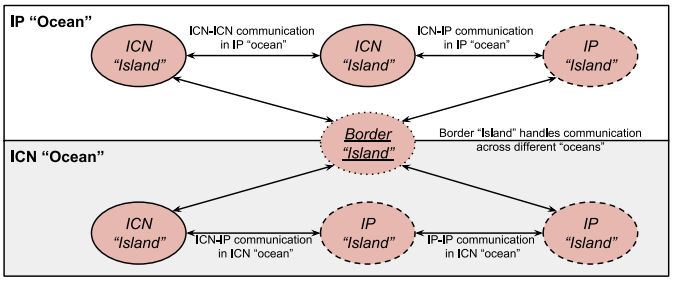


Fig. 3. Deployment scenarios for a coexistence architecture.

generate conflicts that prevent each individual architecture from guaranteeing its main features and properties. For example, since most of the ICN architectures do not preserve the native transport functionalities provided by the TCP protocol of the current Internet, one of their most significant limitations is the traffic management. In a coexistence scenario, there would be a conflict between an IP "island" implementing its own logic for managing the traffic network and an ICN "island", which does not support the same features.

Examining previous works [73], we consider the following requirements as the necessary ones to be supported in a coexistence scenario:

- *Forwarding* - the network forwarding devices should be able to handle packets with diverse routing identifiers (e.g., the variable-lengths of content names lead to dissimilar size of prefix-set and thus, different forwarding table look-ups).
- *Storage* - the network devices should support in-network caching to serve the content request and reduce bandwidth consumption. Nevertheless, the storage capacity of network devices also affects the size of the index table for the cached content and the time required to match the content name in the index table.
- *Security* - the network devices should preserve the security policies enforced in one (source) network to another (destination) network such as authenticating the digital signatures of content objects for content-based security or privacy policies.

- *Management* - the network devices should support management-related operations such as traffic-shaping/engineering, load-balancing, and explicit path steering.

4) Additional Architecture or Technology Used: ICN and IP are not the only architectures that can coexist, and even the coexistence could be improved using other technologies. More specifically, ICN well fits with several different technologies that are already deployed in the current Internet infrastructure. Among those, there are SDN, NFV or CDN. The purpose of this feature is to collect all the architectures that the coexistence solutions involve.

B. Evaluation Parameters

As evaluation parameters, we considered the following challenges arising during the deployment of a coexistence architecture in a real scenario:

- *Access control* - in a networking context, access control uses a set of protocols to define, implement, and maintain policies that describe how the network nodes can be accessed by users/devices. Typically, it includes:
  - Authorization, authentication, and accounting of network connections.
  - Identity and access management.
  - Mitigation of non-zero-day attacks.
  - Policy lifecycle management.
  - Role-based controls of user, device, application.
  - Security posture check.
- *Scalability* - it ensures that the overall performance of a network will be not affected by the size of the network. In other words, scalability describes the ability of a network to grow and manage increasing demand.
- *Dynamic network management* - it is the process of administering and managing dynamic changes in computer networks, such as topology changes and handovers for seamless host mobility.
- *Latency* - it is defined as the amount of time a message takes to traverse a system. In a computer network, it is typically measured as the time required for a packet to be returned to its sender. The major factors for the network latency include propagation delays and delays due to routers, as well as storage devices.
- *Traffic management* - for a detailed description of the traffic management, we refer to Section II-B.

IV. Classification of the Coexistence Architectures

The purpose of this section is to illustrate the classification of the coexistence architectures according to the features and the evaluation parameters described in Section III. The summary of our findings is listed in Table I. Moreover, some of the architectures presented in this section has also shown their functional effectiveness and deployment feasibility with the help of real-world testbed deployments, we discuss the details about such those architectures in Section V-B.

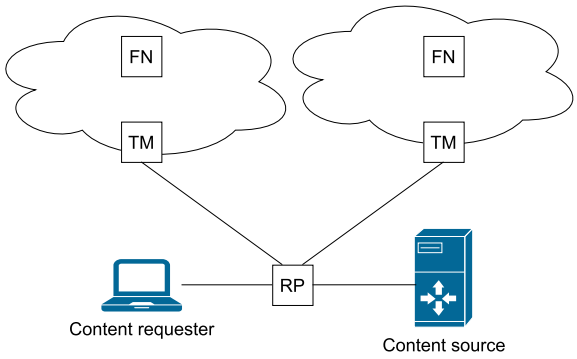


Fig. 4. Simplified view of the PURSUIT architecture.

A. PURSUIT

PURSUIT [74] was a European project financed by the Seventh Framework Program (FP7), started in September 2010 and ended in February 2013. PURSUIT is an evolution of the FP7 project Publish-Subscribe Internet Routing Paradigm (PSIRP) [38], proposing an ICN model based on a source node, that publishes an information, and on a client node, that subscribes to the content it desires. If the information is available, it will be delivered to the client. PURSUIT aims at improving PSIRP, meanwhile evaluating its performance, scalability, and coexistence with the current Internet network. Fig. 4 shows a simplified form of the architecture proposed in PURSUIT project.

PURSUIT architecture relies on the definition of a new data format and on the introduction of three new network components. PURSUIT addresses the data as information items, which consist of pair of identifiers, i.e., RI and SI. The former represents the real piece of information, while the latter specifies the group which the information belongs to. The three additional network functions addressed by PURSUIT are: Rendezvous Function (RF), Topology Function (TF), and Forwarding Function (FF). The RF plays a fundamental role in PURSUIT since it maps subscribers to publishers and supports names resolution. Moreover, it also initializes the delivery of information item to the client. The *Rendezvous Point (RP)* performs the RF and relies on a hierarchical distributed hash table internal data structure. The *TM* implements the TF by deploying a routing protocol to collect the topology of its domain and by exchanging routing information with other domains for global routing. The *Forwarding Node (FN)* implements the FF and it is also responsible for redirecting the information item to the client. In particular, the forwarding mechanism is label-based and uses a bloom filter [86] to speed up the information delivery. In addition, the *FN* offers also a caching facility.

As shown in Fig. 5, the PURSUIT node internal architecture encompasses several components, enabling the publish/subscribe communication model among the different stack layers. The *IPC Elements* implement a non-blocking inter-process mechanism, allowing users-space applications to issue publish/subscribe requests and communicate through the proposed prototype. The functionality of the *Local Proxy* element is to maintain a local record for all the pending subscriptions and, after receiving a request, dispatch

TABLE I
CLASSIFICATION OF THE COEXISTENCE ARCHITECTURES (✓ ADDRESSED ✗ NOT ADDRESSED)

Parameter			PURSUIT [75]	NetInf [40]	NDN [76] & CCN [35]	O-ICN [77]	CONET [78]	GreenICN [79]	coCONET [80]	DOCTOR [81]	POINT [45]	RIFE [82]	CableLabs [83]	NDN-LAN [84]	hICN [85]	OFELIA [86]
Duration of the project/ Year of publication			2010 to 2013	2010 to 2013	2010 till today	2015	2010 to 2013	2013	2012	2014 to 2017	2015 to 2017	2015 to 2018	2016	2017	2018	2012
Features	Deployment approaches	Overlay	✓	✓	✓	✓	✓	✓	✓		✓		✓			
		Underlay								✓		✓			✓	✓
		Hybrid					✓							✓	✓	✓
	Deployment scenarios	ICN-ICN communication in IP "ocean"	✓	✓	✓	✓	✓	✓	✓	✓			✓	✓	✓	
		ICN-IP communication in IP "ocean"						✓	✓	✓			✓	✓	✓	
		ICN-IP communication in ICN "ocean"								✓			✓	✓	✓	
		IP-IP communication in ICN "ocean"								✓			✓	✓	✓	
	Addressed coexistence requirements	Border Island				✓	✓			✓	✓	✓			✓	✓
		Forwarding	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
		Storage	✓	✓	✓	✓	✓		✓	✓	✓	✓	✓	✓	✓	✓
		Security	✓	✓	✓				✓	✓	✓	✓			✓	✓
		Management					✓	✓	✓	✓					✓	✓
	Additional architecture or technology used		PSIRP LAN			SAIL SDN		SDN	SDN	NFV SDN	PURSUIT SDN	PURSUIT DTN	CDN	LAN	DNS	CONET SDN
Evaluation parameters	Other	Traffic management	✗	✗	✗	✗	New IP option overhead.		ICN capable OpenFlow-compliant network.				✗	✗		OpenFlow-compliant networking elements.
		Access control		✗												
		Scalability				✗		✗			✗			✗	✗	
		Dynamic network management				✗					✗			✗		
		Latency						SDN controller must manage every ICN request and rewrite several headers fields for every response packet.		✗	✗		Optimization of CCN router, cache/content implementation, protocol translation between CCN and HTTP.	✗	✗	
		NetInf transport functions Interaction.														

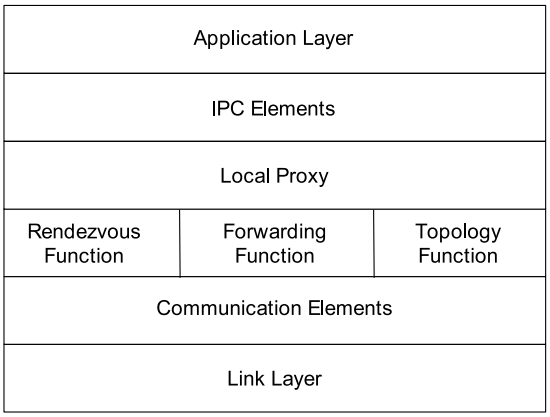


Fig. 5. Internal architecture of a PURSUIT node.

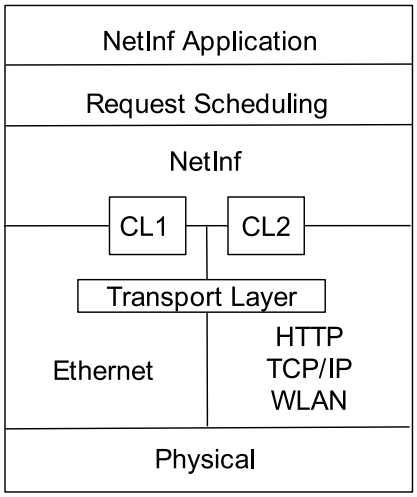


Fig. 6. Internal architecture of a NetInf node.

it to the appropriate functions (i.e., *RF*, *FF*, *TF*). Finally, the *Communication Elements* are responsible for transmitting publications to the network. The design implementation of PURSUIT is based on Click elements [87]: it creates Ethernet frames and forwards them to the appropriate network interface. In addition, it provides the ability to utilize raw IP data packets as an alternative mechanism. This enables the prototype to be tested in Internet-wide scenarios.

Deployment Approach: Trossen and Parisis [74] implemented a Layer-2 Virtual Private Network (VPN)-based *overlay* solution of PURSUIT among multiple nodes located in Europe, U.S. and Asia. The prototype is established and verified on three different testbeds for experimental purposes, functioning as an overlay on LAN environment. To showcase a specimen of native ICN application, multimedia streaming services were hosted as a demonstration, showing a lossless transmission and comparable performance.

Deployment Scenarios: The *ICN-ICN communication in IP "ocean"* is the most suitable scenario for deploying PURSUIT, as it is also confirmed by the *overlay* approach adopted in the testbed.

Addressed Coexistence Requirements: PURSUIT guarantees the following three coexistence requirements:

- Forwarding - this is specifically provided by the *FN*, a software-based forwarder used for ICN messages exchange.
- Storage - the *FN*, which has the responsibility of redirecting information to the client, provides caching facility to furnish storage of information.
- Security - the security measures provided by PURSUIT refer to the access of information. Besides gathering information into groups, PURSUIT supports the information categorization into scopes, used for the definition of access privileges and policy implementations.

Additional Architecture or Technology Used: PURSUIT is an evolution of the PSIRP project and its testbed has been realized as an *overlay* solution over a Local Area Network (LAN) environment.

Evaluation Parameters: The main issue introduced by the *overlay* deployment in the PURSUIT architecture is the traffic management. This is mainly due to the existing Internet applications and protocols, which are not completely compatible with the techniques implementing ICN over TCP/IP or UDP [9], [35], [88], [89] for traffic transport. Thus, many applications and protocols, such as HTTP based multimedia streaming protocols, might face false throughput estimations [90]. This is due to the TCP aggressiveness in presence of variations in content source location (e.g., dynamic caching and interest aggregation) [91].

B. NetInf

The NetInf architecture [39] is the approach proposed by the European FP7 project SAIL [92], started in January 2010 and ended in February 2013. The key component of the NetInf architecture is the Convergence Layer (CL), which is able to map the information, expressed through any protocol (e.g., HTTP, TCP, IP, Ethernet), into specific messages compliant to a general communication paradigm. In particular, when two nodes communicate between each other, the functionality of a CL is to provide framing and message integrity to NetInf requests and responses.

Fig. 6 depicts the different CLs designed within the NetInf stack. In particular, CLs encompass an additional function (i.e., *Request Scheduling*) between the *NetInf Application* and the *NetInf Protocol*. The *CL1* functions over Ethernet, while *CL2* makes NetInf able to function over a variety of networks links and protocols such as HTTP, TCP/IP, Wireless Local Area Network (WLAN). The CLs also provide transport layer functions across different nodes such as flow control, congestion control and reliability.

Deployment Approach: NetInf adheres to the *overlay* deployment approach, as it is confirmed by its first prototypes, deployed as an *overlay* strategy over TCP/UDP.

Deployment Scenarios: The NetInf architecture supports the *ICN-ICN communication in IP "ocean"* scenario.

Addressed Coexistence Requirements: The coexistence requirements provided by NetInf are as follows:

- Forwarding - NetInf guarantees both name-based forwarding and name resolution; NetInf message forwarding protocol relies on the lower-layer networking technology

(e.g., TCP connection between two Internet hosts) and this communication is provided by the CLs.
- Storage - NetInf nodes support both on-path and off-path caching.
- Security - the CLs are responsible for the integrity of the messages exchanged in the architecture.

Additional Architecture or Technology Used: Besides the standard TCP/UDP/IP tunneling, which is part of the *overlay* approach, NetInf does not rely on additional architectures.

Evaluation Parameters: The deployment of the NetInf architecture in a coexistence scenario introduces the following challenges: traffic management, due to the absence of interaction among the CLs transport functions and the NetInf transport functions, and access control. The first issue refers to the CLs, which are responsible for the interconnection of different types of networks into a single ICN network. For example, the interaction among the underlying protocols that provide really different communication services creates new challenges (e.g., from uni-directional, opportunistic message forwarding to flow- and congestion-controlled higher layer communication services; from delay-challenged to high-speed optical backbone networks). Concerning the access control limitation, in NetInf, it is not possible to apply controls over the accessibility levels of the information. Thus, anyone can access the published data without any restriction.

C. NDN and CCN

Among the existing implementations of the CCN paradigm [34], funded by the NSF [93] as part of the Future Internet Architectures program, there is the NDN research project [75]. From its first design late in 2010, the NDN main idea is to shift the existing IP host-to-host communication into a data oriented one by leveraging on an increased responsibility of the routers. Upon receiving a request for a content, the routers first check whether the content is already present in their cache (i.e., Content Store). If this is the case, they immediately return the content back, otherwise, they check the Pending Interest Table (PIT), searching for a pending request issued for the same content. If the PIT already contains an entry for the specific content, routers just collapse the current request into the PIT. If none of the previous cases verifies, routers forward the request to the next node in the network using the FIB, and keep waiting for the associated data to return back. Once the data packet arrives, all the pending interests for that content are satisfied just by sending the copy of data back to all the hosts which have requested it.

As shown in Fig. 7, NDN introduces some changes into the IP stack by adding the *Security* and *Strategy* novel layers: the first refers to the NDN design addressing the security of the content instead of the security of the communication channel between two nodes (which is how IP works); the second substitutes the network layer and provides the forwarding plane to forward *Content chunks* by giving the best choices to maintain multiple connectivities under varying conditions. In addition, the *Strategy* layer also supports security, scalability, efficiency and resiliency. Finally, NDN modifies the *Transport Layer* making it consumer-driven instead of producer-driven [94], [95], importing it into the NDN forwarding plane.

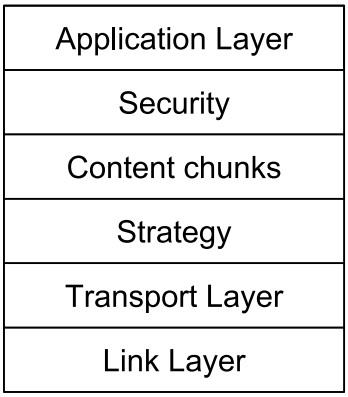


Fig. 7. NDN network stack [35].

Deployment Approach: The common implementation of NDN and CCN includes *overlay* protocols, such as CCNx [9] and NDNLP [89], which are deployed over existing IP infrastructure. For instance, CCNx [88] showcases the explicit example of overlay by implementing CCN-over-UDP. In particular, it provides a method to transport CCNx messages between two nodes over UDP. Moreover, a concrete example of NDN overlay architecture is provided by the ndn-testbed,[2] which connects multiple NDN nodes located in several continents over existing TCP/IP. The services provided in the trials of CCN/NDN include various projects, such as real-time video-conferencing [96], adaptive bit-rate streaming (not limited to end-to-end) [46], [48], [50] and ndnSIM (NDN simulator module on NS-3) [97].

Deployment Scenarios: NDN supports the *ICN-ICN communication in IP "ocean"* scenario, as it is confirmed by the ndn-testbed.

Addressed Coexistence Requirements: NDN guarantees the following three coexistence requirements:
- Forwarding - the router's FIB is responsible for forwarding interests towards the content provider via one or more network interfaces based on the routes to the origin node(s). The requested data packet is then forwarded towards the requester by simply traversing, in reverse, the path of the preceding interest [35]. NDN supports also the multicast data routing, which improves receiver-driven multimedia delivery.
- Storage - NDN routers are enabled to cache contents.
- Security - NDN provides a data-centric security model where each data unit is uniquely signed by the data producer [98].

Additional Architecture or Technology Used: Besides the standard TCP/UDP/IP tunneling, which is part of the *overlay* approach, the NDN project does not rely on additional architectures.

[2]https://named-data.net/ndn-testbed/

Evaluation Parameters: The tunneling approach, where NDN/CCN endpoints communicate over IP [99], [100], disowns the fundamental advantages of the content oriented networking (i.e., in-network caching and multicast forwarding) and the architectures implementing hop-to-hop connectionless (/oriented) connectivity (i.e., over TCP/UDP) suffer from a lack of traffic management [91]. In NDN/CCN networks, Congestion Avoidance (CA) is operated by the consumer rather than by the producer (server). This means that the Interests transmission rate is adapted in order to ensure that the delivery of a requested resource can make maximum fair use of the network. Existing NDN/CCN CA algorithms are largely based on the TCP CA algorithms, which assume that the bandwidth-delay product of the network fluctuates relatively slowly, as all the data packets traverse the same path from server to client. However, in NDN/CCN network content objects may be retrieved from various locations and may reach the consumer through different paths. Thus, the concept of a bandwidth-delay related to a single path and the use of TCP CA algorithms do not fit for NDN/CCN networks. In the NDN/CCN community, this is an active research area [101].

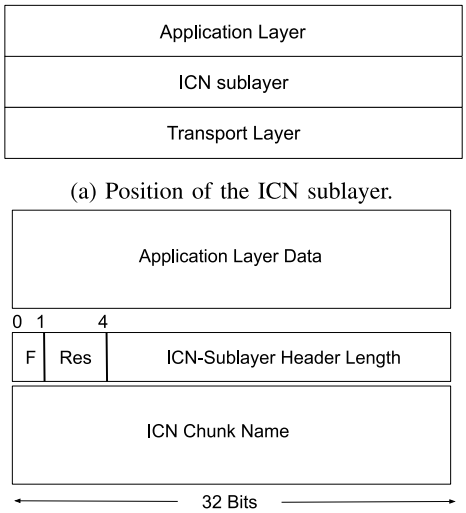
D. O-ICN

Overlay for Information-Centric Networking (O-ICN) [76] is a novel architecture, which leverages the SDN technology for separating data plane activities (i.e., forwarding and storing/caching of ICN contents) from control plane activities (i.e., naming, name resolution and routing). In particular, O-ICN introduces the ICN Manager as an extended version of a DNS server, which performs name resolution for both ICN and non-ICN requests. In case of an ICN request, the ICN Manager identifies the source of the content and sends to it the user's address, so that the source can route back the requested content to the user. In case of a non-ICN request, the standard routing mechanism of TCP/IP is followed. The naming scheme adopted by O-ICN is hybrid, i.e., both human readable and self-certifying as in the SAIL architecture [92]. Finally, the existing routers are modified to cache contents and communicate with the ICN Manager.

Fig. 8(a) depicts the position of the novel *ICN-sublayer* proposed by O-ICN, which lies between the TCP/IP *Application Layer* and *Transport Layer*. More specifically, Fig. 8(b) describes the fields used by the new layer: the ICN flag bit (*F*), equal to 0 for an ICN request or to 1 for an ICN content; the three subsequent bits (1-4) reserved for additional purposes, and the remaining 28 bits for the total ICN header [102].

Deployment Approach: O-ICN relies on an *overlay* deployment solution by leveraging on the ICN Manager, which performs dual tasks: name resolution, along with routing functionalities for ICN requests, and standard DNS resolution for the existing Internet requests. To evaluate the O-ICN architecture, authors in [102] present the Overlay ICN simulator (OICNSIM),[3] an ns-3 based simulator where each O-ICN component is provided with helper classes and it is able to satisfy a wide variety of deployment scenarios. As an example, in [102], the authors studied the performance of OICNSIM for different ICN caching policies.

[3]https://www.nsnam.org/wiki/Contributed_Code



(a) Position of the ICN sublayer.

(b) Detail of the ICN sublayer header format.

Fig. 8. Internal architecture of an O-ICN node.

Deployment Scenarios: O-ICN supports the *ICN-ICN communication in IP "ocean"* scenario. Moreover, thanks to the ICN manager capability of manipulating both ICN and not-ICN requests, O-ICN can support also the *Border Island* deployment scenario.

Addressed Coexistence Requirements: The coexistence requirements addressed by O-ICN are as follows:

- Forwarding - the ICN Manager is responsible for the forwarding strategy.
- Storage - the data plane activities involve tactical storing/caching of ICN contents at different locations/routers/gateways and are managed by ICN routers.

Additional Architecture or Technology Used: O-ICN exploits the SAIL solution for the naming scheme and the SDN technology for a separate management of data plane and control plane activities.

Evaluation Parameters: As for the previous *overlay* approaches, O-ICN is affected from a lack of traffic management. In addition, the overall solution suffers from scalability problems and the ICN manager is not able to guarantee its DNS functionalities in case of dynamic network conditions.

E. CONET

CONET [77] is an architecture designed for connecting several *CONET Sub System (CSS)*, which could be the whole Internet network, an IP autonomous system or a couple of network connected components. The main components of the CONET design, shown in Fig. 9, are as follows: *End-Node (EN)*, *Serving-Node (SN)*, *Border-Node (BN)*, *Internal-Node (IN)*, and *Name-System-Node (NSN)*. An *EN* requests some named-data by issuing an interest routed by the *BNs*, which are located at the border of *CSSs*. The route-by-name process identifies the *CSS* address of the next *BN*, which is closest to the *SN* as soon as the appropriate *CSS* is

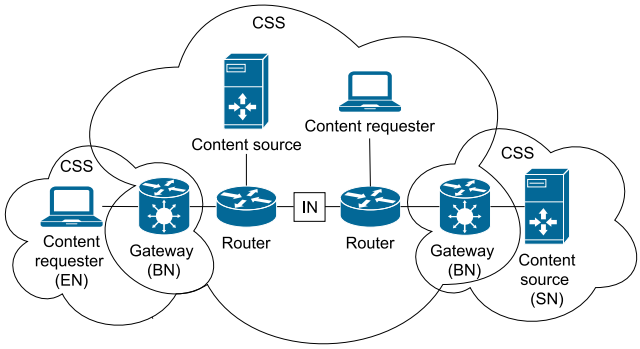


Fig. 9. Simplified view of the CONET architecture.

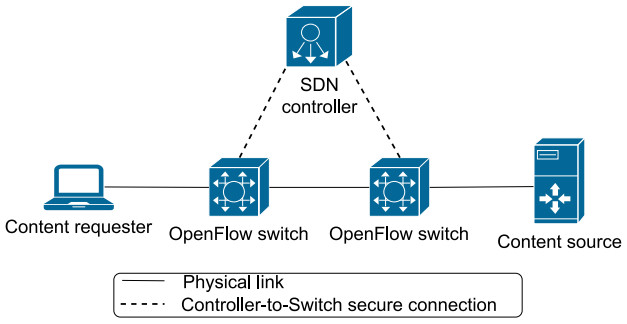


Fig. 10. Simplified view of the GreenICN architecture.

reached. Then, the *INs* forward the packet using the under-CONET routing engine. The *CSS* address of *EN* and the *CSS* addresses of the traversed nodes are appended to the packet. As soon as a CONET node is found to be able to provide the requested named-data, this is sent back on the reverse path to serve the requesting *EN*. All *BNs* and *INs* along the traversed path may cache the content.

Deployment Approach: The CONET architecture can follow either an *overlay* or a *hybrid* deployment approach. In the first case, CONET works on top of the IP layer and the *CSSs* are nodes connected by overlay links (e.g., UDP/IP tunnels). In the second approach, the purpose is to make IP content-aware by introducing a novel IPv4 option or an IPv6 extension header. The network components will have then hybrid routing tables with both IP network addresses and names.

Deployment Scenarios: Considering the *overlay* solution, CONET supports the *ICN-ICN communication in IP "ocean"* scenario. On the contrary, the *hybrid* approach allows it to be deployed in the *Border Island* scenario as well.

Addressed Coexistence Requirements: CONET guarantees the following three coexistence requirements.

- Forwarding and Management - these are guaranteed by *BNs* and *NSNs*. In addition, *ENs* provide transport-level functionalities such as reliability and flow control. Since the logic for requesting a content involves sending separate interests, containing a small part of the named-data, the control of interest sending rate can be used as a TCP-like flow control mechanism.
- Storage - *BNs* are able to store contents.

Additional Architecture or Technology Used: Besides the standard TCP/UDP/IP tunneling, which is part of the *overlay* approach, the CONET project does not rely on additional architectures.

Evaluation Parameters: The *hybrid* deployment solution is hard to be introduced since it requires a new IP option. However, with respect to the *clean-slate* approach, the *hybrid* one is less disruptive, and it allows the architecture deployment in different scenarios.

F. GreenICN

The SDN technology decouples control plane from data plane, and it provides a programmable, centrally managed network control that improves network performance and monitoring. SDN-based implementations of ICN exploit the centralized view available to SDN controller, which enables the SDN controller to install appropriate forwarding rules for ICN requests/responses in such a manner that the network elements only have to support IP forwarding. Vahlenkamp *et al.* in [78] proposed an implementation of ICN using SDN under their GreenICN project. The proposal leverages ICN protocol's Message IDs and features of SDN instantiations such as OpenFlow to rewrite packet header information. Fig. 10 presents a simplified view of this solution. Here, both the *Content requester* and the *Content source* are connected to *OpenFlow-enabled switches* that are managed by the *SDN controller*. Routing information for the content requests and responses, upon arriving on OpenFlow switches, is handled/rewritten by the instructions from the controller.

Deployment Approach: The proposed solution is an *overlay* ICN implementation as ICN data is sent over the SDN-managed IP packets.

Deployment Scenarios: Essentially, the authors in [78] propose ICN deployment over IP network, where an ICN-aware content source delivers the content to an ICN-aware requester over IP network. Hence, this solution supports both the *ICN-ICN communication in IP "ocean"* and the *ICN-IP communication in IP "ocean"* scenarios.

Addressed Coexistence Requirements: The architecture addresses the following coexistence requirements:

- Forwarding - network programmability offered by SDN enables forwarding and routing for ICN.
- Management - SDN centrally managed network control supports load-balancing, traffic engineering, and explicit path steering (e.g., through ICN caches).

Additional Architecture or Technology Used: The authors argue that an ideal or native deployment of ICN, in which user devices, content sources, and intermediary network elements are ICN aware, may not be viable. Hence, the authors proposed to implement ICN-awareness in the SDN-enabled switches, where ICN packets are carried over the IP transport protocol. By using SDN, the authors target all the services/applications of the TCP/IP protocol stack.

Evaluation Parameters: In the proposed ICN implementation, SDN controller must manage every ICN request and rewrite several headers fields for every response packet, which might not scale with increased network size. Given that this

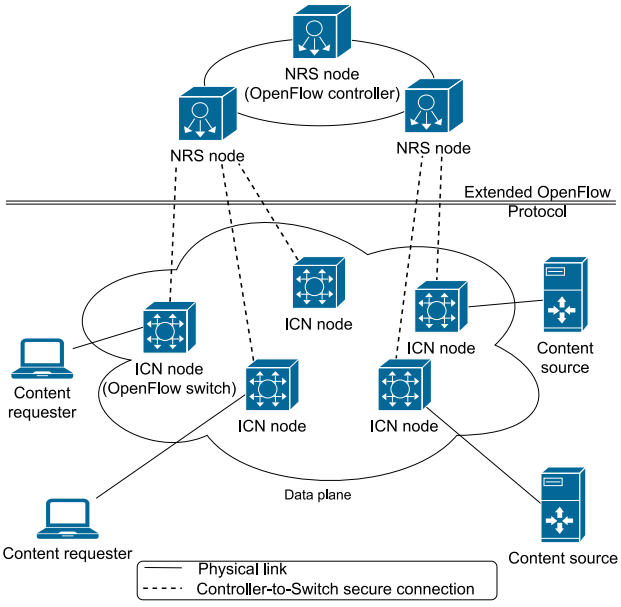


Fig. 11. Simplified view of the coCONET architecture.

solution is based on the widely accepted SDN technology - that supports agile deployment and rapid alternation in networking - the hardware modifications required for its deployment are low in those scenarios where SDN infrastructure already exists. Consequently, the time required for its deployment is also low. Nevertheless, the time and the hardware modifications required for its deployment would be higher if the SDN infrastructure does not already exists.

G. coCONET

Similar to the work [78], Veltri *et al.* [79] proposed a CONET [77] inspired SDN-based implementation of ICN, called coCONET. Fig. 11 presents a simplified view of this solution. In this architecture, ICN nodes and user-terminals form the data plane and *Name Resolution Service (NRS)* nodes are placed in the control plane. Moreover, *ICN node* works as an OpenFlow switch, while *NRS node* works as an OpenFlow controller. To this end, the authors proposed to extend the OpenFlow protocol [59].

Deployment Approach: Similar to the work [78], the proposed solution is an *overlay* ICN implementation as ICN data is encapsulated inside the SDN-based IP packets.

Deployment Scenarios: The proposed solution enables the *ICN-ICN communication in IP "ocean"* and the *ICN-IP communication in IP "ocean"* scenarios, where the underlying IP network is managed by OpenFlow-based SDN network.

Addressed Coexistence Requirements: The present architecture provides the following coexistence requirements:

- Forwarding and Management - SDN-based operations of the proposed approach support both forwarding and management of ICN traffic.
- Storage - ICN capable nodes cache the contents.
- Security - contents are cryptographically protected in order to assure content (and content generator) authentication and data integrity. This security service is provided through digital signature and can be verified through the public key associated to the private key of the content (or of the content generator). The proposed system enforces every ICN node to verify such signature before forwarding the content toward the interested end-nodes, to protect the network against Denial of Service (DoS) or other attacks.

Additional Architecture or Technology Used: Here, the authors focus specifically on OpenFlow-based SDN implementations and target all the services/applications of the TCP/IP protocol stack. OpenFlow is a flavor of SDN.

Evaluation Parameters: The proposed solution requires ICN capable OpenFlow network devices for ICN operations. Due to such specific requirements, the hardware modifications and the time required for its deployment are high.

H. DOCTOR

DeplOyment and seCurisaTion of new functiOnalities in virtualized networking enviRonments (DOCTOR) [80] is an ongoing project funded by French Nation Research Agency. The project provides support towards the adoption of new standards by developing a secure use of virtualized network equipment. This leads to ease the deployment of novel networking architectures, thus enabling the coexistence of IP and emerging stacks, such as NDN, as well as the progressive migration of traffic from one stack to the other. DOCTOR proposes the use of NFV infrastructure to achieve the incremental deployment of NDN at a low cost. The project proposes an HTTP/NDN gateway to interconnect ICN "islands" to the IP world, and an experimental architecture able to process the Web traffic passing through a virtualized NDN network.

In particular, DOCTOR first deploys a virtual network based on OpenvSwitch to provide an end-to-end network connectivity between the virtualized network services and to enable a software control of the networking infrastructure. Then, it selects NDN as an ICN protocol stack. More specifically, the NDNx software is *dockerized* to become a Virtualized Network Function (VNF), deployable in DOCTOR architecture. In DOCTOR, NDN is used both over IP and over Ethernet since most NFV tools are still IP-dependent. To test the functionality of the coexistence, the Web is considered as an application layer service due to its high popularity and predominance in the global network shares. However, since the current Web clients and servers do not yet implement NDN, dedicated gateways are used to perform an HTTP/NDN conversion. Since these gateways are conceived as VNFs, they can be deployed where and when required. In particular, two types of gateways are defined: (1) an ingress GateWay (iGW), aimed at converting HTTP requests into NDN Interest messages and NDN Data messages into HTTP replies; (2) an egress GateWay (eGW), aimed at converting NDN messages into HTTP requests, if the content is not available in the ICN network, and HTTP replies into NDN Data messages. Fig. 12 shows the high level architecture of a virtualized node in DOCTOR. The virtualized node is implemented on a single Linux server and it provides the required hardware resources for the VNFs, which

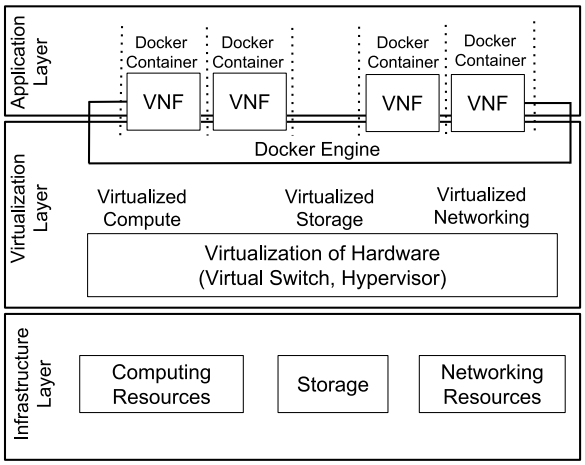


Fig. 12. Internal architecture of a DOCTOR virtualized node.

can act as various components (e.g., NDN stack, IP stack, and HTTP/NDN gateway).

Deployment Approach: DOCTOR uses an *underlay* approach with the help of HTTP/NDN gateways, that can map the HTTP protocol with NDN messages and properly deliver the Web content.

Deployment Scenarios: The iGW and eGW allow DOCTOR to support all the different deployment scenarios.

Addressed Coexistence Requirements: The DOCTOR architecture addresses the following coexistence requirements:

- Forwarding - explicit name based routing of NDN is performed at each router through the use of virtualized NDN stack.
- Storage - content stores perform the content caching.
- Security - DOCTOR supports the same content oriented security as NDN.
- Management - the control and management plane of VNFs in DOCTOR has been designed with respect to the recommendations of the ETSI NFV group, concerning the NFV MANagement and Orchestration (MANO) [103].

Additional Architecture or Technology Used: The architecture of DOCTOR is flexible, as it is based on NFV and SDN principles. Its main component is the NFV infrastructure, which enables the resource virtualization to deploy the ICN protocol stack over the data plane and the MANO aspects over the control plane. As a computing virtualization framework, the architecture uses Docker, which relies on a lightweight virtualization principle

Evaluation Parameters: Among the key limitations of DOCTOR there is the latency, which occurs due to the repeated sending of requests to the ICN servers, acting as gateways and attached to the content source. Since content names are different among each other, each new content name represents a new routing identifier to be given to the gateways. This results in a continuous interaction between content publisher and gateways for each HTTP request.

I. POINT

The H2020 project iP Over IcN- the betTer IP (POINT) [44] started in January 2015 and ended in December 2017. Its main purpose is to evaluate both quantitatively and qualitatively the improvements introduced by running ICN over an IP network. To achieve this aim, POINT designs an evolution of the PURSUIT architecture, which both leverages on the SDN technology and on additional network components that enable IP-based applications to run in the new setup without any modification. Those new elements are the Network Attachment Point (NAP) and the ICN Border GateWay (ICN BGW). The former directly interacts with the end user devices and is responsible for the translation of all the IP protocol abstraction layers (e.g., HTTP, TCP and IP) into the ICN paradigm, while the latter controls the communication between ICN and IP networks. Furthermore, the NAP provides standard gateway functions such as NAT, firewall, and dynamic IP address assignment. The core ICN functionalities are provided by the PURSUIT components (i.e., TM, FN, and RP). Usually, content items are assigned a Routing IDentifier (RID) and are stored on the publisher, which advertises the contents availability in the network. Then, a user device sends a request for a content item and the NAP transforms the interest into a subscription for a specific RID. The subscription is then sent to the RP, which triggers the TM towards the identification of a path between publisher and subscriber. The TM identifies all the nodes that need to be traversed and it calculates the associated FIs, which are placed in the packet header. At this point, the SDN switches are responsible for forwarding the packets by using only the FIs and not the routing tables. The SDN switches are not aware of the POINT architecture and are, instead, coordinated by an SDN controller, which communicates directly with the TM. This communication is bidirectional since the SDN controller informs the TM about any topology modification, and the TM notifies the SDN controller about the configuration to be placed on the SDN switches.

Fig. 13 shows the internal architecture of a POINT node. In the upper layer of the node, there are generic applications (i.e., *App1*, *App2*, *App3*, *App4*) which interact with a set of abstractions provided by POINT (i.e., *IP Abstraction*, *TCP Abstraction*, *HTTP Abstraction*, *CoAP Abstraction*). Those are aimed at enabling the communication between applications and ICN networks without requiring any modification from the application interface side. Each abstraction, then, cooperates with the *Pub/Sub (Information-centric) Service Abstraction* to adhere to a publish/subscribe paradigm, where information is delivered according to specific strategies (i.e., *LIPSIN*, *MSBF*, *POINT Alternative3*). Finally, POINT exploits also the SDN technology by introducing two new layers (i.e., *ICN-over-SDN shim layer* and *SDN*) just above the *L2 Transport Network* layer.

Deployment Approach: The POINT project falls under the *underlay* deployment approach due to the gateway components, which are responsible for the translation from the IP semantics into the ICN semantics.

Deployment Scenarios: The main purpose of the POINT architecture is to enable different subnetworks to communicate between each other. Thus, POINT supports the *Border Island* scenario.

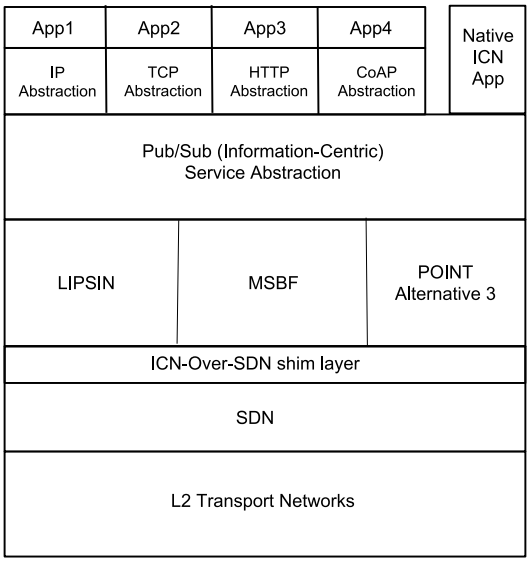


Fig. 13. Internal architecture of a POINT node.

Addressed Coexistence Requirements: Given that POINT is an evolution of PURSUIT, they both share the same coexistence requirements, i.e., forwarding, storage, and security.

Additional Architecture or Technology Used: The POINT solution relies on both the PURSUIT architecture and the SDN technology.

Evaluation Parameters: The challenges introduced by the POINT project involve scalability, dynamic network management and latency of data transmission. The first two challenges refer to the appropriate configuration of SDN switches to face an automatic update of the network topology (e.g., a new host being attached). On the contrary, the third challenge might be due to the high frequency of interaction between NAPs and RPs.

J. RIFE

The architectuRe for an Internet For Everybody (RIFE) [81] architecture is a Horizon2020 funded project, which started in February 2015 and ended in January 2018. Its aim is to develop a new network infrastructure that brings connectivity to communities living in remote locations or unable to afford the communication network costs. To achieve the purpose, the RIFE project focuses on three different challenges regarding the current end-to-end communication paradigm: reduction of capacity, energy, and redundant contents available in the network. The first can be achieved through a time-shifted access to network services and applications. The energy consumed by connected devices can be reduced by introducing a tolerance delay in the communication, so that devices can stay in an idle mode during the absence of network activity. Finally, the third aim is achievable by serving the same content to all the clients that require it, instead of releasing each time a new copy. The architecture addressing those objectives is a combination of IP, ICN, and DTN paradigms.

Deployment Approach: The RIFE architecture follows the *underlay* approach because of the gateway components, which are responsible for the translation from the IP semantics into the ICN semantics.

Deployment Scenarios: RIFE supports the *Border Island* scenario.

Addressed Coexistence Requirements: RIFE is an evolution of the PURSUIT architecture. Thus, the coexistence requirements addressed are the same, i.e., forwarding, storage, and security.

Additional Architecture or Technology Used: The architecture proposed in the RIFE project is a modification of the PURSUIT architecture and it relies on the coexistence of IP, ICN and DTN. This last architecture is responsible for introducing the delay and disruption tolerance required to enable the time-shift requirement.

Evaluation Parameters: No challenges have been found for the RIFE project.

K. CableLabs

Among the different *underlay* approaches, there is a solution designed by CableLabs, which is a non-profit Innovation and R&D lab focused on the introduction of fast and secure release of data, video, voice, and services to end users. CableLas proposes an incremental introduction of CCN/NDN in the existing CDNs to improve the overall content distribution without modifying IP routers [82]. The architecture designed by CableLabs requires first a migration of some services/applications to the ICN paradigm, and then the introduction of proxies. Those are able to manage the translation between HTTP and CCN. Once several ICN "islands" are deployed in the network, the communication among them is provided through IP tunneling.

Deployment Approach: The solution proposed by CableLabs adopts the *underlay* approach because of the gateway components, which are responsible for the translation from the IP semantics into the ICN semantics.

Deployment Scenarios: Except for the *Border Island*, the CableLabs architecture supports all the deployment scenarios.

Addressed Coexistence Requirements: The CableLabs architecture addresses the following coexistence requirements:

- Forwarding - the additional proxies introduced in the network to support the translations, i.e., HTTP to CCN and CCN to HTTP, also work as CCN forwarder.
- Storage - as the architecture is an evolution of a CDN, by design the network nodes can cache contents.

Additional Architecture or Technology Used: Throughout this project, CableLabs investigates how the CCN infrastructure is better in supporting a content-oriented network with respect to the current solutions, such as CDNs. Thus, CableLabs illustrates an incremental deployment of a CCN network over a CDN existing one.

Evaluation Parameters: The challenges identified by CableLabs with respect to their own architecture are as follows: traffic management, optimization of CCN router implementation (e.g., FIB/PIT sizing and memory bandwidth), optimization of CCN cache implementation, content object size and fragmentation (i.e., definition of the maximum content object size transmissible inside a network), CCN to HTTP and

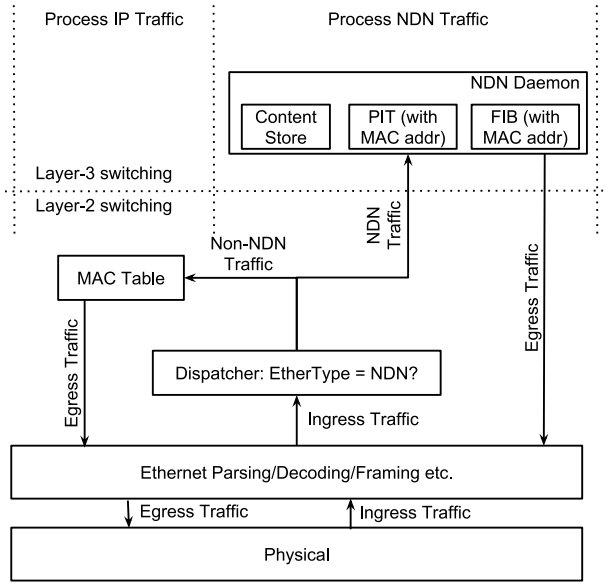


Fig. 14. Dual-stack switch internal architecture.

HTTP to CCN conversions (e.g., the computational complexity of the translation function).

L. NDN-LAN

The authors in [83] propose a *hybrid* ICN architecture in which content names are mapped to the MAC addresses. In particular, the authors present the design of a Dual-Stack switch (D-switch), which provides name-based forwarding for NDN traffic and address-based forwarding for conventional traffic such as IP. It can be seen from Fig. 14 that the key component of D-switch architecture is the *Dispatcher*, which checks the *EtherType* field in the header of a received frame. When an IP frame is detected, the D-switch works like a traditional Ethernet switch and it forwards the frame using the MAC address. If an NDN frame (i.e., Interest or Data packet) is detected, the D-switch processes/forwards the frame based on the content name carried in the NDN header (i.e., Layer 3). In particular, the dispatcher either selects the *Process IP Traffic* or *Process NDN Traffic* module in the D-switch based on the value of *EtherType* field. In the *Process NDN Traffic* module, the PIT and FIB tables are modified to store the mapping between the content names and MAC addresses. For instance, when an Interest packet is received, the D-switch will forward it by searching the content name and its corresponding MAC in the FIB, and then fill the destination MAC address field in Ethernet header with the recorded MAC address.

Deployment Approach: This coexistence approach falls under the *hybrid* approach because the D-switches are able to process both types of traffic (i.e., IP and NDN). In particular, a LAN consists (fully or partially) of D-switches that can process the data traffic received from NDN-enabled hosts, as well as IP hosts. However, a fully *hybrid* scenario needs to be consistent with D-switches only, else other techniques or polices/rules are required to perform the data forwarding.

Deployment Scenarios: Since the D-switches allow NDN traffic to run within the IP network, except for the *Border Island*, NDN-LAN supports all the deployment scenarios. As a matter of fact, due to the use of MAC-layer encapsulation only, the inter-network communications are not possible and the *Border island* scenario cannot be supported.

Addressed Coexistence Requirements: The present architecture provides the following coexistence requirements:

- Forwarding - full advantage of ICN features, such as in-network caching and native multicast, is supported when the underlying LAN consists of D-switches only. However, when the LAN has both D-switch and conventional Ethernet switches, it has to be carefully designed to avoid conflict between name-based forwarding and address-based forwarding.
- Storage - in-network caching is only supported at D-switches, and it is responsibility of the network manager to prevent the conventional Ethernet switches from receiving ICN packets.
- Management - management of such a deployment is challenging due to limitations of topology creation and forwarding rules installation.

Additional Architecture or Technology Used: NDN-LAN is mainly suitable for NDN applications that run in small and private networks such as university campus and within an organization. However, the proposed coexistence solution aims to support a variety of applications which includes NDN as well as IP applications. This is achieved through the following design goals: (i) coexistence with IP traffic, which ensures that the common mechanisms should run without any change or performance penalty, (ii) native NDN support, by not relying on tunnels or overlays, and (iii) incremental deployment and general applicability. The proposed solution does not make use of any specific technology to implement the D-switch logic. Minor hardware and software changes in the D-switches allow them to process the IP and NDN traffic in a controlled environment (i.e., LAN).

Evaluation Parameters: To implement the required logic and functionalities at D-switches so that it can support NDN-enabled traffic processing, some changes are required in the switch hardware, as well as software. Additional forwarding polices need to be installed in scenarios where D-switches coexist with conventional Ethernet switches. Without any standardization of these new software/hardware components, the applicability of the proposed solution in real-world coexistence applications is limited. Designing mechanisms that support the name-based forwarding, meanwhile coexisting with address-based forwarding within the same LAN, is a challenging task. Additionally, the process for D-switches to learn the forwarding table at Layer-2 and build name-based FIB at Layer-3 is an open problem that needs to be addressed. In LAN, the implementation of the proposed solution is simple and straightforward. However, as the LAN size increases and communication between different LANs is needed, the deployment cost will increase significantly, and the current solution needs to be extended to deal with new issues such as interoperability and scalability.

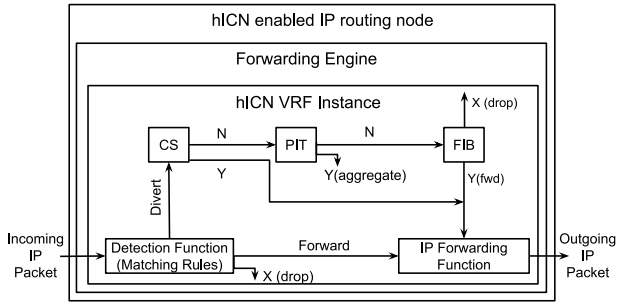


Fig. 15. Internal architecture of an hICN node.

M. hICN

Authors in [84] propose methods and systems to facilitate the integration of ICN into IP networks. The hybrid ICN (hICN) communication system claims to have the ability to preserve ICN features and advantages, while, at the same time, benefiting from exploiting an existing IP infrastructure. The major components of hICN communication system are as follows: (i) hICN-enabled IP router(s), capable of processing and forwarding both regular IP packets and IP packets enhanced with ICN semantics, (ii) IP router(s), capable of handling IP packets, and (iii) hICN router(s), being provisioned with a consumer or producer application. The traditional IP packet headers have been modified to add the ICN semantics. As it is shown in Fig. 15, when a router receives an IP packet, then according to the IP header content, it can identify how to process it, i.e., using ICN or IP stack. The authors suggest two possible name mapping schemes for hICN content names to IP: (i) pure IP mapping, in which content name components can be directly encoded in the IP header, and (ii) optimized mapping, in which a subset of the content name component is encoded in the network header, while the remainder is encoded in the transport header.

Deployment Approach: As the hICN-enabled IP routers are able to process the IP, as well as the ICN traffic, hICN falls under the *hybrid* deployment approach. However, unlike NDN-LAN, in which MAC-to-content name mapping and conversely is performed, in hICN, the IP-to-content name and conversely is done.

Deployment Scenarios: Due to the presence of dual stack routers, the proposed architecture supports all the deployment scenarios.

Addressed Coexistence Requirements: hICN is among the best proposals supporting the coexistence because it retains most of the ICN basic features (e.g., layer-3 name-based routing, partial symmetric routing, object-based security, anchorless mobility, and in-network reactive caching). This is because hICN exploits the IPv4 and IPv6 header fields content semantic to identify whether the received packet is an IP Data packet or an IP Interest packet. The use of IPv4 or IPv6 RFC compliant packet formats guarantees the communication between an IPv4/IPv6 router and a hICN router. More specifically, the hICN router processes and forwards both the regular IP packets and the ICN-semantic-based packets. Hence, it preserves pure ICN behavior at Layer-3 and above by guaranteeing end-to-end service delivery between data producers and data consumers using ICN communication principles. The present architecture provides the following coexistence requirements:

- Forwarding - the hICN-enabled IP routers as well as IP routers use the same forwarding module.
- Storage - the cache stores are available on hICN-enabled IP routers, and the Interest packets could be satisfied by these routers if the requested content is available in the router cache.
- Management - for large scale usage of this architecture, the consumer and producer applications must have the mapping of content-names with the corresponding IP addresses, so that the ICN packets can be processed seamlessly by the non-ICN enabled routers as well.
- Security - the architecture provides the same security features that are provided by ICN. However, the IP-only routers are not able to check the received data packets integrity and authentication, hence, at least one hICN-enabled IP router must be available in the route between the consumer and producer.

Additional Architecture or Technology Used: The hICN proposal uses the IP packet header semantics to differentiate the ICN and IP packets, and the mapping table at hICN-enabled router or DNS is used for performing the mapping task. To support the interoperablity among different networks, the edge router could translate the incoming packets to hICN compliant packets using a proxy. Therefore, hICN does not use any specific architecture (e.g., SDN) or technology (e.g., virtualization or tunnelling) to perform the coexistence.

Evaluation Parameters: The major challenges of hICN are similar to the other *hybrid* approaches and include a lack of support for heterogeneity, scalability, and standardization of the proposed changes in the traditional Internet protocols and network components. Moreover, the communication delay caused by the additional time used by hICN routers for the mapping could be an issue for delay sensitive applications. The hardware modifications are minimal because the hICN routers can be created by installing a software bundle in the existing IP routers. However, the memory requirements will increase due to the need of storage cache. The deployment effort will be considerable due to the need of the modifications in the consumers and producers applications.

N. OFELIA

Melazzi *et al.* [85] proposed an SDN-based *hybrid* implementation of ICN under the OFELIA project. The proposed approach is an extension of the CONET architecture [77] for OpenFlow networks, where dedicated BNs perform name-to-location resolution, using an external system, for any requested Named Data Object (NDO). Fig. 16 presents a simplified view of this solution. The authors propose to include two different forwarding strategies in an *ICN node*: (1) to forward content requests; and (2) to deliver the data. *Forward-by-name* feature of an *ICN node* applies to Interest packets, while *Data Forwarding* is the mechanism that allows the content to be sent back to the device that issued a content request. *Content routing* is used to disseminate information about location of contents, and *Caching* is the ability of ICN

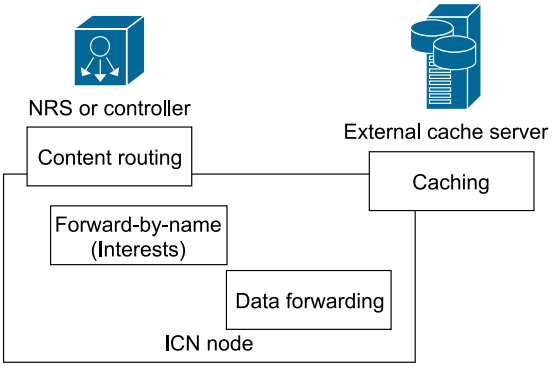


Fig. 16. Simplified view of the solution proposed by OFELIA.

nodes to cache data and to directly reply to incoming content requests. The OFELIA testbed was used in IRATI [8] project for experimental activities.

Deployment Approach: The proposed architecture adheres to a *hybrid* approach.

Deployment Scenarios: The proposed implementation of ICN is an extension of the CONET framework, in which BNs interconnect different CSSs. Hence, this solution supports the *Border Island* scenario.

Addressed Coexistence Requirements: The proposed system is based on CONET framework. Extending the primary goals of CONET framework, this architecture aims to support forwarding, storage, security and management for ICN deployment.

Additional Architecture or Technology Used: The present solution strongly relies on the architecture proposed in the CONET project and, through SDN/OpenFlow, it targets all the services/applications of the TCP/IP protocol stack.

Evaluation Parameters: The architecture of the solution requires the networking elements to be OpenFlow compliant. Given that OpenFlow (SDN) has been widely adopted in the networking domain, the hardware modifications and the time required for its deployment are low in scenarios where OpenFlow-based network is already present. On another side, the hardware modifications and the time required for its deployment would be higher if OpenFlow-based network is not already present.

V. Discussion

The purpose of this section is to summarize the findings achieved through our systematic analysis of all the existing coexistence architectures (Section V-A), discuss their deployment in a real-world scenario (Section V-B) and the open challenges (Section V-C), along with some future directions concerning the coexistence between the current and the future Internet architectures (Section V-D).

A. Summary of the Survey

The main aim of this survey is to provide the necessary overview of the available solutions that already address the coexistence. We believe that it will help to move the research community towards the design of the most appropriate architecture for the future Internet. Thus, to guide the reader towards the interpretation of Table I, we add here two new tables, which are a summary of Table I. In particular, among all the features and evaluation parameters considered in this survey, the only ones that can be chosen by a network designer are the deployment approach and the possible additional architecture or technology used in the design of his solution. Thus, Table II and Table III are aimed at comparing each deployment approach and each additional architecture or technology used with respect to all the other features and evaluation parameters, respectively. As a matter of fact, the deployment scenarios, as well as the addressed coexistence requirements, directly depend on the deployment approach or on the additional architecture or technology, while the evaluation parameters are dynamic properties evaluated during the runtime deployment of an architecture.

The content of the cells as well as their meaning is shared between Table II and Table III. More specifically, the content of each cell corresponds to the number of coexistence architectures addressing both the properties specified in the corresponding row and column (e.g., in the first cell of Table II the value equal to 7 means that there are 7 coexistence architectures adhering to the *overlay* approach and supporting the *forwarding* functionality). The meaning of the values in the cells is different throughout the table. In the upper part (i.e., rows referring to addressed coexistence requirements and deployment scenarios), the value in the cell refers to the number of architectures that guarantee a specific addressed coexistence requirement or a deployment scenario by adopting a deployment approach (listed in the columns). On the contrary, in the lower part of the table (i.e., rows referring to

TABLE II
Comparison of All the Deployment Approaches for Coexistence Architectures—The Value of Each Cell Refers to the Number of Coexistence Architectures Addressing Both the Properties Specified in the Corresponding Row and Column

		Deployment Approach		
		Overlay	Underlay	Hybrid
Addressed coexistence requirements	Forwarding	7	4	4
	Storage	6	4	4
	Security	4	3	2
	Management	3	1	3
Deployment scenarios	ICN-ICN communication in IP "ocean"	7	2	3
	ICN-IP communication in IP "ocean"	2	2	2
	ICN-IP communication in ICN "ocean"	0	2	2
	IP-IP communication in ICN "ocean"	0	2	2
	Border Island	2	3	3
Evaluation parameter	Traffic management	4	1	1
	Access control	1	0	0
	Scalability	2	1	2
	Dynamic network management	1	1	1
	Latency	0	2	2
	Other	4	4	2

TABLE III
COMPARISON OF ALL THE ADDITIONAL ARCHITECTURES OR TECHNOLOGIES USED IN COEXISTENCE ARCHITECTURES—THE VALUE OF EACH CELL REFERS TO THE NUMBER OF COEXISTENCE ARCHITECTURES ADDRESSING BOTH THE PROPERTIES SPECIFIED IN THE CORRESPONDING ROW AND COLUMN

		Additional architecture or technology used								
		PSIRP	LAN	SAIL	SDN	PURSUIT	CDN	DTN	CONET	DNS
Addressed coexistence requirements	Forwarding	1	2	1	6	2	1	1	1	1
	Storage	1	2	1	5	2	1	1	1	1
	Security	1	1	0	4	2	0	1	1	1
	Management	0	0	0	4	0	0	0	1	1
Deployment scenarios	ICN-ICN communication in IP "ocean"	1	2	1	4	0	1	0	0	1
	ICN-IP communication in IP "ocean"	0	1	0	3	0	1	0	0	1
	ICN-IP communication in ICN "ocean"	0	1	0	1	0	1	0	0	1
	IP-IP communication in ICN "ocean"	0	1	0	1	0	1	0	0	1
	Border Island	0	0	1	4	2	0	1	1	1
Evaluation parameter	Traffic management	1	2	1	1	0	1	0	0	0
	Access control	0	0	0	0	0	0	0	0	0
	Scalability	0	1	1	3	1	0	0	0	1
	Dynamic network management	0	1	1	2	1	0	0	0	0
	Latency	0	1	0	2	1	0	0	0	1
	Other	0	0	0	3	0	4	0	1	0

the evaluation parameters), the value in the cells refers to the number of limitations an architecture is affected from.

Table II shows on the columns the three different deployment approaches (i.e., *overlay*, *underlay* and *hybrid*), while on the rows there are all the other features, except for the architectures or technologies used, considered in Table III. Considering the deployment approaches, we found six architectures adopting the *overlay* solution, four the *underlay*, three the *hybrid* and one architecture (i.e., CONET) adhering to both *overlay* and *hybrid*. As it is shown in the table, a plausible reason for this greater adoption of the *overlay* approach might be the higher number of addressed coexistence requirements provided by it. As a matter of fact, almost all the *overlay* architectures guarantee the forwarding and storage features and the number of the architectures supporting security and management is higher than in the *underlay* and *hybrid* cases. While, adopting an *overlay* approach prevents architectures from being deployed in all the deployment scenarios: none of the *overlay* architectures covers either the *ICN-IP communication in ICN "ocean"* or the *IP-IP communication in ICN "ocean"* scenarios. Finally, considering the evaluation parameters, most *overlay* architectures are not able to properly manage the network traffic, but the other limitations are comparable with the ones affecting the *underlay* and *hybrid* solutions. Moreover, even if the number of challenges under the last class (i.e., *Other*) might be significant, we note that those limitations strongly depend on the design of each coexistence architecture.

Table III contains the same rows as Table II, while on the columns it shows all the additional architectures or technologies used in the analyzed coexistence solutions. Throughout this survey, we found the following results: one coexistence solution relying on the PSIRP architecture, two on LAN, one on SAIL, six on SDN, two on PURSUIT, one on CDN, one on DTN, one on CONET, and one on DNS. As it is clearly visible from the table, the reason for adopting the SDN technology in a coexistence scenario is given by its numerous benefits in terms of both features and evaluation parameters with respect to the other possible solutions.

B. Deployment in a Real-World Scenario

A clean slate deployment of ICN requires overhauling the entire Internet infrastructure and changing all the host and producer applications. Thus, researchers have realized that it is difficult, as well as infeasible, to replace a greatly successful imperative architecture, such as the IP one, with a clean slate approach, and considered the three deployment configurations (e.g., overlay [74], [75], underlay [80], [44], and hybrid [84]). Moreover, moving from research testbeds to operational networks is very difficult and requires several trials on different large scale testbeds with different number of users.

The first ICN testbed, deployed within the framework of the NDN[4] project, is a shared research testbed that includes software routers, installed in several participating institutions, application host nodes, and other devices. In recent years, a significant number of trials have been conducted to evaluate the CCN/NDN-related software, which are the ones that mostly support the deployment on real networks by providing the set of specifications of the relative architectures (e.g., security, fragmentation, encapsulation, and packet format) [25]. At the same time, since improving the network capacity and minimizing the service latency, even at high network loads, are

[4]https://named-data.net/ndn-testbed/

the key advantages of ICN, many real-world trials addressing the video streaming application scenario have been setup: Cisco and Verizon demonstrated the feasibility and possible advantages of hybrid-ICN in Verizon's labs, applying hICN to live video distribution over a mobile and multi-homed access network; Huawei and China Unicom recently started trials for the ICN-as-a-Slice configuration, using video conferencing as application scenario to evaluate the security, mobility and bandwidth efficiency of ICN over a wired infrastructure [104]. Both such deployments plan to extend their prototypes to demonstrate the benefits of ICN over a 5G network. Considering the underlay deployment strategy, the Cisco, Internet2 and the U.S. Research and Education community funded the National Research and Education Network (NREN) ICN Testbed project. The purpose of the project is to advance the research in data-intensive science and network, by improving data movement, searchability, and accessibility. The project involves several Universities and U.S. federal Government entities. The testbed has around 15 nodes connected through a nation-wide VPN-based layer-2 underlay across the USA, relies on CCN implementation, and uses the Community Information-Centric Networking (CICN) [105] open-source software. In the ICN2020[5] EU project, a testbed has been created in an overlay deployment configuration over the public Internet. The testbed contains 37 nodes and measures the throughput of video applications under certain scenarios. ICN2020 also proposes the use of the GEANT Testbed Service[6] (GTS) to create an independent and isolated global-scale ICN testbed, and to extend the functionalities of the existing ones (e.g., NDN testbed). The above-mentioned deployments in a real-world scenario are the first efforts towards the adoption of ICN in a real network. However, many more evaluations and tests still need to be done.

C. Open Challenges

According to our findings, the following challenges need to be addressed while designing an efficient and secure coexistence architecture.

- *Traffic management:* the existing Internet applications are not completely compatible with architectures implementing the *overlay* approach [9], [35], [88], [89] due to the issues that these applications introduce on the transport layer. Changing the addressing scheme from host-based to content-based, as well as changing network models from push to pull, are indeed the two obstacles in adapting the existing transport layer protocols to the NDN and CCN architectures. A vast number of existing applications and protocols, such as the HTTP based multimedia streaming protocols, might face false throughput estimations due to the aggressiveness of the underlying TCP in case of content source location variations [90], [91].
- *Latency:* one fundamental issue introduced by the solutions supporting the translation of IP and HTTP-level semantics into ICN [44], [81] is latency. This occurs due to the frequent requests sent to the NAP, that is attached to the source (also referred to as sNAP). Assuming a meaningful interaction between consumer and producer, the URIs are likely different for each content and for each new published content at sNAP, a new RID has to be added to the consumer NAP (cNAP) through the RF. Thus, for each HTTP get request, sNAP and RF have to interact, causing an increasing network latency.
- *Topological limitations:* in *underlay* approaches, there might be several publishers for the same content that belong to the same network. In this case, whenever a consumer asks for a content released by different publishers, the RF should identify the best publisher and suggest the best content route. However, in the current architectures, the RF only announces which is the most appropriate publisher, leaving the other ones in a *silent* phase. This might lead to the generation of multi-point forwarding identifiers, which create unnecessarily long routing tables.
- *Routing and scalability:* the number of content objects, and its continuous growing in the current Internet, introduce a limitation in ICN solutions, which have to handle content names of a possibly indefinite length. Thus, the existing networking devices might not support the content-based routing and might have to face special requirements and optimizations.
- *Security issues in coexistence architectures:* below, we illustrate the security risks affecting the coexistence architectures.
  - *Attacks against NAP nodes:* in *underlay* approaches, an attack performed against a NAP node can cause much more damage than one performed against the rendezvous system. This is because a NAP is a node in an ICN network, which can be used by an attacker to launch prefix hijacking, replay attacks and many more attacks against the ICN core network.
  - *DoS attacks:* an external user sending a new IP address causes the introduction of a state into a NAP. The same action can cause the introduction of states in centralized functions, such as the TF or the RF. Thus, if arbitrary users have a direct access to the centralized TF/RF, as it was the case in pure PURSUIT/PSIRP architectures [74], they could also easily generate a DoS attack.
  - *Lack of authorization and access control:* for every new node added to a network, the entire topology needs to be updated to guarantee the proper link among the new and the old network nodes. Thus, an enhanced access control policy is required in ICN networks.
  - *Attacks against the SDN controller:* there have been increasing concerns about the security of SDN-based networks. Many of these concerns are related to the fact that SDN controller may parse an arbitrary part of a packet's content, and use this information to set up states in the flow tables (and possibly in the controller). Moreover, systems that parse user generated packet input (e.g., Wireshark packet analyzer and Snort intrusion detection system) have been the

[5]http://www.icn2020.org/

[6]https://www.geant.org/Services/Connectivity_and_network/GTS

frequent cause of security vulnerabilities due to the large permutation of potential cases. Since numerous ICN coexistence solutions propose to use SDN, they are potentially open to the inherent vulnerabilities of an SDN controller. Moreover, considering that an SDN controller is the logically centralized entity that affects the entire network, the risk is even higher.

D. Future Research Directions

As confirmed by the large number of coexistence projects (e.g., POINT, DOCTOR, and hICN) that we surveyed in this paper, Governments, Industry, and Academia are pushing towards the definition of a new Internet architecture (i.e., ICN) and its coexistence with the current one (i.e., IP). The significant effort put to assess the feasibility and effectiveness of ICN indicates that the ICN paradigm is being considered as a possible replacement for the current IP-based host-centric Internet infrastructure. Hence, we now present few research directions that need to be explored in this research field.

- *Secure transition phase:* from its start, ICN was purposefully designed to have certain inherent security properties such as authentication of delivered content and (optional) encryption of the content. Additionally, relevant advances in the ICN research community have occurred, promising to address each of the identified security gaps [106], [23]. However, due to the lack of real deployments, an array of security features in ICN networks are still under-investigated, including access control [107], security of in-network caches, protection against various network attacks (e.g., DDoS), and consumer privacy [24]. For instance, due to the distributed nature of content availability in ICN, securing the content itself is much more important than securing the infrastructure or the end points. This lack of addressing security goals in the final ICN paradigm is even more critical when considering the coexistence of TCP/IP and ICN, which could lead to the introduction of new attacks and security issues. One of the main limitations of existing projects is that all of them address only the existence of a transition phase without investigating the impact of coexistence on the security and privacy of the system. We believe that not only passing through this intermediate step is unavoidable, but also that it is important to assess the security and privacy vulnerabilities that might come up under the coexistence of both architectures.
- *Selection of an efficient coexistence approach:* in the literature, three main approaches (i.e., *underlay* [108], *overlay* [77], and *hybrid* [84]) have been used to deploy coexistence architectures. The *underlay* approach introduces communication latency due to the required mapping between IP and name addresses, which limits its usability for real-time and delay-sensitive applications. On the contrary, the *underlay* approach maintains an unaltered quality of service under both normal and exceptional conditions, such as failure, server and link congestion, which are common in operator networks. Considering the *overlay* approach, a major drawback is that it requires the definition and standardization of a new packet format, together with protocols that manage the mapping between ICN faces and IP addresses in the ICN routers FIB. Thus, *overlay* poses a significant challenge to network operators and developers. Additionally, upon new deployment, the tunnel configurations in *overlay* needs to be manually changed to include the newly deployed ICN nodes, and these point-to-point tunnels limit the ICN capability in utilizing the underlying broadcast media. Finally, the *hybrid* approach offers an interesting alternative as it allows ICN semantics to be embedded in standard IPv4 and IPv6 packets so that the packets can be routed through either IP routers or hybrid ICN routers. However, the detailed performance results for *hybrid* solutions are still incomplete, which limits its usage in real deployment scenarios.
- *Coexistence solutions that preserve inherent ICN advantages:* due to its inherent features such as in-network caching, interest aggregation, and content oriented security, ICN provides improved communication system and security by design. Therefore, these essential features of ICN should be protected while designing a coexistence architecture.
- *Optimized ICN-IP name-space mapping:* an important issue in the state-of-the-art solutions, that provide translation of IP/HTTP-level services into ICN (or vice versa), is to ensure that the communication latency is comparable with the one in the current network. In most of the coexistence solutions, that use some sort of translation at any networking layer (e.g., transport or network), the main problem is the repeated sending of newly published content information towards the translation server, which generates delay in the response path of requester and congestion in the network. The problem lies in the fact that the URL is likely different for every request (assuming some form of meaningful service interaction between IP client and ICN producer). Additionally, the existing channel semantics cannot be applied directly because the corresponding routing identifier at the ICN level is different for each publication, from the translation server to IP client. Also, realizing the rendezvous function approach, which is responsible for the response of new publications, requires continue interaction between server and content publisher. This causes an additional latency for the client requests, waiting for a fresh mapping of ICN-IP at each published event.
- *Data protection and confidentiality:* ensuring privacy for network entities (e.g., consumer and producer) in coexistence architecture is not a trivial task, mainly due to the poor privacy support provided in ICN [109]. Hence, it is important to investigate how the privacy issues were dealt in the current coexistence architectures. Ideally, names should reveal no more than what is currently revealed by an IP address and port. However, in ICN the name prefix reveals some information about the content, and the in-network caching and data in PIT might expose the consumer identity [110]. Therefore, the researchers should focus on the specific issues concerning the privacy and

data protection in the coexistence scenarios. For instance, in a coexistence architecture, IP to name-prefix mapping is performed when an IP packet travels from IP to ICN network. In this scenario, the IP header does not reveal any information about the payload, but the prefix name does, thus, the data confidentiality is threatened when these data packets are traveling through the ICN "island". In particular, since the use of name prefix for addressing the data in ICN reveals sufficient information to the passive eavesdropper, ensuring privacy means that names and payloads cannot be correlated. However, such privacy requirement would need an upper-layer service similar to the one that would resolve non-topological identifiers (e.g., ICN name prefix) to topological names (e.g., IP network address).

- *SDN/NFV for efficient coexistence:* as mentioned earlier, the SDN technology separates the control plane from the data plane. The decoupled control plane is programmable and has a global view of the network that provides easier network management monitoring. SDN-based implementations of ICN exploit the centralized view available to the SDN controller, which enables the SDN controller to install appropriate rules in the data-plane to process ICN requests/responses. In the state-of-the-art, both *overlay* and *hybrid* ICN deployments have leveraged SDN to address different coexistence requirements, e.g., forwarding, storage, management, security, and interoperability. SDN has already been successfully adopted for network deployment; it makes SDN an appropriate choice for quick deployment of ICN with low hardware modifications. On the another side, NFV can help to virtualize several network functions that were previously implemented via physical devices.

VI. Conclusion

In this paper, we survey various efforts done by researchers and industries in recent years to propose a design of ICN-IP coexistence architecture. All these architectures differ from each other according to their specific design, but they all adhere to the ICN paradigm, which means a content-oriented communication model in replacement of the current host-centric one. In our survey, we identify that all these architectures have important limitations: none of them has been designed through a comprehensive approach that considers all the new challenges introduced by a coexistence scenario. Instead, the main aim for most of them is to improve the current Internet by exploiting some of the core ICN features (i.e., forwarding, storage, management, and security). Even though security also belongs to that list of features, none of the existing architectures has considered it as the main purpose. In future, we believe appropriate coexistence architecture designs are needed to build a secure path towards the future Internet. This can be done by considering the limitations and necessary improvements of the existing coexistence solutions we have analyzed in this survey. With the set of future research directions and open questions that we have raised, our work will motivate researchers towards designing a complete solution for ICN-IP coexistence while tackling the key security and privacy issues.

References

[1] *Internet of Things (IoT) Connected Devices Installed Base Worldwide From 2015 to 2025 (in Billions)*. Accessed: May 11, 2020. [Online]. Available: https://tinyurl.com/yyfukyk2

[2] P. Srisuresh and K. Egevang, "Traditional IP network address translator (traditional NAT)," IETF, RFC 3022, 2001.

[3] (Sep. 2017). *Cisco Visual Networking Index: Forecast and Methodology: 2016–2021*. Accessed: Jul. 28, 2019. [Online]. Available: https://tinyurl.com/y2jptcbn

[4] (Jun. 2017). *The Zettabyte Era: Trends and Analysis*. Accessed: Jul. 28, 2019. [Online]. Available: https://tinyurl.com/y32f4jwe

[5] *What Happens in an Internet Minute in 2016*, Intel, Santa Clara, CA, USA, Aug. 2019. [Online]. Available: https://tinyurl.com/y5wrz8p6

[6] K. Seo and S. Kent, "Security architecture for the Internet protocol," IETF, RFC 4301, Dec. 2005.

[7] E. Rescorla, "The transport layer security (TLS) protocol version 1.3," IETF, RFC 8446, Aug. 2018. [Online]. Available: https://rfc-editor.org/rfc/rfc8446.txt

[8] E. Grasa *et al.*, "Recursive Internet work architecture (RINA), investigating RINA as an alternative to TCP/IP (IRATI)," in *Building the Future Internet Through FIRE*. New York, NY, USA: Rivers, 2016. [Online]. Available: https://tinyurl.com/y4kbyt5j

[9] M. Mosko. (2014). *CCNx 1.0 Protocol Specification Roadmap*. Accessed: May 11, 2020. [Online]. Available: https://tinyurl.com/yd3qjk2d

[10] M. Diallo, S. Fdida, V. Sourlas, P. Flegkas, and L. Tassiulas, "Leveraging caching for Internet-scale content-based publish/subscribe networks," in *Proc. IEEE Int. Conf. Commun. (ICC)*, 2011, pp. 1–5.

[11] Y. Tang, K. Guo, J. Ma, Y. Shen, and T. Chi, "A smart caching mechanism for mobile multimedia in information centric networking with edge computing," *Future Gener. Comput. Syst.*, vol. 91, pp. 590–600, Feb. 2019. [Online]. Available: https://tinyurl.com/y5qnozwp

[12] A. Ioannou and S. Weber, "A survey of caching policies and forwarding mechanisms in information-centric networking," *IEEE Commun. Surveys Tuts.*, vol. 18, no. 4, pp. 2847–2886, 4th Quart., 2016.

[13] A. Seetharam, "On caching and routing in information-centric networks," *IEEE Commun. Mag.*, vol. 56, no. 3, pp. 204–209, Mar. 2018.

[14] X. Fu, D. Kutscher, S. Misra, and R. Li, "Information-centric networking security," *IEEE Commun. Mag.*, vol. 56, no. 11, pp. 60–61, Nov. 2018.

[15] C. Anastasiades, T. Braun, and V. A. Siris, "Information-centric networking in mobile and opportunistic networks," in *Wireless Networking for Moving Objects* (LNCS 8611), Cham, Switzerland: Springer, 2014, pp. 14–30.

[16] C. Fang, H. Yao, Z. Wang, W. Wu, X. Jin, and F. R. Yu, "A survey of mobile information-centric networking: Research issues and challenges," *IEEE Commun. Surveys Tuts.*, vol. 20, no. 3, pp. 2353–2371, 3rd Quart., 2018.

[17] G. Xylomenos *et al.*, "A survey of information-centric networking research," *IEEE Commun. Surveys Tuts.*, vol. 16, no. 2, pp. 1024–1049, 2nd Quart., 2014.

[18] B. Ahlgren, C. Dannewitz, C. Imbrenda, D. Kutscher, and B. Ohlman, "A survey of information-centric networking," *IEEE Commun. Mag.*, vol. 50, no. 7, pp. 26–36, Jul. 2012.

[19] I. Abdullahi, S. Arif, and S. Hassan, "Survey on caching approaches in information centric networking," *J. Netw. Comput. Appl.*, vol. 56, pp. 48–59, Oct. 2015. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S1084804515001381

[20] G. Zhang, Y. Li, and T. Lin, "Caching in information centric networking: A survey," *Comput. Netw.*, vol. 57, no. 16, pp. 3128–3141, 2013. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S1389128613002235

[21] M. Zhang, H. Luo, and H. Zhang, "A survey of caching mechanisms in information-centric networking," *IEEE Commun. Surveys Tuts.*, vol. 17, no. 3, pp. 1473–1499, 3rd Quart., 2015.

[22] M. F. Bari, S. R. Chowdhury, R. Ahmed, R. Boutaba, and B. Mathieu, "A survey of naming and routing in information-centric networks," *IEEE Commun. Mag.*, vol. 50, no. 12, pp. 44–53, Dec. 2012.

[23] E. G. AbdAllah, H. S. Hassanein, and M. Zulkernine, "A survey of security attacks in information-centric networking," *IEEE Commun. Surveys Tuts.*, vol. 17, no. 3, pp. 1441–1454, 3rd Quart., 2015.

[24] R. Tourani, S. Misra, T. Mick, and G. Panwar, "Security, privacy, and access control in information-centric networking: A survey," *IEEE Commun. Surveys Tuts.*, vol. 20, no. 1, pp. 566–600, 1st Quart., 2018.

[25] M. Tortelli, D. Rossi, G. Boggia, and L. Grieco, "ICN software tools: Survey and cross-comparison," *Simulat. Model. Pract. Theory*, vol. 63, pp. 23–46, Apr. 2016. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S1569190X15300654

[26] X. Liu, Z. Li, P. Yang, and Y. Dong, "Information-centric mobile ad hoc networks and content routing: A survey," *Ad Hoc Netw.*, vol. 58, pp. 255–268, Apr. 2017. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S1570870516301019

[27] A. Rahman, D. Trossen, D. Kutscher, and R. Ravindran, *Deployment Considerations for Information-Centric Networking (ICN)*, ICNRG, New Delhi, India, 2019. [Online]. Available: https://tinyurl.com/y24d4oly

[28] S. Arshad, M. A. Azam, M. H. Rehmani, and J. Loo, "Recent advances in information-centric networking-based Internet of Things (ICN-IoT)," *IEEE Internet Things J.*, vol. 6, no. 2, pp. 2128–2158, Apr. 2019.

[29] B. Nou *et al.*, "A survey of Internet of Things communication using ICN: A use case perspective," *Comput. Commun.*, vols. 142–143, pp. 95–123, Jun. 2019.

[30] C. Xu, M. Wang, X. Chen, L. Zhong, and L. A. Grieco, "Optimal information centric caching in 5G device-to-device communications," *IEEE Trans. Mobile Comput.*, vol. 17, no. 9, pp. 2114–2126, Sep. 2018.

[31] I. U. Din, S. Hassan, M. K. Khan, M. Guizani, O. Ghazali, and A. Habbal, "Caching in information-centric networking: Strategies, challenges, and future research directions," *IEEE Commun. Surveys Tuts.*, vol. 20, no. 2, pp. 1443–1474, 2nd Quart., 2018.

[32] D. Cheriton and M. Gritter. (2000). *TRIAD: A New Next-Generation Internet Architecture*. Accessed: May 11, 2020. [Online]. Available: https://tinyurl.com/y939takg

[33] T. Koponen *et al.*, "A data-oriented (and beyond) network architecture," in *Proc. Conf. Appl. Technol. Archit. Protocols Comput. Commun. (SIGCOMM)*, 2007, pp. 181–192.

[34] V. Jacobson *et al.*, "Networking named content," in *Proc. 5th ACM Int. Conf. Emerg. Netw. Exp. Technol. (CoNEXT)*, 2009, pp. 1–12.

[35] L. Zhang *et al.*, "Named data networking," *ACM SIGCOMM Comput. Commun. Rev.*, vol. 44, no. 3, pp. 66–73, 2014.

[36] *Information-Centric Networking Research Group (ICNRG)*. Accessed: May 11, 2020. [Online]. Available: https://irtf.org/icnrg

[37] J. McQuillan, I. Richer, and E. Rosen, "The new routing algorithm for the ARPANET," *IEEE Trans. Commun.*, vol. C-28, no. 5, pp. 711–719, May 1980.

[38] V. Dimitrov and V. Koptchev, "PSIRP project—Publish-subscribe Internet routing paradigm: New ideas for future Internet," in *Proc. 11th ACM Int. Conf. Comput. Syst. Technol. (CompSysTech)*, 2010, pp. 167–171.

[39] C. Dannewitz, D. Kutscher, B. Ohlman, S. Farrell, B. Ahlgren, and H. Karl, "Network of information (NetInf)—An information-centric networking architecture," *Comput. Commun.*, vol. 36, no. 7, pp. 721–735, 2013.

[40] H. Zimmermann, "OSI reference model—The ISO model of architecture for open systems interconnection," *IEEE Trans. Commun.*, vol. C-28, no. 4, pp. 425–432, Apr. 1980.

[41] R. Braden, "Requirements for Internet hosts—Communication layers," IETF, RFC 1122, 1989.

[42] C. Safitri, Y. Yamada, S. Baharun, S. Goudarzi, Q. Nguyen, and T. Sato, "An intelligent quality of service architecture for information-centric vehicular networking," *Internetworking Indonesia J.*, vol. 10, pp. 15–20, Jan. 2018.

[43] F. Drijver, "Assessment of benefits and drawbacks of ICN for IoT applications," M.S. thesis, Elect. Eng., TU Delf, Delf, The Netherlands, 2018.

[44] *POINT (IP Over ICN—The Better IP)*. Accessed: May 11, 2020. [Online]. Available: https://www.point-h2020.eu/

[45] *White Paper: Cisco Visual Networking Index (VNI): Forecast and Methodology, 2017–2022*. Accessed: May 11, 2020. [Online]. Available: https://www.cisco.com/c/en/us/solutions/collateral/service-provider/visual-networking-index-vni/white-paper-c11-741490.html

[46] S. Lederer, C. Mueller, C. Timmerer, and H. Hellwagner, "Adaptive multimedia streaming in information-centric networks," *IEEE Netw.*, vol. 28, no. 6, pp. 91–96, Nov./Dec. 2014.

[47] S. Lederer, C. Mueller, B. Rainer, C. Timmerer, and H. Hellwagner, "Adaptive streaming over content centric networks in mobile networks using multiple links," in *Proc. IEEE Int. Conf. Commun. Workshops (ICC)*, Jun. 2013, pp. 677–681.

[48] Y. Liu *et al.*, "Dynamic adaptive streaming over CCN: A caching and overhead analysis," in *Proc. IEEE Int. Conf. Commun. (ICC)*, 2013, pp. 3629–3633.

[49] S. Petrangeli, N. Bouten, M. Claeys, and F. D. Turck, "Towards SVC-based adaptive streaming in information-centric networks," in *Proc. IEEE Int. Conf. Multimedia Expo Workshops (ICMEW)*, Jun. 2015, pp. 1–6.

[50] B. Rainer, D. Posch, and H. Hellwagner, "Investigating the performance of pull-based dynamic adaptive streaming in NDN," *IEEE J. Sel. Areas Commun.*, vol. 34, no. 8, pp. 2130–2140, Aug. 2016.

[51] J. Samain *et al.*, "Dynamic adaptive video streaming: Towards a systematic comparison of ICN and TCP/IP," *IEEE Trans. Multimedia*, vol. 19, no. 10, pp. 2166–2181, Oct. 2017.

[52] Y. Zhang, A. Afanasyev, J. Burke, and L. Zhang, "A survey of mobility support in named data networking," in *Proc. IEEE Conf. Comput. Commun. Workshops (INFOCOM WKSHPS)*, Apr. 2016, pp. 83–88.

[53] J. Ott and D. Kutscher, "Why seamless? Towards exploiting WLAN-based intermittent connectivity on the road," in *Proc. TERENA Netw. Conf.*, 2004, pp. 1–15.

[54] K. Fall, "A delay-tolerant network architecture for challenged Internet," in *Proc. Conf. Appl. Technol. Architect. Protocols Comput. Commun. (SIGCOMM)*, 2003, pp. 27–34. [Online]. Available: http://doi.acm.org/10.1145/863955.863960

[55] L. Torgerson *et al.*, "Delay-tolerant networking architecture," IETF, RFC 4838, Apr. 2007. [Online]. Available: https://rfc-editor.org/rfc/rfc4838.txt

[56] A. Compagno, M. Conti, and M. Hassan, "An ICN-based authentication protocol for a simplified LTE architecture," in *Proc. Int. Worskhop Commun. Security*, vol. 447, 2017, pp. 125–140.

[57] H. Farhady, H. Lee, and A. Nakao, "Software defined networking: A survey," *Comput. Netw.*, vol. 81, pp. 79–95, May 2015.

[58] D. Kreutz, F. M. Ramos, P. E. Verissimo, C. E. Rothenberg, S. Azodolmolky, and S. Uhlig, "Software defined networking: A comprehensive survey," *Proc. IEEE*, vol. 103, no. 1, pp. 14–76, Jan. 2015.

[59] N. McKeown *et al.*, "OpenFlow: Enabling innovation in campus networks," *ACM SIGCOMM Comput. Commun. Rev.*, vol. 38, no. 2, pp. 69–74, 2008.

[60] Y. Li and M. Chen, "Software defined network function virtualization: A survey," *IEEE Access*, vol. 3, pp. 2542–2553, 2015.

[61] A. Vakali and G. Pallis, "Content delivery networks: Status and trends," *IEEE Internet Comput.*, vol. 7, no. 6, pp. 68–74, Nov./Dec. 2003.

[62] A. Binder and I. Kotuliak, "Content delivery network interconnect: Practical experience," in *Proc. 11th IEEE Int. Conf. Emerg. e-Learn. Technol. Appl. (ICETA)*, 2013, pp. 29–33.

[63] M. A. Salahuddin, J. Sahoo, R. Glitho, H. Elbiaze, and W. Ajib, "A survey on content placement algorithms for cloud-based content delivery networks," *IEEE Access*, vol. 6, pp. 91–114, 2018.

[64] Y. Nicolas, D. Wolff, D. Rossi, and A. Finamore, "I Tube, YouTube, P2PTube: Assessing ISP benefits of peer-assisted caching of YouTube content," in *Proc. IEEE P2P*, 2013, pp. 1–2.

[65] L. Sun, M. Ma, W. Hu, H. Pang, and Z. Wang, "Beyond 1 million nodes—Crowdsourced video CDN: Architecture, technology, and economy," *IEEE Multimedia*, early access, Jun. 15, 2017, doi: 10.1109/MMUL.2017.265091309.

[66] V. Stocker, G. Smaragdakis, W. Lehr, and S. Bauer, "The growing complexity of content delivery networks: Challenges and implications for the Internet ecosystem," *Telecommun. Policy*, vol. 41, no. 10, pp. 1003–1016, 2017.

[67] D. Clark, Lehr, S. Bauer, P. Faratin, R. Sami, and J. Wroclawski, "The growth of Internet overlay networks: Implications for architecture, industry structure and policy," in *Proc. Telecommun. Policy Res. Conf. (TPRC)*, 2005, pp. 93–106.

[68] P. Medagliani, S. Paris, J. Leguay, L. Maggi, C. Xue, and H. Zhou, "Overlay routing for fast video transfers in CDN," in *Proc. IFIP/IEEE Symp. Integr. Netw. Service Manag. (IM)*, 2017, pp. 531–536.

[69] B. Niven-Jenkins, F. L. Faucheur, and N. Bitar, "Content distribution network interconnection (CDNI) problem statement," IETF, RFC 6707, 2012.

[70] M. J. Khabbaz, C. M. Assi, and W. F. Fawaz, "Disruption-tolerant networking: A comprehensive survey on recent developments and persisting challenges," *IEEE Commun. Surveys Tuts.*, vol. 14, no. 2, pp. 607–640, 2nd Quart., 2012.

[71] A. V. Vasilakos, Y. Zhang, and T. Spyropoulos, *Delay Tolerant Networks: Protocols and Applications*, 1st ed. Boca Raton, FL, USA: CRC Press, 2011.

[72] J. Liu, X. Jiang, H. Nishiyama, and N. Kato, "A general model for store-carry-forward routing schemes with multicast in delay tolerant networks," in *Proc. 6th Int. ICST Conf. Commun. Netw. China (CHINACOM)*, Aug. 2011, pp. 494–500.

[73] J. Ren *et al.*, "On the deployment of information-centric Network: Programmability and virtualization," in *Proc. IEEE Int. Conf. Comput. Netw. Commun. (ICNC)*, 2015, pp. 690–694.

[74] D. Trossen and G. Parisis, "Designing and realizing an information-centric Internet," *IEEE Commun. Mag.*, vol. 50, no. 7, pp. 60–67, Jul. 2012.

[75] L. Zhang *et al. Named Data Networking (NDN) Project*. Accessed: May 11, 2020. [Online]. Available: https://tinyurl.com/y3o5auhw

[76] S. Shailendra, B. Panigrahi, H. K. Rath, and A. Simha, "A novel overlay architecture for information centric networking," in *Proc. 21st Nat. Conf. Commun. (NCC)*, 2015, pp. 1–6.

[77] A. Detti, N. B. Melazzi, S. Salsano, and M. Pomposini, "CONET: A content centric inter-networking architecture," in *Proc. ACM SIGCOMM Workshop Inf. Centric Netw.*, 2011, pp. 50–55.

[78] M. Vahlenkamp, F. Schneider, D. Kutscher, and J. Seedorf, "Enabling ICN in IP networks using SDN," in *Proc. IEEE Int. Conf. Netw. Protocols (ICNP)*, 2013, pp. 1–2.

[79] L. Veltri, G. Morabito, S. Salsano, N. B. Melazzi, and A. Detti, "Supporting information centric functionality in software defined networks," in *Proc. IEEE Int. Conf. Commun. (ICC)*, 2012, pp. 6645–6650.

[80] *Deployment and Securisation of New FunctiOnalities in Virtualized Networking Environments*. Accessed: May 11, 2020. [Online]. Available: http://www.doctor-project.org/

[81] *Architecture for an Internet for Everybody (RIFE)*. Accessed: May 11, 2020. [Online]. Available: https://rife-project.eu/

[82] G. White and G. Rutz. *Content Delivery With Content-Centric Networking*. Accessed: May 11, 2020. [Online]. Available: https://tinyurl.com/y5328v4s

[83] H. Wu *et al.*, "On incremental deployment of named data networking in local area networks," in *Proc. ACM/IEEE Archit. Netw. Commun. Syst. (ANCS)*, 2017, pp. 82–94.

[84] L. Muscariello, G. Carofiglio, and J. Augé, *System and Method to Facilitate Integration of Information-Centric Networking Into Internet Protocol Networks*, CISCO Technol., Inc., San Jose, CA, USA, 2018. [Online]. Available: https://tinyurl.com/y46pc6w4

[85] N. B. Melazzi, A. Detti, G. Mazza, G. Morabito, S. Salsano, and L. Veltri, "An OpenFlow-based testbed for information centric networking," in *Proc. IEEE Future Netw. Mobile Summit (FutureNetw)*, 2012, pp. 1–9.

[86] P. Jokela, A. Zahemszky, C. E. Rothenberg, S. Arianfar, and P. Nikander, "LIPSIN: Line speed publish/subscribe inter-networking," in *Proc. SIGCOMM*, 2009, pp. 195–206.

[87] E. Kohler, R. Morris, B. Chen, J. Jannotti, and M. F. Kaashoek, "The click modular router," *ACM Trans. Comput. Syst.*, vol. 18, no. 3, pp. 263–297, 2000.

[88] *CCNx Over UDP*, PARC, Palo Alto, CA, USA, 2015.

[89] (2012). *NDNLP: A Link Protocol for NDN*. [Online]. Available: https://tinyurl.com/yyx8ezmq

[90] M. Conti, R. Droms, M. Hassan, and S. Valle, "QoE degradation attack in dynamic adaptive streaming over ICN," in *Proc. 19th IEEE Int. Symp. World Wireless Mobile Multimedia Netw. (WoWMoM)*, 2018, pp. 1–9.

[91] M. Conti, R. Droms, M. Hassan, and C. Lal, "Fair-RTT-DAS: A robust and efficient dynamic adaptive streaming over ICN," *Comput. Commun.*, vol. 129, pp. 209–225, Sep. 2018.

[92] *Scalable & Adaptive Internet Solutions (SAIL) European Commission's 7th Framework Program*. Accessed: May 11, 2020. [Online]. Available: http://www.sail-project.eu/about-sail/index.html

[93] *NSF Future Internet Architecture Project*. Accessed: May 11, 2020. [Online]. Available: http://www.nets-fia.net/

[94] G. Carofiglio, M. Gallo, and L. Muscariello, "ICP: Design and evaluation of an interest control protocol for content-centric networking," in *Proc. IEEE INFOCOM Workshops*, Mar. 2012, pp. 304–309.

[95] G. Carofiglio, M. Gallo, L. Muscariello, and M. Papalini, "Optimal multipath congestion control and request forwarding in information-centric networks," in *Proc. 21st IEEE Int. Conf. Netw. Protocols (ICNP)*, 2013, pp. 1–10.

[96] P. Gusev and J. Burke, "NDN-RTC: Real-time video conferencing over named data networking," in *Proc. 2nd ACM Conf. Inf. Centric Netw.*, 2015, pp. 117–126.

[97] S. Mastorakis, A. Afanasyev, I. Moiseenko, and L. Zhang, "NDNSIM 2: An updated NDN simulator for NS-3, revision 2," NDN, Washington, DC, USA, Rep. NDN-0028, 2016.

[98] Z. Zhang *et al.*, "An overview of security support in named data networking," *IEEE Commun. Mag.*, vol. 56, no. 11, pp. 62–68, Nov. 2018.

[99] I. Moiseenko and D. Oran, "TCP/ICN: Carrying TCP over content centric and named data networks," in *Proc. 3rd ACM Conf. Inf. Centric Netw.*, 2016, pp. 112–121.

[100] A. Afanasyev, I. Moiseenko, and L. Zhang, "NDNSIM: NDN simulator for NS-3," NDN, Washington, DC, USA, Rep. NDN-0005, 2012. [Online]. Available: https://tinyurl.com/y2ysqp8v

[101] K. Schneider, C. Yi, B. Zhang, and L. Zhang, "A practical congestion control scheme for named data networking," in *Proc. 3rd ACM Conf. Inf. Centric Netw.*, 2016, pp. 21–30.

[102] A. Suvrat, S. Samar, P. Bighnaraj, R. Hemant, and S. Anantha, "O-ICN simulator (OICNSIM): An ns-3 based simulator for overlay information centric networking (O-ICN)," in *Proc. 1st Workshop Complex Netw. Syst. Smart Infrastructure (CNetSys)*, 2018, pp. 13–15. [Online]. Available: https://doi.org/10.1145/3265997.3266000

[103] *Network Functions Virtualisation (NFV); Management and Orchestration, ETSI GS NFV-MAN 001 V1.1.1 (2014–12)*. Accessed: May 11, 2020. [Online]. Available: https://tinyurl.com/y966jvg4

[104] A. Chakraborti, S. O. Amin, A. Azgin, and R. Ravindran, "Design and evaluation of a multi-source multi-destination real-time application on content centric network," in *Proc. 1st IEEE Int. Conf. Hot Inf. Centric Netw. (HotICN)*, Aug. 2018, pp. 186–192.

[105] CICN. (2017). *Community Information-Centric Networking (CICN)*. [Online]. Available: https://wiki.fd.io/view/Cicn

[106] F. Khan and H. Li, "Ensuring trust and confidentiality for adaptive video streaming in ICN," *J. Commun. Netw.*, vol. 21, no. 6, pp. 539–547, Dec. 2019.

[107] B. Li, D. Huang, Z. Wang, and Y. Zhu, "Attribute-based access control for ICN naming scheme," *IEEE Trans. Depend. Secure Comput.*, vol. 15, no. 2, pp. 194–206, Mar. 2018.

[108] G. Xylomenos *et al.*, "IP over ICN goes live," in *Proc. Eur. Conf. Netw. Commun. (EuCNC)*, Jun. 2018, pp. 319–323.

[109] C. Bernardini, S. Marchal, M. R. Asghar, and B. Crispo, "PrivICN: Privacy-preserving content retrieval in information-centric networking," *Comput. Netw.*, vol. 149, pp. 13–28, Feb. 2019.

[110] G. Acs, M. Conti, P. Gasti, C. Ghali, G. Tsudik, and C. A. Wood, "Privacy-aware caching in information-centric networking," *IEEE Trans. Depend. Secure Comput.*, vol. 16, no. 2, pp. 313–328, Mar. 2019.

Mauro Conti (Senior Member, IEEE) received the Ph.D. degree from Sapienza University, Rome, Italy, in 2009. He was a Postdoctoral Researcher with Vrije Universiteit Amsterdam, The Netherlands. He is a Full Professor with the University of Padua, Padua, Italy, and an Affiliate Professor with the University of Washington, Seattle, WA, USA. In 2011, he joined as an Assistant Professor with the University of Padua, where he became an Associate Professor in 2015 and a Full Professor in 2018. He has been a Visiting Researcher with GMU in 2008 and 2016, UCLA in 2010, UCI in 2012, 2013, 2014, and 2017, TU Darmstadt in 2013, UF in 2015, and FIU in 2015, 2016, and 2018. His research is also funded by companies, including Cisco, Intel, and Huawei. His main research interest is in the area of security and privacy. He published more than 250 papers in topmost international peer-reviewed journals and conference in the above areas. He has been awarded with the Marie Curie Fellowship by the European Commission in 2012 and the Fellowship by the German DAAD in 2013. He is an Area Editor-in-Chief of IEEE COMMUNICATIONS SURVEYS & TUTORIALS, and an Associate Editor for several journals, including IEEE COMMUNICATIONS SURVEYS & TUTORIALS, the IEEE TRANSACTIONS ON INFORMATION FORENSICS AND SECURITY, the IEEE TRANSACTIONS ON DEPENDABLE AND SECURE COMPUTING, and the IEEE TRANSACTIONS ON NETWORK AND SERVICE MANAGEMENT. He was the Program Chair for TRUST 2015, ICISS 2016, and WiSec 2017, and the General Chair for SecureComm 2012 and ACM SACMAT 2013.

Ankit Gangwal received the B.Tech. degree in information technology from RTU Kota, India, in 2011, and the M.Tech. degree in computer engineering from MNIT Jaipur, India, in 2016. He is currently pursuing the Ph.D. degree with the Department of Mathematics, University of Padua, Italy, with a fellowship for international students funded by Fondazione Cassa di Risparmio di Padova e Rovigo. His current research interests are in the area of security and privacy of the blockchain technology and novel network architectures.

Muhammad Hassan received the Ph.D. degree in cyber security from the University of Padua, Italy, in 2019, where he is a Postdoctoral Fellow working with the SPRITZ Research Group. He has been a Visiting Fellow with the Technical University Dresden in 2018. He also worked as a Team Lead Researcher with the Blockchain Security Lab, NCCS, ITU. He has published several papers in peer-reviewed journals and IEEE conferences. His research focuses on security, privacy and access control issues in future Internet architectures (FIA), and related studies, such as secure integration and exploitation of existing technologies (e.g., blockchain, future mobile networks, multimedia streaming) in FIA.

Chhagan Lal received the Ph.D. degree in computer science and engineering from the Malaviya National Institute of Technology, Jaipur, India, in 2014. He was a Postdoctoral Fellow with the Department of Mathematics, University of Padua, Italy, where he was a part of SPRITZ Research Group. He is currently working as a Postdoctoral Research Fellow with Simula Research Laboratory, Norway. During his Ph.D., he has been awarded with the Canadian Commonwealth Scholarship under the Canadian Commonwealth Scholarship Program to work in University of Saskatchewan, Saskatoon, SK, Canada. His current research areas include applications of blockchain technologies, security in software-defined networking, information-centric networks, and Internet of Things networks.

Eleonora Losiouk received the Ph.D. degree in bioengineering and bioinformatics from the University of Pavia, Italy, in 2018. She is a Postdoctoral Fellow working with the SPRITZ Group, University of Padova, Italy. In 2017, she has been a Visiting Fellow with the École Polytechnique Federale de Lausanne. She has published several papers in peer-reviewed journals and IEEE conferences. Her main research interests regard the security and privacy evaluation of the Android operating system and the information-centric networking.

Security and Privacy Analysis of National Science Foundation Future Internet Architectures

Moreno Ambrosin, Alberto Compagno, Mauro Conti, *Senior Member, IEEE*, Cesar Ghali, and Gene Tsudik, *Fellow, IEEE*

***Abstract*—The Internet protocol (IP) is the lifeblood of the modern Internet. Its simplicity and universality have fueled the unprecedented and lasting global success of the current Internet. Nonetheless, some limitations of IP have been emerging in recent years. Furthermore, starting in mid-1990s, the advent of mobility, wirelessness, and the Web substantially shifted Internet usage and communication paradigms. This accentuated long-term concerns about the current Internet architecture and prompted interest in alternative designs. The U.S. National Science Foundation (NSF) has been one of the key supporters of efforts to design a set of candidate next-generation Internet architectures. As a prominent design requirement, NSF emphasized "security and privacy by design" in order to avoid the long and unhappy history of incremental patching and retrofitting that characterizes the current Internet architecture. To this end, as a result of a competitive process, four prominent research projects were funded by the NSF in 2010: nebula, named-data networking, MobilityFirst, and expressive Internet architecture. This paper provides a comprehensive and neutral analysis of salient security and privacy features (and issues) in these NSF-funded future Internet architectures. Prior surveys on future Internet architectures provide a limited, or even no, comparison on security and privacy features. In addition, this paper also compares the four candidate designs with the current IP-based architecture and discusses similarities, differences, and possible improvements.**

***Index Terms*—Network security, privacy, trust, future Internet architectures.**

I. Introduction

THE ORIGINAL Internet was intended to support thousands of users, mainly in North America, accessing shared resources via dumb terminals. Nowadays, the Internet connects over 3 billion of mobile and desktop devices with a variety of applications ranging from simple Web browsing to video conferencing and content distribution. These extreme changes in Internet usage accentuated limitations of the current IP-based architecture and prompted research into alternative internetworking architectures.

Manuscript received August 18, 2016; revised February 15, 2017, June 21, 2017, and October 15, 2017; accepted November 27, 2017. Date of publication January 25, 2018; date of current version May 22, 2018. *(Corresponding author: Alberto Compagno.)*

M. Ambrosin and M. Conti are with the Department of Mathematics, University of Padua, 35121 Padua, Italy (e-mail: ambrosin@math.unipd.it; conti@math.unipd.it).

A. Compagno is with the Department of Computer Science, University of Rome "La Sapienza," 00198 Rome, Italy (e-mail: compagno@di.uniroma1.it).

C. Ghali and G. Tsudik are with the Department of Computer Science, University of California at Irvine, Irvine, CA 92697-3435 USA (e-mail: cghali@uci.edu; gene.tsudik@uci.edu).

Digital Object Identifier 10.1109/COMST.2018.2798280

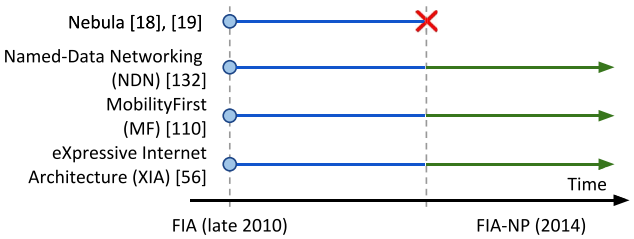


Fig. 1. Timeline of FIA & FIA-NP programs.

In 2010, the National Science Foundation (NSF) launched its "Future Internet Architecture" (FIA) program [95]. Originally, FIA was a 5-year program with the goal of designing a set of candidate next-generation Internet architectures. In 2015, NSF renewed its commitment with a follow-on "Future Internet Architecture – Next Phase" (FIA-NP) program. Unlike FIA which focused on architectural research, FIA-NP emphasizes evaluation, via prototypes, testbeds, trial deployments, and extensive experimentation.

FIA originally included four research projects: Nebula [18], [19], Named-Data Networking (NDN) [132], MobilityFirst (MF) [110], and eXpressive Internet Architecture (XIA) [56]. Each project focuses on a new Internet architecture with a distinct vision and design principles. Nebula envisions a highly-available and extensible core network interconnecting numerous data centers that enable new means of distributed communication and computing. NDN focuses on scalable and efficient data distribution – thus addressing inadequacies of the current Internet's host-centric design – by naming data instead of its location. MF concentrates on scalable and ubiquitous mobility and wireless connections. Meanwhile, XIA stresses flexibility and addresses the need to support different communication models by creating a single network that offers inherent support for communication between various principals (including hosts, content and services) while remaining extensible to future ones. Only three of the original four FIA architectures were selected for continued funding under FIA-NP: NDN, MF and XIA. Figure 1 illustrates the timeline of each project in FIA and FIA-NP programs.

Security and Privacy *by design* is one main goals of all FIA projects. It is mainly motivated by increasing reliance on Internet services, growing range and sophistication of

attacks,[1] and increasing demand for privacy. Given the rocky history of security and privacy in the current Internet, this goal is both very sensible and extremely important.

Contribution. In this paper, we survey and evaluate security and privacy features in the aforementioned four FIA projects. In doing so, we consider the network layer of the current Internet architecture as a point of reference. This is because the network layer reflects the most innovative choices and differences with respect to today's Internet. We also show how each FIA architecture succeeds or fails with respect to security and privacy features of current Internet's network layer, i.e., Internet Protocol (IP) [102] and IP Security Extensions (IPsec) [109]. We also discuss potential vulnerabilities that can be exploited to attack transmission channels, end-nodes, and the network infrastructure. Since different types of resolution services are needed in all FIA architectures, we compare security and privacy features of such services to those of Domain Name System (DNS) [88] and its prominent security extensions, such as the DNS Security Extensions (DNSSEC) [21], and DNSCurve protocol [44].

While we recognize that there are several future Internet architecture proposals outside the FIA program, such as DONA [66], PSIRP [47], NetInf [39] and SINET [131], we decided to focus our survey only on the four involved in the FIA project. We believe that this choice allows us to provide a fair and clear plane for comparison since all FIA projects share the same design principles, which includes security and privacy from the outset.

To the best of our knowledge, this paper represents the first comprehensive security and privacy treatment of four FIA architectures. Since it is impossible to predict which FIA architecture(s), if any, will ultimately succeed, we strive to remain neutral, i.e., to provide a complete and fair analysis.

Prior FIA surveys. An early article by Pan *et al.* [97] overviews Global Environment for Network Innovations (GENI). Unlike this paper, [97] provides a general overview and does not dwell on security and privacy aspects. The work in [64] provides a broad security analysis of the four NSF-founded FIA architectures, plus Recursive InterNetwork Architecture (RINA), Service Oriented Network Architecture (SONATE), and Netlet-based Node Architecture (NENA). The analysis in [64] considers four security features: confidentiality, integrity, availability and authentication. This paper provides a more in-depth security analysis and comparison of the four NSF-funded FIA architectures. In contrast with [64], it: (1) offers a thorough description of the said architectures and their resolution services; (2) treats a larger set of security features; and (3) discusses in detail security mechanisms at the network layer including the resolution services.

Other more focused surveys addresses security and privacy aspects of Information-Centric Networking (ICN) architectures [6], [7], [77], [120]; [77] analyzes security and privacy of NDN alone. Reference [6] investigates denial of service attacks in NDN, while [7] and [120] give a broader security analysis of several ICN architectures, considering several types of attacks on: naming, routing, and caching. Finally, [120] separately considers ICN security, privacy and access control.

Other, more general, surveys of ICN architectures do not focus on security and privacy aspects [14], [23], [121], [122]. Reference [14] analyze Data-Oriented Network Architecture (DONA), Named Data Networking (NDN), Publish-Subscribe Internet Routing Paradigm (PSIRP), and Network of Information (NetInf). The work concentrates on naming, routing and forwarding, caching, and mobility. It marginally considers security and privacy aspects. Reference [23] compares various naming and routing schemes in DONA, NetInf, PURSUIT and PSIRP architectures. References [121] and [122] compare mobility features of NDN, DONA, NetInf, and PURSUIT. None of them [23], [121], [122] discusses security and privacy in much detail.

Organization. We begin by overviewing IP, IPsec, DNS and its security extensions in Section II. Next, Sections III–VI summarize Nebula, NDN, MF and XIA, respectively. Section VII evaluates security and privacy features of these architectures, and compares them with those of IP and IPsec. Section VIII analyzes security and privacy of resolution services used by each new architecture. Section IX summarizes our comparative analysis, and highlights open issues, and possible future research directions. Finally, Section X concludes our paper.

Given familiarity with any FIA architectures, the corresponding sections, can be skipped without the loss of continuity.

II. The Internet of Today

Today's Internet architecture was designed over three decades ago to seamlessly inter-connect multiple heterogeneous networks. At the core of today's Internet is the TCP/IP protocol suite, which puts together protocols, applications and network mediums, and organizes them into four abstraction layers: Link, Internet, Transport and Application. This design leads to an hourglass shape with IP as the network layer as its "thin waist" [15].

We consider IP, which operates at the Internet layer, to be our point of reference when analyzing security and privacy of FIA architectures. IP is responsible for forwarding packets (a.k.a. datagrams) from the source IP interface to its destination counterpart. A host may have one or more IP interfaces, while a router has at least two. Each IP interface is identified by at least one distinct fixed-length IP address.

Another fundamental component of today's Internet architecture, and subject of our analysis, is DNS. DNS is a distributed service that translates application-specific domain names (specified in URLs) into their corresponding IP addresses, allowing hosts to communicate using meaningful names, rather than IP addresses.

In what follows, we briefly describe IP and DNS.

A. Internet Protocol

The cornerstone of IP is addressing of network devices. Every network-layer entity (router or host) is identified by at least one IP address which consists of a network prefix and a

[1] https://www.theguardian.com/technology/2016/oct/21/ddos-attack-dyn-internet-denial-service

host identifier. The boundary between them is flexible, which allows IP addressing to scale.

An IP datagram contains source and destination addresses along with other fields that convey control information. Actual data is carried in the payload field. When a packet is received, a router searches its Forwarding Information Base (FIB) to identify the next hop for that packet. A FIB contains a set of entries, each mapping one or more network prefixes to a router's interface and a next-hop IP address. This allows routers to perform longest-prefix matching on the destination address to identify the next hop. If a packet can not be forwarded, it is dropped and an error message is generated via the Internet Control Message Protocol (ICMP) [101].[2]

One important IPv4 feature is packet fragmentation. If the size of an IP packet is larger than the forwarding interface's Maximum Transmission Unit (MTU) [30], the packet must be divided into smaller chunks, called fragments. A destination host must reassemble fragments to recover the original packet. Other network entities, such as Network Address Translation tables (NATs) [55], [117] and firewalls *might* also assemble fragments.

As mentioned earlier, IP was originally designed for a small and relatively amicable research community. Neither its longevity nor its popularity was foreseen. Thus, it is unsurprising that IP lacks any security and privacy features. In late 1980-s and early 1990-s, as IP started to gain global popularity and the Internet transcended into the commercial sector, IPsec suite [109] was designed to provide basic security services, such as: origin authentication, data integrity, and confidentiality for IP datagrams. The first two are attained via Authentication Header (AH) protocol [62], while Encapsulation Security Payload (ESP) protocol [63] provides all three security features. IPsec supports two modes of operation:

- *Transport:* provides end-to-end communication, e.g., client-server communications. Only packet payloads are encrypted and authenticated in transport mode. Transport and application layers of packets are secured by a hash, thus, they can not be modified, e.g., using NAT. NAT-Traversal (NAT-T) [65] is developed to overcome this issue.
- *Tunnel:* typically used between gateways to provide a secure connection (pipe) between physically separate networks, e.g., different sites of the same organization. Tunnel mode also supports secure host-to-gateway communication. An IP packet is encrypted in its entirety and encapsulated as a datagram with a new outer IP header. One popular application of tunnel mode is Virtual Private Networks (VPN) [83].

IPv6 [42] is a newer version of IP developed to overcome some limitations of IPv4. One of its main new features is extended 128-bit address space (as opposed to 32 bits in IPv4). Another departure from IPv4 is lack of in-network fragmentation. Before sending an IP datagram must first discover the smallest MTU on the path to the destination and fragment the datagram accordingly. To help with this, the Path MTU Discovery protocol [85] was designed and implemented. IPv6 also takes into consideration security and privacy by implementing some features similar to IPsec – such as AH and ESP – as extension headers [1].

In the rest of this paper, we use the term "IP" to refer to both IPv4 and IPv6, unless otherwise specified.

[2]ICMP is also used for sending control messages, such as routing redirect for networks and hosts.

B. Domain Name System

The purpose of DNS is translation of domain names (e.g., those found in URL prefixes) into IP addresses. Domain names are organized in a hierarchical fashion: a top-level domain (e.g., "`.com`") is followed by many sub-level domains (e.g., second-level domain "`example.com`", and third-level domain "`sub.example.com`"). For each domain, DNS assigns an *authoritative name* server that stores information of, and responds to queries for, a specific contiguous portion of the domain name space, called *DNS zone*. This information is contained in *Resource Records* (RR-s) – basic DNS elements which are also carried in DNS replies. Moreover, authoritative name servers might delegate authority over sub-domains to other name servers, thus increasing DNS's scalability.

A user interacts with DNS by issuing a query to a local *resolver*: a process running on the end-user's device which forwards the query to the appropriate name server(s). The resolver sends a UDP (User Datagram Protocol [100]) packet containing the query to the DNS server, which is usually located in the resolver's local network. The server then checks if it can reply to the query from its cache. Otherwise, it fetches the response from other local or remote DNS servers.

DNS queries can be of two types: iterative or recursive. An iterative query allows a DNS server to return the best answer to the resolver, based on its local information, i.e., either a cached RR or an RR belonging to its zone. If the server does not have an exact match for the queried name, it returns a *referral*: a pointer to a DNS server authoritative for a lower level of the domain namespace. The resolver then queries the DNS server in the referral which can also reply with a referral. This process continues until the resolver receives requested information, or an error is generated. In the recursive query, DNS servers reply with either the requested RR or an error. If the DNS server does not have the requested information, it recursively queries other DNS servers.

Although DNS was originally designed as a static distributed database, it now allows dynamic records updates [29], [128] and zone transfers [69]. Also, a recent proposal envisions DNS as a distributed database to store IP related information [8]. For instance, [106] proposes storing IPsec keys related information in DNS records and mapping them to IP addresses.

The original DNS did not include any security or privacy features. DNS Security Extensions (DNSSEC) [21] was added to provide data integrity and origin authentication for DNS messages. In DNSSEC, RR-s are signed by their authoritative servers' keys. The basic mechanism and the query-response protocol of DNS remain unaltered.

There have been other attempts to secure DNS, e.g., DNSCurve [44]. It provides link-level security between clients and DNS servers, using elliptic-curve cryptography. DNSCurve guarantees hop-by-hop query/response confidentiality, authenticity and integrity.

III. NEBULA

Nebula [17]–[19] is a FIA project focused on providing a secure and cloud-oriented networking infrastructure. Its architecture is composed of three tiers:

- *Network core* (NCore) is a collection of routers and interconnections that provide reliable connectivity between routers and data centers. NCore is based on high-performance core routers and rich interconnected topologies [72].
- *Nebula Virtual and Extensible Networking Techniques* (NVENT) represents the control plane of Nebula. NVENT helps in establishing trustworthy routes based on policy routing [22] and service naming [94].
- *Nebula Data Plane* (NDP) is responsible for routing packets along the paths established by NVENT. To guarantee confidentiality, availability, and integrity, NDP ensures that packets for a specific communication can only be carried when all parties, i.e., end nodes and routers in between, have agreed to participate.

A. Nebula Network Layer

The original design of Nebula specifies different candidate network layer stacks for NDP [19], e.g., ICING [91], TorIP [73], and Transit-as-a-Service (TaaS) [99]. From this list, ICING was picked as the most suitable candidate and was included in the Zodiac Nebula prototype implementation [18].

ICING provides a new primitive, called *Path Verification Mechanism* (PVM), which guarantees the following two properties:

- *Path Consent* – every entity in a path between two hosts *consents* the use of the whole path before the communication starts.
- *Path Compliance* – the possibility for each node in a path between two hosts to verify that a received packet: (1) follows the approved path; and (2) has been "correctly" forwarded by all the previous nodes in the path, i.e., according to a specific pre-established policy.

ICING can be deployed either at the network layer or as an overlay on top of IP. In the former case, service providers can deploy ICING nodes as ingress gateways to their networks. However, in the latter case, ICING nodes may become *waypoints*, interconnected using IP, providing waypoint-level path guarantees.

To start communication, a sender must first establish a complete path. Such a path can be provided by DNS with policy enforcement [91]. Figures 2(a) and 2(b) show how forwarding works in ICING and a high-level representation of how the ICING header evolves.

Once a path is selected, the sender requests a *Proof of Consent* (PoC_j), for each node j on the path (action ① in Figure 2(a)). PoCs are cryptographic tokens created by each node transit provider, which attest to the provider's consent to carry packets along the specified path. Each PoC certifies that the corresponding network provider consents to (1) the full path, and (2) a specific policy-based set of local actions (e.g., forwarding) to be performed on packets traversing the path. PoCs are generated by a *consent server*, which is owned by the transit provider or acts on its behalf. Such servers share secret keys with each node (router) in their corresponding providers. Once all PoCs are received, the path is established and packet transmission can begin.

Each packet contains a header (shown in Figure 2(b)) including: (1) the path P consisting of all ICING nodes N_j forming it, and, (2) a list of verifiers V_j, one per node N_j in the path except the sender. This allows each verifier to prove that the packet passed through all previous nodes.

A sender builds a packet header as follows:

1) *Proof of Provenance* (PoP) token, one for each node on the path (action ② in Figure 2(b)), is generated using a PoP key k_j shared with the corresponding node j. In Figure 2(b), PoPs are denoted as $\text{PoP}_{i,j}$, where i is the index of the node generating the PoP and j is the index of the node for which PoP is generated. Specifically, $\text{PoP}_{0,j}$ is computed by node 0 using k_j, path P, and message M itself.
2) Authenticator A_j is computed for each node j using PoC_j, P and M.
3) Verifiers V_j, one per node, are computed by XORing the corresponding A_j and $\text{PoP}_{i,j}$.

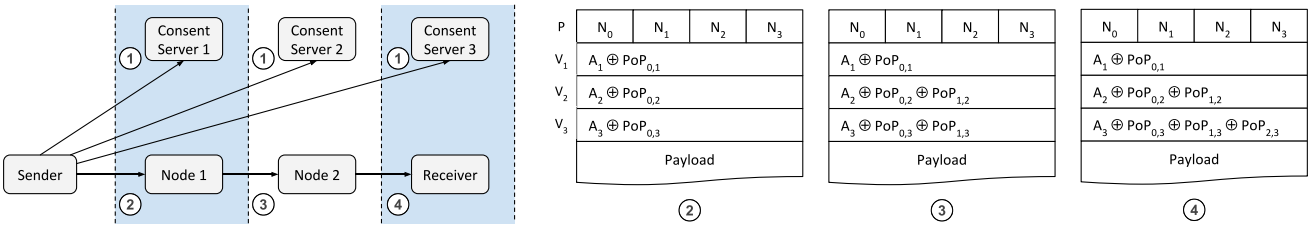
PoP tokens are used by each node on the path to prove that downstream nodes have handled the received packets based on the established policies. When an intermediate node N_i receives a packet, it performs the following actions:

1) Computes the corresponding PoC_i.
2) Computes $\text{PoP}_{j,i}$ using k_j, for each downstream node N_j.
3) Verifies that the received PoC_i and $\text{PoP}_{j,i}$ match the two values computed in the previous two steps. If this verification fails, N_i drops the packet.
4) Derives a shared PoP key k_l, for each upstream node N_l, and computes $\text{PoP}_{i,l}$ as described above.
5) Modifies the verifiers to include the computed PoP, and forwards the packet upstream (actions ③ and ④ in Figures 2(a) and 2(b)).

The previous steps allow any node to guarantee that all packets are forwarded by all the consenting nodes while establishing the path.

B. Nebula Control Plane

The control plane in Nebula is provided by NVENT. NVENT uses declarative networking [75], [76], and allows administrators to provide high-level specifications of their routing policies. NVENT also involves special interfaces, called *service interfaces*, that enable service access and specify the required level of availability. For instance, an emergency service can request high availability, which can be provided by multi-path interdomain routing. A distributed resolution service is used for discovery of other NVENT services. This



(a) Packet forwarding in ICING

(b) ICING packet high-level structure

Fig. 2. ICIN [91] architecture.

service is populated by service providers, e.g., NCore data centers [17].

Serval is an implementation of NVENT based on the concept of service-centric networking [53], [86], which decouples service instances (e.g., Web or email services) from their physical locations (i.e., IP address and port). Serval introduces a new layer, the Service Access Layer (SAL), between the network layer and the transport layer. With Serval, each service is identified by a *serviceID*, a unique identifier that applications use to communicate with the service. In addition, each local traffic flow, representing a connection between two hosts, is identified by a unique *flowID*. The request is handled by SAL, which uses local control plane policies to map the *serviceID* to a service instance. SAL eventually creates a new *flowID* that identifies the established connection. This *flowID* is delivered to the destination host during connection setup, and used by both parties for connection identification. Finally, SAL routes the packet based on specific control plane rules contained in its *SAL table*. For instance, a host application that wants to connect to a specific service might direct the first request to a default Serval router (using its IP address). The SAL of the router then processes the request and take further decisions based on its *SAL table* (e.g., forward to another router or send directly to a known service instance). Moreover, Serval does not directly provide clients a way to learn *serviceID*s: It simply suggests the use of directory services or search engines [94].

Figure 3 presents a view of how all Nebula components integrate to allow a user to negotiate a custom end-to-end path to a specific data center and send the desired packets. First, the user (either the mobile phone or the laptop in the figure) contacts NVENT to request a path to NCore. NVENT determines a suitable path that complies with each transit network's policies and contacts the corresponding consent servers to obtain the necessary PoCs. Once the path and all PoCs are delivered to the user, the latter generates appropriate packet headers and forwards them, using the NDP forwarders network, to the nearest NCore router. This router ensures that all header fields are valid (as described above) and verifies that the negotiated path has actually been traversed. Once verified, the core router forwards received packets to the correct data center using its NCore links.

IV. Named-Data Networking

While IP traffic consists of packets sent between communicating end-points, NDN traffic is comprised of explicit

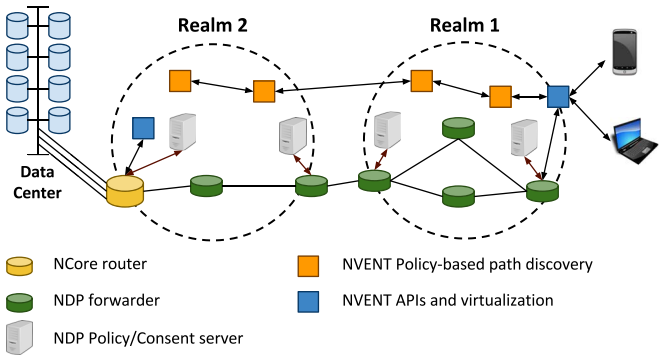


Fig. 3. High-level view of Nebula components integration [17]–[19].

requests for, and responses to, named content objects. NDN is based on the principle of Content-Centric Networking, where content, rather than hosts, occupies the central role in the architecture. NDN is primarily oriented towards efficient large-scale content distribution. Rather than directly addressing specific hosts, NDN users (called consumers) request pieces of content by name. The network is in charge of finding the closest copy of the content, and delivering it. This decoupling of content and location allows NDN to efficiently implement multicast, content replication and fault tolerance.

A. NDN Network Layer

The NDN network layer uses hierarchical structured names to directly address content. Names are composed of a number of components, e.g., `/ndn/bbc/frontpage/news` where "/" represents the boundary between two components. Since names are opaque to the network, they can contain binary or human-readable components.

To support content distribution, NDN defines two types of packets: interest and content (the latter is also called data packet). NDN communication adhered to the *pull* model, that is: every content is delivered to consumers only upon explicit request. Specifically, a consumer issues an *interest* packet carrying the name of the desired content. The network will then forward the interest towards the producer.

One important feature of NDN is in-network caching: any router can store a copy of the content it receives or forwards, and use it to satisfy subsequent interests. Therefore, an NDN interest might be satisfied by the actual content producer or any intermediate router. Along with in-network caching, NDN introduces another important feature called interest collapsing:

only the first of multiple closely spaced (and timed) interests requesting the same content is forwarded by each router.

Each NDN entity (not only routers) maintains the following three components [132]:

- *Content Store* (CS) – cache used for content caching and retrieval. A router's cache size is determined by local resource availability. Each router unilaterally determines what content to cache and for how long. From here on, we use the terms *CS* and *cache* interchangeably.
- *Forwarding Interest Base* (FIB) – table of name prefixes and corresponding outgoing interfaces. FIB is used to route interests based on longest-prefix matching of their names.
- *Pending Interest Table* (PIT) – table of outstanding (pending) interest names and a set of corresponding incoming interfaces, denoted as *arrival-interfaces*

When an NDN entity receives an interest, it searches its PIT to determine whether another interest for the same content is pending. There are three possible outcomes:

1) If a PIT entry for the same name exists, and the arrival interface of the present interest is already in *arrival-interfaces*, the interest is discarded.
2) If a PIT entry for the same name exists, yet the arrival interface is new, the router appends the new incoming interface to *arrival-interfaces*, and the interest is not forwarded further.
3) Otherwise, the router looks up its cache for a matching content. If it succeeds, the cached content is returned and no new PIT entry is needed. Conversely, if no matching content is found, the router creates a new PIT entry and forwards the interest using its FIB.

Upon receipt of the interest, the producer, or an intermediate router, responds with a matching content, thus *satisfying* the interest. The content is then forwarded towards the consumer, traversing the reverse path of the preceding interest. Each router on the path flushes the corresponding PIT entry and forwards the content out on all interfaces specified by that entry. If a content is received by a router with no prior matching interest, the content is considered unsolicited and is discarded. Since no additional information is needed to deliver content, interests do not carry any form of *source addresses*.

The last component at the end of content name can carry an implicit digest (hash) component of the content that is recomputed at every hop. This effectively provides each content with a unique name. Names carrying such digest forms what is called as Self-Certifying Names (SCNs). If an interest is issued using SCN, the retrieved content is guaranteed, due to longest-prefix matching, to be the same content requested by the consumer. However, in most cases, the hash component is not present in interest packets, since NDN does not provide any secure mechanism to learn a content hash *a priori*.

Apart from the name, content packets carry a `Signature` generated by the content producer and covering the entire content. For this reason, each producer is required to have at least one public key, represented as a *bona fide* named content object. Other notable fields in content packets are: the `Payload` containing the actual data of the content and the `ContentType` defining the type of the content, e.g., data or key. Other important fields in interest packets are: the `KeyLocator` which references to the public key required to verify the signature, and the `InterestLifetime` which specifies the lifetime of an interest before it expires (and its corresponding PIT entry is flushed).

Similar to IP, fragmentation of NDN packets can not be avoided. The fact that names can grow arbitrary long might cause interests length to span beyond some link MTU values. In this case, fragmentation must occur. However, since FIB forwarding is based on the availability of the entire name, reassembling of fragmented interests at every hop is a must. Furthermore, interest collapsing can cause content objects to be fragmented (or even re-fragmented) by routers [49]. The question remains to whether to perform a hop-by-hop reassembly [12], or cut-through processing of content fragments [49]. Regardless of its claimed benefits, it is trivial to see that hop-by-hop reassembly incurs unnecessary overhead and end-to-end latency.

Not all interests result in content being returned. If an interest encounters either: (1) a router that can not forward it further or (2) a producer that has no matching content, no error is generated. PIT entries in intervening routers simply expire when no matching content is received. In such case, the consumer can choose to re-issue the same interest after a timeout.

B. NDNS Distributed Database

Since content can be addressed using human-readable names, NDN in principle does not require a resolution service that translate user-friendly names into network addresses. However, as discussed in [11], a distributed database similar to DNS, if existed, provides several benefits to the NDN architecture:

- Cryptographic credential management: Since each data packet is required to be signed, a distributed database is optimal to store and serve security information (e.g., keys and certificates) for namespaces.
- Namespace regulation in the global routing: Similar to the ROVER project [48], a DNS-like service can store information that certifies the authorization of ASes to announce a particular prefix in the global routing.
- Scaling NDN routing: The fact that NDN names can be arbitrary long renders the namespace infinitely large. This exceeds the number of possible routable IP prefixes. Therefore, a DNS-like service can be used to implement a Map-n-encap solution to increase scalability in NDN routing [13].

Two distributed database systems that resemble the DNS design, KRS and NDNS, are proposed in [11] and [79], respectively. These two proposals adopt a similar design and provide the same features. In the rest of this paper we use NDNS to refer to such a distributed system.

Similar to domain names in DNS, NDNS organizes namespaces in a hierarchical set of zones and assigns replicated authoritative servers for each of them. NDNS queries are expressed via interest, in which, names carry all query's necessary information. NDNS responses are carried in content

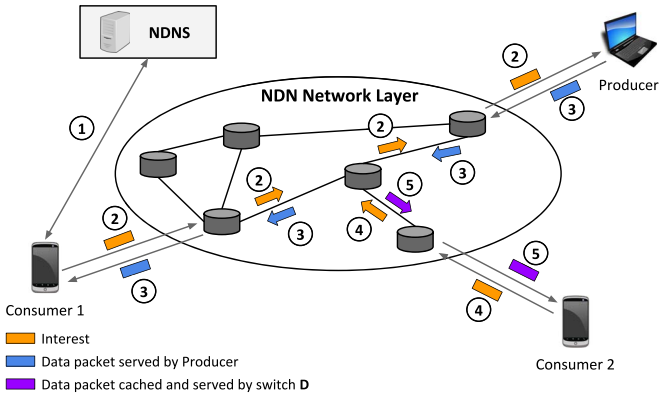


Fig. 4. NDN architecture and forwarding.

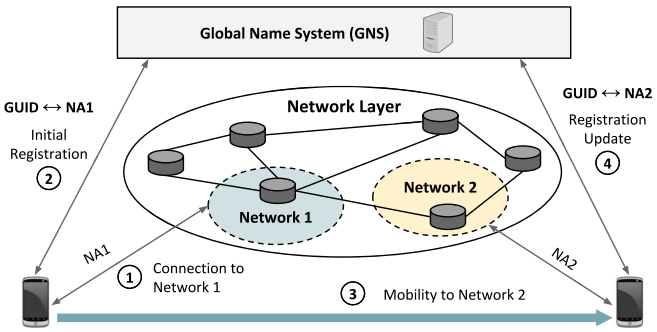


Fig. 5. MF architecture and GUID-NA mapping registration at GNS.

objects where their payloads contain the information requested by the corresponding query.

NDNS reflects many of the DNS protocol machinery: a resolver issues an iterative or recursive query to a local NDNS server. In case of iterative query, the server can reply with the answer (if known), a referral, or a negative response. In case of recursive query, if the NDNS server does not know the answer, it recursively queries another NDNS server until it receives an answer (i.e., the requested data or a negative response). Moreover, secure dynamic updates are provided as in DNS [126].

Figure 4 shows the basic dynamics of NDN naming resolution and forwarding. Consumer 1 retrieves a routable content name from NDNS (Step ① in Figure 4), then issues an interest, which is routed to its Producer (Step ② in Figure 4). The Producer, then responds with a data packet matching the interest, which is routed back to Consumer 1 on the reverse path (Step ③ in Figure 4). Subsequent interests for the same data packet (Step ④ in Figure 4) may be satisfied by intermediate network caches (Step ⑤ in Figure 4).

V. MobilityFirst

MobilityFirst (MF) architecture aims to overcome the inefficiencies and limitations of today's Internet due to mobility. It focuses on scenarios where wireless connections are *ubiquitous* and *pervasive*. To this end, MF has been designed around the concepts of *mobility* and *trustworthiness*. All endpoints must be able to seamlessly switch network connection, and the network must be resilient to compromised endpoints and routers.

MF treats *principals* – devices, content, interfaces, services, human end-users, or a collection of identifiers – as primary addressable network entities. To promote mobility, the (constant) identity of a principal and its (dynamic) network location are strictly separated. This requires a distributed Global Name Service (GNS) to bind principal identities to network addresses. Furthermore, identity and network address separation: (1) facilitates service implementation and deployment; and (2) supports designing routing protocols that overcome link fluctuation and disconnections [93].

We now briefly describe MF's network layer and its GNS.

A. Network Layer

Two types of identifiers are used to differentiate between principal identities and their physical locations.

- *Global Unique Identifier (GUID):* a flat self-certifying identifier that uniquely identifies a principal. GUIDs can be generated using multiple methods depending on the provided service type. For instance, they can be derived from the public key of a host or a service principal or the hash of a content principal. For the sake of usability, a human readable name can be assigned to a principal and later resolved (by GNS) to the corresponding GUID.
- *Network Address (NA):* a flat address that identifies a *network* to which a particular principal (GUID) is connected. MF networks are equivalent to ASes on today's Internet. NAs can be used to identify finer-grained networks such as subnets or organizations. In cases where principals are connected to multiple networks (e.g., using 3G and WiFi simultaneously), multiple NAs can correspond to the same principal.

As a consequence of this addressing scheme, MF defines a new packet type called Packet Data Unit (PDU). PDUs contain source and destination GUIDs, lists of source and destination NAs, payload, and other control fields.

Figure 5 shows a simplified architecture of MF, and registration and mobility handling. When connecting to a network with network address `NA1` (Step ① in Figure 5), a MF host first registers itself at GNS under a specific GUID (Step ② in Figure 5). When such host moves into a different network and obtains address `NA2` (Step ③ in Figure 5), it updates GNS database to reflect its new location (Step ④ in Figure 5).

In order to communicate with a specific GUID, endpoints need to query GNS to obtain the corresponding NA. The retrieved tuple (GUID, NA) is then carried in the PDU header as a routable destination identifier. PDUs are first delivered to their corresponding destination NAs (using inter-domain routing), and then to the destination GUIDs (using intra-domain routing). In case of delivery failure, the packet is stored inside the network (in routers) and GNS is periodically queried for a new or updated GUID-NA mapping.

Multihoming, anycast, and multicast are supported by multicast GUIDs (MIDs). MID has the same format as a regular GUID, except its resolution results in a *set of NAs* (instead of, at most, one). Technically, GNS associates one MID with

several GUIDs (the ones belonging to the multicast group). Resolving all of them results in one or more elements of the output NAs set.

MF can also support content distribution networks. In this case, GUIDs are composed of two parts:

- Content GUID (CID): uniquely identifies the content and is generated by computing the hash of the corresponding content.
- Publisher GUID (PID): points to the network entity providing the content. Such an entity can be the actual content provider, or a third-party content repository.

A router may be equipped with a cache. This opportunistic caching feature facilitates content distribution at the network layer by reducing end-to-end latency and bandwidth consumption. Moreover, MF exploits in-network caching to implement a per-segment (i.e., a continuous set of links with caching routers at each end) reliable chunk (few hundred of megabytes) transfer. Each chunk is fragmented and transmitted according to the segment MTU. Then, the caching router at the other end assembles the entire chunk and stores it. In case of transferring failure, caching routers can re-transmit a chunk via the same, or even a different, path.

B. Global Name Service

GNS is an essential part of the MF architecture. Its main task is to map endpoint identifiers (GUIDs or human readable names) to a set of attributes including the endpoint network address. GNS relies on the following two services:

- *Name Certification Service (NCS):* is equivalent to a Certificate Authority (CA). Its purpose is to (1) assign GUIDs to human-readable names and (2) attest this mapping by generating certificates. MF allows multiple NCSs without a global root of trust. Moreover, if GUID space is large enough, the need for coordination between different NCSs is eliminated.
- *Global Name Resolution Service (GNRS):* a distributed naming service similar to Domain Name System (DNS) that stores the mapping between GUIDs and NAs [74], [89], [125].[3] Two GNRS implementations are evaluated: (1) a distributed hash table maintained among all ASes of the Internet (DMap [127]), and (2) a number of replica-controllers that migrate data (GUID-NA mappings) between a variable number of active replicas (Auspice [111]).

Regardless of its implementation, GNRS clients interact with the service by issuing the following requests to the GNRS resolver:

- `insert`: register a new GUID-NA mapping when a principal joins the network.
- `update`: keep the GUID-NA mapping up-to-date when the corresponding principal migrates to a new network location.
- `query`: retrieve the list of NAs associated with a specific GUID.

[3]GNRS is the actual GNS service that is responsible for maintaining GUID-NA mappings.

In [74], a secure version of the above three GNRS request types is proposed. The secure `insert` and `update` requests adopt a two-step approach to check validity of a GUID-NA mapping. Four network entities are involved in this process: (1) the user issuing the new GUID-NA mapping, (2) the local router to which the user is connected, (3) the border gateway router that connects the user's AS to the rest of the Internet, and (4) the DHCP server which assigns the user's address.

The user generates and signs the request containing the GUID-NA mapping. Local and border routers are in charge of verifying validity of the announced mapping. This is achieved by verifying that the announced NA is the network connected to the user (and the local router), and querying the DHCP server to ensure that the returned NA corresponds to the announced GUID. If the NA matches the one contained in the `update` or the `insert` request, the mapping is accepted and added or updated in the GNRS table.

In the secure `query` request, the protocol involves three entities: the user, the border gateway, and GNRS. The user issues an authenticated request and the border router checks its validity. The router then forwards the request to the appropriate GNRS replica. On receipt, the GNRS satisfies the request with a signed GUID-NA mapping response.

There are several differences [111] between GNRS and DNS [88]. First, GNRS does not restrict the structure of the names, while DNS only supports hierarchical names. Second, scalability of GNRS does not rely on TTL-based caching, which has been proven to be ineffective in the presence of high mobility. Third, GNRS does not statically give the authority to a replicated server for a specific set of names. Active and on-demand replication reduce reliance on passive caching and ensure that mapping replicas are always accessible close to clients.

VI. Expressive Internet Architecture

eXpressive Internet Architecture (XIA) is another research effort aiming to design a new architecture. XIA is based on three types of principals. *Host*-centric networking can support end-to-end communication, such as video conferencing and file sharing. *Service*-centric networking allows users to access various network services such as printing and data storage services. Meanwhile, *content*-centric networking can support Web browsing and content distribution. However, XIA's design is extensible in that it can adaptively provide network evolution and support any new principal type that might emerge in the future.

A core architectural property of XIA is *intrinsic security* of all principals. Any entity should be able to authenticate the principal it is communicating with, without trusted third parties. This can be achieved by binding one or more security properties with principal names. For instance, using the hash of a service (or a host) public key as its name allows entities to verify that they are communicating with the desired principal. Similarly, binding content with its name can be achieved using the hash of the content as its name, allowing users to verify the integrity of a requested content.

XIA defines three main design requirements:

1) All network entities must be capable of clearly expressing their intent. This is achieved by designing the network to be *principal*-centric and allowing in-network optimization. Routers can perform principal-specific operations when receiving, processing, and forwarding packets.
2) The network must be able to adapt to new types of principals. This is essential to support network evolution.
3) Principal identifiers must be intrinsically secure. This depends on the principal type, e.g., authenticating hosts in *host*-centric networking is different than verifying content integrity in *content*-centric networking.

Principal identifiers are denoted as XID, where X defines the type of principle. For instance, HID identifies a host, SID a service, NID a network, and CID a content.

A. Expressive Internet Protocol

In order to comply with the aforementioned requirements, eXpressive Internet Protocol (XIP) is designed. XIP defines packet format, addressing schemes, and behavior of all nodes while processing incoming and outgoing packets from/to various principal types. One of the main features of the XIP addressing scheme is flexibility of defining multiple (fallback) paths to destinations. This prevents downtime and service interruption, especially while gradually deploying new principal types. An XIP address is a directed acyclic graph (DAG) with several properties:

- Each address is a single connected component.
- Each DAG starts with an untyped entry node and ends with one or multiple "sink" nodes. Thus, each node in the address graph has a unique XID except for the entry node.
- Edges define next hops in the path.
- Multiple outgoing edges of a single node are processed in the order they are listed.
- Out-degree of each node is upper bounded to restrict performance overhead.

Using DAGs as a basis for XIP addresses allows applications to build several "styles" of addresses, such as:

- *Shortcut routing* – this style, shown in Figure 6(a) is best suitable for requesting content principal. Each node has a direct edge to the destination principal CID_1, which enables in-network caching. If a node does not have the content cached, the fallback path is processed and the packet is forwarded to the next hop.
- *Binding* – some services require that communication is bound to a specific source or destination. For instance, a service hosted in multiple geographical locations. Users can establish a session with the closest host providing this service. Then, all further communications must be directed to this particular host. Figure 6(b) shows an example of this addressing style. The first packet is destined to SID_1, i.e., the closest host, while the second packet is destined to SID_1 provided by a specific host HID_1.
- *Infrastructure evolution* – as mentioned above, XIA supports gradual network evolution for emerging principal types using fallback paths. Figure 6(c) shows an example of this style. Assume that NID_1 is gradually deploying service SID_1. All NID_1 routers that are not yet updated to recognize and process SID_1 use the fallback path through HID_1 and HID_2.
- *Source routing* – Figure 6(d) gives an example of this addressing style, in which the source routes the packet to the destination through a third party domain and service, NID_a and SID_a, respectively.
- *Multiple paths* – this supports recovery from link failures. An example of this style is shown in Figure 6(e).

Figure 7 shows a high level overview of an XIA router. Its modular design allows efficient multi-principal processing and supports network evolution. Each router contains two main XID-specific processing modules:

- *Source XID-specific processing:* necessary for certain XID types. For instance, in case of a reply to a CID request, the "CID processing" unit can implement in-network content caching.
- *Next Destination XID-specific processing:* invoked by the *Next Destination XID-specific Classifier* which determines the appropriate forwarding action. Similar to source processing, this module consists of several units that carry on XID-specific operations right before forwarding the packet.

If all outgoing DAG edges of a node lead to unrecognizable XIDs, the packet is dropped and an unreachable destination error is generated. It is the responsibility of user applications to provide appropriate fallback paths to avoid forwarding failures at any router. Usually fallback paths are built using well-supported principals, e.g., HID and NID.

B. Principals

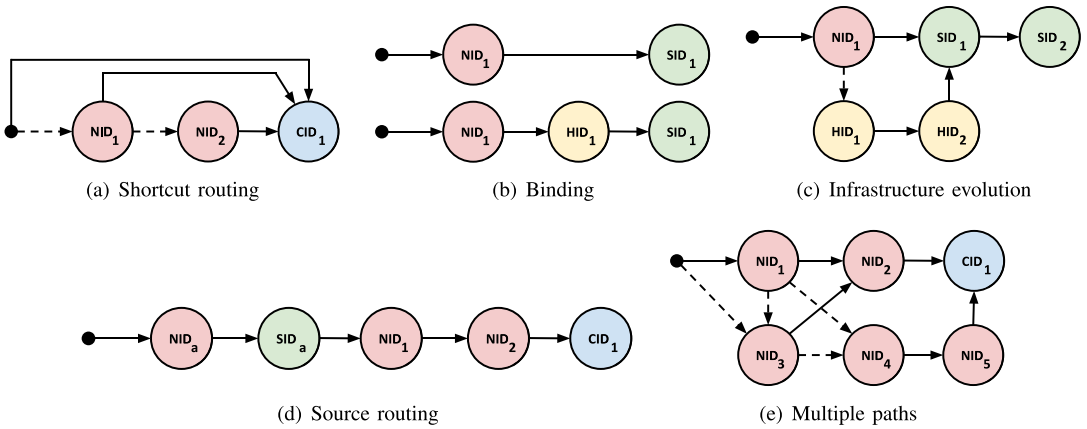
As mentioned above, principals in XIA support emerging communication paradigm on the current Internet. When introducing a new principal, the following issues arise:

- What does it mean to communicate with a principal of this type?
- How is the principal's unique XID generated and how does it map to intrinsic security properties?
- What are the source and next destination XID-specific processing actions that routers should perform and how can such actions be implemented?

We now describe several principal types and discuss their addressing schemes, in-router processing behaviors, and security properties.

1) Network and Host: Network and host principal identifiers are denoted as NID and HID, respectively. They are generated by using the public key hash of the network or the host. Unlike hosts on current Internet, each XIA host has a unique HID regardless of the interface it is communicating through. This feature helps support host mobility. In order to support fallback paths, all XIA routers should implement NID and HID processing modules.

As mentioned above, the fact that the network and host addresses are derived from their corresponding public keys allows users to verify the identity of entities



(a) Shortcut routing (b) Binding (c) Infrastructure evolution

(d) Source routing (e) Multiple paths

Fig. 6. XIP Addressing Styles [56].

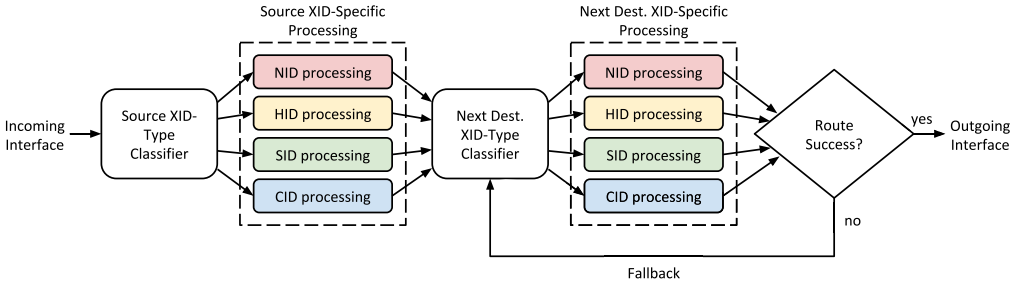


Fig. 7. XIA Router Diagram [56].

with whom they are communicating. Furthermore, this security requirement helps defend against address spoofing, Denial of Service (DoS), and cache poisoning attacks.

2) Service: Services in XIA represent applications in today's Internet. Users communicating with a service SID can use a destination address of the form NID:HID:SID. In today's terminology, this is analogous to sending a packet to a specific host in a specific network and indicating the associated protocol and port number.

Since different services might require different specialized processing, implementing in-router source and next destination processing modules is a challenge. Therefore, routers are only required to perform default processing, routing, and forwarding of SID packets. All other specialized processing should be handled by end-nodes.

SIDs are generated by computing the hash of the service public key. This inherits security properties similar to NIDs and HIDs.

3) Content Principals: This principal type signifies user's intent to retrieve content. Packets carrying content identifiers (CID) as destination addresses will be routed all the way to the node hosting the content. Routers can use a cached version of the content as a reply to such packets. As mentioned above, caching is implemented by routers source XID-specific processing module.

CIDs are generated based on the cryptographic hash of the content they address. This binds the content to its name, forming a self-certifying name.

VII. Network-Layer Security and Privacy

As mentioned earlier, security and privacy by design is one of the key NSF-stipulated guidelines for all FIA projects. In this section, we provide a comparison between the security and privacy features offered at the network layer of each architecture introduced above, and compare them with IP/IPsec. We consider the following security and privacy features, which we consider to be essential [112].

- *Trust:* confidence (sometimes based on inconclusive evidence) (1) that an entity will behave as expected; and (2) that a content comes from a trusted source.
- *Data origin authentication:* corroboration that the source of the received data is as claimed.
- *Peer entity authentication:* corroboration that a peer entity in an association is the one claimed.
- *Data integrity:* assurance that data has not been tampered with, in any (unauthorized or accidental) manner.
- *Authorization and access control:* respectively: (1) a secure means for an entity to access (e.g., read, write, delete) some resource, and (2) protection of a resource against unauthorized access.
- *Accountability:* assurance that all actions can be securely traced to their source(s).
- *Availability:* accessibility and usability of a resource upon demand by an authorized entity.
- *Data confidentiality:* unavailability of data to unauthorized entities.
- *Traffic flow confidentiality:* a set of countermeasures to traffic analysis.

- *Anonymous communication:* inability to determine identities of communicating entities.

We consider the last three as instrumental to privacy at the network layer. In the rest of this section, we discuss each feature separately. Tables I and II summarize our comparison on security and privacy features, respectively.

A. Trust

IPsec defines trust as a one-way relationship between two or more entities (hosts or networks). This relationship is captured by a Security Association (SA). An SA can be viewed as a "contract" between the peers. It describes security services and contains security information needed to protect communication.

Entities involved in secure communication in IPsec establish SAs via the ISAKMP[4] [84] protocol and exchange necessary cryptographic material using the Internet Key Exchange (IKE) protocol [32]. Host authentication in ISAKMP and IKE can be achieved via either digital signatures, or pre-shared keys. Digital signatures require the use of certificates to bind entity identifiers to public keys. This implies the existence of a CA to create, revoke, and distribute certificates.

Nebula's ICING-based network layer defines trust orthogonally to IPsec: between a host and all nodes forwarding its packets. As described in Section III, a host agrees on a "contract" with the network providers carrying its data (i.e., the path negotiated using NVENT) to specify the operation executed at each hop. Such contracts are cryptographically enforced. ICING assumes mutual trust between forwarding nodes and consent servers that are responsible for creating PoC tokens. Therefore, this notion of trust does not require any PKI [91]. However, ICING does not provide an end-to-end definition of trust, which can be added by adopting an IPsec-like approach.

Unlike IP, the notion of trust in NDN is directly associated with contents and not with hosts and networks. Trust in content can be expressed at different levels of granularity: from a single content object to an entire namespace. Recall that a content object is signed by its producer, which allows anyone to verify its origin and authenticity. Origin refers to the content producer and not to any entity that might store a copy of that content. To authenticate a content and its origin, its signature must be verified. This requires the verifier to retrieve, and establish trust in, the corresponding public key.[5] However, network-layer trust management is unspecified and relegated to individual applications. A detailed discussion of this topic can be found in [51].

MF places trust in the principal. Depending on the principal type, trust may be established with: (1) hosts, similar to IPsec, (2) content, similar to NDN, and (3) centralized or distributed services. Trust semantics in XIA also vary depending on the principal type. However, the intrinsic security feature of these principals (described in Section VI) increases trustworthiness of end-to-end communication and content retrieval. For instance, ensuring that a content hash matches its identifier allows receivers (and caching routers) to trust that content.

As shown in Section VI, an XIA address consists of a DAG containing a (partial) path to the destination. To provide trusted path selection for host-to-host communication, SCION is integrated with XIA [92]. SCION [133] is an architecture that provides control and isolation for secure and highly available end-to-end communication. The network is divided into multiple trust domains consisting of several Autonomous Systems (ASes) that trust each other. Each domain has a trusted root AS responsible for relaying packets to and from other domains. Roots initiate path establishment to all hosts in their domains based on local policies and available bandwidth. This process results in constructing a path between each host and its domain root. Whenever two XIA hosts, in different domains, want to communicate, the two half paths (from each host to its domain root) are combined to establish a complete end-to-end path. Such path is trusted since it is created by the trusted roots of each domain.

B. Data Origin Authentication

IP (IPv4 in particular) does not provide any form of authentication. A separate add-on method, IPsec, provides entity authentication via AH and ESP protocols.[6] In transport mode, two hosts securely negotiate a shared secret key. This key is later used to generate a Message Authentication Code (MAC) [68] for each packet. Successful MAC verification ensures authenticity of received packets and their origin. In case of gateway-to-gateway communication, gateways can only verify that the received data originated by *any* (not a specific) host connected to the network at the other end of the tunnel. In host-to-gateway communication, the gateway can actually verify that the data originated by the involved host, while the latter can only verify that received data is originated by the network located behind the gateway. This partial authentication opens the door for insider attacks.

Nebula's ICING-based network layer does not directly provide data origin authentication, which is delegated to applications.

NDN provides data origin authentication via content signatures. Before consuming content, consumers are required to verify its signature [132]. However, this operation is optional for routers because signature verification is an expensive operation at line speed and comprehensive trust management is not viable at the network layer. Even if we assume that routers know all possible application trust models, establishing trust in content is complicated and expensive. For instance, traversing a PKI hierarchy requires routers to fetch and verify public key certificates until a trusted anchor is reached.

On the other hand, NDN interests can optionally be authenticated using digital signature [3]. In a signed interest, the last component of the name carries a signature computed by the consumer. In case the interest carries a small payload,[7] the

[4]ISAKMP: Internet Security Association and Key Management Protocol.

[5]Public keys in NDN are distributed as special content objects with type `KEY`. An object of this type is signed by its issuer, i.e., a CA.

[6]Recall that IPv6 implements both AH and ESP as extension headers.

[7]An interest can carry a small payload to minimize delay in a communications (e.g., 0.5 RTT rather than 1 RTT to retrieve the data), as well as to support push-model communication of small data that fits in a single packet.

TABLE I
NETWORK SECURITY FEATURES COMPARISON

		IP/IPsec	Nebula	NDN	MF	XIA
Trust	*Trusted Model*	Hierarchical.	Hierarchical.	Hierarchical.	Hierarchical.	Distributed.
	Trusted Entity	Host.	Forwarding nodes.	Content.	Host, Content, Service.	Host, Content, Service.
	Trust Management	PKI + certificates, shared keys.	PoC tokens.	Certificates.	Distributed architecture + certificates.	Undefined.
Data Origin Auth.	*Coverage*	Gateway-to-Gateway, Host-to-Gateway.	None.	Producer-to-Consumer.	None.	None.
	Type of packet	Every that belongs to an IPsec SA.	None.	Content, Interest if signed.	None.	None.
	Mechanism	HMAC.	None.	Signature.	None.	None.
Peer Auth.	*Authenticated Entity*	Host, Gateway.	Consent Server.	None.	None.	None.
	Mechanism	Signature, Shared key.	PoC.	None.	None.	None.
Integrity	*Coverage*	Gateway-to-Gateway, Host-to-Gateway.	Host-to-Forwarding nodes.	Producer/Cache-to-Consumer.	Producer/Cache-to-Consumer.	Producer/Cache-to-Consumer.
	Type of packet	Every.	Every.	Content, Interest if signed.	Content.	Content.
	Mechanism	IPsec tunnel + HMAC.	Hash.	Signature.	Hash.	Hash.
Authz. & Access Cont.	*Enforced on*	Router, Host.	Consent Server.	Router, Producer.	Router, Host.	Router, Host.
	Mechanism	ACL.	ACL.	ACL, Encryption.	ACL.	ACL.
Accountability	*Entity accounted*	Host.	Host.	Producer, Consumer.	None.	None.
	Mechanism	Egress Filter on router, IPsec channel.	Path Consent.	Signature in Content or Interest packets.	None.	None.
Availability	*Bandwidth Depletion Attacks*	No architectural design countermeasures	Path consent can block attacks. Consent Servers are Single Point Of Failure.	Pull model prevents flooding with content. Weak to interest flooding.	No architectural design countermeasures.	No architectural design countermeasures.
	Routers Resource Exhaustionn	Weak to fragmentation attack on NAT-enabled IP routers.	Weak to attack targeting path consent verification on routers.	Weak to Interest Flooding attack on PIT.	None.	None.
	Cache-Related Attacks	None.	None.	Weak to Content poisoning. IKB rules as countermeasure to content poisoning. Weak to cache pollution.	SCN as countermeasure to Content poisoning. Weak to cache pollution.	SCN as countermeasure to Content poisoning. Weak to cache pollution.

signature will authenticate the consumer generating it, thus providing data origin authentication for interest payload.

MF and XIA do not provide any data origin authentication at the network layer.

C. Peer Entity Authentication

IPsec provides peer entity authentication during SA establishment of a secure communication. ISAKMP and IKE, the IPsec's protocols used to establish SAs, can achieve peer entity authentication using digital signature or pre-shared key. Digital signatures requires the use of certificates, which bind entity identities to their public keys. The use of certificates implies the existence of a trusted third party or a CA to create, sign and properly distribute certificates. Pre-shared keys on the other hand requires the communicating parties to agree on the shared secret key before communication begins.

TABLE II
NETWORK PRIVACY FEATURES COMPARISON

		IP/IPsec	Nebula	NDN	MF	XIA
Data Conf.	*Coverage*	Gateway-to-Gateway, Host-to-Gateway.	Host-to-Host.	Producer-to-Consumer.	Host-to-Host.	Host-to-Host.
	Mechanism	IPsec ESP header (payload encryption).	ICING + end-to-end encryption.	Payload encryption. Human-readable name might leak information.	End-to-end encryption.	End-to-end encryption.
Traffic flow Conf.	*Coverage*	Gateway-to-Gateway, Host-to-Gateway.	None.	Producer-to-Consumer.	None.	None.
	Mechanism	Padding (no mandatory on IPsec specification).	None.	None.	None.	None.
Anonymous comm.	*Level of anonymity*	Partial using IPsec in tunnel mode.	None.	Partial, consumer has no address. Router's state can be used to de-anonymize consumers.	None.	None.
	Additional Mechanism to achieve full anonymity	Tor.	TorIP.	Andana.	Tor.	Tor.

In Nebula, ICING allows a sender to authenticate the entities issuing the necessary cryptographic tokens, i.e., PoCs. However, ICING design does not specify how PoCs are retrieved, nor does it specify how entities are authenticated [91].

At its current state, NDN does not provide peer entity authentication for consumers and producers. However, in case the authentication of one or both entities is necessary, applications can exploit some features provided by the network layer. Considering consumers, signed interests can facilitate their authentication. Whereas for producers, content signature can ensure that the content is generated by the expected producer. Moreover, if interests must be satisfied by producers only (and not in-network caches), they should carry unique names that avoid cache hits and guarantee their delivery to corresponding producers.

Similar to NDN, MF and XIA do not provide peer entity authentication. However, the usage of self-certifying identities as principal identifiers facilitates entity authentication. Recall that for host, network, and service principals, identifiers are generated by computing the hash of the public key associated with these principals. Therefore, entity authentication can be achieved by ensuring that such principal identifiers match their keys. Peer authentication for content principals can be achieved similar to NDN since such principals are self-authenticating. Neither MF nor XIA provides a secure mechanism for securely retrieving content identifiers.

D. Data Integrity

Although IPv4 header contains the *Header Checksum* field that provides transmission error detection (a form of integrity check), it does not prevent packet manipulation. In fact, both versions of IP, introduced in Section II-A, completely delegate integrity to IPsec AH and ESP protocols. Specifically, the HMAC values in these protocol headers are used to achieve integrity. Depending on the IPsec mode used, host-to-host, gateway-to-gateway, or host-to-gateway integrity guarantees can be provided by both AH and ESP protocols. However, AH provides integrity for the entire packet (except for mutable fields), while ESP guarantees packet headers integrity only.

Each packet in Nebula carries a sequence of cryptographic verifiers V_j, one for each hop on the path (see Section III-A for details). The packet hash is used as part of V_j's calculation. Therefore, ICING guarantees that neither the packet nor the path can be modified. Also, ICING is recommended only at domain gateways [91]. Thus, integrity can only be guaranteed by border routers. Within domains, such guarantees are deferred to either the network-layer protocol or the application.

The way NDN provides integrity is through content signature. By verifying this signature, consumers and routers can always detect malicious manipulation. However, when content is requested using SCNs, data integrity is achieved by comparing the content hash to the last name component of its name. Furthermore, only signed interests can provide interest integrity.

In both MF and XIA, integrity is only available for content principal types. This is again due to the fact that such a principal identifier is generated based on the content hash itself. Whenever a content is received, its hash is compared with its identifier to ensure content integrity. For other principal types, MF and XIA defer integrity guarantees to the application.

E. Authorization and Access Control

Access control in IP is achieved by restricting access based on source and destination addresses. This is implemented using Access Control Lists (ACLs) [31], which contain a set of rules that grant or deny access to network resources. When implemented in routers, ACLs specify whether a received packet will be forwarded further to the next hop, or simply getting dropped. Whereas host ACLs are used to decide whether to forward packets up the stack towards the application. Since IP does not natively provide packet integrity, address spoofing can

be used to circumvent ACL rules. Employing IPsec, however, prevents such actions.

In Nebula, paths must be established before communication begins, i.e., clients must obtain required PoC tokens. Therefore, access control can be implemented by the consent server granting or denying PoC requests. Traffic sent without valid PoC tokens can be easily detected and dropped.

Unlike IP, enforcing access control in NDN should be done based on content and not network entities. Although not implemented in practice, ACLs can still be used to implement access control. In this case, rules are applied on interest messages and content objects based on the names they carry. Longest-prefix can also be employed to grant or restrict access to entire namespaces. Due to the fact that NDN interests do not carry source addresses, access control on the consumer granularity can only be achieved in cases where interests are signed or carry some form of consumer identity [50].

One way of providing access control in NDN is by using encryption. Producers can encrypt their content and disseminate decryption keys to authorized consumers only. Such keys can be encapsulated in content objects and should not be cached. One drawback of this approach is that it requires consumers to issue at least two interests for each content (one to request the content itself and one to request the corresponding key).

Since MF and XIA can support different principal types, they facilitate the combination of both NDN- and IP-based access control schemes. For content principals, access control is done at the content granularity, similar to NDN, e.g., content is encrypted using keys disseminated to only authorized users. For all other principal types, ACLs can restrict access to hosts and other network services.

F. Accountability

One of the main problems in IP is accountability. In fact, IP is subject to source address spoofing that lead to the inability of tracing back the entity responsible for a particular action. A simple countermeasure against IP spoofing requires ASes to implement egress filtering and ensure that all outgoing traffic carries source addresses owned by these ASes. IPsec guarantees peer entity authentication when establishing SAs between hosts. Thus, accountability can be achieved.

Nebula provides accountability through path establishment. All routers on a path consent to use the whole path before the communication begins. Moreover, the fact that these routers pre-agree on performing a specific set of actions on each packet passing through allows the detection of any malicious activities.

NDN provides full accountability of producers. Since every content is signed by its producer, tracing the producer responsible for generating content is a trivial task. However, accountability can not be provided if content is served from router caches. Consumers accountability, on the other hand, can only be achieved when they issue signed interests, or include their identities in the interests themselves. Otherwise, accountability can not be provided.

Both MF and XIA do not provide accountability at the network layer. However, signing requests and responses can provide this feature in a similar fashion to NDN, especially for content principals. Also, IPsec-similar techniques can be employed to provide accountability for other principal types.

G. Availability

Denial-of-Service (DoS) and Distributed Denial-of-Service (DDoS) are well- known attacks on availability. In the following, we discuss DoS and DDoS attacks on the network layers of current and future Internet architectures discussed above. We also discuss new (and possibly more serious) types of attacks triggered by the new architectures. We specifically exclude availability issues that arise due to network misconfigurations, disasters, hardware/software faults, or any other causes that are not a direct consequence of an attack.

Bandwidth Depletion Attacks. The current Internet is susceptible to bandwidth depletion attacks [116] that aim to exhaust bandwidth of a specific link. These attacks can be conducted in two ways: (1) distributed – with packets sent at a relatively low rate by each attack source, or (2) centralized – a single powerful adversary flooding the target link at a high rate. Due to today's high bandwidth allocation and redundancy, the latter are harder to perform.

Several mitigation and prevention techniques have been proposed and implemented. Examples include: (1) tracing malicious traffic back to the source of the attack [25], [108], [113], (2) distinguishing between legitimate and malicious traffic [20], [129], (3) using puzzles to increase the cost for bandwidth consumption [40], [60], and (4) rate-limiting traffic that causes congestion [59], [80], [118]. However, none of the above is a panacea.

Bandwidth depletion at the data plane is harder to mount in Nebula, since senders (potential adversaries) must obtain consent of all nodes on a path before sending packets. Thus, unauthorized packets are dropped by adversary-facing routers. Unfortunately, this only shifts the attack focus from the network layer to consent servers: a single consent server might be responsible for numerous routers in its domain. Thus, lowering the former's ability to issue PoCs can effectively disable all routers in the domain.

NDN is more resilient to bandwidth depletion attacks as compared to IP counterpart. Recall that NDN communication adheres to the *pull* model, i.e., content is forwarded only in response to a prior interest. This prevents adversaries from flooding the network with unsolicited content. However, an adversary can still flood the network with a large number of interests.

Since MF and XIA use a communication model similar to IP, both are susceptible to bandwidth depletion attacks. Countermeasures similar to those applicable to IP can be adopted. However, as mentioned above they can only lower, not eliminate, the impact of such attacks.

The work in [96] suggested integrating STRIDE into XIA to protect against DoS attacks. STRIDE [58] is an architecture resilient to bandwidth depletion (D)DoS attacks. It modifies SCION path establishment to perform tree-based bandwidth allocation. Whenever a trusted domain root initiates the path establishment process, bandwidth is allocated as the path is branching out as a tree from that root. This guarantees required

bandwidth for benign flows. STRIDE also supports long-term static paths to provide high available connectivity.

Routers Resource Exhaustion. Exhausting router resources is another usual DoS and DDoS attack goal. For example, a NAT-enabled IP router might assemble fragments and, at any given time, might maintain multiple reassembly buffers. Each IP packet fragment includes a 16-bit field indicating the original packet size. An attack can involve sending a single fragment with a very large original packet size. This forces the target router to allocate a new buffer and wait for remaining fragments, which will never arrive.

Other types of attacks might be computational in nature. In Nebula, an adversary can generate numerous packets forcing a router to perform all verification operations described in Section III. Such attacks cost very little for an adversary since malicious packets can simply include invalid (e.g., random) PoC and PoP values.

In NDN, the PIT is a mandatory router component that enables interest collapsing and content delivery without source or destination addresses. However, since it is a limited and valuable resource makes the PIT susceptible to exhaustion. An adversary can easily generate and send (in close proximity) a large number of interests that fill up a router's PIT. To prevent interests from being collapsed, each can refer to some *nonsensical* content. Once the PIT is full, a router can either: drop new incoming interests, or remove old PIT entries to make space for new ones. Both options, however, can adversely impact past or future interests. This type of attack is called Interest Flooding (IF).

Unfortunately, there is no comprehensive remedy for IF attacks. Although several countermeasures have been proposed, they are ineffective against smart adversaries and only manage to lower the volume of IF attacks [10], [34]. One comprehensive, though drastic, remedy is to eliminate the PIT – the main target of IF attacks. For this reason, [52] suggests a modified Content-Centric Networking (CCN) architecture without router PITs.

Router resource exhaustion attacks do not apply to MF. Similar to IP routers, MF routers do not perform expensive cryptographic operation nor do they handle per connection-state. Moreover, MF does not implement NAT, rendering NAT-based attacks irrelevant.

Cache-Related Attacks. Despite the benefits of in-network caching, it prompts new types of (D)DoS attacks that are not relevant to today's Internet: content poisoning and cache pollution. We now discuss resiliency of NDN, MF, and XIA against these attacks. Nebula is excluded since it does not provide in-network caching.

a) Content Poisoning: Content poisoning occurs when an adversary injects fake content into router caches. A fake content is not generated by a benign producer and, consequently, does not satisfy user requests. If cached in routers, such content is used to satisfy future requests.

Since NDN adheres to the pull model makes it harder, yet still feasible, to inject fake content into router caches, in at least two ways:

- Reactive: the adversary is a node eavesdropping or controlling a link, e.g., an upstream malicious router. The adversary responds to interests on that links with a fake content that is cached in all downstream routers.
- Proactive: this method involves the adversary that, anticipating demand for certain content, issues one or more bogus interests (perhaps from strategically placed zombie consumers), before genuine interests are issued. The adversary then replies with fake content (from a set of compromised routers or compromised producers) thus pre-poisoning the caches of all routers forwarding the bogus interests.

Reference [51] investigated the causes of content poisoning and proposed a simple approach for its full mitigation. The main idea is for consumers and producers to collaborate in providing routers with enough contextual trust information to perform a single signature verification per content.[8] This approach is captured by a simple tenet called *Interest Key Binding (IKB)* rule.

An orthogonal content poisoning mitigation mean is the use of SCNs. By definition, an SCN contains a value that (uniquely) identifies the principal to which it refers. In case of a content principal in MF and XIA, and content objects in NDN, an SCN is the hash of the data itself. When a user requests content using SCN, the network guarantees that requested content will be correctly delivered. As a result, MF and XIA obviate content poisoning attacks, by design. It is worth mentioning that using SCNs does not prevent adversaries from injecting fake content in router caches. Instead, it guarantees that benign users will not receive such fake content. In order to use SCNs, the system needs to be bootstrapped *without* them, e.g., by using IKB, as described above.

b) Cache Pollution: Pollution is another type of (D)DoS attacks against router caches. In such attacks, adversaries attempt to manipulate reference locality of caches, causing incorrect decisions made by cache eviction strategies. This causes routers to possibly evict popular content reducing the overall content distribution performance. NDN, MF, and XIA are all susceptible to this attack.

Conti *et al.* discuss this attack in [37]. It is shown that with even limited adversarial resources, a highly effective cache pollution attack can be mounted. In fact, even small cache locality manipulation can cause a significant content distribution disruption [45]. It is also shown in [37] that launching pollution attacks on large networks is relatively easy, and smart adversaries reduce the effectiveness of proposed countermeasures.

Cache pollution attacks do not prevent users from retrieving the requested data. Instead, they negatively effect the performance of content distribution, and eliminate the benefits of in-network caching.

H. Data Confidentiality

The natural way to achieve data confidentiality at the network layer is by using encryption.

IP does not provide data confidentiality. This is done by using IPsec. The level of confidentiality depends on the

[8]This assumes that in the near future, public key-based signature verification will be available in hardware and will be achievable at line speed.

mode of operation. In transport mode, ESP only encrypts the IP packet payload and data confidentiality is host-to-host. Tunnel mode extends confidentiality to the entire encapsulated IP packet, including both payload and header. However, data confidentiality can only be achieved in host-to-gateway or gateway-to-gateway scenarios. Also, ESP confidentiality is not generally effective against active adversaries. It has been demonstrated that achieving confidentiality without a strong integrity mechanism, or even applying integrity before encryption, can only protect against passive adversaries [24], [43], [67]. Thus, even though IPsec provides confidentiality, poor usage practices can negate its benefits.

Nebula's ICING-based network layer does not natively provide data confidentiality. Instead, it can be achieved by combining ICING with end-to-end encryption of the packet payload.

Data confidentiality in NDN can be attained by encrypting content payload. This is not supported by the architecture and is left to the application. However, even if content is encrypted, the fact that it carries a human-readable name might leak information about its data.

MF and XIA provides content principal confidentiality using methods similar to the those used in NDN. Fortunately, and due to the fact that such principal identifiers are generated using the hash of the content itself, inspecting them does not leak information about the encrypted content. Moreover, confidentiality of data communicated between host, network and service principals can be achieved using similar techniques to IPsec.

I. Traffic Flow Confidentiality

It is well known that encryption does not protect against statistical traffic analysis – attacks that monitor traffic in order to extract properties, such as volume and timing [104].

IPsec provides some traffic flow confidentiality by padding packet payloads to hide their size patterns. However, according to IPsec specifications, this is not mandatory and, therefore, may not be supported in all IPsec implementations [109].

NDN, MF, Nebula and XIA are all susceptible to traffic analysis attacks Fortunately, padding can be used to provide traffic and flow confidentiality.

Another architecture-agnostic alternative is to add artificial delays to communications to better hinder time-based attacks. This, however, comes at the expense of increasing end-to-end latency and reducing overall network performance, especially for time-sensitive traffic.

J. Anonymous Communication

IP (with or without IPsec) does not support anonymous communication. This is mainly because source and destination addresses are in the clear in packet headers. However, partial anonymity can be achieved using the tunnel mode of IPsec along with ESP. This is because tunnel mode allows the ESP protocol to encrypt the original IP packet along with the source and destination addresses, and it encapsulates that packet into a new one with a new header reflecting gateway addresses. This combination hides end-host identities among the set of other hosts connected to respective end-networks. However, this is only effective if the adversary is eavesdropping on the link between the two gateways and is not located inside one of the end-networks. Furthermore, in case of host-to-gateway tunnel mode, only anonymity of hosts located behind the gateway is preserved.

Crowds [105] is one of the first proposals to achieve user anonymity. In it, a message is randomly forwarded between group members before it reaches its destination. Therefore, none of the group members nor the end recipient learn the actual source of the message. The Onion Router (Tor) [119] is another method that provides anonymous communication through a "circuit." Circuits are multi-hop encrypted communication channels established using at least three Tor nodes. Theoretically, Tor guarantees anonymity with respect to an adversary controlling, at most, two Tor nodes. However, flawed Tor implementations can reduce its provided anonymity level [26].

Hosts anonymity is not provided by ICING-based Nebula. By inspecting packet headers, eavesdroppers can easily determine a packet's source, as well as the path it traversed. However, host anonymity can be achieved by replacing ICING with TorIP [73], thus resulting in a level of anonymity similar to that provided by Tor in today's Internet.

Unlike IP, NDN has some features that facilitate anonymous communication. A PIT allows interest messages and content objects to only carry the requested content name without any consumer-related information. While this help to protect consumer's privacy form on-path observers inspecting packets deep in the core network, its efficacy will decrease as the observer moves closer to the consumer. At the network access (e.g., wireless access point or a broadband network gateway) an observer can easily correlate interests with the consumer issuing them. If the observer sits multiple hops far away from the consumer, he will only be able to correlate interests with a set of consumers (i.e., all the consumers reachable from the observer). Moreover, Compagno *et al.* show in [33] that off-path adversaries can abuse the NDN content caching and forwarding state to determine consumers' location. DiBenedetto *et al.* [46] proposed ANDāNA, a tool that provides a level of anonymity similar to Tor, while requiring only two intermediate nodes, instead of three.

MF and XIA suffer from the same privacy and anonymity problems as IP. Packets contain both source and destination GUIDs (or principal identifiers), thus revealing the hosts involved. To make the matter worse, XIA packets path can be revealed by inspecting their destination DAG addresses. This is because such addresses might include (part of) the path to the destination, as described in Section VI. Due to the similarity to IP with respect to the communication model adopted, both MF and XIA can use approaches developed to preserve users anonymity in IP networks. For instance, Tor can be used to protect MF and XIA host principals' anonymity [81].

VIII. Resolution Services Security

Resolution service is a fundamental part of the current Internet architecture. It maps human-readable names to

routable network addresses. As mentioned above, new Internet architectures also require similar resolution services to operate. In this section we compare the security and privacy features of the various resolution services proposed in FIA projects. We exclude Nebula and XIA since they do not propose a new resolution service and only exploit the existing DNS, and its security extensions.

We consider the following security features: trust, data origin authentication, data integrity, peer entity authentication, authorization and access control, accounting, data confidentiality and availability. We consciously exclude other privacy-related features, i.e., traffic flow confidentiality and anonymous communication, since we believe they should be provided at the network layer. We summarize our comparison on security and privacy features in resolution services in Table III.

A. Trust

DNSSEC introduces the notion of trust into DNS. It considers authoritative servers as trusted entities responsible of maintaining and securely providing the correct mapping between human-readable domain names and corresponding IP addresses. Recall that each DNSSEC server signs the resource records (name-to- IP mappings) of its respective domain. Trustworthiness of such servers is ensured by a chain-of-trust model that resembles the domains hierarchical organization. The top-level domain resides at the root of this chain. DNSCurve considers a different attacker model, where external attackers (malicious non-DNSCurve servers) try to modify queries and/or responses, or to mount DoS attacks. Mitigating such attacks requires additional trust in every DNSCurve-enabled server in the system.

Similar to DNSSEC, NDNS applies the same notion. Authoritative server are trusted entities and their trustworthiness is ensured by a similar chain-of-trust model.

GNRS, on the other hand, adopts a different approach. Every network is responsible of providing signed GUID-NA mappings. Thus, verifying these signatures ensures their validity. This also prevents (compromised) GNRS from manipulating GUID-NA mappings without being detected.

B. Data-Origin Authentication and Data Integrity

DNSSEC provides data-origin authentication and data integrity by requiring: (1) authoritative servers to sign each of their resource records, and (2) resolvers to verify the validity of these signatures and their corresponding public keys. This prevents adversaries from injecting bogus data into the DNS system. Signing every response resource record is an expensive operation that authoritative server should not perform at run-time. Adversaries can abuse such costly operation to launch (D)DoS attacks against authoritative servers. To this end, resource record signatures should always be generated in advance. While this method can be easily applied in case resolvers asks for existing domain names, it does not work for a non-existing domain names. DNS uses the NXDOMAIN resource records to inform a resolver that the queried name does not exist. However, providing data-origin authentication and integrity for NXDOMAIN resource records can not be done by generating the signature in advance, because of the number of possible non-existing names is infinite. To solve this problem, DNSSEC introduces a new record type called the NextSECure (NSEC) resource record. Specifically, assuming a canonical ordering of the domain names, a NSEC record contains two consecutive existing names in the canonical ordering, thus describing the gaps between them. Such records are signed and used as authenticated denial of existence for non-existing names. Since NSEC records contain existing names, their signatures can be calculated *a priori*. DNSCurve guarantees integrity of queries and responses exchanged with next-hop servers. However, it does not guarantee data-origin authentication.

In NDNS, query responses are carried in content object payloads, thus data-origin authentication and integrity is inherited from NDN. One difference between DNSSEC and NDNS resides in the granularity of these authentication and integrity guarantees. While DNSSEC can offer such security properties per individual resource record or resource record set, the fact that a NDNS record is carried in a content object can only guarantee the authentication and integrity of said record. Although this is a clear restriction in the flexibility and scalability of the protocol, it does not jeopardize its security [11]. In order to overcome DoS attacks due to requests for non-existing names, NDNS adopts methods similar to DNSSEC. In particular, NDNS servers can sign the gaps between existing names. Furthermore, Compagno *et al.* [35] proposed the use of a Bloom filter [27] to defeat the aforementioned DoS attacks. This, however, requires some changes in the NDN architecture as well as the introduction of a new content type.

GNRS also provides data-origin authentication and integrity by means of GUID-NA mapping signatures. The main different between GNRS and DNSSEC is that the former does not assume that GNRS servers are trusted [74], while the latter requires all authoritative servers to be trusted.

C. Peer Entity Authentication

In DNS, and DNSSEC, there is no entity authentication between a resolver and a DNS server. Resolvers usually know the IP address of a DNS server which is used to initiate queries. Usually, such IP address is manually configured on the resolver or obtained through DHCP and considered valid. Using a TLS connection between resolvers and DNS servers can provide entity authentication [28], [134]. Authentication among DNS servers is obtained through Transaction signatures (TSIG) [126]. TSIG involves pairwise keys shared among DNS servers and used to secure dynamic updates, zone transfers and recursive queries. Moreover, in case of dynamic updates generated from DNS clients, a signature is used to authenticate these clients, and validate the update. Similar to DNS and DNSSEC, DNSCurve does not provide entity authentication by default. However, this could be easily achieved by associating client public keys with certificates.

NDNS follows the same approach of DNS by not providing entity authentication. However, the fact that both NDNS users and servers are regular NDN consumers and producers, respectively, allows approaches similar to what is proposed in

TABLE III
RESOLUTION SERVICES SECURITY & PRIVACY FEATURES COMPARISON

		DNS/DNSSEC	NDNS	GNRS
Trust	*Trusted Model*	Hierarchical. Authoritative Servers are trusted for their respective domains. Delegation is supported	Hierarchical. Authoritative Servers are trusted for their respective domains. Delegation is supported	Distributed. Hosts, DHCP servers, Border routers must cooperate.
	Trusted entity	Authoritative Server.	Authoritative Server.	None.
	Trust Management	PKI + certificates, shared keys.	Certificates.	Distributed architecture + certificates.
Data Origin Auth.	*Coverage*	Authoritative Server-to-Client.	Authoritative Server-to-Client.	GNRS server-to-Client.
	Type of packet	Response.	Response.	Response.
	Mechanism	Signature.	Signature.	Self Certifying name + Signature.
Peer Auth.	*Authenticated Entity*	None.	None.	Client and GNRS server.
	Mechanism	None.	None.	Signature.
Integrity	*Coverage*	Authoritative Server-to-Client.	Authoritative Server-to-Client.	GNRS Server-to-Client.
	Type of packet	Response.	Response.	Request.
	Mechanism	Signature.	Signature.	Self Certifying name + Signature.
Authz. & Access Cont.	*Enforced on*	None.	None.	GNRS Server.
	Mechanism	None.	None.	ACL.
Accountability	*Entity accounted*	Host updating DNS.	None	Client and host updating GNRS.
	Mechanism	Signature.	None.	Signature.
Availability	*DNS Request Flooding Attacks*	IP "Anycast" distribute the number of request across many authoritative servers.	Secondary server to distribute the load + in network load-balancing techniques.	More resilient to flooding due to its distributed design.
Data Conf.	*Mechanism*	None.	None.	None.

Section VII-C to be adopted. Furthermore, securing dynamic updates requires NDNS clients to have previously shared their certificate with the NDNS servers. Signing the updates will then authenticate them.

Unlike DNS and NDNS, GNRS authenticates every client issuing requests (query, update and delete) by retrieving the corresponding certificate, from NCS, and verifying the GUID authenticity. GNRS clients can ensure server authentication by performing similar steps, requesting certificates from NCS and verifying servers identities.

D. Authorization and Access Control

Both DNS/DNSSEC and NDNS do not provide any form of access control. All resource records are publicly available to every host in the network. However, authorization and access

control is provided for dynamic updates. With DNSCurve, on the other hand, exchanged packets between are encrypted, and thus cannot be publicly accessed by other unauthorized entities in the network.

GNRS does not follow the same trend and consider access control a crucial part of its design. Specifically, GNRS stores a set of access control policies along with GUID-NA mappings. Such policies regulate access to particular GNRS resources by specifying read and write permissions which blacklist and whitelist certain user GUIDs.

E. Accountability

DNS/DNSSEC guarantees accountability only for secure DNS dynamic updates [128]. This is because such requests must in fact be signed by their originators. DNSCurve allows accountability for queries, since they contain client public keys.

NDNS does not provide any mechanism for accountability. The fact that NDNS users and servers are consumers and producers allows the adoption of similar approaches described in Section VII-F.

GNRS uses a different approach and mandates GNRS clients to sign every request. By doing so, accountability is provided for all insert, update and query requests.

F. Availability

As a public available service, DNS is subject to (D)DoS attacks which jeopardize its availability. In particular, adversaries can flood authoritative servers with a large number of query requests to exhaust their resources.[9] Although the use of DNS caching and redundancy servers reduce the effect of DDoS, a number of such attacks have been successfully directed against root and top-level DNS servers in past years [2], [4], [5]. Many solutions have been presented that either: (1) require some changes in the DNS protocol [41], [87], [98], [103], or (2) propose new resolution services [57], [130]. Nowadays, DNS uses a single approach that does not require any modification to its architecture, which is the adoption of "Anycast" [9]. In this case, a single DNS server is replicated in different geographically locations among several ASes. Therefore, routing protocols forward DNS requests to the nearest server that can satisfy them [36]. However, this approach can not achieve an efficient load balancing because it does not consider replica workloads and network traffic.

DNS can be exploited to launch (D)DoS amplification attacks against other hosts. These attackstake advantage of open DNS resolvers, and involve forwarding (D)DoS traffic through DNS servers to amplify the volume of data being sent to a target [38], [61]. This issue is exacerbated by DNSSEC [107], which may be exploited by attackers to cause an amplification effect of 50 or more times the original attack bandwidth [16], [123]. DNSCurve has approximately the same amplification effect as regular DNS.

DNS resolvers (not implementing DNSSEC protocol), and DNSCurve resolvers, can be subject to cache poisoning attacks. This attack is similar, in concept, to content poisoning described in Section VII-G. In DNS cache poisoning attacks, the goal of the adversary is to inject spoofed responses (name-IP mappings) in the resolver caches [115]. Injecting false DNS responses can be achieved using a man-in-the-middle attack in which the adversary satisfies requests with false DNS responses [115]. The introduction of data- origin authentication in DNSSEC allows resolvers to verify the origin of data in DNS response, thus eliminating this attack.

NDNS follows the same hierarchical design of DNS. In principles NDNS authoritative servers seems to offer the same level of resilience to DDoS as DNS authoritative servers. Furthermore, NDNS envisions the use of a set of secondary servers to balance the workload, which was proven to be not effective. The same anycast approach used in DNS could be employed in NDNS. Such forwarding strategy must be implemented by NDN routers. Moreover, NDNS is susceptible to content poisoning attacks. Fortunately, the same countermeasures described in Section VII-G can be effective.

GNRS design appears to be more resilient to DDoS than DNS design. In fact, GNRS does not adopt a hierarchical structure, instead it distributes the GUID-NA mappings among a number of replicas using Distributed Hashtables (DHT) maintained by all the ASes in the Internet. This allows GNRS to easily scale and distribute a DDoS attack.

G. Data Confidentiality

Neither DNS nor DNSSEC provide confidentiality. Queries and resource records are never encrypted by authoritative servers and are always exchanged in cleartext. One way of achieving confidentiality in DNS/DNSSEC is to establish a TLS session between resolvers and authoritative servers [28], [134]. DNSCurve, however, uses symmetric encryption to provide data confidentiality of DNS messages.

Similarly, both NDNS and GNRS designs do not take confidentiality into consideration. Fortunately, similar approaches to using TLS channels can be adopted. This feature is important to provide private communication.

IX. Lessons Learned and Future Directions

In this section, we summarize security and privacy features provided by the network layer in the four studied FIA architectures, as well as their respective resolution services. We also highlight missing features and outline directions for future work. Finally, we briefly discuss the current implementation status and performance characteristics of each architecture.

A. Network Layer

Overall, we believe that the NSF mandatory requirement of *Security and Privacy by Design* has only been partially accomplished. Table IV summarizes security and privacy features provided by each FIA architecture.

Nebula. At its current state, Nebula includes trust, data origin authentication, peer entity authentication, data integrity, authorization and access control, accountability, and availability features. All of them are provided between senders and routers implementing ICING, except for peer entity

[9]The use of secure dynamic updates involving asymmetric encryption increases the effect of the attack.

TABLE IV
NETWORK LAYER SECURITY AND PRIVACY COMPARISON

Security & Privacy Features	Network layers			
	Nebula	NDN	MF	XIA
Trust	✓	✓	✓	✓
Data Origin Authentication	◎	✓	✗	✗
Peer entity Authentication	◎	◎	◎	◎
Data Integrity	◎	✓	✗	✗
Authorization and Access Control	✓	◎	◎	◎
Accountability	✓	◎	◎	◎
Data Confidentiality	✗	✓	✗	✗
Traffic Flow Confidentiality	✗	✗	✗	✗
Anonymous Communication	✗	✗	✗	✗
Availability	◎	◎	◎	◎

✓indicates that a feature is present in the architecture.
✗indicates that a feature is unavailable.
◎ indicates that a feature is partially available or the architecture provides a means to facilitate its implementation by applications.

authentication that is guaranteed between senders and consent servers during PoCs retrieval. With respect to IP and IPsec, Nebula certainly increases the security of intra-domain communication. However, it lacks inter-domain and end-to-end communication security support, even if IPsec is used to establish a secure "pipe" between communicating end-hosts. We believe that integration of new or existing mechanisms to support both inter-domain and end-to-end security deserves further investigation.

Availability in Nebula is provided by path verification mechanism which prevents any adversary from sending unrequested data to perform bandwidth depletion attacks. However, ICING nodes and consent servers could be the target of a DoS attack, due to the heavy use of cryptographic operations. Consent servers can be another target for DoS attacks. In particular, an adversary can flood a server with an abnormal amount of request. If the target server controls a large number of routers, the attack can effectively disable all of them. We believe that such attack deserves further investigation.

By design, Nebula does not provide the three privacy features we considered in this paper: data confidentiality, traffic flow confidentiality, and anonymous communication. The first two can be easily implemented via IPsec-like techniques, while the latter can be achieved by adopting existing solutions, such as TorIP [73]. Nevertheless, we believe these aspects in Nebula deserve further investigation. Specifically, Nebula should make these features available by design before any adoption in the real world. Furthermore, anonymous communications conflicts with the current ICING design, which requires path establishment for each communication. This is an issue of concern that needs to be explored and, ideally, remedied.

From a feasibility perspective, ICING can be effectively deployed in Internet backbone routers [91], despite its heavy use of cryptographic operations at the data plane, and its large header size.[10] However, as reported in [91], this incurs a higher normalized cost of 93% over IP, i.e., hardware cost as equivalent gate count per unit of throughput.

NDN. NDN moves hosts and interfaces into the background and focuses on content as its primary network-layer entity. This strongly influences network-layer security and privacy features. Currently, NDN provides: trust, data origin authentication, data integrity and data confidentiality. Other features (e.g., peer entity authentication and accountability) are available only when derived from data origin authentication for both interest and content. Ubiquitous caching further complicates achieving these two features. In fact, when a consumer obtains a content from an intermediate router's cache, there is currently no way to provide peer entity authentication between the consumer and the router providing the content. Similarly, NDN does not provide accountability for content use: the distinction between a content served by its producer or from a router's cache.

Availability is an aspect for which NDN provides some improvements with respect to IP/IPsec. NDN reduces the amount of traffic both inside the network and at the producer by: (1) aggregating "matching" interests, and (2) serving matching requests from local caches, when possible. Unfortunately, at the same time NDN opens the door to new types of attacks. Indeed, while the pull model communication prevents any adversary from flooding a host with non-requested content, adversaries can exhaust router states (i.e., PIT and CS), decreasing the performance of a network. Even though several countermeasures have been proposed, none of them has been chosen and implemented as part of the architecture.

The remaining security features: authorization and access control, traffic flow confidentiality and anonymous communication are not provided by NDN at the network layer. Even if ACL can be implemented, NDN delegates access control to the application layer, which is responsible for encrypting content and distributing appropriate keys only to authorized consumers. Traffic flow confidentiality and anonymous communication are not natively available. The former can be provided by adopting existing approaches designed for IP and

[10]ICING's header occupies approximately 18.3% of a 1514 bytes packet, compared to 1.3% with IP, on a path through 7 realms [90].

IPsec, without any modification of the NDN architecture. The latter can be obtained by adopting solutions similar to the one presented in [46]. Note that NDN design improves privacy guarantees for consumers, since neither interest nor data packets carry consumers address. Nevertheless, consumer privacy can still be compromised by motivated attackers [33].

Heavy use of digital signatures in NDN introduces a major source of overhead in the network. This has been extensively evaluated in [114]. To the best of our knowledge, no assessment on the implications of router signature verification in NDN has been conducted. The only work considering the performance impact of signature generation by producers has been recently evaluated in [82]. From an interoperability perspective, NDN can be used alongside with IP as an overlay network.

MobilityFirst (MF). MF security features can be viewed as a hybrid of end-to-end and content-based approaches of IP/IPsec and NDN. In its current state, MF provides only few security features: trust as well as authorization and access control. The adoption of SCNs to address hosts facilitates the implementation of data origin authentication, peer entity authentication, and accountability. Peer entity authentication can be achieved involving a simple challenge-response protocol between the two peers. While this would guarantee end-to-end accountability, it is not enough for network accountability. In order to achieve that, edge routers should prevent address spoofing.

Data integrity and data confidentiality are currently not provided at network layer and it remains unclear whether MF will delegate these aspects to upper layers. Traffic flow confidentiality and anonymous communication are also not natively provided. Existing approaches for traffic flow confidentiality designed for IP/IPsec can be easily imported. However, TOR-like solutions for anonymous communication should be further studied and developed. A relatively recent initial effort in this direction is represented in [81]. Additionally, according to [70], the use of Disposable Identifiers [54], [71] is currently under examination to guarantee GUID-NA unlinkability. We believe that more research is needed to construct techniques that take advantage of MF's architectural idiosyncrasies.

MF is somewhat effective against content poisoning attacks due to its usage of SCNs. Meanwhile, it does not consider other network attacks, such as bandwidth depletion and cache pollution. Current IPsec-like methods can be applied to mitigate these issues. Further work is necessary to provide new architecture-specific countermeasures.

To the best of our knowledge there is no large scale performance assessment of MF. Venkataramani *et al.* [124] performed some benchmarks on a commodity hardware, and reached a maximum forwarding rate of 1 Gbps. We believe that further effort should be devoted to study and improve MF performance to meet real-world requirements.

XIA. XIA's main goal is to support communication between multiple and different principals. XIA security approach extends, de-facto, MF approach, where security focuses on two principals: content and hosts. XIA does not limit the number of principals but instead provides the freedom to design new ones. To this end, XIA security is based on each principal intrinsic security feature.

In its current state, XIA offers the same security and privacy features as MF. This is due to their similar approach in addressing principals using SCNs.

It is worth mentioning that XIA's performance is comparable with IP in core network routers [78].

TABLE V
RESOLUTION SERVICES SECURITY & PRIVACY COMPARISON

Security and Privacy Features	Resolution Services	
	NDNS	GNRS
Trust	✓	✓
Data Origin Authentication	✓	✓
Peer entity Authentication	✗	✓
Data Integrity	✓	✓
Authorization and Access Control	✗	✓
Accountability	◎	✓
Data Confidentiality	✗	✗
Availability	◎	✓

✓indicates that a feature is present in the architecture.
✗indicates that a feature is unavailable.
◎ indicates that a feature is partially available or the architecture provides a means to facilitate its implementation by applications.

B. Resolution Services

Nebula's resolution service is used by the NVENT to perform path discovery. In NDN, the same service provides the mapping between namespaces and the corresponding security information, i.e., a mapping between public key(s) and name prefixes. In MF, the resolution service is actively involved in any communication guaranteeing the binding between GUIDs and NAs. Finally, XIA's DNS-based resolution service is used for the address/path resolution.

Although all architectures requires a resolution service, only NDN and MF are actually investigating their own proposal. Table V summarizes security and privacy features of NDNS and GNRS.

NDNS. NDNS borrows many features from DNS and DNSSEC without introducing much novelty. NDNS involves the same notion of trust of DNSSEC in which servers are trusted entities. Moreover, NDNS queries and responses provide: data origin authentication, data integrity, accountability (only for dynamic updates), and availability. The last feature is provided by server replication. However, while DNSSEC uses the "Anycast" technique to choose the closes server to the resolver, in NDNS the network must be aware of all servers and implement specific forwarding strategies to balance the requests among them, which adds more complexities.

Peer entity authentication, authorization and access control, as well as data confidentiality, are not provided. We believe that NDNS should be enhanced to provide better availability and currently missing security features.

GNRS. GNRS is fundamentally different from DNS, DNSSEC, and DNSCurve. In fact, since MF adopts a flat name structure, it also forces GNRS to use the same structure for its servers. This different organization has some side effects

on the provided security features: servers are not assumed to be trusted. Also, data origin authentication is provided by the owner of the GUID and not by servers. GNRS also introduces authorization and access control of its stored information and provides accountability and peer entity authentication for each query and response. Finally, GNRS is more robust to DoS attacks than DNS. First, compromising one server does not affect others. Second, GNRS is designed to easily adapt to network changes and to balance GUID-NA mapping among many replicas based on locality of requests.

We believe that GNRS introduces good security improvements with respect to DNS, DNSSEC, and DNSCurve. The only missing feature is data confidentiality.

X. Conclusion

Despite the unmitigated success of the current IP-based Internet architecture, lack of security considerations in its design led to numerous security and privacy problems, over the years. Thus, a key goal of any future Internet architectures must be to include – from the outset – a set of comprehensive and extensible security and privacy features.

This paper analyzed (mostly network-layer) security and privacy features of four prominent NSF-funded FIA architectures – Nebula, NDN, MobilityFirst and XIA – with IP/IPsec used as a point of reference in the analysis. Prior surveys on future Internet architectures provide a limited, or even no, comparison on security and privacy features. This paper provides a comprehensive and neutral analysis of salient security and privacy features (and issues) in these NSF-funded Future Internet Architectures.

As evident from this work, while each FIA architecture offers some innovative and effective security and privacy features, none provides a comprehensive set thereof.

References

[1] *IPv6 Extension Headers Review and Considerations*. Accessed: May 26, 2016. [Online]. Available: https://www.cisco.com/en/US/technologies/tk648/tk872/technologies_white_paper0900aecd8054d37d.html

[2] *Nameserver DoS Attack October 2002*. Accessed: Jan. 31, 2018. [Online]. Available: https://www.caida.org/projects/dns/dns-rootgtld/oct02dos.xml

[3] *Signed Interest*. Accessed: Jan. 31, 2018. [Online]. Available: http://named-data.net/doc/ndn-cxx/current/tutorials/signed-interest.html

[4] (2002). *Ultradns DoS Attack*. [Online]. Available: http://www.theregister.co.uk/2002/12/14/

[5] (2007). *ICANN Factsheet for the February 6, 2007 Root Server Attack*. [Online]. Available: http://www.icann.org/announcements/factsheet-dns-attack-08mar07.pdf

[6] M. Aamir and S. M. A. Zaidi, "Denial-of-service in content centric (named data) networking: A tutorial and state-of-the-art survey," *Security Commun. Netw.*, vol. 8, no. 11, pp. 2037–2059, 2015.

[7] E. G. AbdAllah, H. S. Hassanein, and M. Zulkernine, "A survey of security attacks in information-centric networking," *IEEE Commun. Surveys Tuts.*, vol. 17, no. 3, pp. 1441–1454, 3rd Quart., 2015.

[8] J. Abley and T. Manderson, "Nameservers for IPv4 and IPv6 reverse zones," IETF, Fremont, CA, USA, Rep. RFC 5855, 2010.

[9] J. Abley and K. E. Lindqvist, "Operation of anycast services," IETF, Fremont, CA, USA, RFC 4786, 2006.

[10] A. Afanasyev, P. Mahadevan, I. Moiseenko, E. Uzun, and L. Zhang, "Interest flooding attack and countermeasures in named data networking," in *Proc. IFIP Netw. Conf.*, Brooklyn, NY, USA, 2013, pp. 1–9.

[11] A. Afanasyev, "Addressing operational challenges in named data networking through NDNS distributed database," Ph.D. dissertation, Dept. Comput. Sci., Univ. California at Los Angeles, Los Angeles, CA, USA, 2013.

[12] A. Afanasyev, J. Shi, L. Wang, B. Zhang, and L. Zhang, "Packet fragmentation in NDN: Why NDN uses hop-by-hop fragmentation," Named Data Netw., Rep. NDN-0032, 2015.

[13] A. Afanasyev, C. Yi, L. Wang, B. Zhang, and L. Zhang, "Map-and-ENCAP for scaling NDN routing," Univ. California at Los Angeles, Los Angeles, CA, USA, Rep. NDN-0004, 2015.

[14] B. Ahlgren, C. Dannewitz, C. Imbrenda, D. Kutscher, and B. Ohlman, "A survey of information-centric networking," *IEEE Commun. Mag.*, vol. 50, no. 7, pp. 26–36, Jul. 2012.

[15] S. Akhshabi and C. Dovrolis, "The evolution of layered protocol stacks leads to an hourglass-shaped architecture," *SIGCOMM Comput. Commun. Rev.*, vol. 41, no. 4, pp. 206–217, 2011.

[16] M. Anagnostopoulos, G. Kambourakis, P. Kopanos, G. Louloudakis, and S. Gritzalis, "DNS amplification attack revisited," *Comput. Security*, vol. 39, pp. 475–485, Nov. 2013.

[17] D. Comer, "A future Internet architecture that supports cloud computing," in *Proc. 6th Int. Conf. Future Internet Technol. (CFI)*, Seoul, South Korea, 2011, pp. 79–83. [Online]. Available: http://doi.acm.org/10.1145/2002396.2002418, doi: 10.1145/2002396.2002418.

[18] T. Anderson *et al.*, "A brief overview of the NEBULA future Internet architecture," *SIGCOMM Comput. Commun. Rev.*, vol. 44, no. 3, pp. 81–86, 2014.

[19] T. Anderson *et al.*, "The NEBULA future Internet architecture," in *The Future Internet* (LNCS 7858), A. Galis and A. Gavras, Eds. Heidelberg, Germany: Springer, 2013, pp. 16–26.

[20] T. Anderson, T. Roscoe, and D. Wetherall, "Preventing Internet denial-of-service with capabilities," *ACM SIGCOMM Comput. Commun. Rev.*, vol. 34, no. 1, pp. 39–44, 2004.

[21] R. Arends, R. Austein, M. Larson, D. Massey, and S. Rose, "DNS security introduction and requirements," IETF, Fremont, CA, USA, RFC 4033, 2005.

[22] M. Arye *et al.*, "Increasing network resilience through edge diversity in NEBULA," *SIGMOBILE Mobile Comput. Commun. Rev.*, vol. 16, no. 3, pp. 14–20, 2012.

[23] M. F. Bari, S. R. Chowdhury, R. Ahmed, R. Boutaba, and B. Mathieu, "A survey of naming and routing in information-centric networks," *IEEE Commun. Mag.*, vol. 50, no. 12, pp. 44–53, Dec. 2012.

[24] S. M. Bellovin, "Problem areas for the IP security protocols," in *Proc. 6th Conf. USENIX Security Symp. Focusing Appl. Cryptograph.*, San Jose, CA, USA, 1996, pp. 21–32.

[25] S. M. Bellovin, M. Leech, and T. Taylor, "ICMP traceback messages," Internet Draft, 2003.

[26] A. Biryukov, I. Pustogarov, and R.-P. Weinmann, "Trawling for Tor hidden services: Detection, measurement, deanonymization," in *Proc. 34th IEEE Symp. Security Privacy*, Berkeley, CA, USA, 2013, pp. 80–94.

[27] B. H. Bloom, "Space/time trade-offs in hash coding with allowable errors," *Commun. ACM*, vol. 13, no. 7, pp. 422–426, 1970.

[28] S. Bortzmeyer, "DNS privacy considerations," IETF, Fremont, CA, USA, RFC 7626, 2015.

[29] J. Bound and Y. Rekhter, "Dynamic updates in the domain name system (DNS update)," IETF, Fremont, CA, USA, RFC 2136, 1997.

[30] R. Braden, "Requirements for Internet hosts-communication layers," IETF, Fremont, CA, USA, RFC 1122, 1989. [Online]. Available: https://tools.ietf.org/html/rfc1122

[31] H. C. Cankaya, "Access control lists," in *Encyclopedia of Cryptography and Security*, H. C. van Tilborg and S. Jajodia, Eds. Boston, MA, USA: Springer, 2011, pp. 9–12.

[32] C. Kaufman, P. Hoffman, Y. Nir, P. Eronen, and T. Kivinen, "Internet key exchange protocol version 2 (IKEv2)," IETF, Fremont, CA, USA, RFC 5996, 2010.

[33] A. Compagno, M. Conti, P. Gasti, L. V. Mancini, and G. Tsudik, "Violating consumer anonymity: Geo-locating nodes in named data networking," in *Proc. 13th Int. Conf. Appl. Cryptograph. Netw. Security (ACNS)*, 2015, pp. 243–262.

[34] A. Compagno, M. Conti, P. Gasti, and G. Tsudik, "Poseidon: Mitigating interest flooding DDoS attacks in named data networking," in *Proc. 38th Annu. IEEE Conf. Local Comput. Netw. (LCN)*, Sydney, NSW, Australia, 2013, pp. 630–638.

[35] A. Compagno, M. Conti, C. Ghali, and G. Tsudik, "To NACK or not to NACK? Negative acknowledgments in information-centric networking," in *Proc. 24th Int. Conf. Comput. Commun. Netw. (ICCCN)*, Las Vegas, NV, USA, 2015, pp. 1–10.

[36] D. Conrad. (2012). *Towards Improving DNS Security, Stability, and Resiliency*. [Online]. Available: http://www.internetsociety.org/sites/default/files/bp-dnsresiliency-201201-en_0.pdf

[37] M. Conti, P. Gasti, and M. Teoli, "A lightweight mechanism for detection of cache pollution attacks in named data networking," *Comput. Netw.*, vol. 57, no. 16, pp. 3178–3191, 2013.

[38] D. Dagon, N. Provos, C. P. Lee, and W. Lee, "Corrupted DNS resolution paths: The rise of a malicious resolution authority," in *Proc. 15th Netw. Distrib. Syst. Security Symp. (NDSS)*, San Diego, CA, USA, 2008.

[39] C. Dannewitz *et al.*, "Network of information (NetInf)—An information-centric networking architecture," *Comput. Commun.*, vol. 36, no. 7, pp. 721–735, 2013.

[40] D. Dean and A. Stubblefield, "Using client puzzles to protect TLS," in *Proc. 10th USENIX Security Symp.*, vol. 42. Washington, DC, USA, 2001.

[41] T. Deegan, J. Crowcroft, and A. Warfield, "The main name system: An exercise in centralized computing," *ACM SIGCOMM Comput. Commun. Rev.*, vol. 35, no. 5, pp. 5–13, 2005.

[42] S. E. Deering and R. Hinden, "Internet protocol, version 6 (IPv6) specification," IETF, Fremont, CA, USA, RFC 2460, 1998.

[43] J. P. Degabriele and K. G. Paterson, "Attacking the IPsec standards in encryption-only configurations," in *Proc. 28th IEEE Symp. Security Privacy (S&P)*, Berkeley, CA, USA, 2007, pp. 335–349.

[44] M. Dempsky, "DNSCurve: Link-level security for the domain name system," IETF, Fremont, CA, USA, Internet-Draft, 2010.

[45] L. Deng, Y. Gao, Y. Chen, and A. Kuzmanovic, "Pollution attacks and defenses for Internet caching systems," *Comput. Netw.*, vol. 52, no. 5, pp. 935–956, 2008.

[46] S. DiBenedetto, P. Gasti, G. Tsudik, and E. Uzun, "ANDaNA: Anonymous named data networking application," in *Proc. 18th Annu. Netw. Distrib. Syst. Security Symp. (NDSS)*, 2011.

[47] N. Fotiou, P. Nikander, D. Trossen, and G. C. Polyzos, "Developing information networking further: From PSIRP to pursuit," in *Proc. Int. Conf. Broadband Commun. Netw. Syst.*, 2010, pp. 1–13.

[48] J. Gersch and D. Massey, "ROVER: Route origin verification using DNS," in *Proc. 22nd Int. Conf. Comput. Commun. Netw. (ICCCN)*, Nassau, Bahamas, 2013, pp. 1–9.

[49] C. Ghali, A. Narayanan, D. Oran, G. Tsudik, and C. A. Wood, "Secure fragmentation for content-centric networks," in *Proc. IEEE 14th Int. Symp. Netw. Comput. Appl. (NCA)*, Cambridge, MA, USA, 2015, pp. 47–56.

[50] C. Ghali, M. A. Schlosberg, G. Tsudik, and C. A. Wood, "Interest-based access control for content centric networks," in *Proc. 2nd Int. Conf. Inf. Centric Netw. (ICN)*, San Francisco, CA, USA, 2015, pp. 147–156.

[51] C. Ghali, G. Tsudik, and E. Uzun, "Network-layer trust in named-data networking," *ACM SIGCOMM Comput. Commun. Rev.*, vol. 44, no. 5, pp. 12–19, 2014.

[52] C. Ghali, G. Tsudik, E. Uzun, and C. A. Wood, "Closing the floodgate with stateless content-centric networking," in *Proc. 26th Int. Conf. Comput. Commun. Netw. (ICCCN)*, Vancouver, BC, Canada, Jul. 2017, pp. 1–10, doi: 10.1109/ICCCN.2017.8038367.

[53] D. Griffin *et al.*, "Service-centric networking," *Handbook of Research on Redesigning the Future of Internet Architectures*. Hershey, PA, USA: IGI Glob., 2015.

[54] M. Gruteser and D. Grunwald, "Enhancing location privacy in wireless LAN through disposable interface identifiers: A quantitative analysis," *Mobile Netw. Appl.*, vol. 10, no. 3, pp. 315–325, 2005.

[55] T. Hain, "Architectural implications of NAT," IETF, Fremont, CA, USA, RFC 2993, 2000.

[56] D. Han *et al.*, "XIA: Efficient support for evolvable internetworking," in *Proc. 9th USENIX Conf. Netw. Syst. Design Implement. (NSDI)*, San Jose, CA, USA, 2012, pp. 309–322.

[57] M. Handley and A. Greenhalgh, "The case for pushing DNS," in *Proc. Hotnets-IV*, 2005.

[58] H.-C. Hsiao *et al.*, "STRIDE: Sanctuary trail–refuge from Internet DDoS entrapment," in *Proc. 8th ACM SIGSAC Symp. Inf. Comput. Commun. Security (CCS)*, 2013, pp. 415–426.

[59] J. Ioannidis and S. M. Bellovin, "Implementing pushback: Router-based defense against DDoS attacks," in *Proc. 9th Netw. Distrib. Syst. Security Symp. (NDSS)*, San Diego, CA, USA, 2002.

[60] A. Juels and J. G. Brainard, "Client puzzles: A cryptographic countermeasure against connection depletion attacks," in *Proc. Netw. Distrib. Syst. Security Symp. (NDSS)*, vol. 99. San Diego, CA, USA, 1999, pp. 151–165.

[61] G. Kambourakis, T. Moschos, D. Geneiatakis, and S. Gritzalis, "Detecting DNS amplification attacks," in *Proc. Int. Workshop Crit. Inf. Infrastruct. Security*, 2007, pp. 185–196.

[62] S. Kent, "IP authentication header," IETF, Fremont, CA, USA, RFC 4302, 2005.

[63] S. Kent, "IP encapsulating security payload (ESP)," IETF, Fremont, CA, USA, RFC 4303, 2005.

[64] R. Khondoker, B. Nugraha, R. Marx, and K. M. Bayarou, "Security of selected future Internet architectures: A survey," in *Proc. 8th Int. Conf. Innov. Mobile Internet Services Ubiquitous Comput. (IMIS)*, Birmingham, U.K., 2014, pp. 433–440.

[65] T. Kivinen, B. Swander, A. Huttunen, and V. Volpe, "Negotiation of NAT-traversal in the IKE," IETF, Fremont, CA, USA, RFC 3947, 2005.

[66] T. Koponen *et al.*, "A data-oriented (and beyond) network architecture," *ACM SIGCOMM Comput. Commun. Rev.*, vol. 37, no. 4, pp. 181–192, 2007.

[67] H. Krawczyk, "The order of encryption and authentication for protecting communications (or: How secure is SSL?)," in *Advances in Cryptology—CRYPTO 2001* (LNCS 2139), J. Kilian, Ed. Heidelberg, Germany: Springer, 2001, pp. 310–331.

[68] H. Krawczyk, R. Canetti, and M. Bellare, "HMAC: Keyed-hashing for message authentication," IETF, Fremont, CA, USA, RFC 2104, 1997.

[69] E. Lewis and A. Hoenes, "DNS zone transfer protocol (AXFR)," Internet Eng. Task Force, Fremont, CA, USA, Rep. 5936, 2010.

[70] J. Lindqvist and M. Gruteser. *Privacy in MobilityFirst Architecture*. Accessed: Jan. 31, 2018. [Online]. Available: http://mobilityfirst.winlab.rutgers.edu/documents/documents/Lindqvist.pdf

[71] J. Lindqvist and J.-M. Tapio, "Protecting privacy with protocol stack virtualization," in *Proc. 7th ACM Workshop Privacy Electron. Soc. (WPES)*, Alexandria, VA, USA, 2008, pp. 65–74.

[72] V. Liu, D. Halperin, A. Krishnamurthy, and T. Anderson, "F10: A fault-tolerant engineered network," in *Proc. 10th USENIX Conf. Netw. Syst. Design Implement. (NSDI)*, Lombard, IL, USA, 2013, pp. 399–412.

[73] V. Liu, S. Han, A. Krishnamurthy, and T. Anderson, "Tor instead of IP," in *Proc. 10th ACM Workshop Hot Topics Netw. (HotNets)*, Cambridge, MA, USA, 2011, pp. 1–6.

[74] X. Liu, W. Trappe, and Y. Zhang, "Secure name resolution for identifier-to-locator mappings in the global Internet," in *Proc. 22nd Int. Conf. Comput. Commun. Netw. (ICCCN)*, 2013, pp. 1–7.

[75] B. T. Loo *et al.*, "Declarative networking: Language, execution and optimization," in *Proc. ACM SIGMOD Int. Conf. Manag. Data (SIGMOD)*, Chicago, IL, USA, 2006, pp. 97–108.

[76] B. T. Loo *et al.*, "Declarative networking," *Commun. ACM*, vol. 52, no. 11, pp. 87–95, 2009.

[77] R. Lutz, "Security and privacy in future Internet architectures—Benefits and challenges of content centric networks," *CoRR*, vol. abs/1601.01278, Jan. 2016.

[78] M. Machado, C. Doucette, and J. W. Byers, "Linux XIA: An interoperable meta network architecture to crowdsource the future Internet," in *Proc. 11th ACM/IEEE Symp. Architect. Netw. Commun. Syst. (ANCS)*, 2015, pp. 147–158.

[79] P. Mahadevan, E. Uzun, S. Sevilla, and J. J. Garcia-Luna-Aceves, "CCN-KRS: A key resolution service for CCN," in *Proc. 1st Int. Conf. Inf. Centric Netw. (ICN)*, Paris, France, 2014, pp. 97–106.

[80] R. Mahajan *et al.*, "Controlling high bandwidth aggregates in the network," *ACM SIGCOMM Comput. Commun. Rev.*, vol. 32, no. 3, pp. 62–73, 2002.

[81] K. Manandhar, B. Adcock, and X. Cao, "Preserving the anonymity in mobilityfirst networks," in *Proc. 23rd Int. Conf. Comput. Commun. Netw. (ICCCN)*, Shanghai, China, 2014, pp. 1–6.

[82] X. Marchal, T. Cholez, and O. Festor, "Server-side performance evaluation of NDN," in *Proc. 3rd ACM Conf. Inf. Centric Netw. (ICN)*, Kyoto, Japan, 2016, pp. 148–153.

[83] A. Mason, *CCSP Self-Study: Cisco Secure Virtual Private Networks (CSVPN)*. Indianapolis, IN, USA: Cisco Press, 2004.

[84] D. Maughan and M. Schneider, "Internet security association and key management protocol (ISAKMP)," IETF, Fremont, CA, USA, RFC 2408, 1998.

[85] J. McCann, J. Mogul, and S. E. Deering, "Path MTU discovery for IP version 6," IETF, Fremont, CA, USA, RFC 1981, 1996.

[86] R. D. McKinney, W. A. Montgomery, H. Ouibrahim, P. Sijben, and J. J. Stanaway, "Service-centric networks," *Bell Labs Tech. J.*, vol. 3, no. 3, pp. 98–115, Jul./Sep. 1998.

[87] E. Meshkova, J. Riihijärvi, M. Petrova, and P. Mähönen, "A survey on resource discovery mechanisms, peer-to-peer and service discovery frameworks," *Comput. Netw.*, vol. 52, no. 11, pp. 2097–2128, 2008.

[88] P. Mockapetris, "Domain names—Implementation and specification," IETF, Fremont, CA, USA, RFC 1035, 1987.
[89] S. Mukherjee, A. Baid, I. Seskar, and D. Raychaudhuri, "Network-assisted multihoming for emerging heterogeneous wireless access scenarios," in *Proc. IEEE 25th Annu. Int. Symp. Pers. Indoor Mobile Radio Commun. (PIMRC)*, Washington, DC, USA, Sep. 2014, pp. 1520–1524, doi: 10.1109/PIMRC.2014.7136409.
[90] J. Naous, M. Walfish, D. Mazieres, A. Nicolosi, and A. Seehra, "Network security via explicit consent," Nebula Project, Rep. TR-09, 2012.
[91] J. Naous *et al.*, "Verifying and enforcing network paths with icing," in *Proc. 7th Conf. Emerg. Netw. Experiments Technol. (CoNEXT)*, Tokyo, Japan, 2011, pp. 1–12.
[92] D. Naylor *et al.*, "XIA: Architecting a more trustworthy and evolvable Internet," *ACM SIGCOMM Comput. Commun. Rev.*, vol. 44, no. 3, pp. 50–57, 2014.
[93] S. C. Nelson, G. Bhanage, and D. Raychaudhuri, "GSTAR: Generalized storage-aware routing for mobilityfirst in the future mobile Internet," in *Proc. MobiArch*, Bethesda, MD, USA, 2011, pp. 19–24.
[94] E. Nordström *et al.*, "Serval: An end-host stack for service-centric networking," in *Proc. 9th USENIX Conf. Netw. Syst. Design Implement. (NSDI)*, San Jose, CA, USA, 2012, pp. 85–98.
[95] NSF. (2014). *NSF Future Internet Architecture Project*. [Online]. Available: http://www.nets-fia.net/
[96] B. Nugraha, R. Khondoker, R. Marx, and K. Bayarou, "A mutual key agreement protocol to mitigate replaying attack in expressive Internet architecture (XIA)," in *Proc. ITU Kaleidoscope Acad. Conf. Living Converged World Impossible Without Standards*, St. Petersburg, Russia, 2014, pp. 233–240.
[97] J. Pan, S. Paul, and R. Jain, "A survey of the research on future Internet architectures," *IEEE Commun. Mag.*, vol. 49, no. 7, pp. 26–36, Jul. 2011.
[98] V. Pappas, D. Massey, and L. Zhang, "Enhancing DNS resilience against denial of service attacks," in *Proc. 37th Annu. IEEE/IFIP Int. Conf. Depend. Syst. Netw. (DNS)*, Edinburgh, U.K., 2007, pp. 450–459.
[99] S. Peter *et al.*, "One tunnel is (often) enough," in *Proc. ACM SIGCOMM Conf. (SIGCOMM)*, Chicago, IL, USA, 2014, pp. 99–110.
[100] J. Postel, "User datagram protocol," IETF, Fermont, CA, USA, RFC 768, 1980.
[101] J. Postel, "Internet control message protocol," IETF, Fermont, CA, USA, RFC 792, 1981.
[102] J. Postel *et al.*, "Internet protocol," IETF, Fermont, CA, USA, RFC 791, 1981.
[103] V. Ramasubramanian and E. G. Sirer, "The design and implementation of a next generation name service for the Internet," *ACM SIGCOMM Comput. Commun. Rev.*, vol. 34, no. 4, pp. 331–342, 2004.
[104] J.-F. Raymond, "Traffic analysis: Protocols, attacks, design issues, and open problems," in *Designing Privacy Enhancing Technologies*. Heidelberg, Germany: Springer, 2001, pp. 10–29.
[105] M. K. Reiter and A. D. Rubin, "Crowds: Anonymity for Web transactions," *ACM Trans. Inf. Syst. Security*, vol. 1, no. 1, pp. 66–92, 1998.
[106] M. C. Richardson, "A method for storing IPsec keying material in DNS," IETF, Fermont, CA, USA, RFC 4025, 2005.
[107] C. Rossow, "Amplification hell: Revisiting network protocols for DDoS abuse," in *Proc. 21st Netw. Distrib. Syst. Security Symp. (NDSS)*, 2014.
[108] S. Savage, D. Wetherall, A. Karlin, and T. Anderson, "Network support for IP traceback," *IEEE/ACM Trans. Netw.*, vol. 9, no. 3, pp. 226–237, Jun. 2001.
[109] K. Seo and S. Kent, "Security architecture for the Internet protocol," IETF, Fremont, CA, USA, RFC 4301, 2005.
[110] I. Seskar, K. Nagaraja, S. Nelson, and D. Raychaudhuri, "MobilityFirst future Internet architecture project," in *Proc. 7th Asian Internet Eng. Conf. (AINTEC)*, Bangkok, Thailand, 2011, pp. 1–3.
[111] A. Sharma *et al.*, "A global name service for a highly mobile internetwork," *SIGCOMM Comput. Commun. Rev.*, vol. 44, no. 4, pp. 247–258, 2014.
[112] R. W. Shirey, "Internet security glossary, version 2," IETF, Fremont, CA, USA, RFC 4301, 2007.
[113] A. C. Snoeren *et al.*, "Hash-based IP traceback," *ACM SIGCOMM Comput. Commun. Rev.*, vol. 31, no. 4, 2001, pp. 3–14.
[114] W. So, A. Narayanan, and D. Oran, "Named data networking on a router: Fast and dos-resistant forwarding with hash tables," in *Proc. 9th ACM/IEEE Symp. Architect. Netw. Commun. Syst.*, San Jose, CA, USA, 2013, pp. 215–226.
[115] S. Son and V. Shmatikov, "The Hitchhiker's guide to DNS cache poisoning," in *Proc. Int. Conf. Security Privacy Commun. Syst.*, Singapore, 2010, pp. 466–483.
[116] S. M. Specht and R. B. Lee, "Distributed denial of service: Taxonomies of attacks, tools, and countermeasures," in *Proc. 17th Int. Conf. Parallel Distrib. Comput. Syst. Int. Workshop Security Parallel Distrib. Syst.*, San Francisco, CA, USA, 2004, pp. 543–550.
[117] P. Srisuresh and K. B. Egevang, "Traditional IP network commaddress translator (traditional NAT)," IETF, Fremont, CA, USA, RFC 3022, 2001.
[118] I. Stoica, S. Shenker, and H. Zhang, "Core-stateless fair queueing: A scalable architecture to approximate fair bandwidth allocations in high-speed networks," *IEEE/ACM Trans. Netw.*, vol. 11, no. 1, pp. 33–46, Feb. 2003.
[119] P. F. Syverson, R. Dingledine, and N. Mathewson, "Tor: The second-generation onion router," in *Proc. 13th USENIX Security Symp.*, San Diego, CA, USA, 2004, pp. 303–320.
[120] R. Tourani, S. Misra, T. Mick, and G. Panwar, "Security, privacy, and access control in information-centric networking: A survey," *IEEE Commun. Surveys Tuts.*, to be published, doi: 10.1109/COMST.2017.2749508.
[121] G. Tyson, N. Sastry, R. Cuevas, I. Rimac, and A. Mauthe, "A survey of mobility in information-centric networks," *Commun. ACM*, vol. 56, no. 12, pp. 90–98, 2013.
[122] G. Tyson, N. Sastry, I. Rimac, R. Cuevas, and A. Mauthe, "A survey of mobility in information-centric networks: Challenges and research directions," in *Proc. 1st ACM Workshop Emerg. Name Oriented Mobile Netw. Design Architect. Algorithms Appl.*, 2012, pp. 1–6.
[123] R. van Rijswijk-Deij, A. Sperotto, and A. Pras, "DNSSEC and its potential for DDoS attacks: A comprehensive measurement study," in *Proc. Conf. Internet Measur. Conf.*, Vancouver, BC, Canada, 2014, pp. 449–460.
[124] A. Venkataramani *et al.*, "Mobilityfirst: A mobility-centric and trustworthy Internet architecture," *ACM SIGCOMM Comput. Commun. Rev.*, vol. 44, no. 3, pp. 74–80, 2014.
[125] A. Venkataramani *et al.*, "Design requirements of a global name service for a mobility-centric, trustworthy internetwork," in *Proc. 5th IEEE Int. Conf. Commun. Syst. Netw. (COMSNETS)*, 2013, pp. 1–9.
[126] P. Vixie, O. Gudmundsson, D. Eastlake, III, and B. Wellington, "Secret key transaction authentication for DNS (TSIG)," IETF, Fremont, CA, USA, Rep. 2845, 2000.
[127] T. Vu *et al.*, "DMAP: A shared hosting scheme for dynamic identifier to locator mappings in the global Internet," in *Proc. IEEE 32nd Int. Conf. Distrib. Comput. Syst. (ICDCS)*, 2012, pp. 698–707.
[128] B. Wellington, "Secure domain name system (DNS) dynamic update," IETF, Fremont, CA, USA, RFC 2137, 2000.
[129] A. Yaar, A. Perrig, and D. Song, "SIFF: A stateless Internet flow filter to mitigate DDoS flooding attacks," in *Proc. IEEE Symp. Security Privacy (S&P)*, Berkeley, CA, USA, 2004, pp. 130–143.
[130] H. Yang, H. Luo, Y. Yang, S. Lu, and L. Zhang, "HOURS: Achieving DoS resilience in an open service hierarchy," in *Proc. 35th Annu. IEEE/IFIP Int. Conf. Depend. Syst. Netw. (DNS)*, 2004, pp. 83–92.
[131] H. Zhang, W. Su, and W. Quan, *Smart Collaborative Identifier Network: A Promising Design for Future Internet*. Heidelberg, Germany: Springer, 2016.
[132] L. Zhang *et al.*, "Named data networking (NDN) project," Named Data Netw., Rep. NDN-0001, 2010. [Online]. Available: http://named-data.net/techreport/TR001ndn-proj.pdf
[133] X. Zhang *et al.*, "SCION: Scalability, control, and isolation on next-generation networks," in *Proc. IEEE Symp. Security Privacy (SP)*, Berkeley, CA, USA, 2011, pp. 212–227.
[134] L. Zhu *et al.*, "T-DNS: Connection-oriented DNS to improve privacy and security," *ACM SIGCOMM Comput. Commun. Rev.*, vol. 44, no. 4, pp. 379–380, 2014.

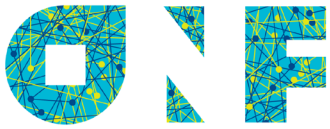
Moreno Ambrosin received the Ph.D. degree from the University of Padua, Italy, in 2017. He is a Research Scientist with Intel Labs. His research interests are security and privacy in the Internet of Things, and in novel networking architectures, in particular 5G and next generation cellular networks, information-centric networking, and software-defined networking.

Alberto Compagno received the Ph.D. degree from Sapienza University of Rome, Italy, in 2017. He is a Researcher Scientist with Cisco Systems. His main research interests are network security and user privacy in novel network architectures, in the Internet of Things, and in distributed computing such as fog and cloud computing. He is currently involved in the open source Cicn/FD.io project.

Mauro Conti received the Ph.D. degree from Sapienza University of Rome, Italy, in 2009. He was a Post-Doctoral Researcher with Vrije Universiteit Amsterdam, The Netherlands. He is an Associate Professor with the University of Padua, Italy. His main research interest is in the area of security and privacy. He has published over 170 papers in the topmost international peer-reviewed journals and conferences in the above areas. He was awarded with a Marie Curie Fellowship in 2012 by the European Commission, and with a Fellowship by the German DAAD in 2013. He is an Associate Editor for several journals, including the IEEE COMMUNICATIONS SURVEYS AND TUTORIALS and the IEEE TRANSACTIONS ON INFORMATION FORENSICS AND SECURITY. He was the Program Chair for TRUST 2015, ICISS 2016, WiSec 2017, and the General Chair for SecureComm 2012 and ACM SACMAT 2013.

Cesar Ghali received the master's degree in electrical and computer engineering from the American University of Beirut (AUB) in 2010, and the master's and Ph.D. degrees in networked systems from the Donald Bren School of Information and Computer Science, University of California, Irvine, in 2016. He was a Research Assistant and a Web Developer from 2008 to 2012 with AUB. He is currently a Software Engineer with Google working with the infrastructure security team on the Application Layer Transport Security project. His research interests include future Internet architecture security, information security, network security, Web services and cloud computing security, routing in the cloud, and service reputations.

Gene Tsudik received the Ph.D. degree in computer science from the University of Southern California (USC) in 1991. He is a Chancellor's Professor of computer science with the University of California at Irvine (UCI). He was with IBM Zurich Research Laboratory from 1991 to 1996, followed by USC/ISI from 1996 to 2000, and has been with UCI since 2000. He authored the first crypto-poem published at a refereed venue. His research interests include numerous topics in security, privacy, and applied cryptography. He was a recipient of the 2017 ACM SIGSAC Outstanding Contribution Award. From 2009 to 2016, he served as the Editor-in-Chief of *ACM Transactions on Information and Systems Security*. He was a Fulbright Scholar, Fulbright Specialist, and a fellow of ACM and AAAS, as well as a Foreign Member of Academia Europaea.



OPEN NETWORKING
FOUNDATION

OpenFlow Switch Specification

Version 1.3.2 (Wire Protocol 0x04)
April 25, 2013

ONF TS-009

ONF Document Type: OpenFlow Spec
ONF Document Name: openflow-spec-v1.3.2

Disclaimer

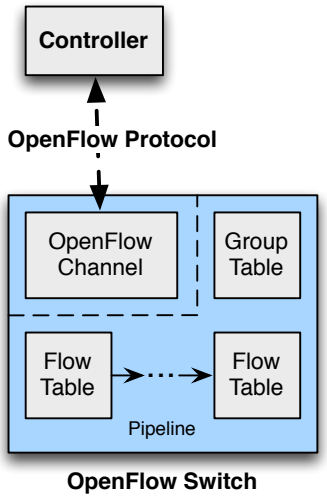
Contents

List of Tables

List of Figures

1 Introduction

This document describes the requirements of an OpenFlow Switch. We recommend that you read the latest version of the OpenFlow whitepaper before reading this specification. The whitepaper is available on the Open Networking Foundation website (`https://www.opennetworking.org/standards/intro-to-openflow`). This specification covers the components and the basic functions of the switch, and the OpenFlow protocol to manage an OpenFlow switch from a remote controller.



Figure˜1: Main components of an OpenFlow switch.

2 Switch Components

An OpenFlow Switch consists of one or more *flow tables* and a *group table*, which perform packet lookups and forwarding, and an *OpenFlow channel* to an external controller (Figure 1). The switch communicates with the controller and the controller manages the switch via the OpenFlow protocol.

Using the OpenFlow protocol, the controller can add, update, and delete *flow entries* in flow tables, both reactively (in response to packets) and proactively. Each flow table in the switch contains a set of flow entries; each flow entry consists of *match fields*, *counters*, and a set of *instructions* to apply to matching packets (see 5.2).

Matching starts at the first flow table and may continue to additional flow tables (see 5.1). Flow entries match packets in priority order, with the first matching entry in each table being used (see 5.3). If a matching entry is found, the instructions associated with the specific flow entry are executed. If no match is found in a flow table, the outcome depends on configuration of the table-miss flow entry: for example, the packet may be forwarded to the controller over the OpenFlow channel, dropped, or may continue to the next flow table (see 5.4).

Instructions associated with each flow entry either contain actions or modify pipeline processing (see 5.9). Actions included in instructions describe packet forwarding, packet modification and group table

processing. Pipeline processing instructions allow packets to be sent to subsequent tables for further processing and allow information, in the form of metadata, to be communicated between tables. Table pipeline processing stops when the instruction set associated with a matching flow entry does not specify a next table; at this point the packet is usually modified and forwarded (see 5.10).

Flow entries may forward to a *port*. This is usually a physical port, but it may also be a logical port defined by the switch or a reserved port defined by this specification (see 4.1). Reserved ports may specify generic forwarding actions such as sending to the controller, flooding, or forwarding using non-OpenFlow methods, such as "normal" switch processing (see 4.5), while switch-defined logical ports may specify link aggregation groups, tunnels or loopback interfaces (see 4.4).

Actions associated with flow entries may also direct packets to a group, which specifies additional processing (see 5.6). Groups represent sets of actions for flooding, as well as more complex forwarding semantics (e.g. multipath, fast reroute, and link aggregation). As a general layer of indirection, groups also enable multiple flow entries to forward to a single identifier (e.g. IP forwarding to a common next hop). This abstraction allows common output actions across flow entries to be changed efficiently.

The group table contains group entries; each group entry contains a list of *action buckets* with specific semantics dependent on group type (see 5.6.1). The actions in one or more action buckets are applied to packets sent to the group.

Switch designers are free to implement the internals in any way convenient, provided that correct match and instruction semantics are preserved. For example, while a flow entry may use an all group to forward to multiple ports, a switch designer may choose to implement this as a single bitmask within the hardware forwarding table. Another example is matching; the pipeline exposed by an OpenFlow switch may be physically implemented with a different number of hardware tables.

3 Glossary

This section describes key OpenFlow specification terms:

- **Byte**: an 8-bit octet.
- **Packet**: an Ethernet frame, including header and payload.
- **Port**: where packets enter and exit the OpenFlow pipeline (see 4.1). May be a physical port, a logical port defined by the switch, or a reserved port defined by the OpenFlow protocol.
- **Pipeline**: the set of linked flow tables that provide matching, forwarding, and packet modification in an OpenFlow switch.
- **Flow Table**: a stage of the pipeline. It contains flow entries.
- **Flow Entry**: an element in a flow table used to match and process packets. It contains a set of match fields for matching packets, a priority for matching precedence, a set of counters to track packets, and a set of instructions to apply.
- **Match Field**: a field against which a packet is matched, including packet headers, the ingress port, and the metadata value. A match field may be wildcarded (match any value) and in some cases bitmasked.

- **Metadata**: a maskable register value that is used to carry information from one table to the next.
- **Instruction**: instructions are attached to a flow entry and describe the OpenFlow processing that happens when a packet matches the flow entry. An instruction either modifies pipeline processing, such as directing the packet to another flow table, or contains a *set* of actions to add to the action set, or contains a *list* of actions to apply immediately to the packet.
- **Action**: an operation that forwards the packet to a port or modifies the packet, such as decrementing the TTL field. Actions may be specified as part of the instruction set associated with a flow entry or in an action bucket associated with a group entry. Actions may be accumulated in the Action Set of the packet or applied immediately to the packet.
- **Action Set**: a set of actions associated with the packet that are accumulated while the packet is processed by each table and that are executed when the instruction set instructs the packet to exit the processing pipeline.
- **Group**: a list of action buckets and some means of choosing one or more of those buckets to apply on a per-packet basis.
- **Action Bucket**: a set of actions and associated parameters, defined for groups.
- **Tag**: a header that can be inserted or removed from a packet via push and pop actions.
- **Outermost Tag**: the tag that appears closest to the beginning of a packet.
- **Controller**: an entity interacting with the OpenFlow switch using the OpenFlow protocol.
- **Meter**: a switch element that can measure and control the rate of packets. The meter triggers a meter band if the packet rate or byte rate passing through the meter exceeds a predefined threshold. If the meter band drops the packet, it is called a **Rate Limiter**.

4 OpenFlow Ports

This section describes the OpenFlow port abstraction and the various types of OpenFlow ports supported by OpenFlow.

4.1 OpenFlow Ports

OpenFlow ports are the network interfaces for passing packets between OpenFlow processing and the rest of the network. OpenFlow switches connect logically to each other via their OpenFlow ports.

An OpenFlow switch makes a number of OpenFlow ports available for OpenFlow processing. The set of OpenFlow ports may not be identical to the set of network interfaces provided by the switch hardware, some network interfaces may be disabled for OpenFlow, and the OpenFlow switch may define additional OpenFlow ports.

OpenFlow packets are received on an **ingress port** and processed by the OpenFlow pipeline (see 5.1) which may forward them to an **output port**. The packet ingress port is a property of the packet throughout the OpenFlow pipeline and represents the OpenFlow port on which the packet was received

into the OpenFlow switch. The ingress port can be used when matching packets (see 5.3). The OpenFlow pipeline can decide to send the packet on an output port using the output action (see 5.12), which defines how the packet goes back to the network.

An OpenFlow switch must support three types of OpenFlow ports: *physical ports*, *logical ports* and *reserved ports*.

4.2 Standard Ports

The OpenFlow **standard ports** are defined as physical ports, logical ports, and the `LOCAL` reserved port if supported (excluding other reserved ports).

Standard ports can be used as ingress and output ports, they can be used in groups (see 5.6), and they have port counters (see 5.8).

4.3 Physical Ports

The OpenFlow **physical ports** are switch defined ports that correspond to a hardware interface of the switch. For example, on an Ethernet switch, physical ports map one-to-one to the Ethernet interfaces.

In some deployments, the OpenFlow switch may be virtualised over the switch hardware. In those cases, an OpenFlow physical port may represent a virtual slice of the corresponding hardware interface of the switch.

4.4 Logical Ports

The OpenFlow **logical ports** are switch defined ports that don't correspond directly to a hardware interface of the switch. Logical ports are higher level abstractions that may be defined in the switch using non-OpenFlow methods (e.g. link aggregation groups, tunnels, loopback interfaces).

Logical ports may include packet encapsulation and may map to various physical ports. The processing done by the logical port must be transparent to OpenFlow processing and those ports must interact with OpenFlow processing like OpenFlow physical ports.

The only differences between *physical ports* and *logical ports* is that a packet associated with a logical port may have an extra metadata field called *Tunnel-ID* associated with it (see 7.2.3.7) and when a packet received on a logical port is sent to the controller, both its logical port and its underlying physical port are reported to the controller (see 7.4.1).

4.5 Reserved Ports

The OpenFlow **reserved ports** are defined by this specification. They specify generic forwarding actions such as sending to the controller, flooding, or forwarding using non-OpenFlow methods, such as "normal" switch processing.

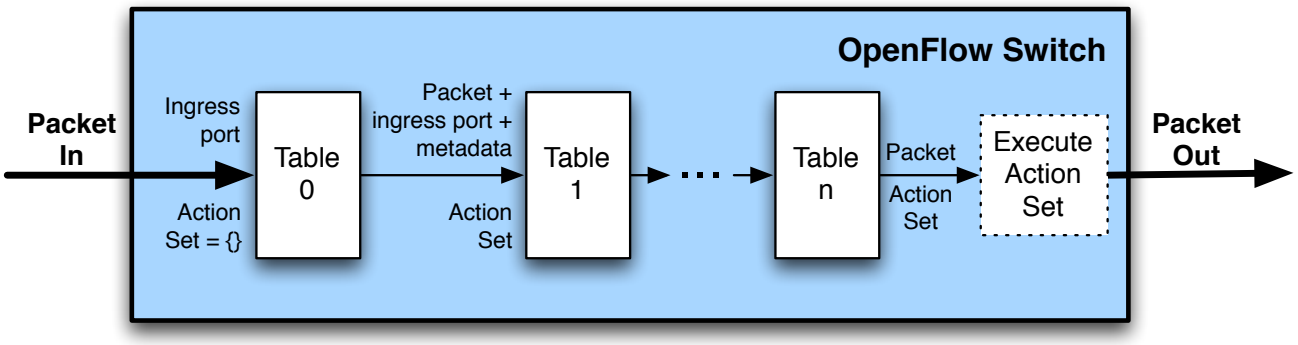
A switch is not required to support all reserved ports, just those marked *"Required"* below.

- *Required:* **ALL:** Represents all ports the switch can use for forwarding a specific packet. Can be used only as an output port. In that case a copy of the packet is sent to all standard ports, excluding the packet ingress port and ports that are configured `OFPPC_NO_FWD`.
- *Required:* **CONTROLLER:** Represents the control channel with the OpenFlow controller. Can be used as an ingress port or as an output port. When used as an output port, encapsulate the packet in a *packet-in* message and send it using the OpenFlow protocol (see 7.4.1). When used as an ingress port, this identifies a packet originating from the controller.
- *Required:* **TABLE:** Represents the start of the OpenFlow pipeline (see 5.1). This port is only valid in an output action in the action list of a *packet-out* message (see 7.3.7), and submits the packet to the first flow table so that the packet can be processed through the regular OpenFlow pipeline.
- *Required:* **IN_PORT:** Represents the packet ingress port. Can be used only as an output port, send the packet out through its ingress port.
- *Required:* **ANY:** Special value used in some OpenFlow commands when no port is specified (i.e. port is wildcarded). Can neither be used as an ingress port nor as an output port.
- *Optional:* **LOCAL:** Represents the switch's local networking stack and its management stack. Can be used as an ingress port or as an output port. The local port enables remote entities to interact with the switch and its network services via the OpenFlow network, rather than via a separate control network. With a suitable set of default flow entries it can be used to implement an in-band controller connection.
- *Optional:* **NORMAL:** Represents the traditional non-OpenFlow pipeline of the switch (see 5.1). Can be used only as an output port and processes the packet using the normal pipeline. If the switch cannot forward packets from the OpenFlow pipeline to the normal pipeline, it must indicate that it does not support this action.
- *Optional:* **FLOOD:** Represents flooding using the normal pipeline of the switch (see 5.1). Can be used only as an output port, in general will send the packet out all standard ports, but not to the ingress port, nor ports that are in `OFPPS_BLOCKED` state. The switch may also use the packet VLAN ID to select which ports to flood.

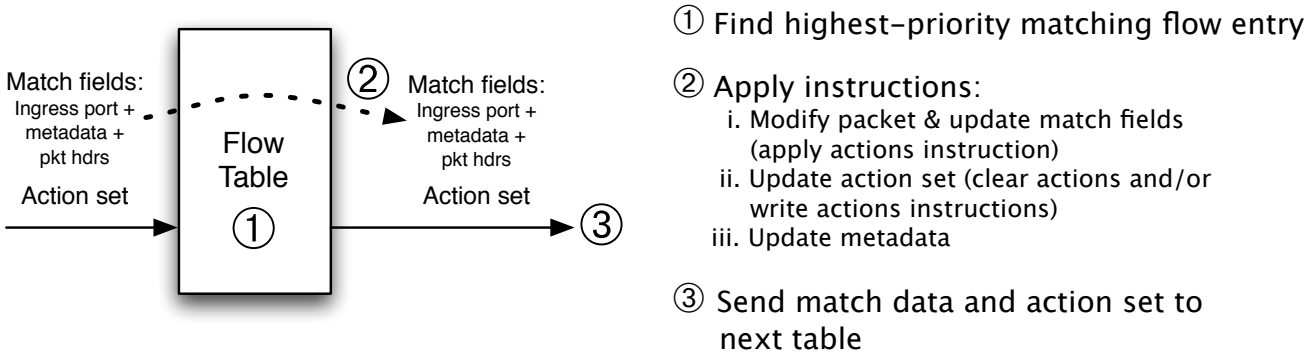
OpenFlow-only switches do not support the **NORMAL** port and **FLOOD** port, while *OpenFlow-hybrid* switches may support them (see 5.1). Forwarding packets to the **FLOOD** port depends on the switch implementation and configuration, while forwarding using a **group** of type *all* enables the controller to more flexibly implement flooding (see 5.6.1).

5 OpenFlow Tables

This section describes the components of flow tables and group tables, along with the mechanics of matching and action handling.



(a) Packets are matched against multiple tables in the pipeline



(b) Per-table packet processing

Figure˜2: Packet flow through the processing pipeline.

5.1 Pipeline Processing

OpenFlow-compliant switches come in two types: *OpenFlow-only*, and *OpenFlow-hybrid*. **OpenFlow-only** switches support only OpenFlow operation, in those switches all packets are processed by the OpenFlow pipeline, and can not be processed otherwise.

OpenFlow-hybrid switches support both OpenFlow operation and *normal* Ethernet switching operation, i.e. traditional L2 Ethernet switching, VLAN isolation, L3 routing (IPv4 routing, IPv6 routing...), ACL and QoS processing. Those switches must provide a classification mechanism outside of OpenFlow that routes traffic to *either* the OpenFlow pipeline *or* the normal pipeline. For example, a switch may use the VLAN tag or input port of the packet to decide whether to process the packet using one pipeline or the other, or it may direct all packets to the OpenFlow pipeline. This classification mechanism is outside the scope of this specification. An OpenFlow-hybrid switch may also allow a packet to go from the OpenFlow pipeline to the normal pipeline through the *NORMAL* and *FLOOD* reserved ports (see 4.5).

The **OpenFlow pipeline** of every OpenFlow switch contains multiple flow tables, each flow table containing multiple flow entries. The OpenFlow pipeline processing defines how packets interact with those flow tables (see Figure 2). An OpenFlow switch is required to have at least one flow table, and can optionally have more flow tables. An OpenFlow switch with only a single flow table is valid, in this case pipeline processing is greatly simplified.

The flow tables of an OpenFlow switch are sequentially numbered, starting at 0. Pipeline processing always starts at the first flow table: the packet is first matched against flow entries of flow table 0. Other flow tables may be used depending on the outcome of the match in the first table.

When processed by a flow table, the packet is matched against the flow entries of the flow table to select a flow entry (see 5.3). If a flow entry is found, the instruction set included in that flow entry is executed. These instructions may explicitly direct the packet to another flow table (using the Goto Instruction, see 5.9), where the same process is repeated again. A flow entry can only direct a packet to a flow table number which is greater than its own flow table number, in other words pipeline processing can only go forward and not backward. Obviously, the flow entries of the last table of the pipeline can not include the Goto instruction. If the matching flow entry does not direct packets to another flow table, pipeline processing stops at this table. When pipeline processing stops, the packet is processed with its associated action set and usually forwarded (see 5.10).

If a packet does not match a flow entry in a flow table, this is a table miss. The behavior on a table miss depends on the table configuration (see 5.4). A table-miss flow entry in the flow table can specify how to process unmatched packets: options include dropping them, passing them to another table or sending them to the controller over the control channel via packet-in messages (see 6.1.2).

The OpenFlow pipeline and various OpenFlow operations process packets of a specific type in conformance with the specifications defined for that packet type, unless the present specification or the OpenFlow configuration specify otherwise. For example, the Ethernet header definition used by OpenFlow must conform to IEEE specifications, and the TCP/IP header definition used by OpenFlow must conform to RFC specifications. Additionally, packet reordering in an OpenFlow switch must conform to the requirements of IEEE specifications, provided that the packets are processed by the same flow entries, group bucket and meter band.

5.2 Flow Table

A flow table consists of flow entries.

Match Fields	Priority	Counters	Instructions	Timeouts	Cookie

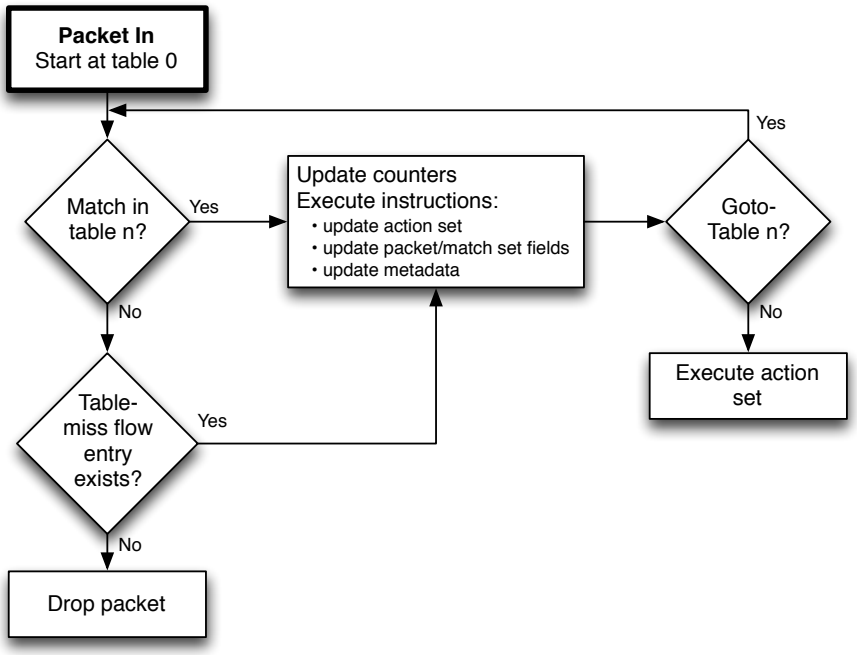
Table˜1: Main components of a flow entry in a flow table.

Each flow table entry (see Table 1) contains:

- **match fields**: to match against packets. These consist of the ingress port and packet headers, and optionally metadata specified by a previous table.
- **priority**: matching precedence of the flow entry.
- **counters**: updated when packets are matched.
- **instructions**: to modify the action set or pipeline processing.
- **timeouts**: maximum amount of time or idle time before flow is expired by the switch.
- **cookie**: opaque data value chosen by the controller. May be used by the controller to filter flow statistics, flow modification and flow deletion. Not used when processing packets.

A flow table entry is identified by its match fields and priority: the match fields and priority taken together identify a unique flow entry in the flow table. The flow entry that wildcards all fields (all fields omitted) and has priority equal to 0 is called the table-miss flow entry (see 5.4).

5.3 Matching



Figure˜3: Flowchart detailing packet flow through an OpenFlow switch.

On receipt of a packet, an OpenFlow Switch performs the functions shown in Figure 3. The switch starts by performing a table lookup in the first flow table, and based on pipeline processing, may perform table lookups in other flow tables (see 5.1).

Packet match fields are extracted from the packet. Packet match fields used for table lookups depend on the packet type, and typically include various packet header fields, such as Ethernet source address or IPv4 destination address (see 7.2.3). In addition to packet headers, matches can also be performed against the ingress port and metadata fields. Metadata may be used to pass information between tables in a switch. The packet match fields represent the packet in its current state, if actions applied in a previous table using the *Apply-Actions* changed the packet headers, those changes are reflected in the packet match fields.

A packet matches a flow table entry if the values in the packet match fields used for the lookup match those defined in the flow table entry. If a flow table entry field has a value of ANY (field omitted), it matches all possible values in the header. If the switch supports arbitrary bitmasks on specific match fields, these masks can more precisely specify matches.

The packet is matched against the table and *only* the highest priority flow entry that matches the packet must be selected. The counters associated with the selected flow entry must be updated and the instruction set included in the selected flow entry must be applied. If there are multiple matching flow

entries with the same highest priority, the selected flow entry is explicitly undefined. This case can only arise when a controller writer never sets the `OFPFF_CHECK_OVERLAP` bit on flow mod messages and adds overlapping entries.

IP fragments must be reassembled before pipeline processing if the switch configuration contains the `OFPC_FRAG_REASM` flag (see 7.3.2).

This version of the specification does *not* define the expected behavior when a switch receives a malformed or corrupted packet.

5.4 Table-miss

Every flow table must support a table-miss flow entry to process table misses. The table-miss flow entry specifies how to process packets unmatched by other flow entries in the flow table (see 5.1), and may, for example, send packets to the controller, drop packets or direct packets to a subsequent table.

The table-miss flow entry is identified by its match and its priority (see 5.2), it wildcards all match fields (all fields omitted) and has the lowest priority (0). The match of the table-miss flow entry may fall outside the normal range of matches supported by a flow table, for example an exact match table would not support wildcards for other flow entries but must support the table-miss flow entry wildcarding all fields. The table-miss flow entry may not have the same capability as regular flow entry (see 7.3.5.5). The table-miss flow entry must support at least sending packets to the controller using the CONTROLLER reserved port (see 4.5) and dropping packets using the Clear-Actions instruction (see 5.9). Implementations are encouraged to support directing packets to a subsequent table when possible for compatibility with earlier versions of this specification.

The table-miss flow entry behaves in most ways like any other flow entry: it does not exist by default in a flow table, the controller may add it or remove it at any time (see 6.4), and it may expire (see 5.5). The table-miss flow entry matches packets in the table as expected from its set of match fields and priority (see 5.3): it matches packets unmatched by other flow entries in the flow table. The table-miss flow entry instructions are applied to packets matching the table-miss flow entry (see 5.9). If the table-miss flow entry directly sends packets to the controller using the CONTROLLER reserved port (see 4.5), the packet-in reason must identify a table-miss (see 7.4.1).

If the table-miss flow entry does not exist, by default packets unmatched by flow entries are dropped (discarded). A switch configuration, for example using the OpenFlow Configuration Protocol, may override this default and specify another behaviour.

5.5 Flow Removal

Flow entries are removed from flow tables in two ways, either at the request of the controller or via the switch flow expiry mechanism.

The switch **flow expiry** mechanism is run by the switch independently of the controller and is based on the state and configuration of flow entries. Each flow entry has an `idle_timeout` and a `hard_timeout` associated with it. If the `hard_timeout` field is non-zero, the switch must note the flow entry's arrival time, as it may need to evict the entry later. A non-zero `hard_timeout` field causes the flow entry to be removed after the given number of seconds, regardless of how many packets it has matched. If the

`idle_timeout` field is non-zero, the switch must note the arrival time of the last packet associated with the flow, as it may need to evict the entry later. A non-zero `idle_timeout` field causes the flow entry to be removed when it has matched no packets in the given number of seconds. The switch must implement flow expiry and remove flow entries from the flow table when one of their timeouts is exceeded.

The controller may actively remove flow entries from flow tables by sending **delete** flow table modification messages (`OFPFC_DELETE` or `OFPFC_DELETE_STRICT` - see 6.4).

When a flow entry is removed, either by the controller or the flow expiry mechanism, the switch must check the flow entry's `OFPFF_SEND_FLOW_REM` flag. If this flag is set, the switch must send a flow removed message to the controller. Each flow removed message contains a complete description of the flow entry, the reason for removal (expiry or delete), the flow entry duration at the time of removal, and the flow statistics at the time of removal.

5.6 Group Table

A group table consists of group entries. The ability for a flow entry to point to a *group* enables OpenFlow to represent additional methods of forwarding (e.g. select and all).

Group Identifier	Group Type	Counters	Action Buckets

Table˜2: Main components of a group entry in the group table.

Each group entry (see Table 2) is identified by its group identifier and contains:

- **group identifier**: a 32 bit unsigned integer uniquely identifying the group
- **group type**: to determine group semantics (see Section 5.6.1)
- **counters**: updated when packets are processed by a group
- **action buckets**: an ordered list of action buckets, where each action bucket contains a set of actions to execute and associated parameters

5.6.1 Group Types

A switch is not required to support all group types, just those marked *"Required"* below. The controller can also query the switch about which of the *"Optional"* group types it supports.

- *Required:* **all**: Execute all buckets in the group. This group is used for multicast or broadcast forwarding. The packet is effectively cloned for each bucket; one packet is processed for each bucket of the group. If a bucket directs a packet explicitly out the ingress port, this packet clone is dropped. If the controller writer wants to forward out the ingress port, the group must include an extra bucket which includes an output action to the `OFPP_IN_PORT` reserved port.

- *Optional:* **select**: Execute one bucket in the group. Packets are processed by a single bucket in the group, based on a switch-computed selection algorithm (e.g. hash on some user-configured tuple or simple round robin). All configuration and state for the selection algorithm is external to OpenFlow. The selection algorithm should implement equal load sharing and can optionally be based on bucket weights. When a port specified in a bucket in a select group goes down, the switch may restrict bucket selection to the remaining set (those with forwarding actions to live ports) instead of dropping packets destined to that port. This behavior may reduce the disruption of a downed link or switch.
- *Required:* **indirect**: Execute the one defined bucket in this group. This group supports only a single bucket. Allows multiple flow entries or groups to point to a common group identifier, supporting faster, more efficient convergence (e.g. next hops for IP forwarding). This group type is effectively identical to an all group with one bucket.
- *Optional:* **fast failover**: Execute the first live bucket. Each action bucket is associated with a specific port and/or group that controls its liveness. The buckets are evaluated in the order defined by the group, and the first bucket which is associated with a live port/group is selected. This group type enables the switch to change forwarding without requiring a round trip to the controller. If no buckets are live, packets are dropped. This group type must implement a *liveness mechanism* (see 6.5).

5.7 Meter Table

A meter table consists of meter entries, defining per-flow meters. Per-flow meters enable OpenFlow to implement various simple QoS operations, such as rate-limiting, and can be combined with per-port queues (see 5.12) to implement complex QoS frameworks, such as DiffServ.

A meter measures the rate of packets assigned to it and enables controlling the rate of those packets. Meters are attached directly to flow entries (as opposed to queues which are attached to ports). Any flow entry can specify a meter in its instruction set (see 5.9): the meter measures and controls the rate of the aggregate of all flow entries to which it is attached. Multiple meters can be used in the same table, but in an exclusive way (disjoint set of flow entries). Multiple meters can be used on the same set of packets by using them in successive flow tables.

Meter Identifier	Meter Bands	Counters

Table˜3: Main components of a meter entry in the meter table.

Each meter entry (see Table 3) is identified by its meter identifier and contains:

- **meter identifier**: a 32 bit unsigned integer uniquely identifying the meter
- **meter bands**: an unordered list of meter bands, where each meter band specifies the rate of the band and the way to process the packet
- **counters**: updated when packets are processed by a meter

5.7.1 Meter Bands

Each meter may have one or more meter bands. Each band specifies the rate at which the band applies and the way packets should be processed. Packets are processed by a single meter band based on the current measured meter rate. The meter applies the meter band with the highest configured rate that is lower than the current measured rate. If the current rate is lower than any specified meter band rate, no meter band is applied.

Band Type	Rate	Counters	Type specific arguments

Table˜4: Main components of a meter band in a meter entry.

Each meter band (see Table 4) is identified by its rate and contains:

- **band type**: defines how packet are processed
- **rate**: used by the meter to select the meter band, defines the lowest rate at which the band can apply
- **counters**: updated when packets are processed by a meter band
- **type specific arguments**: some band types have optional arguments

There is no band type *"Required"* by this specification. The controller can query the switch about which of the *"Optional"* meter band types it supports.

- *Optional:* **drop**: drop (discard) the packet. Can be used to define a rate limiter band.
- *Optional:* **dscp remark**: increase the drop precedence of the DSCP field in the IP header of the packet. Can be used to define a simple DiffServ policer.

5.8 Counters

Counters are maintained for each flow table, flow entry, port, queue, group, group bucket, meter and meter band. OpenFlow-compliant counters may be implemented in software and maintained by polling hardware counters with more limited ranges. Table 5 contains the set of counters defined by the OpenFlow specification. A switch is not required to support all counters, just those marked *"Required"* in Table 5.

Duration refers to the amount of time the flow entry, a port, a group, a queue or a meter has been installed in the switch, and must be tracked with second precision. The Receive Errors field is the total of all receive and collision errors defined in Table 5, as well as any others not called out in the table.

Counters are unsigned and wrap around with no overflow indicator. If a specific numeric counter is not available in the switch, its value must be set to the maximum field value (the unsigned equivalent of -1).

Counter	**Bits**	
Per Flow Table		
Reference Count (active entries)	32	*Required*
Packet Lookups	64	*Optional*
Packet Matches	64	*Optional*
Per Flow Entry		
Received Packets	64	*Optional*
Received Bytes	64	*Optional*
Duration (seconds)	32	*Required*
Duration (nanoseconds)	32	*Optional*
Per Port		
Received Packets	64	*Required*
Transmitted Packets	64	*Required*
Received Bytes	64	*Optional*
Transmitted Bytes	64	*Optional*
Receive Drops	64	*Optional*
Transmit Drops	64	*Optional*
Receive Errors	64	*Optional*
Transmit Errors	64	*Optional*
Receive Frame Alignment Errors	64	*Optional*
Receive Overrun Errors	64	*Optional*
Receive CRC Errors	64	*Optional*
Collisions	64	*Optional*
Duration (seconds)	32	*Required*
Duration (nanoseconds)	32	*Optional*
Per Queue		
Transmit Packets	64	*Required*
Transmit Bytes	64	*Optional*
Transmit Overrun Errors	64	*Optional*
Duration (seconds)	32	*Required*
Duration (nanoseconds)	32	*Optional*
Per Group		
Reference Count (flow entries)	32	*Optional*
Packet Count	64	*Optional*
Byte Count	64	*Optional*
Duration (seconds)	32	*Required*
Duration (nanoseconds)	32	*Optional*
Per Group Bucket		
Packet Count	64	*Optional*
Byte Count	64	*Optional*
Per Meter		
Flow Count	32	*Optional*
Input Packet Count	64	*Optional*
Input Byte Count	64	*Optional*
Duration (seconds)	32	*Required*
Duration (nanoseconds)	32	*Optional*
Per Meter Band		
In Band Packet Count	64	*Optional*
In Band Byte Count	64	*Optional*

Table˜5: List of counters.

5.9 Instructions

Each flow entry contains a set of instructions that are executed when a packet matches the entry. These instructions result in changes to the packet, action set and/or pipeline processing.

A switch is not required to support all instruction types, just those marked *"Required Instruction"* below. The controller can also query the switch about which of the *"Optional Instruction"* types it supports.

- *Optional Instruction:* **Meter *meter_id***: Direct packet to the specified meter. As the result of the metering, the packet may be dropped (depending on meter configuration and state).
- *Optional Instruction:* **Apply-Actions *action(s)***: Applies the specific action(s) immediately, without any change to the Action Set. This instruction may be used to modify the packet between two tables or to execute multiple actions of the same type. The actions are specified as an action list (see 5.11).
- *Optional Instruction:* **Clear-Actions**: Clears all the actions in the action set immediately.
- *Required Instruction:* **Write-Actions *action(s)***: Merges the specified action(s) into the current action set (see 5.10). If an action of the given type exists in the current set, overwrite it, otherwise add it.
- *Optional Instruction:* **Write-Metadata *metadata / mask***: Writes the masked metadata value into the metadata field. The mask specifies which bits of the metadata register should be modified (i.e. new_metadata = old_metadata & ~mask | value & mask).
- *Required Instruction:* **Goto-Table *next-table-id***: Indicates the next table in the processing pipeline. The table-id must be greater than the current table-id. The flow entries of the last table of the pipeline can not include this instruction (see 5.1). OpenFlow switches with only a single flow table are not required to implement this instruction.

The instruction set associated with a flow entry contains a maximum of one instruction of each type. The instructions of the set execute in the order specified by this above list. In practice, the only constraints are that the *Meter* instruction is executed before the *Apply-Actions* instruction, that the *Clear-Actions* instruction is executed before the *Write-Actions* instruction, and that *Goto-Table* is executed last.

A switch must reject a flow entry if it is unable to execute the instructions associated with the flow entry. In this case, the switch must return an unsupported flow error (see 6.4). Flow tables may not support every match, every instruction or every action.

5.10 Action Set

An action set is associated with each packet. This set is empty by default. A flow entry can modify the action set using a *Write-Action* instruction or a *Clear-Action* instruction associated with a particular match. The action set is carried between flow tables. When the instruction set of a flow entry does not contain a *Goto-Table* instruction, pipeline processing stops and the actions in the action set of the packet are executed.

An action set contains a maximum of one action of each type. The *set-field* actions are identified by their field types, therefore the action set contains a maximum of one *set-field* action for each field type (i.e. multiple fields can be set). When multiple actions of the same type are required, e.g. pushing multiple MPLS labels or popping multiple MPLS labels, the *Apply-Actions* instruction should be used (see 5.11).

The actions in an action set are applied in the order specified below, regardless of the order that they were added to the set. If an action set contains a group action, the actions in the appropriate action bucket of the group are also applied in the order specified below. The switch may support arbitrary action execution order through the action list of the *Apply-Actions* instruction.

1. **copy TTL inwards**: apply copy TTL inward actions to the packet
2. **pop**: apply all tag pop actions to the packet
3. **push-MPLS**: apply MPLS tag push action to the packet
4. **push-PBB**: apply PBB tag push action to the packet
5. **push-VLAN**: apply VLAN tag push action to the packet
6. **copy TTL outwards**: apply copy TTL outwards action to the packet
7. **decrement TTL**: apply decrement TTL action to the packet
8. **set**: apply all set-field actions to the packet
9. **qos**: apply all QoS actions, such as set_queue to the packet
10. **group**: if a group action is specified, apply the actions of the relevant group bucket(s) in the order specified by this list
11. **output**: if no group action is specified, forward the packet on the port specified by the output action

The output action in the action set is executed last. If both an output action and a group action are specified in an action set, the output action is ignored and the group action takes precedence. If no output action and no group action were specified in an action set, the packet is dropped. The execution of groups is recursive if the switch supports it; a group bucket may specify another group, in which case the execution of actions traverses all the groups specified by the group configuration.

5.11 Action List

The *Apply-Actions* instruction and the *Packet-out* message include an action list. The semantics of the action list is identical to the OpenFlow 1.0 specification. The actions of an action list are executed in the order specified by the list, and are applied immediately to the packet.

The execution of an action list starts with the first action in the list and each action is executed on the packet in sequence. The effect of those actions is cumulative, if the action list contains two Push VLAN actions, two VLAN headers are added to the packet. If the action list contains an output action, a copy of the packet is forwarded in its current state to the desired port. If the list contains group actions, a copy of the packet in its current state is processed by the relevant group buckets.

After the execution of the action list in an *Apply-Actions* instruction, pipeline execution continues on the modified packet (see 5.1). The action set of the packet is unchanged by the execution of the action list.

5.12 Actions

A switch is not required to support all action types, just those marked *"Required Action"* below. The controller can also query the switch about which of the *"Optional Action"* it supports.

Required Action: **Output**. The Output action forwards a packet to a specified OpenFlow port (see 4.1). OpenFlow switches must support forwarding to physical ports, switch-defined logical ports and the required reserved ports (see 4.5).

Optional Action: **Set-Queue**. The set-queue action sets the queue id for a packet. When the packet is forwarded to a port using the output action, the queue id determines which queue attached to this port is used for scheduling and forwarding the packet. Forwarding behavior is dictated by the configuration of the queue and is used to provide basic Quality-of-Service (QoS) support (see section 7.2.2).

Required Action: **Drop**. There is no explicit action to represent drops. Instead, packets whose action sets have no output actions should be dropped. This result could come from empty instruction sets or empty action buckets in the processing pipeline, or after executing a Clear-Actions instruction.

Required Action: **Group**. Process the packet through the specified group. The exact interpretation depends on group type.

Optional Action: **Push-Tag/Pop-Tag**. Switches may support the ability to push/pop tags as shown in Table 6. To aid integration with existing networks, we suggest that the ability to push/pop VLAN tags be supported.

Newly pushed tags should *always* be inserted as the outermost tag in the outermost valid location for that tag. When a new VLAN tag is pushed, it should be the outermost tag inserted, immediately after the Ethernet header and before other tags. Likewise, when a new MPLS tag is pushed, it should be the outermost tag inserted, immediately after the Ethernet header and before other tags.

When multiple push actions are added to the action set of the packet, they apply to the packet in the order defined by the action set rules, first MPLS, then PBB, than VLAN (se 5.10). When multiple push actions are included in an action list, they apply to the packet in the list order (see 5.11).

Note: Refer to section 5.12.1 for information on default field values.

Action	Associated Data	Description
Push VLAN header	Ethertype	Push a new VLAN header onto the packet. The Ethertype is used as the Ethertype for the tag. Only Ethertype 0x8100 and 0x88a8 should be used.
Pop VLAN header	-	Pop the outer-most VLAN header from the packet.
Push MPLS header	Ethertype	Push a new MPLS shim header onto the packet. The Ethertype is used as the Ethertype for the tag. Only Ethertype 0x8847 and 0x8848 should be used.

Table 6 – Continued on next page

Table 6 – concluded from previous page

Action	Associated Data	Description
Pop MPLS header	Ethertype	Pop the outer-most MPLS tag or shim header from the packet. The Ethertype is used as the Ethertype for the resulting packet (Ethertype for the MPLS payload).
Push PBB header	Ethertype	Push a new PBB service instance header (I-TAG TCI) onto the packet (see 7.2.5). The Ethertype is used as the Ethertype for the tag. Only Ethertype 0x88E7 should be used.
Pop PBB header	-	Pop the outer-most PBB service instance header (I-TAG TCI) from the packet (see 7.2.5).

Table˜6: Push/pop tag actions.

Optional Action: **Set-Field**. The various Set-Field actions are identified by their field type and modify the values of respective header fields in the packet. While not strictly required, the support of rewriting various header fields using Set-Field actions greatly increase the usefulness of an OpenFlow implementation. To aid integration with existing networks, we suggest that VLAN modification actions be supported. Set-Field actions should *always* be applied to the outermost-possible header (e.g. a "Set VLAN ID" action always sets the ID of the outermost VLAN tag), unless the field type specifies otherwise.

Optional Action: **Change-TTL**. The various Change-TTL actions modify the values of the IPv4 TTL, IPv6 Hop Limit or MPLS TTL in the packet. While not strictly required, the actions shown in Table 7 greatly increase the usefulness of an OpenFlow implementation for implementing routing functions. Change-TTL actions should *always* be applied to the outermost-possible header.

Action	Associated Data	Description
Set MPLS TTL	8 bits: New MPLS TTL	Replace the existing MPLS TTL. Only applies to packets with an existing MPLS shim header.
Decrement MPLS TTL	-	Decrement the MPLS TTL. Only applies to packets with an existing MPLS shim header.
Set IP TTL	8 bits: New IP TTL	Replace the existing IPv4 TTL or IPv6 Hop Limit and update the IP checksum. Only applies to IPv4 and IPv6 packets.
Decrement IP TTL	-	Decrement the IPv4 TTL or IPv6 Hop Limit field and update the IP checksum. Only applies to IPv4 and IPv6 packets.
Copy TTL outwards	-	Copy the TTL from next-to-outermost to outermost header with TTL. Copy can be IP-to-IP, MPLS-to-MPLS, or IP-to-MPLS.
Copy TTL inwards	-	Copy the TTL from outermost to next-to-outermost header with TTL. Copy can be IP-to-IP, MPLS-to-MPLS, or MPLS-to-IP.

Table˜7: Change-TTL actions.

The OpenFlow switch checks for packets with invalid IP TTL or MPLS TTL and rejects them. Checking for invalid TTL does not need to be done for every packet, but it must be done at a minimum every time a *decrement TTL* action is applied to a packet. The asynchronous configuration of the switch may

be changed (see 6.1.1) to send packets with invalid TTL to the controller over the control channel via a packet-in message (see 6.1.2).

5.12.1 Default values for fields on push

Field values for all fields specified in Table 8 should be copied from existing outer headers to new outer headers when executing a push action. New fields listed in Table 8 without corresponding existing fields should be set to zero. Fields that cannot be modified via OpenFlow set-field actions should be initialized to appropriate protocol values.

New Fields		Existing Field(s)
VLAN ID	←	VLAN ID
VLAN priority	←	VLAN priority
MPLS label	←	MPLS label
MPLS traffic class	←	MPLS traffic class
MPLS TTL	←	{ MPLS TTL IP TTL
PBB I-SID	←	PBB I-SID
PBB I-PCP	←	VLAN PCP
PBB C-DA	←	ETH DST
PBB C-SA	←	ETH SRC

Table˜8: Existing fields that may be copied into new fields on a push action.

Fields in new headers may be overridden by specifying a "set" action for the appropriate field(s) after the push operation.

6 OpenFlow Channel

The OpenFlow channel is the interface that connects each OpenFlow switch to a controller. Through this interface, the controller configures and manages the switch, receives events from the switch, and sends packets out the switch.

Between the datapath and the OpenFlow channel, the interface is implementation-specific, however all OpenFlow channel messages must be formatted according to the OpenFlow protocol. The OpenFlow channel is usually encrypted using TLS, but may be run directly over TCP.

6.1 OpenFlow Protocol Overview

The OpenFlow protocol supports three message types, *controller-to-switch*, *asynchronous*, and *symmetric*, each with multiple sub-types. Controller-to-switch messages are initiated by the controller and used to directly manage or inspect the state of the switch. Asynchronous messages are initiated by the switch and used to update the controller of network events and changes to the switch state. Symmetric messages are initiated by either the switch or the controller and sent without solicitation. The message types used by OpenFlow are described below.

6.1.1 Controller-to-Switch

Controller/switch messages are initiated by the controller and may or may not require a response from the switch.

Features: The controller may request the identity and the basic capabilities of a switch by sending a features request; the switch must respond with a features reply that specifies the identity and basic capabilities of the switch. This is commonly performed upon establishment of the OpenFlow channel.

Configuration: The controller is able to set and query configuration parameters in the switch. The switch only responds to a query from the controller.

Modify-State: Modify-State messages are sent by the controller to manage state on the switches. Their primary purpose is to add, delete and modify flow/group entries in the OpenFlow tables and to set switch port properties.

Read-State: Read-State messages are used by the controller to collect various information from the switch, such as current configuration, statistics and capabilities.

Packet-out: These are used by the controller to send packets out of a specified port on the switch, and to forward packets received via Packet-in messages. Packet-out messages must contain a full packet or a buffer ID referencing a packet stored in the switch. The message must also contain a list of actions to be applied in the order they are specified; an empty action list drops the packet.

Barrier: Barrier request/reply messages are used by the controller to ensure message dependencies have been met or to receive notifications for completed operations.

Role-Request: Role-Request messages are used by the controller to set the role of its OpenFlow channel, or query that role. This is mostly useful when the switch connects to multiple controllers (see 6.3.4).

Asynchronous-Configuration: The Asynchronous-Configuration message are used by the controller to set an additional filter on the asynchronous messages that it wants to receive on its OpenFlow channel, or to query that filter. This is mostly useful when the switch connects to multiple controllers (see 6.3.4) and commonly performed upon establishment of the OpenFlow channel.

6.1.2 Asynchronous

Asynchronous messages are sent without a controller soliciting them from a switch. Switches send asynchronous messages to controllers to denote a packet arrival, switch state change, or error. The four main asynchronous message types are described below.

Packet-in: Transfer the control of a packet to the controller. For all packets forwarded to the **CONTROLLER** reserved port using a flow entry or the table-miss flow entry, a packet-in event is always sent to controllers (see 5.12). Other processing, such as TTL checking, may also generate packet-in events to send packets to the controller.

Packet-in events can be configured to buffer packets. For packet-in generated by an output action in a flow entries or group bucket, it can be specified individually in the output action itself (see 7.2.5), for other packet-in it can be configured in the switch configuration (see 7.3.2). If the packet-in event is configured to buffer packets and the switch has sufficient memory to buffer them, the packet-in events

contain only some fraction of the packet header and a buffer ID to be used by a controller when it is ready for the switch to forward the packet. Switches that do not support internal buffering, are configured to not buffer packets for the packet-in event, or have run out of internal buffering, must send the full packet to controllers as part of the event. Buffered packets will usually be processed via a **Packet-out** message from a controller, or automatically expired after some time.

If the packet is buffered, the number of bytes of the original packet to include in the packet-in can be configured. By default, it is 128 bytes. For packet-in generated by an output action in a flow entries or group bucket, it can be specified individually in the output action itself (see 7.2.5), for other packet-in it can be configured in the switch configuration (see 7.3.2).

Flow-Removed: Inform the controller about the removal of a flow entry from a flow table. Flow-Removed messages are only sent for flow entries with the `OFPFF_SEND_FLOW_REM` flag set. They are generated as the result of a controller flow delete requests or the switch flow expiry process when one of the flow timeout is exceeded (see 5.5).

Port-status: Inform the controller of a change on a port. The switch is expected to send port-status messages to controllers as port configuration or port state changes. These events include change in port configuration events, for example if it was brought down directly by a user, and port state change events, for example if the link went down.

Error: The switch is able to notify controllers of problems using error messages.

6.1.3 Symmetric

Symmetric messages are sent without solicitation, in either direction.

Hello: Hello messages are exchanged between the switch and controller upon connection startup.

Echo: Echo request/reply messages can be sent from either the switch or the controller, and must return an echo reply. They are mainly used to verify the liveness of a controller-switch connection, and may as well be used to measure its latency or bandwidth.

Experimenter: Experimenter messages provide a standard way for OpenFlow switches to offer additional functionality within the OpenFlow message type space. This is a staging area for features meant for future OpenFlow revisions.

6.2 Message Handling

The OpenFlow protocol provides reliable message delivery and processing, but does *not* automatically provide acknowledgements or ensure ordered message processing. The OpenFlow message handling behaviour described in this section is provided on the main connection and auxiliary connections using reliable transport, however it is not supported on auxiliary connections using unreliable transport (see 6.3.5).

Message Delivery: Messages are guaranteed delivery, unless the OpenFlow channel fails entirely, in which case the controller should not assume anything about the switch state (e.g., the switch may have gone into "fail standalone mode").

Message Processing: Switches must process every message received from a controller in full, possibly generating a reply. If a switch cannot completely process a message received from a controller, it must send back an error message. For packet-out messages, fully processing the message does not guarantee that the included packet actually exits the switch. The included packet may be silently dropped after OpenFlow processing due to congestion at the switch, QoS policy, or if sent to a blocked or invalid port.

In addition, switches must send to the controller all asynchronous messages generated by OpenFlow state changes, such as flow-removed, port-status or packet-in messages, so that the controller view of the switch is consistent with its actual state. Those messages may get filtered out based on the *Asynchronous Configuration* (see 6.1.1). Moreover, conditions that would trigger an OpenFlow state change may get filtered prior to causing such change. For example, packets received on data ports that should be forwarded to the controller may get dropped due to congestion or QoS policy within the switch and generate no packet-in messages. These drops may occur for packets with an explicit output action to the controller. These drops may also occur when a packet fails to match any entries in a table and that table's default action is to send to the controller. The policing of packets destined to the controller using QoS actions or rate limiting is advised, to prevent denial of service of the controller connection, and is outside the scope of the present specification.

Controllers are free to ignore messages they receive, but must respond to echo messages to prevent the switch from terminating the connection.

Message Ordering: Ordering can be ensured through the use of *barrier* messages. In the absence of barrier messages, switches may arbitrarily reorder messages to maximize performance; hence, controllers should not depend on a specific processing order. In particular, flow entries may be inserted in tables in an order different than that of flow mod messages received by the switch. Messages must not be reordered across a barrier message and the barrier message must be processed only when all prior messages have been processed. More precisely:

1. messages before a barrier must be fully processed before the barrier, including sending any resulting replies or errors
2. the barrier must then be processed and a barrier reply sent
3. messages after the barrier may then begin processing

If two messages from the controller depend on each other, they must be separated by a barrier message. Examples of such message dependancies include a group mod add with a flow mod add referencing the group, a port mod with a packet-out forwarding to the port, or a flow mod add with a following packet-out to `OFPP_TABLE`.

6.3 OpenFlow Channel Connections

The OpenFlow channel is used to exchange OpenFlow message between an OpenFlow switch and an OpenFlow controller. A typical OpenFlow controller manages multiple OpenFlow channels, each one to a different OpenFlow switch. An OpenFlow switch may have one OpenFlow channel to a single controller, or multiple channels for reliability, each to a different controller (see 6.3.4).

An OpenFlow controller typically manages an OpenFlow switch remotely over one or more networks. The specification of the networks used for the OpenFlow channels is outside the scope of the present

specification. It may be a separate dedicated network, or the OpenFlow channel may use the network managed by the OpenFlow switch (in-band controller connection). The only requirement is that it should provide TCP/IP connectivity.

The OpenFlow channel is usually instantiated as a single network connection between the switch and the controller, using TLS or plain TCP (see 6.3.3). Alternatively, the OpenFlow channel may be composed of multiple network connections to exploit parallelism (see 6.3.5). The OpenFlow switch must be able to create an OpenFlow channel by initiating a connection to an OpenFlow controller (see 6.3.1). Some switch implementations may optionally allow an OpenFlow controller to connect to the OpenFlow switch, in this case the switch usually should restrict itself to secured connections (see 6.3.3) to prevent unauthorised connections.

6.3.1 Connection Setup

The switch must be able to establish communication with a controller at a user-configurable (but otherwise fixed) IP address, using either a user-specified transport port or the default transport port (see 6.3.3). If the switch is configured with the IP address of the controller to connect to, the switch initiates a standard TLS or TCP connection to the controller. Traffic to and from the OpenFlow channel is not run through the OpenFlow pipeline. Therefore, the switch must identify incoming traffic as local before checking it against the flow tables.

When an OpenFlow connection is first established, each side of the connection must immediately send an `OFPT_HELLO` message with the `version` field set to the highest OpenFlow protocol version supported by the sender (see 7.1). This Hello message may optionally include some OpenFlow elements to help connection setup (see 7.5.1). Upon receipt of this message, the recipient must calculate the OpenFlow protocol version to be used. If both the Hello message sent and the Hello message received contained a `OFPHET_VERSIONBITMAP` hello element, and if those bitmaps have some common bits set, the negotiated version must be the highest version set in both bitmaps. Otherwise, the negotiated version must be the smaller of the version number that was sent and the one that was received in the `version` fields.

If the negotiated version is supported by the recipient, then the connection proceeds. Otherwise, the recipient must reply with an `OFPT_ERROR` message with a `type` field of `OFPET_HELLO_FAILED`, a `code` field of `OFPHFC_INCOMPATIBLE`, and optionally an ASCII string explaining the situation in `data`, and then terminate the connection.

After the switch and the controller have exchanged `OFPT_HELLO` messages and successfully negotiated a common version number, the connection setup is done and standard OpenFlow messages can be exchanged over the connection. One of the first thing that the controller should do is to send a `OFPT_FEATURES_REQUEST` message to get the *Datapath ID* of the switch (see 7.3.1).

6.3.2 Connection Interruption

In the case that a switch loses contact with all controllers, as a result of echo request timeouts, TLS session timeouts, or other disconnections, the switch must *immediately* enter either "fail secure mode" or "fail standalone mode", depending upon the switch implementation and configuration. In "fail secure mode", the only change to switch behavior is that packets and messages destined to the controllers are dropped. Flow entries should continue to expire according to their timeouts in "fail secure mode". In

"fail standalone mode", the switch processes all packets using the `OFPP_NORMAL` reserved port; in other words, the switch acts as a legacy Ethernet switch or router. The "fail standalone mode" is usually only available on Hybrid switches (see 5.1).

Upon connecting to a controller again, the existing flow entries remain. The controller then has the option of deleting all flow entries, if desired.

The first time a switch starts up, it will operate in either "fail secure mode" or "fail standalone mode" mode, until it successfully connects to a controller. Configuration of the default set of flow entries to be used at startup is outside the scope of the OpenFlow protocol.

6.3.3 Encryption

The switch and controller may communicate through a TLS connection. The TLS connection is initiated by the switch on startup to the controller, which is located by default on TCP port 6633 . The switch and controller mutually authenticate by exchanging certificates signed by a site-specific private key. Each switch must be user-configurable with one certificate for authenticating the controller (controller certificate) and the other for authenticating to the controller (switch certificate).

The switch and controller may optionally communicate using plain TCP. The TCP connection is initiated by the switch on startup to the controller, which is located by default on TCP port 6633 . When using plain TCP, it is recommended to use alternative security measures to prevent eavesdropping, controller impersonation or other attacks on the OpenFlow channel.

6.3.4 Multiple Controllers

The switch may establish communication with a single controller, or may establish communication with multiple controllers. Having multiple controllers improves reliability, as the switch can continue to operate in OpenFlow mode if one controller or controller connection fails. The hand-over between controllers is entirely managed by the controllers themselves, which enables fast recovery from failure and also controller load balancing. The controllers coordinate the management of the switch amongst themselves via mechanisms outside the scope of the present specification, and the goal of the multiple controller functionality is only to help synchronise controller handoffs performed by the controllers. The multiple controller functionality only addresses controller fail-over and load balancing, and doesn't address virtualisation which can be done outside the OpenFlow protocol.

When OpenFlow operation is initiated, the switch must connect to all controllers it is configured with, and try to maintain connectivity with all of them concurrently. Many controllers may send controller-to-switch commands to the switch, the reply or error messages related to those commands must only be sent on the controller connection associated with that command. Asynchronous messages may need to be sent to multiple controllers, the message is duplicated for each eligible OpenFlow channel and each message sent when the respective controller connection allows it.

The default role of a controller is `OFPCR_ROLE_EQUAL`. In this role, the controller has full access to the switch and is equal to other controllers in the same role. By default, the controller receives all the switch asynchronous messages (such as packet-in, flow-removed). The controller can send controller-to-switch commands to modify the state of the switch. The switch does not do any arbitration or resource sharing between controllers.

A controller can request its role to be changed to OFPCR_ROLE_SLAVE. In this role, the controller has read-only access to the switch. By default, the controller does not receive switch asynchronous messages, apart from Port-status messages. The controller is denied the ability to execute all controller-to-switch commands that send packets or modify the state of the switch. For example, OFPT_PACKET_OUT, OFPT_FLOW_MOD, OFPT_GROUP_MOD, OFPT_PORT_MOD, OFPT_TABLE_MOD requests, and OFPMP_TABLE_FEATURES multiparts requests with a non-empty body must be rejected. If the controller sends one of those commands, the switch must reply with an OFPT_ERROR message with a type field of OFPET_BAD_REQUEST, a code field of OFPBRC_IS_SLAVE. Other controller-to-switch messages, such as OFPT_ROLE_REQUEST, OFPT_SET_ASYNC and OFPT_MULTIPART_REQUEST that only query data, should be processed normally.

A controller can request its role to be changed to OFPCR_ROLE_MASTER. This role is similar to OFPCR_ROLE_EQUAL and has full access to the switch, the difference is that the switch ensures it is the only controller in this role. When a controller changes its role to OFPCR_ROLE_MASTER, the switch changes all other controllers with the role OFPCR_ROLE_MASTER to have the role OFPCR_ROLE_SLAVE, but does not affect controllers with role OFPCR_ROLE_EQUAL. When the switch performs such role changes, no message is generated to the controller whose role is changed (in most cases that controller is no longer reachable).

Each controller may send a OFPT_ROLE_REQUEST message to communicate its role to the switch (see 7.3.9), and the switch must remember the role of each controller connection. A controller may change role at any time, provided the generation_id in the message is current (see below).

The role request message offers a lightweight mechanism to help the controller master election process, the controllers configure their role and usually still need to coordinate among themselve. The switch can not change the state of a controller on its own, controller state is always changed as a result of a request from one of the controllers. Any Slave controller or Equal controller can elect itself Master. A switch may be *simultaneously* connected to multiple controllers in Equal state, multiple controllers in Slave state, and at most one controller in Master state. The controller in Master state (if any) and all the controllers in Equal state can fully change the switch state, there is no mechanism to enforce partitioning of the switch between those controllers. If the controller in Master role need to be the only controller able to make changes on the switch, then no controllers should be in Equal state and all other controllers should be in Slave state.

A controller can also control which types of switch asynchronous messages are sent over its OpenFlow channel, and change the defaults described above. This is done via a *Asynchronous Configuration* message (see 6.1.1), listing all reasons for each message type that need to be enabled or filtered out (see 7.3.10) for the specific OpenFlow channel. Using this feature, different controllers can receive different notifications, a controller in master mode can selectively disable notifications it does not care about, and a controller in slave mode can enable notifications it wants to monitor.

To detect out-of-order messages during a master/slave transition, the OFPT_ROLE_REQUEST message contains a 64-bit sequence number field, generation_id, that identifies a given mastership view. As a part of the master election mechanism, controllers (or a third party on their behalf) coordinate the assignment of generation_id. generation_id is a monotonically increasing counter: a new (larger) generation_id is assigned each time the mastership view changes, e.g. when a new master is designated. generation_id can wrap around.

On receiving a OFPT_ROLE_REQUEST with role equal to OFPCR_ROLE_MASTER or OFPCR_ROLE_SLAVE the switch must compare the generation_id in the message against the largest generation id seen so far.

A message with a generation_id smaller than a previously seen generation id must be considered stale and discarded. The switch must respond to stale messages with an error message with type OFPET_ROLE_REQUEST_FAILED and code OFPRRFC_STALE.

The following pseudo-code describes the behavior of the switch in dealing with generation_id.

On switch startup:

```
generation_is_defined = false;
```

On receiving OFPT_ROLE_REQUEST with role equal to OFPCR_ROLE_MASTER or OFPCR_ROLE_SLAVE and with a given generation_id, say GEN_ID_X:

```
if (generation_is_defined AND
    distance(GEN_ID_X, cached_generation_id) < 0) {
  <discard OFPT_ROLE_REQUEST message>;
  <send an error message with code OFPRRFC_STALE>;
} else {
  cached_generation_id = GEN_ID_X;
  generation_is_defined = true;
  <process the message normally>;
}
```

where distance() is the *Wrapping Sequence Number Distance* operator defined as following:

```
distance(a, b) := (int64_t)(a - b)
```

I.e. distance() is the unsigned difference between the sequence numbers, interpreted as a two's complement signed value. This results in a positive distance if a is greater than b (in a circular sense) but less than "half the sequence number space" away from it. It results in a negative distance otherwise (a < b).

The switch must ignore generation_id if the role in the OFPT_ROLE_REQUEST is OFPCR_ROLE_EQUAL, as generation_id is specifically intended for the disambiguation of race condition in master/slave transition.

6.3.5 Auxiliary Connections

By default, the *OpenFlow channel* between an OpenFlow switch and an OpenFlow controller is a single network connection. The OpenFlow channel may also be composed of a **main connection** and multiple **auxiliary connections**. Auxiliary connections are created by the OpenFlow switch and are helpful to improve the switch processing performance and exploit the parallelism of most switch implementations.

Each connection from the switch to the controller is identified by the switch *Datapath ID* and a *Auxiliary ID* (see 7.3.1). The main connection must have its Auxiliary ID set to zero, whereas auxiliary connection must have a non-zero Auxiliary ID and the same Datapath ID. Auxiliary connections must use the same source IP address as the main connection, but can use a different transport, for example TLS, TCP,

DTLS or UDP, depending on the switch configuration. The auxiliary connection should have the same destination IP address and same transport destination port as the main connection, unless the switch configuration specifies otherwise. The controller must recognise incoming connections with non-zero Auxiliary ID as auxiliary connections and bind them to the main connection with the same Datapath ID.

The switch must not initiate auxiliary connection before having completed the connection setup over the main connection (see 6.3.1), it must setup and maintain auxiliary connections with the controller only while the corresponding main connection is alive. The connection setup for auxiliary connections is the same as for the main connection (see 6.3.1). If the switch detects that the main connection to a controller is broken, it must immediately close all its auxiliary connections to that controller, to enable the controller to properly resolve Datapath ID conflicts.

Both the OpenFlow switch and the OpenFlow controller must accept any OpenFlow message types and sub-types on all connections : the main connection or an auxiliary connection can not be restricted to a specific message type or sub-type. However, the processing performance of different message types or sub-types on different connections may be different. The switch may service auxiliary connections with different priorities, for example one auxiliary connection may be dedicated to high priority requests and always processed by the switch before other auxiliary connections. A switch configuration, for example using the OpenFlow Configuration Protocol, may optionally configure the priority of auxiliary connections.

A reply to an OpenFlow request must be made on the same connection it came in. There is no synchronisation between connections, and messages sent on different connections may be processed in any order. A barrier message applies only to the connection where it is used (see 6.2). Auxiliary connections using DTLS or UDP may lose or reorder messages, OpenFlow does not provide ordering or delivery guarantees on those connections (see 6.2). If messages must be processed in sequence, they must be sent over the same connection, use a connection that does not reorder packets, and use barrier messages.

The controller is free to use the various switch connections for sending OpenFlow messages at its entire discretion, however to maximise performance on most switches the following guidelines are suggested:

- All OpenFlow controller messages which are not Packet-out (flow-mod, statistic request...) should be sent over the main connection.
- All Packet-Out messages containing a packet from a Packet-In message should be sent on the connection where the Packet-In came from.
- All other Packet-Out messages should be spread across the various auxiliary connections using a mechanism keeping the packets of a same flow mapped to the same connection.
- If the desired auxiliary connection is not available, the controller should use the main connection.

The switch is free to use the various controller connections for sending OpenFlow messages as it wishes, however the following guidelines are suggested :

- All OpenFlow messages which are not Packet-in should be sent over the main connection.
- All Packet-In messages spread across the various auxiliary connection using a mechanism keeping the packets of a same flow mapped to the same connection.

Auxiliary connection on **unreliable transports** (UDP, DTLS) have additional restrictions and rules that don't apply to auxiliary connection on other transport (TCP, TLS). The only message types supported on unreliable auxiliary connections are `OFPT_HELLO`, `OFPT_ERROR`, `OFPT_ECHO_REQUEST`, `OFPT_ECHO_REPLY`, `OFPT_FEATURES_REQUEST`, `OFPT_FEATURES_REPLY`, `OFPT_PACKET_IN`, `OFPT_PACKET_OUT` and `OFPT_EXPERIMENTER`, other messages types are not supported by the specification.

On unreliable auxiliary connection, *Hello messages* are sent at connection initiation to setup the connection (see 6.3.1). If an OpenFlow device receives another message on an unreliable auxiliary connection prior to receiving a *Hello message*, the device must either assume the connection is setup properly and use the version number from that message, or it must return an Error message with `OFPET_BAD_REQUEST` type and `OFPBRC_BAD_VERSION` code. If a OpenFlow device receives a error message with `OFPET_BAD_REQUEST` type and `OFPBRC_BAD_VERSION` code on unreliable auxiliary connection, it must either send a new *Hello message* or terminate the unreliable auxiliary connection (the connection may be retried at a later time). If no message was ever received on an auxiliary connection after some implementation chosen amount of time lower than 5 seconds, the device must either send a new *Hello* message or terminate the unreliable auxiliary connection. If after sending a *Feature Request* message, the controller does not receives a *Feature Reply* message after some implementation chosen amount of time lower than 5 seconds, the device must either send a new *Feature Request* message or terminate the unreliable auxiliary connection. If after receiving a message, a device does not receives any other message after some implementation chosen amount of time lower than 30 seconds, the device must terminate the unreliable auxiliary connection. If a device receives a message for a unreliable auxiliary connection already terminated, it must assume it is a new connection.

OpenFlow devices using unreliable auxiliary connection should follow recommendations in RFC 5405 when possible.

6.4 Flow Table Modification Messages

Flow table modification messages can have the following types:

```
enum ofp_flow_mod_command {
    OFPFC_ADD           = 0, /* New flow. */
    OFPFC_MODIFY        = 1, /* Modify all matching flows. */
    OFPFC_MODIFY_STRICT = 2, /* Modify entry strictly matching wildcards and
                                priority. */
    OFPFC_DELETE        = 3, /* Delete all matching flows. */
    OFPFC_DELETE_STRICT = 4, /* Delete entry strictly matching wildcards and
                                priority. */
};
```

For **add** requests (`OFPFC_ADD`) with the `OFPFF_CHECK_OVERLAP` flag set, the switch must first check for any overlapping flow entries in the requested table. Two flow entries overlap if a single packet may match both, and both entries have the same priority. If an overlap conflict exists between an existing flow entry and the **add** request, the switch must refuse the addition and respond with an `ofp_error_msg` with `OFPET_FLOW_MOD_FAILED` type and `OFPFMFC_OVERLAP` code.

For non-overlapping **add** requests, or those with no overlap checking, the switch must insert the flow entry in the requested table. If a flow entry with identical match fields and priority already resides in

the requested table, then that entry, including its duration, must be cleared from the table, and the new flow entry added. If the OFPFF_RESET_COUNTS flag is set, the flow entry counters must be cleared, otherwise they should be copied from the replaced flow entry. No flow-removed message is generated for the flow entry eliminated as part of an **add** request; if the controller wants a flow-removed message it should explicitly send a **delete** request for the old flow entry prior to adding the new one.

For **modify** requests (OFPFC_MODIFY or OFPFC_MODIFY_STRICT), if a matching entry exists in the table, the instructions field of this entry is updated with the value from the request, whereas its cookie, idle_timeout, hard_timeout, flags, counters and duration fields are left unchanged. If the OFPFF_RESET_COUNTS flag is set, the flow entry counters must be cleared. For **modify** requests, if no flow entry currently residing in the requested table matches the request, no error is recorded, and no flow table modification occurs.

For **delete** requests (OFPFC_DELETE or OFPFC_DELETE_STRICT), if a matching entry exists in the table, it must be deleted, and if the entry has the OFPFF_SEND_FLOW_REM flag set, it should generate a flow removed message. For **delete** requests, if no flow entry currently residing in the requested table matches the request, no error is recorded, and no flow table modification occurs.

Modify and **delete** flow_mod commands have *non-strict* versions (OFPFC_MODIFY and OFPFC_DELETE) and *strict* versions (OFPFC_MODIFY_STRICT or OFPFC_DELETE_STRICT). In the *strict* versions, the set of match fields, all match fields, including their masks, and the priority, are strictly matched against the entry, and only an identical flow entry is modified or removed. For example, if a message to remove entries is sent that has no match fields included, the OFPFC_DELETE command would delete all flow entries from the tables, while the OFPFC_DELETE_STRICT command would only delete a flow entry that applies to all packets at the specified priority.

For *non-strict* **modify** and **delete** commands, all flow entries that match the flow_mod description are modified or removed. In the *non-strict* versions, a match will occur when a flow entry exactly matches or is more specific than the description in the flow_mod command; in the flow_mod the missing match fields are wildcarded, field masks are active, and other flow_mod fields such as priority are ignored. For example, if a OFPFC_DELETE command says to delete all flow entries with a destination port of 80, then a flow entry that wildcards all match fields will not be deleted. However, a OFPFC_DELETE command that wildcards all match fields will delete an entry that matches all port 80 traffic. This same interpretation of mixed wildcard and exact match fields also applies to individual and aggregate flows stats requests.

Delete commands can be optionally filtered by destination group or output port. If the out_port field contains a value other than OFPP_ANY, it introduces a constraint when matching. This constraint is that each matching flow entry must contain an *output* action directed at the specified port in the actions associated with that flow entry. This constraint is limited to only the actions directly associated with the flow entry. In other words, the switch must not recurse through the action sets of pointed-to groups, which may have matching *output* actions. The out_group, if different from OFPG_ANY, introduce a similar constraint on the *group* action. These fields are ignored by OFPFC_ADD, OFPFC_MODIFY and OFPFC_MODIFY_STRICT messages.

Modify and **delete** commands can also be filtered by cookie value, if the cookie_mask field contains a value other than 0. This constraint is that the bits specified by the cookie_mask in both the cookie field of the flow mod and a flow entry's cookie value must be equal. In other words, $(flow\_entry.cookie \,\&\, flow\_mod.cookie\_mask) == (flow\_mod.cookie \,\&\, flow\_mod.cookie\_mask)$.

Delete commands can use the OFPTT_ALL value for table-id to indicate that matching flow entries are to be deleted from all flow tables.

If the flow modification message specifies an invalid table-id, the switch must send an ofp_error_msg with OFPET_FLOW_MOD_FAILED type and OFPFMFC_BAD_TABLE_ID code. If the flow modification message specifies OFPTT_ALL for table-id in a **add** or **modify** request, the switch must send the same error message.

If a switch cannot find any space in the requested table in which to add the incoming flow entry, the switch must send an ofp_error_msg with OFPET_FLOW_MOD_FAILED type and OFPFMFC_TABLE_FULL code.

If the instructions requested in a flow mod message are unknown the switch must return an ofp_error_msg with OFPET_BAD_INSTRUCTION type and OFPBIC_UNKNOWN_INST code. If the instructions requested in a flow mod message are unsupported the switch must return an ofp_error_msg with OFPET_BAD_INSTRUCTION type and OFPBIC_UNSUP_INST code.

If the instructions requested contain a Goto-Table and the next-table-id refers to an invalid table the switch must return an ofp_error_msg with OFPET_BAD_INSTRUCTION type and OFPBIC_BAD_TABLE_ID code.

If the instructions requested contain a Write-Metadata and the metadata value or metadata mask value is unsupported then the switch must return an ofp_error_msg with OFPET_BAD_INSTRUCTION type and OFPBIC_UNSUP_METADATA or OFPBIC_UNSUP_METADATA_MASK code.

If the match in a flow mod message specifies a field that is unsupported in the table, the switch must return an ofp_error_msg with OFPET_BAD_MATCH type and OFPBMC_BAD_FIELD code. If the match in a flow mod message specifies a field more than once, the switch must return an ofp_error_msg with OFPET_BAD_MATCH type and OFPBMC_DUP_FIELD code. If the match in a flow mod message specifies a field but fail to specify its associated prerequisites, for example specifies an IPv4 address without matching the EtherType to 0x800, the switch must return an ofp_error_msg with OFPET_BAD_MATCH type and OFPBMC_BAD_PREREQ code.

If the match in a flow mod specifies an arbitrary bitmask for either the datalink or network addresses which the switch cannot support, the switch must return an ofp_error_msg with OFPET_BAD_MATCH type and either OFPBMC_BAD_DL_ADDR_MASK or OFPBMC_BAD_NW_ADDR_MASK. If the bitmasks specified in *both* the datalink and network addresses are not supported then OFPBMC_BAD_DL_ADDR_MASK should be used. If the match in a flow mod specifies an arbitrary bitmask for another field which the switch cannot support, the switch must return an ofp_error_msg with OFPET_BAD_MATCH type and OFPBMC_BAD_MASK code.

If the match in a flow mod specifies values that cannot be matched, for example, a VLAN ID greater than 4095 and not one of the reserved values, or a DSCP value with one of the two higher bits set, the switch must return an ofp_error_msg with OFPET_BAD_MATCH type and OFPBMC_BAD_VALUE code.

If any action references a port that will never be valid on a switch, the switch must return an ofp_error_msg with OFPET_BAD_ACTION type and OFPBAC_BAD_OUT_PORT code. If the referenced port may be valid in the future, e.g. when a linecard is added to a chassis switch, or a port is dynamically added to a software switch, the switch must either silently drop packets sent to the referenced port, or immediately return an OFPBAC_BAD_OUT_PORT error and refuse the flow mod.

If an action in a flow mod message references a group that is not currently defined on the switch, or is a reserved group, such as OFPG_ALL, the switch must return an ofp_error_msg with OFPET_BAD_ACTION type and OFPBAC_BAD_OUT_GROUP code.

If an action in a flow mod message has a value that is invalid, for example a Set VLAN ID action with value greater than 4095, or a Push action with an invalid Ethertype, the switch must return an ofp_error_msg with OFPET_BAD_ACTION type and OFPBAC_BAD_ARGUMENT code.

If an action in a flow mod message performs an operation which is inconsistent with the match, for example, a pop VLAN action with a match specifying no VLAN, or a set IPv4 address action with a match wildcarding the Ethertype, the switch may optionally reject the flow mod and immediately return an ofp_error_msg with OFPET_BAD_ACTION type and OFPBAC_MATCH_INCONSISTENT code. The effect of any inconsistent actions on matched packets is undefined. Controllers are strongly encouraged to avoid generating combinations of table entries that may yield inconsistent actions.

If an action list contain a sequence of actions that the switch can not support in the specified order, the switch must return an ofp_error_msg with OFPET_BAD_ACTION type and OFPBAC_UNSUPPORTED_ORDER code.

If any other errors occur during the processing of the flow mod message, the switch may return an ofp_error_msg with OFPET_FLOW_MOD_FAILED type and OFPFMC_UNKNOWN code.

6.5 Group Table Modification Messages

Group table modification messages can have the following types:

```
/* Group commands */
enum ofp_group_mod_command {
    OFPGC_ADD    = 0,       /* New group. */
    OFPGC_MODIFY = 1,       /* Modify all matching groups. */
    OFPGC_DELETE = 2,       /* Delete all matching groups. */
};
```

Groups may consist of zero or more buckets. A group with no buckets will not alter the action set associated with a packet. A group may also include buckets which themselves forward to other groups if the switch supports it.

The action set for each bucket must be validated using the same rules as those for flow mods (Section 6.4), with additional group-specific checks. If an action in one of the buckets is invalid or unsupported, the switch should return an ofp_error_msg with OFPET_BAD_ACTION type and code corresponding to the error (see 6.4).

For **add** requests (OFPGC_ADD), if a group entry with the specified group identifier already resides in the group table, then the switch must refuse to add the group entry and must send an ofp_error_msg with OFPET_GROUP_MOD_FAILED type and OFPGMFC_GROUP_EXISTS code.

For **modify** requests (OFPGC_MODIFY), if a group entry with the specified group identifier already resides in the group table, then that entry, including its type and action buckets, must be removed, and the new group entry added. If a group entry with the specified group identifier does not already exist then the switch must refuse the group mod and send an ofp_error_msg with OFPET_GROUP_MOD_FAILED type and OFPGMFC_UNKNOWN_GROUP code.

If a specified group type is invalid (ie: includes fields such as `weight` that are undefined for the specified group type) then the switch must refuse to add the group entry and must send an `ofp_error_msg` with `OFPET_GROUP_MOD_FAILED` type and `OFPGMFC_INVALID_GROUP` code.

If a switch does not support unequal load sharing with select groups (buckets with weight different than 1), it must refuse to add the group entry and must send an `ofp_error_msg` with `OFPET_GROUP_MOD_FAILED` type and `OFPGMFC_WEIGHT_UNSUPPORTED` code.

If a switch cannot add the incoming group entry due to lack of space, the switch must send an `ofp_error_msg` with `OFPET_GROUP_MOD_FAILED` type and `OFPGMFC_OUT_OF_GROUPS` code.

If a switch cannot add the incoming group entry due to restrictions (hardware or otherwise) limiting the number of group buckets, it must refuse to add the group entry and must send an `ofp_error_msg` with `OFPET_GROUP_MOD_FAILED` type and `OFPGMFC_OUT_OF_BUCKETS` code.

If a switch cannot add the incoming group because it does not support the proposed liveliness configuration, the switch must send an `ofp_error_msg` with `OFPET_GROUP_MOD_FAILED` type and `OFPGMFC_WATCH_UNSUPPORTED` code. This includes specifying `watch_port` or `watch_group` for a group that does not support liveness, or specifying a port that does not support liveness in `watch_port`, or specifying a group that does not support liveness in `watch_group`.

For **delete** requests (`OFPGC_DELETE`), if no group entry with the specified group identifier currently exists in the group table, no error is recorded, and no group table modification occurs. Otherwise, the group is removed, and all flow entries containing this group in a Group action are also removed. The group type need not be specified for the **delete** request. **Delete** also differs from an **add** or **modify** with no buckets specified in that future attempts to **add** the group identifier will not result in a group exists error. If one wishes to effectively delete a group yet leave in flow entries using it, that group can be cleared by sending a **modify** with no buckets specified.

To delete all groups with a single message, specify `OFPG_ALL` as the group value.

Groups may be *chained* if the switch supports it, when at least one group forward to another group, or in more complex configuration. For example, a fast reroute group may have two buckets, where each points to a select group. If a switch does not support groups of groups, it must send an `ofp_error_msg` with `OFPET_GROUP_MOD_FAILED` type and `OFPGMFC_CHAINING_UNSUPPORTED` code.

A switch may support checking that no loop is created while chaining groups : if a group mod is sent such that a forwarding loop would be created, the switch must reject the group mod and must send an `ofp_error_msg` with `OFPET_GROUP_MOD_FAILED` type and `OFPGMFC_LOOP` code. If the switch does not support such checking, the forwarding behavior is undefined.

A switch may support checking that groups forwarded to by other groups are not removed : If a switch cannot delete a group because it is referenced by another group, it must refuse to delete the group entry and must send an `ofp_error_msg` with `OFPET_GROUP_MOD_FAILED` type and `OFPGMFC_CHAINED_GROUP` code. If the switch does not support such checking, the forwarding behavior is undefined.

Fast failover group support requires *liveness monitoring*, to determine the specific bucket to execute. Other group types are not required to implement liveness monitoring, but may optionally implement it. If a switch cannot implement liveness checking for any bucket in a group, it must refuse the group mod and return an error. The rules for determining liveness include:

- A port is considered live if it has the OFPPS_LIVE flag set in its port state. Port liveness may be managed by code outside of the OpenFlow portion of a switch, defined outside of the OpenFlow specification, such as Spanning Tree or a KeepAlive mechanism. The port must not be considered live (and the OFPPS_LIVE flag must be unset) if one of the port liveness mechanisms enabled on the switch consider the port not live, or if the port config bit OFPPC_PORT_DOWN indicates the port is down, or if the port state bit OFPPS_LINK_DOWN indicates the link is down.
- A bucket is considered live if either watch_port is not OFPP_ANY and the port watched is live, or if watch_group is not OFPG_ANY and the group watched is live.
- A group is considered live if a least one of its buckets is live.

The controller can infer the liveness state of the group by monitoring the states of the various ports.

6.6 Meter Modification Messages

Meter modification messages can have the following types:

```
/* Meter commands */
enum ofp_meter_mod_command {
    OFPMC_ADD,              /* New meter. */
    OFPMC_MODIFY,           /* Modify specified meter. */
    OFPMC_DELETE,           /* Delete specified meter. */
};
```

For **add** requests (OFPMC_ADD), if a meter entry with the specified meter identifier already exist, then the switch must refuse to add the meter entry and must send an ofp_error_msg with OFPET_METER_MOD_FAILED type and OFPMMFC_METER_EXISTS code.

For **modify** requests (OFPMC_MODIFY), if a meter entry with the specified meter identifier already exists, then that entry, including its bands, must be removed, and the new meter entry added. If a meter entry with the specified meter identifier does not already exists then the switch must refuse the meter mod and send an ofp_error_msg with OFPET_METER_MOD_FAILED type and OFPMMFC_UNKNOWN_METER code.

If a switch cannot add the incoming meter entry due to lack of space, the switch must send an ofp_error_msg with OFPET_METER_MOD_FAILED type and OFPMMFC_OUT_OF_METERS code.

If a switch cannot add the incoming meter entry due to restrictions (hardware or otherwise) limiting the number of bands, it must refuse to add the meter entry and must send an ofp_error_msg with OFPET_METER_MOD_FAILED type and OFPMMFC_OUT_OF_BANDS code.

For **delete** requests (OFPMC_DELETE), if no meter entry with the specified meter identifier currently exists, no error is recorded, and no meter modification occurs. Otherwise, the meter is removed, and all flows that include the meter in their instruction set are also removed. Only the meter identifier need to be specified for the **delete** request, other fields such as bands can be omitted.

To delete all meters with a single message, specify OFPM_ALL as the meter value. Virtual meters can never be deleted and are not removed when deleting all meters.

7 The OpenFlow Protocol

The heart of the OpenFlow switch specification is the set of structures used for OpenFlow Protocol messages.

The structures, defines, and enumerations described below are derived from the file `include/openflow/openflow.h`, which is part of the standard OpenFlow specification distribution. All structures are packed with padding and 8-byte aligned, as checked by the assertion statements. All OpenFlow messages are sent in big-endian format.

7.1 OpenFlow Header

Each OpenFlow message begins with the OpenFlow header:

```
/* Header on all OpenFlow packets. */
struct ofp_header {
    uint8_t version;    /* OFP_VERSION. */
    uint8_t type;       /* One of the OFPT_ constants. */
    uint16_t length;    /* Length including this ofp_header. */
    uint32_t xid;       /* Transaction id associated with this packet.
                           Replies use the same id as was in the request
                           to facilitate pairing. */
};
OFP_ASSERT(sizeof(struct ofp_header) == 8);
```

The `version` specifies the OpenFlow protocol version being used. During the earlier draft phase of the OpenFlow Protocol, the most significant bit was set to indicate an experimental version. The lower bits indicate the revision number of the protocol. The version of the protocol described by the current specification is 1.3.2, and its *ofp_version* is 0x04 .

The `length` field indicates the total length of the message, so no additional framing is used to distinguish one frame from the next. The `type` can have the following values:

```
enum ofp_type {
    /* Immutable messages. */
    OFPT_HELLO              = 0,  /* Symmetric message */
    OFPT_ERROR              = 1,  /* Symmetric message */
    OFPT_ECHO_REQUEST       = 2,  /* Symmetric message */
    OFPT_ECHO_REPLY         = 3,  /* Symmetric message */
    OFPT_EXPERIMENTER       = 4,  /* Symmetric message */

    /* Switch configuration messages. */
    OFPT_FEATURES_REQUEST   = 5,  /* Controller/switch message */
    OFPT_FEATURES_REPLY     = 6,  /* Controller/switch message */
    OFPT_GET_CONFIG_REQUEST = 7,  /* Controller/switch message */
    OFPT_GET_CONFIG_REPLY   = 8,  /* Controller/switch message */
    OFPT_SET_CONFIG         = 9,  /* Controller/switch message */

    /* Asynchronous messages. */
    OFPT_PACKET_IN          = 10, /* Async message */
    OFPT_FLOW_REMOVED       = 11, /* Async message */
```



```
    OFPT_PORT_STATUS        = 12, /* Async message */

    /* Controller command messages. */
    OFPT_PACKET_OUT         = 13, /* Controller/switch message */
    OFPT_FLOW_MOD           = 14, /* Controller/switch message */
    OFPT_GROUP_MOD          = 15, /* Controller/switch message */
    OFPT_PORT_MOD           = 16, /* Controller/switch message */
    OFPT_TABLE_MOD          = 17, /* Controller/switch message */

    /* Multipart messages. */
    OFPT_MULTIPART_REQUEST      = 18, /* Controller/switch message */
    OFPT_MULTIPART_REPLY        = 19, /* Controller/switch message */

    /* Barrier messages. */
    OFPT_BARRIER_REQUEST    = 20, /* Controller/switch message */
    OFPT_BARRIER_REPLY      = 21, /* Controller/switch message */

    /* Queue Configuration messages. */
    OFPT_QUEUE_GET_CONFIG_REQUEST = 22,  /* Controller/switch message */
    OFPT_QUEUE_GET_CONFIG_REPLY   = 23,  /* Controller/switch message */

    /* Controller role change request messages. */
    OFPT_ROLE_REQUEST       = 24, /* Controller/switch message */
    OFPT_ROLE_REPLY         = 25, /* Controller/switch message */

    /* Asynchronous message configuration. */
    OFPT_GET_ASYNC_REQUEST  = 26, /* Controller/switch message */
    OFPT_GET_ASYNC_REPLY    = 27, /* Controller/switch message */
    OFPT_SET_ASYNC          = 28, /* Controller/switch message */

    /* Meters and rate limiters configuration messages. */
    OFPT_METER_MOD          = 29, /* Controller/switch message */
};
```

7.1.1 Padding

Most OpenFlow messages contain padding fields. Those are included in the various message types and in various common structures. Most of those padding fields can be identified by the fact that their name start with `pad`. The goal of padding fields is to align multi-byte entities on natural processor boundaries.

All common structures included in messages are aligned on 64 bit boundaries. Various other types are aligned as needed, for example 32 bits integer are aligned on 32 bit boundaries. An exception to the padding rules are OXM match fields which are never padded (7.2.3.2). In general, the end of OpenFlow messages is not padded, unless explicitly specified. On the other hand, common structures are almost always padded at the end.

The padding fields should be set to zero. An OpenFlow implementation must accept any values set in padding fields, and must just ignore the content of padding fields.

7.2 Common Structures

This section describes structures used by multiple message types.

7.2.1 Port Structures

The OpenFlow pipeline receives and sends packets on ports. The switch may define physical and logical ports, and the OpenFlow specification defines some reserved ports (see 4.1).

The physical ports, switch-defined logical ports, and the `OFPP_LOCAL` reserved port are described with the following structure:

```
/* Description of a port */
struct ofp_port {
    uint32_t port_no;
    uint8_t pad[4];
    uint8_t hw_addr[OFP_ETH_ALEN];
    uint8_t pad2[2];                  /* Align to 64 bits. */
    char name[OFP_MAX_PORT_NAME_LEN]; /* Null-terminated */

    uint32_t config;        /* Bitmap of OFPPC_* flags. */
    uint32_t state;         /* Bitmap of OFPPS_* flags. */

    /* Bitmaps of OFPPF_* that describe features.  All bits zeroed if
     * unsupported or unavailable. */
    uint32_t curr;          /* Current features. */
    uint32_t advertised;    /* Features being advertised by the port. */
    uint32_t supported;     /* Features supported by the port. */
    uint32_t peer;          /* Features advertised by peer. */

    uint32_t curr_speed;    /* Current port bitrate in kbps. */
    uint32_t max_speed;     /* Max port bitrate in kbps */
};
OFP_ASSERT(sizeof(struct ofp_port) == 64);
```

The `port_no` field uniquely identifies a port within a switch. The `hw_addr` field typically is the MAC address for the port; `OFP_ETH_ALEN` is 6. The name field is a null-terminated string containing a human-readable name for the interface. The value of `OFP_MAX_PORT_NAME_LEN` is 16.

The `config` field describes port administrative settings, and has the following structure:

```
/* Flags to indicate behavior of the physical port.  These flags are
 * used in ofp_port to describe the current configuration.  They are
 * used in the ofp_port_mod message to configure the port's behavior.
 */
enum ofp_port_config {
    OFPPC_PORT_DOWN    = 1 << 0,  /* Port is administratively down. */

    OFPPC_NO_RECV      = 1 << 2,  /* Drop all packets received by port. */
    OFPPC_NO_FWD       = 1 << 5,  /* Drop packets forwarded to port. */
    OFPPC_NO_PACKET_IN = 1 << 6   /* Do not send packet-in msgs for port. */
};
```


The OFPPC_PORT_DOWN bit indicates that the port has been administratively brought down and should not be used by OpenFlow. The OFPPC_NO_RECV bit indicates that packets received on that port should be ignored. The OFPPC_NO_FWD bit indicates that OpenFlow should not send packets to that port. The OFPPFL_NO_PACKET_IN bit indicates that packets on that port that generate a table miss should never trigger a packet-in message to the controller.

In general, the port config bits are set by the controller and not changed by the switch. Those bits may be useful for the controller to implement protocols such as STP or BFD. If the port config bits are changed by the switch through another administrative interface, the switch sends an OFPT_PORT_STATUS message to notify the controller of the change.

The state field describes the port internal state, and has the following structure:

```
/* Current state of the physical port.  These are not configurable from
 * the controller.
 */
enum ofp_port_state {
    OFPPS_LINK_DOWN    = 1 << 0,  /* No physical link present. */
    OFPPS_BLOCKED      = 1 << 1,  /* Port is blocked */
    OFPPS_LIVE         = 1 << 2,  /* Live for Fast Failover Group. */
};
```

The port state bits represent the state of the physical link or switch protocols outside of OpenFlow. The OFPPS_LINK_DOWN bit indicates the the physical link is not present. The OFPPS_BLOCKED bit indicates that a switch protocol outside of OpenFlow, such as 802.1D Spanning Tree, is preventing the use of that port with OFPP_FLOOD.

All port state bits are read-only and cannot be changed by the controller. When the port flags are changed, the switch sends an OFPT_PORT_STATUS message to notify the controller of the change.

The port numbers use the following conventions:

```
/* Port numbering. Ports are numbered starting from 1. */
enum ofp_port_no {
    /* Maximum number of physical and logical switch ports. */
    OFPP_MAX        = 0xffffff00,

    /* Reserved OpenFlow Port (fake output "ports"). */
    OFPP_IN_PORT    = 0xfffffff8,  /* Send the packet out the input port.  This
                                      reserved port must be explicitly used
                                      in order to send back out of the input
                                      port. */
    OFPP_TABLE      = 0xfffffff9,  /* Submit the packet to the first flow table
                                      NB: This destination port can only be
                                      used in packet-out messages. */
    OFPP_NORMAL     = 0xfffffffa,  /* Process with normal L2/L3 switching. */
    OFPP_FLOOD      = 0xfffffffb,  /* All physical ports in VLAN, except input
                                      port and those blocked or link down. */
    OFPP_ALL        = 0xfffffffc,  /* All physical ports except input port. */
    OFPP_CONTROLLER = 0xfffffffd,  /* Send to controller. */
    OFPP_LOCAL      = 0xfffffffe,  /* Local openflow "port". */
    OFPP_ANY        = 0xffffffff   /* Wildcard port used only for flow mod
                                      (delete) and flow stats requests. Selects
```



```
                                    all flows regardless of output port
                                    (including flows with no output port). */
};
```

The `curr`, `advertised`, `supported`, and `peer` fields indicate link modes (speed and duplexity), link type (copper/fiber) and link features (autonegotiation and pause). Port features are represented by the following structure:

```
/* Features of ports available in a datapath. */
enum ofp_port_features {
    OFPPF_10MB_HD    = 1 << 0,  /* 10 Mb half-duplex rate support. */
    OFPPF_10MB_FD    = 1 << 1,  /* 10 Mb full-duplex rate support. */
    OFPPF_100MB_HD   = 1 << 2,  /* 100 Mb half-duplex rate support. */
    OFPPF_100MB_FD   = 1 << 3,  /* 100 Mb full-duplex rate support. */
    OFPPF_1GB_HD     = 1 << 4,  /* 1 Gb half-duplex rate support. */
    OFPPF_1GB_FD     = 1 << 5,  /* 1 Gb full-duplex rate support. */
    OFPPF_10GB_FD    = 1 << 6,  /* 10 Gb full-duplex rate support. */
    OFPPF_40GB_FD    = 1 << 7,  /* 40 Gb full-duplex rate support. */
    OFPPF_100GB_FD   = 1 << 8,  /* 100 Gb full-duplex rate support. */
    OFPPF_1TB_FD     = 1 << 9,  /* 1 Tb full-duplex rate support. */
    OFPPF_OTHER      = 1 << 10, /* Other rate, not in the list. */

    OFPPF_COPPER     = 1 << 11, /* Copper medium. */
    OFPPF_FIBER      = 1 << 12, /* Fiber medium. */
    OFPPF_AUTONEG    = 1 << 13, /* Auto-negotiation. */
    OFPPF_PAUSE      = 1 << 14, /* Pause. */
    OFPPF_PAUSE_ASYM = 1 << 15  /* Asymmetric pause. */
};
```

Multiple of these flags may be set simultaneously. If none of the port speed flags are set, the `max_speed` or `curr_speed` are used.

The `curr_speed` and `max_speed` fields indicate the current and maximum bit rate (raw transmission speed) of the link in kbps. The number should be rounded to match common usage. For example, an optical 10 Gb Ethernet port should have this field set to 10000000 (instead of 10312500), and an OC-192 port should have this field set to 10000000 (instead of 9953280).

The `max_speed` fields indicate the maximum configured capacity of the link, whereas the `curr_speed` indicates the current capacity. If the port is a LAG with 3 links of 1Gb/s capacity, with one of the ports of the LAG being down, one port auto-negotiated at 1Gb/s and 1 port auto-negotiated at 100Mb/s, the `max_speed` is 3 Gb/s and the `curr_speed` is 1.1 Gb/s.

7.2.2 Queue Structures

An OpenFlow switch provides limited Quality-of-Service support (QoS) through a simple queuing mechanism. One (or more) queues can attach to a port and be used to map flow entries on it. Flow entries mapped to a specific queue will be treated according to that queue's configuration (e.g. min rate).

A queue is described by the `ofp_packet_queue` structure:


```
/* Full description for a queue. */
struct ofp_packet_queue {
    uint32_t queue_id;     /* id for the specific queue. */
    uint32_t port;         /* Port this queue is attached to. */
    uint16_t len;          /* Length in bytes of this queue desc. */
    uint8_t pad[6];        /* 64-bit alignment. */
    struct ofp_queue_prop_header properties[0]; /* List of properties. */
};
OFP_ASSERT(sizeof(struct ofp_packet_queue) == 16);
```

Each queue is further described by a set of properties, each of a specific type and configuration.

```
enum ofp_queue_properties {
    OFPQT_MIN_RATE      = 1,      /* Minimum datarate guaranteed. */
    OFPQT_MAX_RATE      = 2,      /* Maximum datarate. */
    OFPQT_EXPERIMENTER  = 0xffff  /* Experimenter defined property. */
};
```

Each queue property description starts with a common header:

```
/* Common description for a queue. */
struct ofp_queue_prop_header {
    uint16_t property;    /* One of OFPQT_. */
    uint16_t len;         /* Length of property, including this header. */
    uint8_t pad[4];       /* 64-bit alignemnt. */
};
OFP_ASSERT(sizeof(struct ofp_queue_prop_header) == 8);
```

A *minimum-rate* queue property uses the following structure and fields:

```
/* Min-Rate queue property description. */
struct ofp_queue_prop_min_rate {
    struct ofp_queue_prop_header prop_header; /* prop: OFPQT_MIN, len: 16. */
    uint16_t rate;        /* In 1/10 of a percent; >1000 -> disabled. */
    uint8_t pad[6];       /* 64-bit alignment */
};
OFP_ASSERT(sizeof(struct ofp_queue_prop_min_rate) == 16);
```

If `rate` is not configured, it is set to `OFPQ_MIN_RATE_UNCFG`, which is 0xffff.

A *maximum-rate* queue property uses the following structure and fields:

```
/* Max-Rate queue property description. */
struct ofp_queue_prop_max_rate {
    struct ofp_queue_prop_header prop_header; /* prop: OFPQT_MAX, len: 16. */
    uint16_t rate;        /* In 1/10 of a percent; >1000 -> disabled. */
    uint8_t pad[6];       /* 64-bit alignment */
};
OFP_ASSERT(sizeof(struct ofp_queue_prop_max_rate) == 16);
```

If `rate` is not configured, it is set to `OFPQ_MAX_RATE_UNCFG`, which is 0xffff.

A *experimenter* queue property uses the following structure and fields:


```
/* Experimenter queue property description. */
struct ofp_queue_prop_experimenter {
    struct ofp_queue_prop_header prop_header; /* prop: OFPQT_EXPERIMENTER, len: 16. */
    uint32_t experimenter;          /* Experimenter ID which takes the same
                                       form as in struct
                                       ofp_experimenter_header. */
    uint8_t pad[4];       /* 64-bit alignment */
    uint8_t data[0];      /* Experimenter defined data. */
};
OFP_ASSERT(sizeof(struct ofp_queue_prop_experimenter) == 16);
```

The rest of the experimenter queue property body is uninterpreted by standard OpenFlow processing and is arbitrarily defined by the corresponding experimenter.

7.2.3 Flow Match Structures

An OpenFlow match is composed of a flow match header and a sequence of zero or more flow match fields.

7.2.3.1 Flow Match Header

The flow match header is described by the `ofp_match` structure:

```
/* Fields to match against flows */
struct ofp_match {
    uint16_t type;             /* One of OFPMT_* */
    uint16_t length;           /* Length of ofp_match (excluding padding) */
    /* Followed by:
     *   - Exactly (length - 4) (possibly 0) bytes containing OXM TLVs, then
     *   - Exactly ((length + 7)/8*8 - length) (between 0 and 7) bytes of
     *     all-zero bytes
     * In summary, ofp_match is padded as needed, to make its overall size
     * a multiple of 8, to preserve alignement in structures using it.
     */
    uint8_t oxm_fields[0];     /* 0 or more OXM match fields */
    uint8_t pad[4];            /* Zero bytes - see above for sizing */
};
OFP_ASSERT(sizeof(struct ofp_match) == 8);
```

The `type` field is set to `OFPMT_OXM` and `length` field is set to the actual length of `ofp_match` structure including all match fields. The payload of the OpenFlow match is a set of OXM Flow match fields.

```
/* The match type indicates the match structure (set of fields that compose the
 * match) in use. The match type is placed in the type field at the beginning
 * of all match structures. The "OpenFlow Extensible Match" type corresponds
 * to OXM TLV format described below and must be supported by all OpenFlow
 * switches. Extensions that define other match types may be published on the
 * ONF wiki. Support for extensions is optional.
 */
enum ofp_match_type {
```



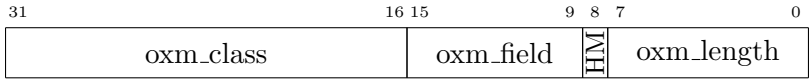
```
    OFPMT_STANDARD = 0,       /* Deprecated. */
    OFPMT_OXM      = 1,       /* OpenFlow Extensible Match */
};
```

The only valid match type in this specification is `OFPMT_OXM`, the OpenFlow 1.1 match type `OFPMT_STANDARD` is deprecated. If an alternate match type is used, the match fields and payload may be set differently, but this is outside the scope of this specification.

7.2.3.2 Flow Match Field Structures

The flow match fields are described using the OpenFlow Extensible Match (OXM) format, which is a compact type-length-value (TLV) format. Each OXM TLV is 5 to 259 (inclusive) bytes long. OXM TLVs are not aligned on or padded to any multibyte boundary. The first 4 bytes of an OXM TLV are its header, followed by the entry's body.

An OXM TLV's header is interpreted as a 32-bit word in network byte order (see figure 4).



Figure˜4: OXM TLV header layout.

The OXM TLV's header fields are defined in Table 9.

Name		Width	Usage
oxm_type	`oxm_class`	16	Match class: member class or reserved class
	`oxm_field`	7	Match field within the class
	`oxm_hasmask`	1	Set if OXM include a bitmask in payload
	`oxm_length`	8	Length of OXM payload

Table˜9: OXM TLV header fields.

The `oxm_class` is a OXM match class that contains related match types, and is described in section 7.2.3.3. `oxm_field` is an class-specific value, identifying one of the match types within the match class. The combination of `oxm_class` and `oxm_field` (the most-significant 23 bits of the header) are collectively `oxm_type`. The `oxm_type` normally designates a protocol header field, such as the Ethernet type, but it can also refer to packet metadata, such as the switch port on which a packet arrived.

`oxm_hasmask` defines if the OXM TLV contains a bitmask, more details is explained in section 7.2.3.5.

`oxm_length` is a positive integer describing the length of the OXM TLV payload in bytes. The length of the OXM TLV, including the header, is exactly 4 + `oxm_length` bytes.

For a given `oxm_class`, `oxm_field`, and `oxm_hasmask` value, `oxm_length` is a constant. It is included only to allow software to minimally parse OXM TLVs of unknown types. (Similarly, for a given `oxm_class`, `oxm_field`, and `oxm_length`, `oxm_hasmask` is a constant.)

7.2.3.3 OXM classes

The match types are structured using OXM match classes. The OpenFlow specification distinguishes two types of OXM match classes, ONF member classes and ONF reserved classes, differentiated by their high order bit. Classes with the high order bit set to 1 are ONF reserved classes, they are used for the OpenFlow specification itself. Classes with the high order bit set to zero are ONF member classes, they are allocated by the ONF on an as needed basis, they uniquely identify an ONF member and can be used arbitrarily by that member. Support for ONF member classes is optional.

The following OXM classes are defined:

```
/* OXM Class IDs.
 * The high order bit differentiate reserved classes from member classes.
 * Classes 0x0000 to 0x7FFF are member classes, allocated by ONF.
 * Classes 0x8000 to 0xFFFE are reserved classes, reserved for standardisation.
 */
enum ofp_oxm_class {
    OFPXMC_NXM_0          = 0x0000,    /* Backward compatibility with NXM */
    OFPXMC_NXM_1          = 0x0001,    /* Backward compatibility with NXM */
    OFPXMC_OPENFLOW_BASIC = 0x8000,    /* Basic class for OpenFlow */
    OFPXMC_EXPERIMENTER   = 0xFFFF,    /* Experimenter class */
};
```

The class `OFPXMC_OPENFLOW_BASIC` contains the basic set of OpenFlow match fields (see 7.2.3.7). The optional class `OFPXMC_EXPERIMENTER` is used for experimenter matches (see 7.2.3.8). Other ONF reserved classes are reserved for future uses such as modularisation of the specification. The first two ONF member classes `OFPXMC_NXM_0` and `OFPXMC_NXM_1` are reserved for backward compatibility with the Nicira Extensible Match (NXM) specification.

7.2.3.4 Flow Matching

A zero-length OpenFlow match (one with no OXM TLVs) matches every packet. Match fields that should be wildcarded are omitted from the OpenFlow match.

An OXM TLV places a constraint on the packets matched by the OpenFlow match:

- If `oxm_hasmask` is 0, the OXM TLV's body contains a value for the field, called `oxm_value`. The OXM TLV match matches only packets in which the corresponding field equals `oxm_value`.
- If `oxm_hasmask` is 1, then the `oxm_entry`'s body contains a value for the field (`oxm_value`), followed by a bitmask of the same length as the value, called `oxm_mask`. Each 1-bit in `oxm_mask` constrains the OXM TLV to match only packets in which the corresponding bit of the field equals the the corresponding bit in `oxm_value`. Each 0-bit in `oxm_mask` places no constraint on the corresponding bit in the field.

When using masking, it is an error for a 0-bit in `oxm_mask` to have a corresponding 1-bit in `oxm_value`. The switch must report an error message of type `OFPET_BAD_MATCH` and code `OFPBMC_BAD_WILDCARDS` in such a case.

The following table summarizes the constraint that a pair of corresponding `oxm_mask` and `oxm_value` bits place upon the corresponding field bit when using masking. Omitting `oxm_mask` is equivalent to supplying an `oxm_mask` that is all 1-bits.

	`oxm_value`	
`oxm_mask`	0	1
0	no constraint	error
1	must be 0	must be 1

Table˜10: OXM mask and value.

When there are multiple OXM TLVs, all of the constraints must be met: the packet fields must match all OXM TLVs part of the OpenFlow match. The fields for which OXM TLVs that are not present are wildcarded to ANY, omitted OXM TLVs are effectively fully masked to zero.

7.2.3.5 Flow Match Field Masking

When `oxm_hasmask` is 1, the OXM TLV contains a bitmask and its length is effectively doubled, so `oxm_length` is always even. The bitmask follows the field value and is encoded in the same way. The masks are defined such that a 0 in a given bit position indicates a "don't care" match for the same bit in the corresponding field, whereas a 1 means match the bit exactly.

An all-zero-bits `oxm_mask` is equivalent to omitting the OXM TLV entirely. An all-one-bits `oxm_mask` is equivalent to specifying 0 for `oxm_hasmask` and omitting `oxm_mask`.

Some `oxm_types` may not support masked wildcards, that is, `oxm_hasmask` must always be 0 when these fields are specified. For example, the field that identifies the ingress port on which a packet was received may not be masked.

Some `oxm_types` that do support masked wildcards may only support certain `oxm_mask` patterns. For example, some fields that have IPv4 address values may be restricted to CIDR masks (subnet masks).

These restrictions are detailed in specifications for individual fields. A switch may accept an `oxm_hasmask` or `oxm_mask` value that the specification disallows, but only if the switch correctly implements support for that `oxm_hasmask` or `oxm_mask` value. A switch must reject an attempt to set up a flow entry that contains a `oxm_hasmask` or `oxm_mask` value that it does not support (see 6.4).

7.2.3.6 Flow Match Field Prerequisite

The presence of an OXM TLV with a given `oxm_type` may be restricted based on the presence or values of other OXM TLVs. In general, matching header fields of a protocol can only be done if the OpenFlow match explitly matches the corresponding protocol.

For example:

- An OXM TLV for `oxm_type`=OXM_OF_IPV4_SRC is allowed only if it is preceded by another entry with `oxm_type=OXM_OF_ETH_TYPE`, `oxm_hasmask`=0, and `oxm_value`=0x0800. That is, matching on the IPv4 source address is allowed only if the Ethernet type is explicitly set to IPv4.

- An OXM TLV for `oxm_type`=OXM_OF_TCP_SRC is allowed only if it is preceded by an entry with `oxm_type`=OXM_OF_ETH_TYPE, `oxm_hasmask`=0, `oxm_value`=0x0800 or 0x86dd, and another with `oxm_type`=OXM_OF_IP_PROTO, `oxm_hasmask`=0, `oxm_value`=6, in that order. That is, matching on the TCP source port is allowed only if the Ethernet type is IP and the IP protocol is TCP.
- An OXM TLV for `oxm_type`=OXM_OF_MPLS_LABEL is allowed only if it is preceded by an entry with `oxm_type`=OXM_OF_ETH_TYPE, `oxm_hasmask`=0, `oxm_value`=0x8847 or 0x8848.
- An OXM TLV for `oxm_type`=OXM_OF_VLAN_PCP is allowed only if it is preceded by an entry with `oxm_type`=OXM_OF_VLAN_VID, `oxm_value`!=OFPVID_NONE.

These restrictions are noted in specifications for individual fields (see 7.2.3.7). A switch may implement relaxed versions of these restrictions. For example, a switch may accept no prerequisite at all. A switch must reject an attempt to set up a flow entry that violates its restrictions (see 6.4), and must deal with inconsistent matches created by the lack of prerequisite (for example matching both a TCP source port and a UDP destination port).

New match fields defined by members (in member classes or as experimenter fields) may provide alternate prerequisites to already specified match fields. For example, this could be used to reuse existing IP match fields over an alternate link technology (such as PPP) by substituting the `ETH_TYPE` prerequisite as needed (for PPP, that could be an hypothetical `PPP_PROTOCOL` field).

An OXM TLV that has prerequisite restrictions must appear after the OXM TLVs for its prerequisites. Ordering of OXM TLVs within an OpenFlow match is not otherwise constrained.

Any given `oxm_type` may appear in an OpenFlow match at most once, otherwise the switch must generate an error (see 6.4). A switch may implement a relaxed version of this rule and may allow in some cases a `oxm_type` to appear multiple time in an OpenFlow match, however the behaviour of matching is then implementation-defined.

If a flow table implements a specific OXM TLV, this flow table must accept valid matches containing the prerequisites of this OXM TLV, even if the flow table does not support matching all possible values for the match fields specified by those prerequisites. For example, if a flow table matches the IPv4 source address, this flow table must accept matching the Ethertype exactly to IPv4, however this flow table does not need to support matching Ethertype to any other value.

7.2.3.7 Flow Match Fields

The specification defines a default set of match fields with `oxm_class`=`OFPXMC_OPENFLOW_BASIC` which can have the following values:

```
/* OXM Flow match field types for OpenFlow basic class. */
enum oxm_ofb_match_fields {
    OFPXMT_OFB_IN_PORT        = 0,  /* Switch input port. */
    OFPXMT_OFB_IN_PHY_PORT    = 1,  /* Switch physical input port. */
    OFPXMT_OFB_METADATA       = 2,  /* Metadata passed between tables. */
    OFPXMT_OFB_ETH_DST        = 3,  /* Ethernet destination address. */
    OFPXMT_OFB_ETH_SRC        = 4,  /* Ethernet source address. */
    OFPXMT_OFB_ETH_TYPE       = 5,  /* Ethernet frame type. */
    OFPXMT_OFB_VLAN_VID       = 6,  /* VLAN id. */
```



```
    OFPXMT_OFB_VLAN_PCP       = 7,  /* VLAN priority. */
    OFPXMT_OFB_IP_DSCP        = 8,  /* IP DSCP (6 bits in ToS field). */
    OFPXMT_OFB_IP_ECN         = 9,  /* IP ECN (2 bits in ToS field). */
    OFPXMT_OFB_IP_PROTO       = 10, /* IP protocol. */
    OFPXMT_OFB_IPV4_SRC       = 11, /* IPv4 source address. */
    OFPXMT_OFB_IPV4_DST       = 12, /* IPv4 destination address. */
    OFPXMT_OFB_TCP_SRC        = 13, /* TCP source port. */
    OFPXMT_OFB_TCP_DST        = 14, /* TCP destination port. */
    OFPXMT_OFB_UDP_SRC        = 15, /* UDP source port. */
    OFPXMT_OFB_UDP_DST        = 16, /* UDP destination port. */
    OFPXMT_OFB_SCTP_SRC       = 17, /* SCTP source port. */
    OFPXMT_OFB_SCTP_DST       = 18, /* SCTP destination port. */
    OFPXMT_OFB_ICMPV4_TYPE    = 19, /* ICMP type. */
    OFPXMT_OFB_ICMPV4_CODE    = 20, /* ICMP code. */
    OFPXMT_OFB_ARP_OP         = 21, /* ARP opcode. */
    OFPXMT_OFB_ARP_SPA        = 22, /* ARP source IPv4 address. */
    OFPXMT_OFB_ARP_TPA        = 23, /* ARP target IPv4 address. */
    OFPXMT_OFB_ARP_SHA        = 24, /* ARP source hardware address. */
    OFPXMT_OFB_ARP_THA        = 25, /* ARP target hardware address. */
    OFPXMT_OFB_IPV6_SRC       = 26, /* IPv6 source address. */
    OFPXMT_OFB_IPV6_DST       = 27, /* IPv6 destination address. */
    OFPXMT_OFB_IPV6_FLABEL    = 28, /* IPv6 Flow Label */
    OFPXMT_OFB_ICMPV6_TYPE    = 29, /* ICMPv6 type. */
    OFPXMT_OFB_ICMPV6_CODE    = 30, /* ICMPv6 code. */
    OFPXMT_OFB_IPV6_ND_TARGET = 31, /* Target address for ND. */
    OFPXMT_OFB_IPV6_ND_SLL    = 32, /* Source link-layer for ND. */
    OFPXMT_OFB_IPV6_ND_TLL    = 33, /* Target link-layer for ND. */
    OFPXMT_OFB_MPLS_LABEL     = 34, /* MPLS label. */
    OFPXMT_OFB_MPLS_TC        = 35, /* MPLS TC. */
    OFPXMT_OFP_MPLS_BOS       = 36, /* MPLS BoS bit. */
    OFPXMT_OFB_PBB_ISID       = 37, /* PBB I-SID. */
    OFPXMT_OFB_TUNNEL_ID      = 38, /* Logical Port Metadata. */
    OFPXMT_OFB_IPV6_EXTHDR    = 39, /* IPv6 Extension Header pseudo-field */
};
```

A switch must support the required match fields listed in Table 11 in its pipeline. Each required match field must be supported in at least one flow table of the switch : that flow table must enable matching that field and the match field prerequisites must be met in that table (see 7.2.3.6). The required fields don't need to be implemented in all flow tables, and don't need to be implemented in the same flow table. Flow tables can support non-required and experimenter match fields. The controller can query the switch about which match fields are supported in each flow table (see 7.3.5.5).

Field		Description
OXM_OF_IN_PORT	*Required*	Ingress port. This may be a physical or switch-defined logical port.
OXM_OF_ETH_DST	*Required*	Ethernet destination address. Can use arbitrary bitmask
OXM_OF_ETH_SRC	*Required*	Ethernet source address. Can use arbitrary bitmask
OXM_OF_ETH_TYPE	*Required*	Ethernet type of the OpenFlow packet payload, after VLAN tags.
OXM_OF_IP_PROTO	*Required*	IPv4 or IPv6 protocol number
OXM_OF_IPV4_SRC	*Required*	IPv4 source address. Can use subnet mask or arbitrary bitmask
OXM_OF_IPV4_DST	*Required*	IPv4 destination address. Can use subnet mask or arbitrary bitmask
OXM_OF_IPV6_SRC	*Required*	IPv6 source address. Can use subnet mask or arbitrary bitmask
OXM_OF_IPV6_DST	*Required*	IPv6 destination address. Can use subnet mask or arbitrary bitmask
OXM_OF_TCP_SRC	*Required*	TCP source port
OXM_OF_TCP_DST	*Required*	TCP destination port
OXM_OF_UDP_SRC	*Required*	UDP source port
OXM_OF_UDP_DST	*Required*	UDP destination port

Table˜11: Required match fields.

All match fields have different size, prerequisites and masking capability, as specified in Table 12. If not explictely specified in the field description, each field type refer the the outermost occurrence of the field in the packet headers.

Field	Bits	Mask	Pre-requisite	Description
OXM_OF_IN_PORT	32	No	None	Ingress port. Numerical representation of incoming port, starting at 1. This may be a physical or switch-defined logical port.
OXM_OF_IN_PHY_PORT	32	No	IN_PORT present	Physical port. In `ofp_packet_in` messages, underlying physical port when packet received on a logical port.
OXM_OF_METADATA	64	Yes	None	Table metadata. Used to pass information between tables.
OXM_OF_ETH_DST	48	Yes	None	Ethernet destination MAC address.
OXM_OF_ETH_SRC	48	Yes	None	Ethernet source MAC address.
OXM_OF_ETH_TYPE	16	No	None	Ethernet type of the OpenFlow packet payload, after VLAN tags.
OXM_OF_VLAN_VID	12+1	Yes	None	VLAN-ID from 802.1Q header. The CFI bit indicate the presence of a valid VLAN-ID, see below.
OXM_OF_VLAN_PCP	3	No	VLAN_VID!=NONE	VLAN-PCP from 802.1Q header.
OXM_OF_IP_DSCP	6	No	ETH_TYPE=0x0800 or ETH_TYPE=0x86dd	Diff Serv Code Point (DSCP). Part of the IPv4 ToS field or the IPv6 Traffic Class field.
OXM_OF_IP_ECN	2	No	ETH_TYPE=0x0800 or ETH_TYPE=0x86dd	ECN bits of the IP header. Part of the IPv4 ToS field or the IPv6 Traffic Class field.
OXM_OF_IP_PROTO	8	No	ETH_TYPE=0x0800 or ETH_TYPE=0x86dd	IPv4 or IPv6 protocol number.
OXM_OF_IPV4_SRC	32	Yes	ETH_TYPE=0x0800	IPv4 source address. Can use subnet mask or arbitrary bitmask
OXM_OF_IPV4_DST	32	Yes	ETH_TYPE=0x0800	IPv4 destination address. Can use subnet mask or arbitrary bitmask

Table 12 – Continued on next page

Table 12 – concluded from previous page

Field	Bits	Mask	Pre-requisite	Description
OXM_OF_TCP_SRC	16	No	IP_PROTO=6	TCP source port
OXM_OF_TCP_DST	16	No	IP_PROTO=6	TCP destination port
OXM_OF_UDP_SRC	16	No	IP_PROTO=17	UDP source port
OXM_OF_UDP_DST	16	No	IP_PROTO=17	UDP destination port
OXM_OF_SCTP_SRC	16	No	IP_PROTO=132	SCTP source port
OXM_OF_SCTP_DST	16	No	IP_PROTO=132	SCTP destination port
OXM_OF_ICMPV4_TYPE	8	No	IP_PROTO=1	ICMP type
OXM_OF_ICMPV4_CODE	8	No	IP_PROTO=1	ICMP code
OXM_OF_ARP_OP	16	No	ETH_TYPE=0x0806	ARP opcode
OXM_OF_ARP_SPA	32	Yes	ETH_TYPE=0x0806	Source IPv4 address in the ARP payload. Can use subnet mask or arbitrary bitmask
OXM_OF_ARP_TPA	32	Yes	ETH_TYPE=0x0806	Target IPv4 address in the ARP payload. Can use subnet mask or arbitrary bitmask
OXM_OF_ARP_SHA	48	Yes	ETH_TYPE=0x0806	Source Ethernet address in the ARP payload.
OXM_OF_ARP_THA	48	Yes	ETH_TYPE=0x0806	Target Ethernet address in the ARP payload.
OXM_OF_IPV6_SRC	128	Yes	ETH_TYPE=0x86dd	IPv6 source address. Can use subnet mask or arbitrary bitmask
OXM_OF_IPV6_DST	128	Yes	ETH_TYPE=0x86dd	IPv6 destination address. Can use subnet mask or arbitrary bitmask
OXM_OF_IPV6_FLABEL	20	Yes	ETH_TYPE=0x86dd	IPv6 flow label.
OXM_OF_ICMPV6_TYPE	8	No	IP_PROTO=58	ICMPv6 type
OXM_OF_ICMPV6_CODE	8	No	IP_PROTO=58	ICMPv6 code
OXM_OF_IPV6_ND_TARGET	128	No	ICMPV6_TYPE=135 or ICMPV6_TYPE=136	The target address in an IPv6 Neighbor Discovery message.
OXM_OF_IPV6_ND_SLL	48	No	ICMPV6_TYPE=135	The source link-layer address option in an IPv6 Neighbor Discovery message.
OXM_OF_IPV6_ND_TLL	48	No	ICMPV6_TYPE=136	The target link-layer address option in an IPv6 Neighbor Discovery message.
OXM_OF_MPLS_LABEL	20	No	ETH_TYPE=0x8847 or ETH_TYPE=0x8848	The LABEL in the first MPLS shim header.
OXM_OF_MPLS_TC	3	No	ETH_TYPE=0x8847 or ETH_TYPE=0x8848	The TC in the first MPLS shim header.
OXM_OF_MPLS_BOS	1	No	ETH_TYPE=0x8847 or ETH_TYPE=0x8848	The BoS bit (Bottom of Stack bit) in the first MPLS shim header.
OXM_OF_PBB_ISID	24	Yes	ETH_TYPE=0x88E7	The I-SID in the first PBB service instance tag.
OXM_OF_TUNNEL_ID	64	Yes	None	Metadata associated with a logical port.
OXM_OF_IPV6_EXTHDR	9	Yes	ETH_TYPE=0x86dd	IPv6 Extension Header pseudo-field.

Table˜12: Match fields details.

The ingress port OXM_OF_IN_PORT is a valid standard OpenFlow port, either a physical, a logical port, the OFPP_LOCAL reserved port or the OFPP_CONTROLLER reserved port. The physical port OXM_OF_IN_PHY_PORT is used in *Packet-in* messages to identify a physical port underneath a logical port (see 7.4.1).

The metadata field OXM_OF_METADATA is used to pass information between lookups across multiple tables. This value can be arbitrarily masked.

The Tunnel ID field OXM_OF_TUNNEL_ID carries optional metadata associated with a logical port. The mapping of this metadata is defined by the logical port implementation. If the logical port does not provide such data or if the packet was received on a physical port, its value is zero. For example, for a packet received via GRE tunnel including a (32-bit) key, the key is stored in the low 32-bits and the

high bits are zeroed. For a MPLS logical port, the low 20 bits represent the MPLS Label. For a VxLAN logical port, the low 24 bits represent the VNI.

Omitting the OFPXMT_OFB_VLAN_VID field specifies that a flow entry should match packets regardless of whether they contain the corresponding tag. Special values are defined below for the VLAN tag to allow matching of packets with any tag, independent of the tag's value, and to supports matching packets without a VLAN tag. The special values defined for OFPXMT_OFB_VLAN_VID are:

```
/* The VLAN id is 12-bits, so we can use the entire 16 bits to indicate
 * special conditions.
 */
enum ofp_vlan_id {
    OFPVID_PRESENT = 0x1000, /* Bit that indicate that a VLAN id is set */
    OFPVID_NONE    = 0x0000, /* No VLAN id was set. */
};
```

The OFPXMT_OFB_VLAN_PCP field must be rejected when the OFPXMT_OFB_VLAN_VID field is wildcarded (not present) or when the value of OFPXMT_OFB_VLAN_VID is set to OFPVID_NONE. Table 13 summarizes the combinations of wildcard bits and field values for particular VLAN tag matches.

OXM field	oxm_value	oxm_mask	Matching packets
absent	-	-	Packets *with* and *without* a VLAN tag
present	OFPVID_NONE	absent	Only packets *without* a VLAN tag
present	OFPVID_PRESENT	OFPVID_PRESENT	Only packets *with* a VLAN tag regardless of its value
present	*value* | OFPVID_PRESENT	absent	Only packets with VLAN tag and VID equal *value*

Table˜13: Match combinations for VLAN tags.

The field OXM_OF_IPV6_EXTHDR is a pseudo field that indicates the presence of various IPv6 extension headers in the packet header. The IPv6 extension header bits are combined together in the fields OXM_OF_IPV6_EXTHDR, and those bits can have the following values:

```
/* Bit definitions for IPv6 Extension Header pseudo-field. */
enum ofp_ipv6exthdr_flags {
    OFPIEH_NONEXT = 1 << 0,     /* "No next header" encountered. */
    OFPIEH_ESP    = 1 << 1,     /* Encrypted Sec Payload header present. */
    OFPIEH_AUTH   = 1 << 2,     /* Authentication header present. */
    OFPIEH_DEST   = 1 << 3,     /* 1 or 2 dest headers present. */
    OFPIEH_FRAG   = 1 << 4,     /* Fragment header present. */
    OFPIEH_ROUTER = 1 << 5,     /* Router header present. */
    OFPIEH_HOP    = 1 << 6,     /* Hop-by-hop header present. */
    OFPIEH_UNREP  = 1 << 7,     /* Unexpected repeats encountered. */
    OFPIEH_UNSEQ  = 1 << 8,     /* Unexpected sequencing encountered. */
};
```

- OFPIEH_HOP is set to 1 if a hop-by-hop IPv6 extension header is present as the first extension header in the packet.
- OFPIEH_ROUTER is set to 1 if a router IPv6 extension header is present.

- OFPIEH_FRAG is set to 1 if a fragmentation IPv6 extension header is present.
- OFPIEH_DEST is set to 1 if one or more Destination options IPv6 extension headers are present. It is normal to have either one or two of these in one IPv6 packet (see RFC 2460).
- OFPIEH_AUTH is set to 1 if an Authentication IPv6 extension header is present.
- OFPIEH_ESP is set to 1 if an Encrypted Security Payload IPv6 extension header is present.
- OFPIEH_NONEXT is set to 1 if a No Next Header IPv6 extension header is present.
- OFPIEH_UNSEQ is set to 1 if IPv6 extension headers were not in the order preferred (but not required) by RFC 2460.
- OFPIEH_UNREP is set to 1 if more than one of a given IPv6 extension header is unexpectedly encountered. (Two destination options headers may be expected and would not cause this bit to be set.)

7.2.3.8 Experimenter Flow Match Fields

Support for experimenter-specific flow match fields is optional. Experimenter-specific flow match fields may defined using the oxm_class=OFPXMC_EXPERIMENTER. The first four bytes of the OXM TLV's body contains the experimenter identifier, which takes the same form as in struct ofp_experimenter (see 7.5.4). Both oxm_field and the rest of the OXM TLV is experimenter-defined and does not need to be padded or aligned.

```
/* Header for OXM experimenter match fields. */
struct ofp_oxm_experimenter_header {
    uint32_t oxm_header;        /* oxm_class = OFPXMC_EXPERIMENTER */
    uint32_t experimenter;      /* Experimenter ID which takes the same
                                   form as in struct ofp_experimenter_header. */
};
OFP_ASSERT(sizeof(struct ofp_oxm_experimenter_header) == 8);
```

7.2.4 Flow Instruction Structures

Flow instructions associated with a flow table entry are executed when a flow matches the entry. The list of instructions that are currently defined are:

```
enum ofp_instruction_type {
    OFPIT_GOTO_TABLE = 1,       /* Setup the next table in the lookup
                                   pipeline */
    OFPIT_WRITE_METADATA = 2,   /* Setup the metadata field for use later in
                                   pipeline */
    OFPIT_WRITE_ACTIONS = 3,    /* Write the action(s) onto the datapath action
                                   set */
    OFPIT_APPLY_ACTIONS = 4,    /* Applies the action(s) immediately */
    OFPIT_CLEAR_ACTIONS = 5,    /* Clears all actions from the datapath
                                   action set */
    OFPIT_METER = 6,            /* Apply meter (rate limiter) */
```



```
    OFPIT_EXPERIMENTER = 0xFFFF  /* Experimenter instruction */
};
```

The instruction set is described in section 5.9. Flow tables may support a subset of instruction types. An instruction definition contains the instruction type, length, and any associated data:

```
/* Instruction header that is common to all instructions.  The length includes
 * the header and any padding used to make the instruction 64-bit aligned.
 * NB: The length of an instruction *must* always be a multiple of eight. */
struct ofp_instruction {
    uint16_t type;                /* Instruction type */
    uint16_t len;                 /* Length of this struct in bytes. */
};
OFP_ASSERT(sizeof(struct ofp_instruction) == 4);
```

The OFPIT_GOTO_TABLE instruction uses the following structure and fields:

```
/* Instruction structure for OFPIT_GOTO_TABLE */
struct ofp_instruction_goto_table {
    uint16_t type;                /* OFPIT_GOTO_TABLE */
    uint16_t len;                 /* Length of this struct in bytes. */
    uint8_t table_id;             /* Set next table in the lookup pipeline */
    uint8_t pad[3];               /* Pad to 64 bits. */
};
OFP_ASSERT(sizeof(struct ofp_instruction_goto_table) == 8);
```

table_id indicates the next table in the packet processing pipeline.

The OFPIT_WRITE_METADATA instruction uses the following structure and fields:

```
/* Instruction structure for OFPIT_WRITE_METADATA */
struct ofp_instruction_write_metadata {
    uint16_t type;                /* OFPIT_WRITE_METADATA */
    uint16_t len;                 /* Length of this struct in bytes. */
    uint8_t pad[4];               /* Align to 64-bits */
    uint64_t metadata;            /* Metadata value to write */
    uint64_t metadata_mask;       /* Metadata write bitmask */
};
OFP_ASSERT(sizeof(struct ofp_instruction_write_metadata) == 24);
```

Metadata for the next table lookup can be written using the metadata and the metadata_mask in order to set specific bits on the match field. If this instruction is not specified, the metadata is passed, unchanged.

The OFPIT_WRITE_ACTIONS, OFPIT_APPLY_ACTIONS, and OFPIT_CLEAR_ACTIONS instructions use the following structure and fields:


```
/* Instruction structure for OFPIT_WRITE/APPLY/CLEAR_ACTIONS */
struct ofp_instruction_actions {
    uint16_t type;              /* One of OFPIT_*_ACTIONS */
    uint16_t len;               /* Length of this struct in bytes. */
    uint8_t pad[4];             /* Align to 64-bits */
    struct ofp_action_header actions[0];  /* 0 or more actions associated with
                                             OFPIT_WRITE_ACTIONS and
                                             OFPIT_APPLY_ACTIONS */
};
OFP_ASSERT(sizeof(struct ofp_instruction_actions) == 8);
```

For the Apply-Actions instruction, the `actions` field is treated as a list and the actions are applied to the packet *in-order*. For the Write-Actions instruction, the `actions` field is treated as a set and the actions are merged into the current action set.

For the Clear-Actions instruction, the structure does not contain any actions.

The `OFPIT_METER` instruction uses the following structure and fields:

```
/* Instruction structure for OFPIT_METER */
struct ofp_instruction_meter {
    uint16_t type;                /* OFPIT_METER */
    uint16_t len;                 /* Length is 8. */
    uint32_t meter_id;            /* Meter instance. */
};
OFP_ASSERT(sizeof(struct ofp_instruction_meter) == 8);
```

`meter_id` indicates which meter to apply on the packet.

An `OFPIT_EXPERIMENTER` instruction uses the following structure and fields:

```
/* Instruction structure for experimental instructions */
struct ofp_instruction_experimenter {
    uint16_t type; /* OFPIT_EXPERIMENTER */
    uint16_t len;               /* Length of this struct in bytes */
    uint32_t experimenter;      /* Experimenter ID which takes the same form
                                   as in struct ofp_experimenter_header. */
    /* Experimenter-defined arbitrary additional data. */
};
OFP_ASSERT(sizeof(struct ofp_instruction_experimenter) == 8);
```

The `experimenter` field is the Experimenter ID, which takes the same form as in struct `ofp_experimenter` (see 7.5.4).

7.2.5 Action Structures

A number of actions may be associated with flow entries, groups or packets. The currently defined action types are:


```
enum ofp_action_type {
    OFPAT_OUTPUT       = 0,  /* Output to switch port. */
    OFPAT_COPY_TTL_OUT = 11, /* Copy TTL "outwards" -- from next-to-outermost
                                to outermost */
    OFPAT_COPY_TTL_IN  = 12, /* Copy TTL "inwards" -- from outermost to
                                next-to-outermost */
    OFPAT_SET_MPLS_TTL = 15, /* MPLS TTL */
    OFPAT_DEC_MPLS_TTL = 16, /* Decrement MPLS TTL */

    OFPAT_PUSH_VLAN    = 17, /* Push a new VLAN tag */
    OFPAT_POP_VLAN     = 18, /* Pop the outer VLAN tag */
    OFPAT_PUSH_MPLS    = 19, /* Push a new MPLS tag */
    OFPAT_POP_MPLS     = 20, /* Pop the outer MPLS tag */
    OFPAT_SET_QUEUE    = 21, /* Set queue id when outputting to a port */
    OFPAT_GROUP        = 22, /* Apply group. */
    OFPAT_SET_NW_TTL   = 23, /* IP TTL. */
    OFPAT_DEC_NW_TTL   = 24, /* Decrement IP TTL. */
    OFPAT_SET_FIELD    = 25, /* Set a header field using OXM TLV format. */
    OFPAT_PUSH_PBB     = 26, /* Push a new PBB service tag (I-TAG) */
    OFPAT_POP_PBB      = 27, /* Pop the outer PBB service tag (I-TAG) */
    OFPAT_EXPERIMENTER = 0xffff
};
```

Output, group, and set-queue actions are described in Section 5.12, tag push/pop actions are described in Table 6, and Set-Field actions are described from their OXM types in Table 12. An action definition contains the action type, length, and any associated data:

```
/* Action header that is common to all actions.  The length includes the
 * header and any padding used to make the action 64-bit aligned.
 * NB: The length of an action *must* always be a multiple of eight. */
struct ofp_action_header {
    uint16_t type;                  /* One of OFPAT_*. */
    uint16_t len;                   /* Length of action, including this
                                       header.  This is the length of action,
                                       including any padding to make it
                                       64-bit aligned. */
    uint8_t pad[4];
};
OFP_ASSERT(sizeof(struct ofp_action_header) == 8);
```

An *Output* action uses the following structure and fields:

```
/* Action structure for OFPAT_OUTPUT, which sends packets out 'port'.
 * When the 'port' is the OFPP_CONTROLLER, 'max_len' indicates the max
 * number of bytes to send.  A 'max_len' of zero means no bytes of the
 * packet should be sent. A 'max_len' of OFPCML_NO_BUFFER means that
 * the packet is not buffered and the complete packet is to be sent to
 * the controller. */
struct ofp_action_output {
    uint16_t type;                  /* OFPAT_OUTPUT. */
    uint16_t len;                   /* Length is 16. */
    uint32_t port;                  /* Output port. */
    uint16_t max_len;               /* Max length to send to controller. */
    uint8_t pad[6];                 /* Pad to 64 bits. */
```



```
};
OFP_ASSERT(sizeof(struct ofp_action_output) == 16);
```

The `port` specifies the port through which the packet should be sent. The `max_len` indicates the maximum amount of data from a packet that should be sent when the port is `OFPP_CONTROLLER`. If `max_len` is zero, the switch must send zero bytes of the packet. A `max_len` of `OFPCML_NO_BUFFER` means that the complete packet should be sent, and it should not be buffered.

```
enum ofp_controller_max_len {
OFPCML_MAX       = 0xffe5, /* maximum max_len value which can be used
                              to request a specific byte length. */
OFPCML_NO_BUFFER = 0xffff  /* indicates that no buffering should be
                              applied and the whole packet is to be
                              sent to the controller. */
};
```

A *Group* action uses the following structure and fields:

```
/* Action structure for OFPAT_GROUP. */
struct ofp_action_group {
    uint16_t type;                  /* OFPAT_GROUP. */
    uint16_t len;                   /* Length is 8. */
    uint32_t group_id;              /* Group identifier. */
};
OFP_ASSERT(sizeof(struct ofp_action_group) == 8);
```

The `group_id` indicates the group used to process this packet. The set of buckets to apply depends on the group type.

The *Set-Queue* action sets the queue id that will be used to map a flow entry to an already-configured queue on a port, regardless of the ToS and VLAN PCP bits. The packet should not change as a result of a Set-Queue action. If the switch needs to set the ToS/PCP bits for internal handling, the original values should be restored before sending the packet out.

A switch may support only queues that are tied to specific PCP/ToS bits. In that case, we cannot map an arbitrary flow entry to a specific queue, therefore the Set-Queue action is not supported. The user can still use these queues and map flow entries to them by setting the relevant fields (ToS, VLAN PCP).

A *Set Queue* action uses the following structure and fields:

```
/* OFPAT_SET_QUEUE action struct: send packets to given queue on port. */
struct ofp_action_set_queue {
    uint16_t type;            /* OFPAT_SET_QUEUE. */
    uint16_t len;             /* Len is 8. */
    uint32_t queue_id;        /* Queue id for the packets. */
};
OFP_ASSERT(sizeof(struct ofp_action_set_queue) == 8);
```

A *Set MPLS TTL* action uses the following structure and fields:


```
/* Action structure for OFPAT_SET_MPLS_TTL. */
struct ofp_action_mpls_ttl {
    uint16_t type;                  /* OFPAT_SET_MPLS_TTL. */
    uint16_t len;                   /* Length is 8. */
    uint8_t mpls_ttl;               /* MPLS TTL */
    uint8_t pad[3];
};
OFP_ASSERT(sizeof(struct ofp_action_mpls_ttl) == 8);
```

The `mpls_ttl` field is the MPLS TTL to set.

A *Decrement MPLS TTL* action takes no arguments and consists only of a generic `ofp_action_header`. The action decrements the MPLS TTL.

A *Set IPv4 TTL* action uses the following structure and fields:

```
/* Action structure for OFPAT_SET_NW_TTL. */
struct ofp_action_nw_ttl {
    uint16_t type;                  /* OFPAT_SET_NW_TTL. */
    uint16_t len;                   /* Length is 8. */
    uint8_t nw_ttl;                 /* IP TTL */
    uint8_t pad[3];
};
OFP_ASSERT(sizeof(struct ofp_action_nw_ttl) == 8);
```

The `nw_ttl` field is the TTL address to set in the IP header.

An *Decrement IPv4 TTL* action takes no arguments and consists only of a generic `ofp_action_header`. This action decrement the TTL in the IP header if one is present.

A *Copy TTL outwards* action takes no arguments and consists only of a generic `ofp_action_header`. The action copies the TTL from the next-to-outermost header with TTL to the outermost header with TTL.

A *Copy TTL inwards* action takes no arguments and consists only of a generic `ofp_action_header`. The action copies the TTL from the outermost header with TTL to the next-to-outermost header with TTL.

A *Push VLAN header* action, *Push MPLS header* and *Push PBB header* action use the following structure and fields:

```
/* Action structure for OFPAT_PUSH_VLAN/MPLS/PBB. */
struct ofp_action_push {
    uint16_t type;                  /* OFPAT_PUSH_VLAN/MPLS/PBB. */
    uint16_t len;                   /* Length is 8. */
    uint16_t ethertype;             /* Ethertype */
    uint8_t pad[2];
};
OFP_ASSERT(sizeof(struct ofp_action_push) == 8);
```


The `ethertype` indicates the Ethertype of the new tag. It is used when pushing a new VLAN tag, new MPLS header or PBB service header.

The *Push PBB header* action logically pushes a new PBB service instance header onto the packet (I-TAG TCI), and copy the original Ethernet addresses of the packet into the customer addresses (C-DA and C-SA) of the tag. The customer addresses of the I-TAG are in the location of the original Ethernet addresses of the encapsulated packet, therefore this operations can be seen as adding both the backbone MAC-in-MAC header and the I-SID field to the front of the packet. The *Push PBB header* action does not add a backbone VLAN header (B-TAG) to the packet, it can be added via the *Push VLAN header* action after the push PBB header operation. After this operation, regular set-field actions can be used to modify the outer Ethernet addresses (B-DA and B-SA).

A *Pop VLAN header* action takes no arguments and consists only of a generic `ofp_action_header`. The action pops the outermost VLAN tag from the packet.

A *Pop PBB header* action takes no arguments and consists only of a generic `ofp_action_header`. The action logically pops the outer-most PBB service instance header from the packet (I-TAG TCI) and copy the customer addresses (C-DA and C-SA) in the Ethernet addresses of the packet. This operation can be seen as removing the backbone MAC-in-MAC header and the I-SID field from the front of the packet. The *Pop PBB header* action does not remove the backbone VLAN header (B-TAG) from the packet, it should be removed prior to this operation via the *Pop VLAN header* action.

A *Pop MPLS header* action uses the following structure and fields:

```
/* Action structure for OFPAT_POP_MPLS. */
struct ofp_action_pop_mpls {
    uint16_t type;                  /* OFPAT_POP_MPLS. */
    uint16_t len;                   /* Length is 8. */
    uint16_t ethertype;             /* Ethertype */
    uint8_t pad[2];
};
OFP_ASSERT(sizeof(struct ofp_action_pop_mpls) == 8);
```

The `ethertype` indicates the Ethertype of the MPLS payload. The `ethertype` is used as the Ethertype for the resulting packet regardless of whether the "bottom of stack (BoS)" bit was set in the removed MPLS shim. It is recommended that flow entries using this action match both the MPLS label and the MPLS BoS fields to avoid applying the wrong Ethertype to the MPLS payload.

The MPLS specification does not allow to set arbitrary Ethertype to MPLS payload when BoS is not equal to 1, and the controller is responsible in complying with this requirement and only set 0x8847 or 0x8848 as the Ethertype for those MPLS payloads. The switch can optionally enforce this MPLS requirement: in this case the switch should reject any flow entry matching a wildcard BoS and any flow entry matching BoS to 0 with the wrong `ethertype` in the Pop MPLS header action, and in both cases should return an `ofp_error_msg` with `OFPET_BAD_ACTION` type and `OFPBAC_MATCH_INCONSISTENT` code.

Set Field actions uses the following structure and fields:

```
/* Action structure for OFPAT_SET_FIELD. */
struct ofp_action_set_field {
    uint16_t type;                  /* OFPAT_SET_FIELD. */
```



```
    uint16_t len;                   /* Length is padded to 64 bits. */
    /* Followed by:
     *   - Exactly oxm_len bytes containing a single OXM TLV, then
     *   - Exactly ((oxm_len + 4) + 7)/8*8 - (oxm_len + 4) (between 0 and 7)
     *     bytes of all-zero bytes
     */
    uint8_t field[4];               /* OXM TLV - Make compiler happy */
};
OFP_ASSERT(sizeof(struct ofp_action_set_field) == 8);
```

The `field` contains a header field described using a single OXM TLV structure (see 7.2.3). Set-Field actions are defined by `oxm_type`, the type of the OXM TLV, and modify the corresponding header field in the packet with the value of `oxm_value`, the payload of the OXM TLV. The value of `oxm_hasmask` must be zero and no `oxm_mask` is included. The match of the flow entry must contain the OXM prerequisite curresponding to the field to be set (see 7.2.3.6), otherwise an error must be generated (see 6.4).

The type of a set-field action can be any valid OXM header type, the list of possible OXM types are described in Section 7.2.3.7 and Table 12. Set-Field actions for OXM types `OFPXMT_OFB_IN_PORT`, `OXM_OF_IN_PHY_PORT` and `OFPXMT_OFB_METADATA` are not supported, because those are not header fields. The Set-Field action overwrite the header field specified by the OXM type, and perform the necessary CRC recalculation based on the header field. The OXM fields refers to the outermost-possible occurence in the header, unless the field type explictely specifies otherwise, and therefore in general the set-field actions applies to the outermost-possible header (e.g. a "Set VLAN ID" set-field action always sets the ID of the outermost VLAN tag).

An *Experimenter* action uses the following structure and fields:

```
/* Action header for OFPAT_EXPERIMENTER.
 * The rest of the body is experimenter-defined. */
struct ofp_action_experimenter_header {
    uint16_t type;                  /* OFPAT_EXPERIMENTER. */
    uint16_t len;                   /* Length is a multiple of 8. */
    uint32_t experimenter;          /* Experimenter ID which takes the same
                                       form as in struct
                                       ofp_experimenter_header. */
};
OFP_ASSERT(sizeof(struct ofp_action_experimenter_header) == 8);
```

The `experimenter` field is the Experimenter ID, which takes the same form as in struct `ofp_experimenter` (see 7.5.4).

7.3 Controller-to-Switch Messages

7.3.1 Handshake

The `OFPT_FEATURES_REQUEST` message is used by the controller to identify the switch and read its basic capabilities. Upon session establishment (see 6.3.1), the controller should send an `OFPT_FEATURES_REQUEST` message. This message does not contain a body beyond the OpenFlow header. The switch must respond with an `OFPT_FEATURES_REPLY` message:


```
/* Switch features. */
struct ofp_switch_features {
    struct ofp_header header;
    uint64_t datapath_id;   /* Datapath unique ID.  The lower 48-bits are for
                               a MAC address, while the upper 16-bits are
                               implementer-defined. */

    uint32_t n_buffers;     /* Max packets buffered at once. */

    uint8_t n_tables;       /* Number of tables supported by datapath. */
    uint8_t auxiliary_id;   /* Identify auxiliary connections */
    uint8_t pad[2];         /* Align to 64-bits. */

    /* Features. */
    uint32_t capabilities;  /* Bitmap of support "ofp_capabilities". */
    uint32_t reserved;
};
OFP_ASSERT(sizeof(struct ofp_switch_features) == 32);
```

The `datapath_id` field uniquely identifies a datapath. The lower 48 bits are intended for the switch MAC address, while the top 16 bits are up to the implementer. An example use of the top 16 bits would be a VLAN ID to distinguish multiple virtual switch instances on a single physical switch. This field should be treated as an opaque bit string by controllers.

The `n_buffers` field specifies the maximum number of packets the switch can buffer when sending packets to the controller using *packet-in* messages (see 6.1.2).

The `n_tables` field describes the number of tables supported by the switch, each of which can have a different set of supported match fields, actions and number of entries. When the controller and switch first communicate, the controller will find out how many tables the switch supports from the Features Reply. If it wishes to understand the size, types, and order in which tables are consulted, the controller sends a `OFPMP_TABLE_FEATURES` multipart request (see 7.3.5.5). A switch must return these tables in the order the packets traverse the tables.

The `auxiliary_id` field identify the type of connection from the switch to the controller, the main connection has this field set to zero, an auxiliary connection has this field set to a non-zero value (see 6.3.5).

The `capabilities` field uses a combination of the following flags:

```
/* Capabilities supported by the datapath. */
enum ofp_capabilities {
    OFPC_FLOW_STATS     = 1 << 0,  /* Flow statistics. */
    OFPC_TABLE_STATS    = 1 << 1,  /* Table statistics. */
    OFPC_PORT_STATS     = 1 << 2,  /* Port statistics. */
    OFPC_GROUP_STATS    = 1 << 3,  /* Group statistics. */
    OFPC_IP_REASM       = 1 << 5,  /* Can reassemble IP fragments. */
    OFPC_QUEUE_STATS    = 1 << 6,  /* Queue statistics. */
    OFPC_PORT_BLOCKED   = 1 << 8   /* Switch will block looping ports. */
};
```

The `OFPC_PORT_BLOCKED` bit indicates that a switch protocol outside of OpenFlow, such as 802.1D Spanning Tree, will detect topology loops and block ports to prevent packet loops. If this bit is not set, in most cases the controller should implement a mechanism to prevent packet loops.

7.3.2 Switch Configuration

The controller is able to set and query configuration parameters in the switch with the OFPT_SET_CONFIG and OFPT_GET_CONFIG_REQUEST messages, respectively. The switch responds to a configuration request with an OFPT_GET_CONFIG_REPLY message; it does not reply to a request to set the configuration.

There is no body for OFPT_GET_CONFIG_REQUEST beyond the OpenFlow header. The OFPT_SET_CONFIG and OFPT_GET_CONFIG_REPLY use the following:

```
/* Switch configuration. */
struct ofp_switch_config {
    struct ofp_header header;
    uint16_t flags;             /* Bitmap of OFPC_* flags. */
    uint16_t miss_send_len;     /* Max bytes of packet that datapath
                                   should send to the controller. See
                                   ofp_controller_max_len for valid values.
                                   */
};
OFP_ASSERT(sizeof(struct ofp_switch_config) == 12);
```

The configuration flags include the following:

```
enum ofp_config_flags {
    /* Handling of IP fragments. */
    OFPC_FRAG_NORMAL   = 0,       /* No special handling for fragments. */
    OFPC_FRAG_DROP     = 1 << 0,  /* Drop fragments. */
    OFPC_FRAG_REASM    = 1 << 1,  /* Reassemble (only if OFPC_IP_REASM set). */
    OFPC_FRAG_MASK     = 3,
};
```

The OFPC_FRAG_* flags indicate whether IP fragments should be treated normally, dropped, or reassembled. "Normal" handling of fragments means that an attempt should be made to pass the fragments through the OpenFlow tables. If any field is not present (e.g., the TCP/UDP ports didn't fit), then the packet should not match any entry that has that field set.

The miss_send_len field defines the number of bytes of each packet sent to the controller by the OpenFlow pipeline when not using an output action to the OFPP_CONTROLLER logical port, for example sending packets with invalid TTL if this message reason is enabled. If this field equals 0, the switch must send zero bytes of the packet in the ofp_packet_in message. If the value is set to OFPCML_NO_BUFFER the complete packet must be included in the message, and should not be buffered.

7.3.3 Flow Table Configuration

Flow tables are numbered from 0 and can take any number until OFPTT_MAX. OFPTT_ALL is a reserved value.


```
/* Table numbering. Tables can use any number up to OFPT_MAX. */
enum ofp_table {
    /* Last usable table number. */
    OFPTT_MAX        = 0xfe,

    /* Fake tables. */
    OFPTT_ALL        = 0xff   /* Wildcard table used for table config,
                                 flow stats and flow deletes. */
};
```

The controller can configure and query table state in the switch with the OFP_TABLE_MOD and OFPMP_TABLE requests, respectively. The switch responds to a table multipart request with a OFPT_MULTIPART_REPLY message. The OFP_TABLE_MOD use the following structure and fields:

```
/* Configure/Modify behavior of a flow table */
struct ofp_table_mod {
    struct ofp_header header;
    uint8_t table_id;       /* ID of the table, OFPTT_ALL indicates all tables */
    uint8_t pad[3];         /* Pad to 32 bits */
    uint32_t config;        /* Bitmap of OFPTC_* flags */
};
OFP_ASSERT(sizeof(struct ofp_table_mod) == 16);
```

The table_id chooses the table to which the configuration change should be applied. If the table_id is OFPTT_ALL, the configuration is applied to all tables in the switch.

The config field is a bitmap that is provided for backward compatibility with earlier version of the specification, it is reserved for future use. There is no flags that are currently defined for that field. The following values are defined for that field :

```
/* Flags to configure the table. Reserved for future use. */
enum ofp_table_config {
    OFPTC_DEPRECATED_MASK       = 3,  /* Deprecated bits */
};
```

7.3.4 Modify State Messages

7.3.4.1 Modify Flow Entry Message

Modifications to a flow table from the controller are done with the OFPT_FLOW_MOD message:

```
/* Flow setup and teardown (controller -> datapath). */
struct ofp_flow_mod {
    struct ofp_header header;
    uint64_t cookie;            /* Opaque controller-issued identifier. */
    uint64_t cookie_mask;       /* Mask used to restrict the cookie bits
                                   that must match when the command is
                                   OFPFC_MODIFY* or OFPFC_DELETE*. A value
                                   of 0 indicates no restriction. */

    /* Flow actions. */
```



```
    uint8_t table_id;            /* ID of the table to put the flow in.
                                    For OFPFC_DELETE_* commands, OFPTT_ALL
                                    can also be used to delete matching
                                    flows from all tables. */
    uint8_t command;             /* One of OFPFC_*. */
    uint16_t idle_timeout;       /* Idle time before discarding (seconds). */
    uint16_t hard_timeout;       /* Max time before discarding (seconds). */
    uint16_t priority;           /* Priority level of flow entry. */
    uint32_t buffer_id;          /* Buffered packet to apply to, or
                                    OFP_NO_BUFFER.
                                    Not meaningful for OFPFC_DELETE*. */
    uint32_t out_port;           /* For OFPFC_DELETE* commands, require
                                    matching entries to include this as an
                                    output port.  A value of OFPP_ANY
                                    indicates no restriction. */
    uint32_t out_group;          /* For OFPFC_DELETE* commands, require
                                    matching entries to include this as an
                                    output group.  A value of OFPG_ANY
                                    indicates no restriction. */
    uint16_t flags;              /* Bitmap of OFPFF_* flags. */
    uint8_t pad[2];
    struct ofp_match match;      /* Fields to match. Variable size. */
    /* The variable size and padded match is always followed by instructions. */
    //struct ofp_instruction instructions[0]; /* Instruction set - 0 or more.
                                                 The length of the instruction
                                                 set is inferred from the
                                                 length field in the header. */
};
OFP_ASSERT(sizeof(struct ofp_flow_mod) == 56);
```

The `cookie` field is an opaque data value chosen by the controller. This value appears in flow removed messages and flow statistics, and can also be used to filter flow statistics, flow modification and flow deletion (see 6.4). It is not used by the packet processing pipeline, and thus does not need to reside in hardware. The value -1 (0xffffffffffffffff) is reserved and must not be used. When a flow entry is inserted in a table through an `OFPFC_ADD` message, its `cookie` field is set to the provided value. When a flow entry is modified (`OFPFC_MODIFY` or `OFPFC_MODIFY_STRICT` messages), its `cookie` field is unchanged.

If the `cookie_mask` field is non-zero, it is used with the `cookie` field to restrict flow matching while modifying or deleting flow entries. This field is ignored by `OFPFC_ADD` messages. The `cookie_mask` field's behavior is explained in Section 6.4.

The `table_id` field specifies the table into which the flow entry should be inserted, modified or deleted. Table 0 signifies the first table in the pipeline. The use of `OFPTT_ALL` is only valid for delete requests.

The `command` field must be one of the following:

```
enum ofp_flow_mod_command {
    OFPFC_ADD           = 0, /* New flow. */
    OFPFC_MODIFY        = 1, /* Modify all matching flows. */
    OFPFC_MODIFY_STRICT = 2, /* Modify entry strictly matching wildcards and
                                priority. */
    OFPFC_DELETE        = 3, /* Delete all matching flows. */
    OFPFC_DELETE_STRICT = 4, /* Delete entry strictly matching wildcards and
                                priority. */
};
```


The differences between OFPFC_MODIFY and OFPFC_MODIFY_STRICT are explained in Section 6.4 and differences between OFPFC_DELETE and OFPFC_DELETE_STRICT are explained in Section 6.4.

The idle_timeout and hard_timeout fields control how quickly flow entries expire (see 5.5). When a flow entry is inserted in a table, its idle_timeout and hard_timeout fields are set with the values from the message. When a flow entry is modified (OFPFC_MODIFY or OFPFC_MODIFY_STRICT messages), the idle_timeout and hard_timeout fields are ignored.

If the idle_timeout is set and the hard_timeout is zero, the entry must expire after idle_timeout seconds with no received traffic. If the idle_timeout is zero and the hard_timeout is set, the entry must expire in hard_timeout seconds regardless of whether or not packets are hitting the entry. If both idle_timeout and hard_timeout are set, the flow entry will timeout after idle_timeout seconds with no traffic, or hard_timeout seconds, whichever comes first. If both idle_timeout and hard_timeout are zero, the entry is considered permanent and will never time out. It can still be removed with a flow_mod message of type OFPFC_DELETE.

The priority indicates priority within the specified flow table table. Higher numbers indicate higher priorities. This field is used only for OFPFC_ADD messages when matching and adding flow entries, and for OFPFC_MODIFY_STRICT or OFPFC_DELETE_STRICT messages when matching flow entries.

The buffer_id refers to a packet buffered at the switch and sent to the controller by a *packet-in* message. If no buffered packet is associated with the flow mod, it must be set to OFP_NO_BUFFER. A flow mod that includes a valid buffer_id removes the corresponding packet from the buffer and processes it through the entire OpenFlow pipeline after the flow is inserted, starting at the first flow table. This is effectively equivalent to sending a two-message sequence of a flow mod and a packet-out forwarding to the OFPP_TABLE logical port (see 7.3.7), with the requirement that the switch must fully process the flow mod before the packet out. These semantics apply regardless of the table to which the flow mod refers, or the instructions contained in the flow mod. This field is ignored by OFPFC_DELETE and OFPFC_DELETE_STRICT flow mod messages.

The out_port and out_group fields optionally filter the scope of OFPFC_DELETE and OFPFC_DELETE_STRICT messages by output port and group. If either out_port or out_group contains a value other than OFPP_ANY or OFPG_ANY respectively, it introduces a constraint when matching. This constraint is that the flow entry must contain an output action directed at that port or group. Other constraints such as ofp_match structs and priorities are still used; this is purely an *additional* constraint. Note that to disable output filtering, both out_port and out_group must be set to OFPP_ANY and OFPG_ANY respectively. These fields are ignored by OFPFC_ADD, OFPFC_MODIFY or OFPFC_MODIFY_STRICT messages.

The flags field may include a combination of the following flags:

```
enum ofp_flow_mod_flags {
    OFPFF_SEND_FLOW_REM = 1 << 0,  /* Send flow removed message when flow
                                    * expires or is deleted. */
    OFPFF_CHECK_OVERLAP = 1 << 1,  /* Check for overlapping entries first. */
    OFPFF_RESET_COUNTS  = 1 << 2,  /* Reset flow packet and byte counts. */
    OFPFF_NO_PKT_COUNTS = 1 << 3,  /* Don't keep track of packet count. */
    OFPFF_NO_BYT_COUNTS = 1 << 4,  /* Don't keep track of byte count. */
};
```


When the OFPFF_SEND_FLOW_REM flag is set, the switch must send a flow removed message when the flow entry expires or is deleted.

When the OFPFF_CHECK_OVERLAP flag is set, the switch must check that there are no conflicting entries with the same priority prior to inserting it in the flow table. If there is one, the flow mod fails and an error message is returned (see 6.4).

When the OFPFF_NO_PKT_COUNTS flag is set, the switch does not need to keep track of the flow packet count. When the OFPFF_NO_BYT_COUNTS flag is set, the switch does not need to keep track of the flow byte count. Setting those flags may decrease the processing load on some OpenFlow switches, however those counters may not be available in flow statistics and flow removed messages for this flow entry. A switch is not required to honor those flags and may keep track of a flow count and return it despite the corresponding flag being set. If a switch does not keep track of a flow count, the corresponding counter is not available and must be set to the maximum field value (see 5.8).

When a flow entry is inserted in a table, its **flags** field is set with the values from the message. When a flow entry is matched and modified (OFPFC_MODIFY or OFPFC_MODIFY_STRICT messages), the **flags** field is ignored.

The **instructions** field contains the instruction set for the flow entry when adding or modifying entries. If the instruction set is not valid or supported, the switch must generate an error (see 6.4).

7.3.4.2 Modify Group Entry Message

Modifications to the group table from the controller are done with the OFPT_GROUP_MOD message:

```
/* Group setup and teardown (controller -> datapath). */
struct ofp_group_mod {
    struct ofp_header header;
    uint16_t command;             /* One of OFPGC_*. */
    uint8_t type;                 /* One of OFPGT_*. */
    uint8_t pad;                  /* Pad to 64 bits. */
    uint32_t group_id;            /* Group identifier. */
    struct ofp_bucket buckets[0]; /* The length of the bucket array is inferred
                                     from the length field in the header. */
};
OFP_ASSERT(sizeof(struct ofp_group_mod) == 16);
```

The semantics of the type and group fields are explained in Section 6.5.

The **command** field must be one of the following:

```
/* Group commands */
enum ofp_group_mod_command {
    OFPGC_ADD    = 0,       /* New group. */
    OFPGC_MODIFY = 1,       /* Modify all matching groups. */
    OFPGC_DELETE = 2,       /* Delete all matching groups. */
};
```

The **type** field must be one of the following:


```
/* Group types.  Values in the range [128, 255] are reserved for experimental
 * use. */
enum ofp_group_type {
    OFPGT_ALL      = 0, /* All (multicast/broadcast) group.  */
    OFPGT_SELECT   = 1, /* Select group. */
    OFPGT_INDIRECT = 2, /* Indirect group. */
    OFPGT_FF       = 3, /* Fast failover group. */
};
```

The `group_id` field uniquely identifies a group within a switch. The following special group identifiers are defined:

```
/* Group numbering. Groups can use any number up to OFPG_MAX. */
enum ofp_group {
    /* Last usable group number. */
    OFPG_MAX        = 0xffffff00,

    /* Fake groups. */
    OFPG_ALL        = 0xfffffffc,  /* Represents all groups for group delete
                                      commands. */
    OFPG_ANY        = 0xffffffff   /* Wildcard group used only for flow stats
                                      requests. Selects all flows regardless of
                                      group (including flows with no group).
                                      */
};
```

The `buckets` field is an array of buckets. For *Indirect* group, the arrays must contain exactly one bucket (see 5.6.1), other group types may have multiple buckets in the array. For *Fast Failover* group, the bucket order does define the bucket priorities (see 5.6.1), and the bucket order can be changed by modifying the group (for example using a `OFPT_GROUP_MOD` message with command `OFPGC_MODIFY`).

Buckets in the array use the following struct:

```
/* Bucket for use in groups. */
struct ofp_bucket {
    uint16_t len;                   /* Length the bucket in bytes, including
                                       this header and any padding to make it
                                       64-bit aligned. */
    uint16_t weight;                /* Relative weight of bucket.  Only
                                       defined for select groups. */
    uint32_t watch_port;            /* Port whose state affects whether this
                                       bucket is live.  Only required for fast
                                       failover groups. */
    uint32_t watch_group;           /* Group whose state affects whether this
                                       bucket is live.  Only required for fast
                                       failover groups. */
    uint8_t pad[4];
    struct ofp_action_header actions[0]; /* 0 or more actions associated with
                                            the bucket - The action list length
                                            is inferred from the length
                                            of the bucket. */
};
OFP_ASSERT(sizeof(struct ofp_bucket) == 16);
```


The `weight` field is only defined for select groups, and its support is optional. The bucket's share of the traffic processed by the group is defined by the individual bucket's weight divided by the sum of the bucket weights in the group. When a port goes down, the change in traffic distribution is undefined. The precision by which a switch's packet distribution should match bucket weights is undefined.

The `watch_port` and `watch_group` fields are only required for fast failover groups, and may be optionally implemented for other group types. These fields indicate the port and/or group whose liveness controls whether this bucket is a candidate for forwarding. For fast failover groups, the first bucket defined is the highest-priority bucket, and only the highest-priority live bucket is used (see 5.6.1).

The `actions` field is the action set associated with the bucket. When the bucket is selected for a packet, its action set is applied to the packet (see 5.10).

7.3.4.3 Port Modification Message

The controller uses the `OFPT_PORT_MOD` message to modify the behavior of the port:

```
/* Modify behavior of the physical port */
struct ofp_port_mod {
    struct ofp_header header;
    uint32_t port_no;
    uint8_t pad[4];
    uint8_t hw_addr[OFP_ETH_ALEN]; /* The hardware address is not
                                      configurable.  This is used to
                                      sanity-check the request, so it must
                                      be the same as returned in an
                                      ofp_port struct. */
    uint8_t pad2[2];        /* Pad to 64 bits. */
    uint32_t config;        /* Bitmap of OFPPC_* flags. */
    uint32_t mask;          /* Bitmap of OFPPC_* flags to be changed. */

    uint32_t advertise;     /* Bitmap of OFPPF_*.  Zero all bits to prevent
                               any action taking place. */
    uint8_t pad3[4];        /* Pad to 64 bits. */
};
OFP_ASSERT(sizeof(struct ofp_port_mod) == 40);
```

The `mask` field is used to select bits in the `config` field to change. The `advertise` field has no mask; all port features change together.

7.3.4.4 Meter Modification Message

Modifications to a meter from the controller are done with the `OFPT_METER_MOD` message:

```
/* Meter configuration. OFPT_METER_MOD. */
struct ofp_meter_mod {
    struct ofp_header header;
    uint16_t            command;        /* One of OFPMC_*. */
    uint16_t            flags;          /* Bitmap of OFPMF_* flags. */
    uint32_t            meter_id;       /* Meter instance. */
```



```
    struct ofp_meter_band_header bands[0]; /* The band list length is
                                            inferred from the length field
                                            in the header. */
};
OFP_ASSERT(sizeof(struct ofp_meter_mod) == 16);
```

The `meter_id` field uniquely identifies a meter within a switch. Meter are defined starting with `meter_id`=1 up to the maximum number of meters that the switch can support. The OpenFlow protocol also defines some additional virtual meters that can not be associated with flows:

```
/* Meter numbering. Flow meters can use any number up to OFPM_MAX. */
enum ofp_meter {
    /* Last usable meter. */
    OFPM_MAX        = 0xffff0000,

    /* Virtual meters. */
    OFPM_SLOWPATH   = 0xfffffffd,  /* Meter for slow datapath. */
    OFPM_CONTROLLER = 0xfffffffe,  /* Meter for controller connection. */
    OFPM_ALL        = 0xffffffff,  /* Represents all meters for stat requests
                                      commands. */
};
```

Virtual meters are provided to support existing implementations of OpenFlow. New implementations are encouraged to use regular per-flow meters or priority queues instead.

- `OFPM_CONTROLLER`: Virtual meter controlling all packets sent to the controller via *Packet-in* messages, either using the **CONTROLLER** reserved port or in other processing (see 6.1.2). Can be used to limit the amount of traffic sent to the controller.
- `OFPM_SLOWPATH`: Virtual meter controlling all packets processed by the slow datapath of the switch. Many switch implementations have a fast and slow datapath, for example a hardware switch may have a slow software datapath, or a software switch may have a slow userspace datapath.

The `command` field must be one of the following:

```
/* Meter commands */
enum ofp_meter_mod_command {
    OFPMC_ADD,              /* New meter. */
    OFPMC_MODIFY,           /* Modify specified meter. */
    OFPMC_DELETE,           /* Delete specified meter. */
};
```

The `flags` field may include a combination of following flags:

```
/* Meter configuration flags */
enum ofp_meter_flags {
    OFPMF_KBPS    = 1 << 0,     /* Rate value in kb/s (kilo-bit per second). */
    OFPMF_PKTPS   = 1 << 1,     /* Rate value in packet/sec. */
    OFPMF_BURST   = 1 << 2,     /* Do burst size. */
    OFPMF_STATS   = 1 << 3,     /* Collect statistics. */
};
```


The `bands` field is a list of rate bands. It can contain any number of bands, and each band type can be repeated when it make sense. Only a single band is used at a time, if the current rate of packets exceed the rate of multiple bands, the band with the highest configured rate is used.

All the rate bands are defined using the same common header:

```
/* Common header for all meter bands */
struct ofp_meter_band_header {
    uint16_t        type;    /* One of OFPMBT_*. */
    uint16_t        len;     /* Length in bytes of this band. */
    uint32_t        rate;    /* Rate for this band. */
    uint32_t        burst_size; /* Size of bursts. */
};
OFP_ASSERT(sizeof(struct ofp_meter_band_header) == 12);
```

The `rate` field indicates the rate value above which the corresponding band may apply to packets (see 5.7.1). The rate value is in kilobit per seconds, unless the `flags` field includes `OFPMF_PKTPS`, in which case the rate is in packets per seconds.

The `burst_size` field is used only if the `flags` field includes `OFPMF_BURST`. It defines the granularity of the meter, for all packet or byte burst which length is greater than burst value, the meter rate will always be strictly enforced. The burst value is in kilobits, unless the `flags` field includes `OFPMF_PKTPS`, in which case the burst value is in packets.

The `type` field must be one of the following:

```
/* Meter band types */
enum ofp_meter_band_type {
    OFPMBT_DROP            = 1,      /* Drop packet. */
    OFPMBT_DSCP_REMARK     = 2,      /* Remark DSCP in the IP header. */
    OFPMBT_EXPERIMENTER    = 0xFFFF  /* Experimenter meter band. */
};
```

An OpenFlow switch may not support all band types, and may not allow to use all its supported bands on all meters, i.e. some meters may be specialised.

The band `OFPMBT_DROP` defines a simple rate limiter that drop packets that exceed the band rate value, and uses the following structure:

```
/* OFPMBT_DROP band - drop packets */
struct ofp_meter_band_drop {
    uint16_t        type;    /* OFPMBT_DROP. */
    uint16_t        len;     /* Length in bytes of this band. */
    uint32_t        rate;    /* Rate for dropping packets. */
    uint32_t        burst_size; /* Size of bursts. */
    uint8_t         pad[4];
};
OFP_ASSERT(sizeof(struct ofp_meter_band_drop) == 16);
```

The band `OFPMBT_DSCP_REMARK` defines a simple DiffServ policer that remark the drop precedence of the DSCP field in the IP header of the packets that exceed the band rate value, and uses the following structure:


```
/* OFPMBT_DSCP_REMARK band - Remark DSCP in the IP header */
struct ofp_meter_band_dscp_remark {
    uint16_t        type;    /* OFPMBT_DSCP_REMARK. */
    uint16_t        len;     /* Length in bytes of this band. */
    uint32_t        rate;    /* Rate for remarking packets. */
    uint32_t        burst_size; /* Size of bursts. */
    uint8_t         prec_level; /* Number of drop precedence level to add. */
    uint8_t         pad[3];
};
OFP_ASSERT(sizeof(struct ofp_meter_band_dscp_remark) == 16);
```

The `prec_level` field indicates by which amount the drop precedence of the packet should be increased if the band is exceeded.

The band `OFPMBT_EXPERIMENTER` is experimenter defined and uses the following structure:

```
/* OFPMBT_EXPERIMENTER band - Write actions in action set */
struct ofp_meter_band_experimenter {
    uint16_t        type;    /* One of OFPMBT_*. */
    uint16_t        len;     /* Length in bytes of this band. */
    uint32_t        rate;    /* Rate for this band. */
    uint32_t        burst_size;   /* Size of bursts. */
    uint32_t        experimenter; /* Experimenter ID which takes the same
                                     form as in struct
                                     ofp_experimenter_header. */
};
OFP_ASSERT(sizeof(struct ofp_meter_band_experimenter) == 16);
```

7.3.5 Multipart Messages

Multipart messages are used to encode requests or replies that potentially carry a large amount of data and would not always fit in a single OpenFlow message, which is limited to 64KB. The request or reply is encoded as a sequence of multipart messages with a specific multipart type, and re-assembled by the receiver. Multipart messages are primarily used to request statistics or state information from the switch.

The request is carried in one or more `OFPT_MULTIPART_REQUEST` messages:

```
struct ofp_multipart_request {
    struct ofp_header header;
    uint16_t type;              /* One of the OFPMP_* constants. */
    uint16_t flags;             /* OFPMPF_REQ_* flags. */
    uint8_t pad[4];
    uint8_t body[0];            /* Body of the request. 0 or more bytes. */
};
OFP_ASSERT(sizeof(struct ofp_multipart_request) == 16);


enum ofp_multipart_request_flags {
    OFPMPF_REQ_MORE  = 1 << 0  /* More requests to follow. */
};
```

The switch responds with one or more `OFPT_MULTIPART_REPLY` messages:


```
struct ofp_multipart_reply {
    struct ofp_header header;
    uint16_t type;              /* One of the OFPMP_* constants. */
    uint16_t flags;             /* OFPMPF_REPLY_* flags. */
    uint8_t pad[4];
    uint8_t body[0];            /* Body of the reply. 0 or more bytes. */
};
OFP_ASSERT(sizeof(struct ofp_multipart_reply) == 16);


enum ofp_multipart_reply_flags {
    OFPMPF_REPLY_MORE  = 1 << 0  /* More replies to follow. */
};
```

The only value defined for `flags` in a request and reply is whether more requests/replies will follow this one - this has the value `0x0001`. To ease implementation, the controller is allowed to send requests and the switch is allowed to send replies with no additional entries (i.e. an empty `body`). However, another message must always follow a message with the *more* flag set. A request or reply that spans multiple messages (has one or more messages with the *more* flag set), must use the same multipart `type` and transaction id (xid) for all messages in the message sequence. Messages from a multipart request or reply may be interleaved with other OpenFlow message types, including other multipart requests or replies, but must have distinct transaction ids if multiple unanswered multipart requests or replies are in flight simultaneously. Transaction ids of replies must always match the request that prompted them.

In both the request and response, the `type` field specifies the kind of information being passed and determines how the `body` field is interpreted:

```
enum ofp_multipart_type {
    /* Description of this OpenFlow switch.
     * The request body is empty.
     * The reply body is struct ofp_desc. */
    OFPMP_DESC = 0,

    /* Individual flow statistics.
     * The request body is struct ofp_flow_stats_request.
     * The reply body is an array of struct ofp_flow_stats. */
    OFPMP_FLOW = 1,

    /* Aggregate flow statistics.
     * The request body is struct ofp_aggregate_stats_request.
     * The reply body is struct ofp_aggregate_stats_reply. */
    OFPMP_AGGREGATE = 2,

    /* Flow table statistics.
     * The request body is empty.
     * The reply body is an array of struct ofp_table_stats. */
    OFPMP_TABLE = 3,

    /* Port statistics.
     * The request body is struct ofp_port_stats_request.
     * The reply body is an array of struct ofp_port_stats. */
    OFPMP_PORT_STATS = 4,
```



```
    /* Queue statistics for a port
     * The request body is struct ofp_queue_stats_request.
     * The reply body is an array of struct ofp_queue_stats */
    OFPMP_QUEUE = 5,

    /* Group counter statistics.
     * The request body is struct ofp_group_stats_request.
     * The reply is an array of struct ofp_group_stats. */
    OFPMP_GROUP = 6,

    /* Group description.
     * The request body is empty.
     * The reply body is an array of struct ofp_group_desc. */
    OFPMP_GROUP_DESC = 7,

    /* Group features.
     * The request body is empty.
     * The reply body is struct ofp_group_features. */
    OFPMP_GROUP_FEATURES = 8,

    /* Meter statistics.
     * The request body is struct ofp_meter_multipart_requests.
     * The reply body is an array of struct ofp_meter_stats. */
    OFPMP_METER = 9,

    /* Meter configuration.
     * The request body is struct ofp_meter_multipart_requests.
     * The reply body is an array of struct ofp_meter_config. */
    OFPMP_METER_CONFIG = 10,

    /* Meter features.
     * The request body is empty.
     * The reply body is struct ofp_meter_features. */
    OFPMP_METER_FEATURES = 11,

    /* Table features.
     * The request body is either empty or contains an array of
     * struct ofp_table_features containing the controller's
     * desired view of the switch. If the switch is unable to
     * set the specified view an error is returned.
     * The reply body is an array of struct ofp_table_features. */
    OFPMP_TABLE_FEATURES = 12,

    /* Port description.
     * The request body is empty.
     * The reply body is an array of struct ofp_port. */
    OFPMP_PORT_DESC = 13,

    /* Experimenter extension.
     * The request and reply bodies begin with
     * struct ofp_experimenter_multipart_header.
     * The request and reply bodies are otherwise experimenter-defined. */
    OFPMP_EXPERIMENTER = 0xffff
};
```

If a multipart request spans multiple messages and grows to a size that the switch is unable to buffer, the switch must respond with an error messsage of type `OFPET_BAD_REQUEST` and code

OFPBRC_MULTIPART_BUFFER_OVERFLOW. If a multipart request contains a type that is not supported, the switch must respond with an error messsage of type OFPET_BAD_REQUEST and code OFPBRC_BAD_MULTIPART.

In all types of multipart reply containing statistics, if a specific numeric counter is not available in the switch, its value must be set to the maximum field value (the unsigned equivalent of -1). Counters are unsigned and wrap around with no overflow indicator.

7.3.5.1 Description

Information about the switch manufacturer, hardware revision, software revision, serial number, and a description field is available from the OFPMP_DESC multipart request type:

```
/* Body of reply to OFPMP_DESC request.  Each entry is a NULL-terminated
 * ASCII string. */
struct ofp_desc {
    char mfr_desc[DESC_STR_LEN];       /* Manufacturer description. */
    char hw_desc[DESC_STR_LEN];        /* Hardware description. */
    char sw_desc[DESC_STR_LEN];        /* Software description. */
    char serial_num[SERIAL_NUM_LEN];   /* Serial number. */
    char dp_desc[DESC_STR_LEN];        /* Human readable description of datapath. */
};
OFP_ASSERT(sizeof(struct ofp_desc) == 1056);
```

Each entry is ASCII formatted and padded on the right with null bytes (\0). DESC_STR_LEN is 256 and SERIAL_NUM_LEN is 32. The dp_desc field is a free-form string to describe the datapath for debugging purposes, e.g., "switch3 in room 3120". As such, it is not guaranteed to be unique and should not be used as the primary identifier for the datapath—use the datapath_id field from the switch features instead (see 7.3.1).

7.3.5.2 Individual Flow Statistics

Information about individual flow entries is requested with the OFPMP_FLOW multipart request type:

```
/* Body for ofp_multipart_request of type OFPMP_FLOW. */
struct ofp_flow_stats_request {
    uint8_t table_id;         /* ID of table to read (from ofp_table_stats),
                                 OFPTT_ALL for all tables. */
    uint8_t pad[3];           /* Align to 32 bits. */
    uint32_t out_port;        /* Require matching entries to include this
                                 as an output port.  A value of OFPP_ANY
                                 indicates no restriction. */
    uint32_t out_group;       /* Require matching entries to include this
                                 as an output group.  A value of OFPG_ANY
                                 indicates no restriction. */
    uint8_t pad2[4];          /* Align to 64 bits. */
    uint64_t cookie;          /* Require matching entries to contain this
                                 cookie value */
    uint64_t cookie_mask;     /* Mask used to restrict the cookie bits that
                                 must match. A value of 0 indicates
```



```
                                no restriction. */
    struct ofp_match match;   /* Fields to match. Variable size. */
};
OFP_ASSERT(sizeof(struct ofp_flow_stats_request) == 40);
```

The `match` field contains a description of the flow entries that should be matched and may contain wildcarded and masked fields. This field's matching behavior is described in Section 6.4.

The `table_id` field indicates the index of a single table to read, or `OFPTT_ALL` for all tables.

The `out_port` and `out_group` fields optionally filter by output port and group. If either `out_port` or `out_group` contain a value other than `OFPP_ANY` and `OFPG_ANY` respectively, it introduces a constraint when matching. This constraint is that the flow entry must contain an output action directed at that port or group. Other constraints such as `ofp_match` structs are still used; this is purely an *additional* constraint. Note that to disable output filtering, both `out_port` and `out_group` must be set to `OFPP_ANY` and `OFPG_ANY` respectively.

The usage of the `cookie` and `cookie_mask` fields is defined in Section 6.4.

The `body` of the reply to a `OFPMP_FLOW` multipart request consists of an array of the following:

```
/* Body of reply to OFPMP_FLOW request. */
struct ofp_flow_stats {
    uint16_t length;          /* Length of this entry. */
    uint8_t table_id;         /* ID of table flow came from. */
    uint8_t pad;
    uint32_t duration_sec;    /* Time flow has been alive in seconds. */
    uint32_t duration_nsec;   /* Time flow has been alive in nanoseconds beyond
                                 duration_sec. */
    uint16_t priority;        /* Priority of the entry. */
    uint16_t idle_timeout;    /* Number of seconds idle before expiration. */
    uint16_t hard_timeout;    /* Number of seconds before expiration. */
    uint16_t flags;           /* Bitmap of OFPFF_* flags. */
    uint8_t pad2[4];          /* Align to 64-bits. */
    uint64_t cookie;          /* Opaque controller-issued identifier. */
    uint64_t packet_count;    /* Number of packets in flow. */
    uint64_t byte_count;      /* Number of bytes in flow. */
    struct ofp_match match;   /* Description of fields. Variable size. */
    /* The variable size and padded match is always followed by instructions. */
    //struct ofp_instruction instructions[0]; /* Instruction set - 0 or more. */
};
OFP_ASSERT(sizeof(struct ofp_flow_stats) == 56);
```

The fields consist of those provided in the `flow_mod` that created the flow entry (see 7.3.4.1), plus the `table_id` into which the entry was inserted, the `packet_count`, and the `byte_count` counting all packets processed by the flow entry.

The `duration_sec` and `duration_nsec` fields indicate the elapsed time the flow entry has been installed in the switch. The total duration in nanoseconds can be computed as $duration\_sec \times 10^9 + duration\_nsec$. Implementations are required to provide second precision; higher precision is encouraged where available.

7.3.5.3 Aggregate Flow Statistics

Aggregate information about multiple flow entries is requested with the OFPMP_AGGREGATE multipart request type:

```
/* Body for ofp_multipart_request of type OFPMP_AGGREGATE. */
struct ofp_aggregate_stats_request {
    uint8_t table_id;         /* ID of table to read (from ofp_table_stats)
                                 OFPTT_ALL for all tables. */
    uint8_t pad[3];           /* Align to 32 bits. */
    uint32_t out_port;        /* Require matching entries to include this
                                 as an output port.  A value of OFPP_ANY
                                 indicates no restriction. */
    uint32_t out_group;       /* Require matching entries to include this
                                 as an output group.  A value of OFPG_ANY
                                 indicates no restriction. */
    uint8_t pad2[4];          /* Align to 64 bits. */
    uint64_t cookie;          /* Require matching entries to contain this
                                 cookie value */
    uint64_t cookie_mask;     /* Mask used to restrict the cookie bits that
                                 must match. A value of 0 indicates
                                 no restriction. */
    struct ofp_match match;   /* Fields to match. Variable size. */
};
OFP_ASSERT(sizeof(struct ofp_aggregate_stats_request) == 40);
```

The fields in this message have the same meanings as in the individual flow stats request type (OFPMP_FLOW).

The body of the reply consists of the following:

```
/* Body of reply to OFPMP_AGGREGATE request. */
struct ofp_aggregate_stats_reply {
    uint64_t packet_count;    /* Number of packets in flows. */
    uint64_t byte_count;      /* Number of bytes in flows. */
    uint32_t flow_count;      /* Number of flows. */
    uint8_t pad[4];           /* Align to 64 bits. */
};
OFP_ASSERT(sizeof(struct ofp_aggregate_stats_reply) == 24);
```

7.3.5.4 Table Statistics

Information about tables is requested with the OFPMP_TABLE multipart request type. The request does not contain any data in the body.

The body of the reply consists of an array of the following:

```
/* Body of reply to OFPMP_TABLE request. */
struct ofp_table_stats {
    uint8_t table_id;        /* Identifier of table.  Lower numbered tables
                                are consulted first. */
    uint8_t pad[3];          /* Align to 32-bits. */
```



```
    uint32_t active_count;   /* Number of active entries. */
    uint64_t lookup_count;   /* Number of packets looked up in table. */
    uint64_t matched_count;  /* Number of packets that hit table. */
};
OFP_ASSERT(sizeof(struct ofp_table_stats) == 24);
```

The array has one structure for each table supported by the switch. The entries are returned in the order that packets traverse the tables.

7.3.5.5 Table Features

The `OFPMP_TABLE_FEATURES` multipart type allows a controller to both query for the capabilities of existing tables, and to optionally ask the switch to reconfigure its tables to match a supplied configuration. In general, the table feature capabilities represents all possible features of a table, however some features may be mutually exclusive and the current capabilities structures do not allow us to represent such exclusions.

7.3.5.5.1 Table Features request and reply

If the `OFPMP_TABLE_FEATURES` request body is empty the switch will return an array of struct `ofp_table_features` containing the capabilities of the currently configured flow tables.

If the request body contains an array of one or more `ofp_table_features` structs, the switch will attempt to change its flow tables to match the requested flow table configuration. This operation configures the entire pipeline, and the set of flow tables in the pipeline must match the set in the request, or an error must be returned. In particular, if the requested configuration does not contain an `ofp_table_features` struct for one or more flow tables that the switch supports, these flow tables are to be removed from the pipeline if the configuration is successfully set. A successful configuration change will modify the features for all flow tables in the request, that is, either all the flow tables specified in the request are modified or none, and the new capabilities for each flow table must be either a superset of, or equal to the requested capabilities. If the flow table configuration is successful, flow entries from flow tables that have been removed or flow tables that had their capabilities change between the prior and new configuration are removed from the flow table, however no `ofp_flow_removed` messages are sent. The switch then replies with the new configuration. If the switch is unable to set the requested configuration, an error of type `OFPET_TABLE_FEATURES_FAILED` is returned with the appropriate error code.

Requests and replies containing `ofp_table_features` are expected to meet the following minimum requirements:

- Each `ofp_table_features` struct's `table_id` field value should be unique amongst all `ofp_table_features` structs in the message
- Each `ofp_table_features` struct's `properties` field must contain exactly one of each of the `ofp_table_feature_prop_type` properties, with two exceptions. First, properties with the `_MISS` suffix may be ommited if it is the same as the corresponding property for regular flow entries. Second, properties of type `OFPTFPT_EXPERIMENTER` and `OFPTFPT_EXPERIMENTER_MISS` may be ommited or included many times. Ordering is unspecified, but implementers are encouraged to use the ordering listed in the specification (see 7.3.5.5.2).

A switch receiving a request that does not meet these requirements should return an error of type OFPET_TABLE_FEATURES_FAILED with the appropriate error code.

The following structure describes the body of the table features request and reply:

```
/* Body for ofp_multipart_request of type OFPMP_TABLE_FEATURES./
 * Body of reply to OFPMP_TABLE_FEATURES request. */
struct ofp_table_features {
    uint16_t length;         /* Length is padded to 64 bits. */
    uint8_t table_id;        /* Identifier of table.  Lower numbered tables
                                are consulted first. */
    uint8_t pad[5];          /* Align to 64-bits. */
    char name[OFP_MAX_TABLE_NAME_LEN];
    uint64_t metadata_match; /* Bits of metadata table can match. */
    uint64_t metadata_write; /* Bits of metadata table can write. */
    uint32_t config;         /* Bitmap of OFPTC_* values */
    uint32_t max_entries;    /* Max number of entries supported. */

    /* Table Feature Property list */
    struct ofp_table_feature_prop_header properties[0]; /* List of properties */
};
OFP_ASSERT(sizeof(struct ofp_table_features) == 64);
```

The array has one structure for each flow table supported by the switch. The entries are always returned in the order that packets traverse the flow tables. OFP_MAX_TABLE_NAME_LEN is 32 .

The metadata_match field indicates the bits of the metadata field that the table can match on, when using the metadata field of struct ofp_match. A value of 0xFFFFFFFFFFFFFFFF indicates that the table can match the full metadata field.

The metadata_write field indicates the bits of the metadata field that the table can write using the OFPIT_WRITE_METADATA instruction. A value of 0xFFFFFFFFFFFFFFFF indicates that the table can write the full metadata field.

The config field is the table configuration that was set on the table via a table configuration message (see 7.3.3).

The max_entries field describes the maximum number of flow entries that can be inserted into that table. Due to limitations imposed by modern hardware, the max_entries value should be considered advisory and a best effort approximation of the capacity of the table. Despite the high-level abstraction of a table, in practice the resource consumed by a single flow table entry is not constant. For example, a flow table entry might consume more than one entry, depending on its match parameters (e.g., IPv4 vs. IPv6). Also, tables that appear distinct at an OpenFlow-level might in fact share the same underlying physical resources. Further, on OpenFlow hybrid switches, those table may be shared with non-OpenFlow functions. The result is that switch implementers should report an approximation of the total flow entries supported and controller writers should not treat this value as a fixed, physical constant.

The properties field is a list of table feature properties, describing various capabilities of the table.

7.3.5.5.2 Table Features properties

The list of table feature property types that are currently defined are:

```
/* Table Feature property types.
 * Low order bit cleared indicates a property for a regular Flow Entry.
 * Low order bit set indicates a property for the Table-Miss Flow Entry.
 */
enum ofp_table_feature_prop_type {
    OFPTFPT_INSTRUCTIONS           = 0,  /* Instructions property. */
    OFPTFPT_INSTRUCTIONS_MISS      = 1,  /* Instructions for table-miss. */
    OFPTFPT_NEXT_TABLES            = 2,  /* Next Table property. */
    OFPTFPT_NEXT_TABLES_MISS       = 3,  /* Next Table for table-miss. */
    OFPTFPT_WRITE_ACTIONS          = 4,  /* Write Actions property. */
    OFPTFPT_WRITE_ACTIONS_MISS     = 5,  /* Write Actions for table-miss. */
    OFPTFPT_APPLY_ACTIONS          = 6,  /* Apply Actions property. */
    OFPTFPT_APPLY_ACTIONS_MISS     = 7,  /* Apply Actions for table-miss. */
    OFPTFPT_MATCH                  = 8,  /* Match property. */
    OFPTFPT_WILDCARDS              = 10, /* Wildcards property. */
    OFPTFPT_WRITE_SETFIELD         = 12, /* Write Set-Field property. */
    OFPTFPT_WRITE_SETFIELD_MISS    = 13, /* Write Set-Field for table-miss. */
    OFPTFPT_APPLY_SETFIELD         = 14, /* Apply Set-Field property. */
    OFPTFPT_APPLY_SETFIELD_MISS    = 15, /* Apply Set-Field for table-miss. */
    OFPTFPT_EXPERIMENTER           = 0xFFFE, /* Experimenter property. */
    OFPTFPT_EXPERIMENTER_MISS      = 0xFFFF, /* Experimenter for table-miss. */
};
```

The properties with the `_MISS` suffix describe the capabilities for the table-miss flow entry (see 5.4), whereas other properties describe the capabilities for regular flow entry. If a specific property does not have any capability (for example no Set-Field support), a property with an empty list must be included in the property list. When a property of the table-miss flow entry is the same as the corresponding property for regular flow entries (i.e. both properties have the same list of capabilities), this table-miss property can be omited from the property list.

A property definition contains the property type, length, and any associated data:

```
/* Common header for all Table Feature Properties */
struct ofp_table_feature_prop_header {
    uint16_t         type;    /* One of OFPTFPT_*. */
    uint16_t         length;  /* Length in bytes of this property. */
};
OFP_ASSERT(sizeof(struct ofp_table_feature_prop_header) == 4);
```

The `OFPTFPT_INSTRUCTIONS` and `OFPTFPT_INSTRUCTIONS_MISS` properties use the following structure and fields:

```
/* Instructions property */
struct ofp_table_feature_prop_instructions {
    uint16_t         type;    /* One of OFPTFPT_INSTRUCTIONS,
                                 OFPTFPT_INSTRUCTIONS_MISS. */
    uint16_t         length;  /* Length in bytes of this property. */
    /* Followed by:
     *   - Exactly (length - 4) bytes containing the instruction ids, then
```



```
     *   - Exactly (length + 7)/8*8 - (length) (between 0 and 7)
     *     bytes of all-zero bytes */
    struct ofp_instruction   instruction_ids[0];   /* List of instructions */
};
OFP_ASSERT(sizeof(struct ofp_table_feature_prop_instructions) == 4);
```

The `instruction_ids` is the list of instructions supported by this table (see 5.9). The elements of that list are variable size to enable expressing experimenter instructions, non-experimenter instructions are 4 bytes.

The `OFPTFPT_NEXT_TABLES` and `OFPTFPT_NEXT_TABLES_MISS` properties use the following structure and fields:

```
/* Next Tables property */
struct ofp_table_feature_prop_next_tables {
    uint16_t         type;    /* One of OFPTFPT_NEXT_TABLES,
                                 OFPTFPT_NEXT_TABLES_MISS. */
    uint16_t         length;  /* Length in bytes of this property. */
    /* Followed by:
     *   - Exactly (length - 4) bytes containing the table_ids, then
     *   - Exactly (length + 7)/8*8 - (length) (between 0 and 7)
     *     bytes of all-zero bytes */
    uint8_t          next_table_ids[0];        /* List of table ids. */
};
OFP_ASSERT(sizeof(struct ofp_table_feature_prop_next_tables) == 4);
```

The `next_table_ids` is the array of tables that can be directly reached from the present table using the `OFPIT_GOTO_TABLE` instruction (see 5.1).

The `OFPTFPT_WRITE_ACTIONS`, `OFPTFPT_WRITE_ACTIONS_MISS`, `OFPTFPT_APPLY_ACTIONS` and `OFPTFPT_APPLY_ACTIONS_MISS` properties use the following structure and fields:

```
/* Actions property */
struct ofp_table_feature_prop_actions {
    uint16_t         type;    /* One of OFPTFPT_WRITE_ACTIONS,
                                 OFPTFPT_WRITE_ACTIONS_MISS,
                                 OFPTFPT_APPLY_ACTIONS,
                                 OFPTFPT_APPLY_ACTIONS_MISS. */
    uint16_t         length;  /* Length in bytes of this property. */
    /* Followed by:
     *   - Exactly (length - 4) bytes containing the action_ids, then
     *   - Exactly (length + 7)/8*8 - (length) (between 0 and 7)
     *     bytes of all-zero bytes */
    struct ofp_action_header  action_ids[0];      /* List of actions */
};
OFP_ASSERT(sizeof(struct ofp_table_feature_prop_actions) == 4);
```

The `action_ids` is the list of actions for the feature (see 5.12). The elements of that list are variable size to enable expressing experimenter actions, non-experimenter actions are 4 bytes. The `OFPTFPT_WRITE_ACTIONS` and `OFPTFPT_WRITE_ACTIONS_MISS` properties describe actions supported by the table using the `OFPIT_WRITE_ACTIONS` instruction, whereas the `OFPTFPT_APPLY_ACTIONS`

and OFPTFPT_APPLY_ACTIONS_MISS properties describe actions supported by the table using the OFPIT_APPLY_ACTIONS instruction.

The OFPTFPT_MATCH, OFPTFPT_WILDCARDS, OFPTFPT_WRITE_SETFIELD, OFPTFPT_WRITE_SETFIELD_MISS, OFPTFPT_APPLY_SETFIELD and OFPTFPT_APPLY_SETFIELD_MISS properties use the following structure and fields:

```
/* Match, Wildcard or Set-Field property */
struct ofp_table_feature_prop_oxm {
    uint16_t         type;    /* One of OFPTFPT_MATCH,
                                 OFPTFPT_WILDCARDS,
                                 OFPTFPT_WRITE_SETFIELD,
                                 OFPTFPT_WRITE_SETFIELD_MISS,
                                 OFPTFPT_APPLY_SETFIELD,
                                 OFPTFPT_APPLY_SETFIELD_MISS. */
    uint16_t         length;  /* Length in bytes of this property. */
    /* Followed by:
     *   - Exactly (length - 4) bytes containing the oxm_ids, then
     *   - Exactly (length + 7)/8*8 - (length) (between 0 and 7)
     *     bytes of all-zero bytes */
    uint32_t         oxm_ids[0];   /* Array of OXM headers */
};
OFP_ASSERT(sizeof(struct ofp_table_feature_prop_oxm) == 4);
```

The oxm_ids is the list of OXM types for the feature (see 7.2.3.2). The elements of that list are 32-bit OXM headers or 64-bit OXM headers for experimenter OXM fields.

The OFPTFPT_MATCH property indicates the fields for which that particular table supports matching on (see 7.2.3.7). For example, if the table can match the ingress port, an OXM header with the type OXM_OF_IN_PORT should be included in the list. If the HASMASK bit is set on the OXM header then the switch must support masking for the given type. The OFPTFPT_WILDCARDS property indicates the fields for which that particular table supports wildcarding (omiting). For example, a direct look-up hash table would have that list empty, while a TCAM or sequentially searched table would have it set to the same value as the OFPTFPT_MATCH property.

The OFPTFPT_WRITE_SETFIELD and OFPTFPT_WRITE_SETFIELD_MISS properties describe Set-Field action types supported by the table using the OFPIT_WRITE_ACTIONS instruction, whereas the OFPTFPT_APPLY_SETFIELD and OFPTFPT_APPLY_SETFIELD_MISS properties describe Set-Field action types supported by the table using the OFPIT_APPLY_ACTIONS instruction.

All fields in ofp_table_features may be requested to be changed by the controller with the exception of the max_entries field, this is read only and returned by the switch.

The OFPTFPT_APPLY_ACTIONS, OFPTFPT_APPLY_ACTIONS_MISS, OFPTFPT_APPLY_SETFIELD, and OFPTFPT_APPLY_SETFIELD_MISS properties contain actions and fields the table is capable of applying. For each of these lists, if an element is present it means the table is at least capable of applying the element in isolation one time. There is currently no way to indicate which elements can be applied together, in which order, and how many time an element can be applied in a single flow entry.

The OFPTFPT_EXPERIMENTER and OFPTFPT_EXPERIMENTER_MISS properties uses the following structure and fields:


```
/* Experimenter table feature property */
struct ofp_table_feature_prop_experimenter {
    uint16_t         type;    /* One of OFPTFPT_EXPERIMENTER,
                                 OFPTFPT_EXPERIMENTER_MISS. */
    uint16_t         length;  /* Length in bytes of this property. */
    uint32_t         experimenter;  /* Experimenter ID which takes the same
                                       form as in struct
                                       ofp_experimenter_header. */
    uint32_t         exp_type;      /* Experimenter defined. */
    /* Followed by:
     *   - Exactly (length - 12) bytes containing the experimenter data, then
     *   - Exactly (length + 7)/8*8 - (length) (between 0 and 7)
     *     bytes of all-zero bytes */
    uint32_t         experimenter_data[0];
};
OFP_ASSERT(sizeof(struct ofp_table_feature_prop_experimenter) == 12);
```

The `experimenter` field is the Experimenter ID, which takes the same form as in struct `ofp_experimenter` (see 7.5.4).

7.3.5.6 Port Statistics

Information about ports statistics is requested with the `OFPMP_PORT_STATS` multipart request type:

```
/* Body for ofp_multipart_request of type OFPMP_PORT. */
struct ofp_port_stats_request {
    uint32_t port_no;        /* OFPMP_PORT message must request statistics
                              * either for a single port (specified in
                              * port_no) or for all ports (if port_no ==
                              * OFPP_ANY). */
    uint8_t pad[4];
};
OFP_ASSERT(sizeof(struct ofp_port_stats_request) == 8);
```

The `port_no` field optionally filters the stats request to the given port. To request all port statistics, `port_no` must be set to `OFPP_ANY`.

The `body` of the reply consists of an array of the following:

```
/* Body of reply to OFPMP_PORT request. If a counter is unsupported, set
 * the field to all ones. */
struct ofp_port_stats {
    uint32_t port_no;
    uint8_t pad[4];          /* Align to 64-bits. */
    uint64_t rx_packets;     /* Number of received packets. */
    uint64_t tx_packets;     /* Number of transmitted packets. */
    uint64_t rx_bytes;       /* Number of received bytes. */
    uint64_t tx_bytes;       /* Number of transmitted bytes. */
    uint64_t rx_dropped;     /* Number of packets dropped by RX. */
    uint64_t tx_dropped;     /* Number of packets dropped by TX. */
    uint64_t rx_errors;      /* Number of receive errors.  This is a super-set
                                of more specific receive errors and should be
```



```
                                greater than or equal to the sum of all
                                rx_*_err values. */
    uint64_t tx_errors;      /* Number of transmit errors.  This is a super-set
                                of more specific transmit errors and should be
                                greater than or equal to the sum of all
                                tx_*_err values (none currently defined.) */
    uint64_t rx_frame_err;   /* Number of frame alignment errors. */
    uint64_t rx_over_err;    /* Number of packets with RX overrun. */
    uint64_t rx_crc_err;     /* Number of CRC errors. */
    uint64_t collisions;     /* Number of collisions. */
    uint32_t duration_sec;   /* Time port has been alive in seconds. */
    uint32_t duration_nsec;  /* Time port has been alive in nanoseconds beyond
                                duration_sec. */
};
OFP_ASSERT(sizeof(struct ofp_port_stats) == 112);
```

The `duration_sec` and `duration_nsec` fields indicate the elapsed time the port has been configured into the OpenFlow pipeline. The total duration in nanoseconds can be computed as $duration\_sec \times 10^9 + duration\_nsec$. Implementations are required to provide second precision; higher precision is encouraged where available.

7.3.5.7 Port Description

The port description request `OFPMP_PORT_DESCRIPTION` enables the controller to get a description of all the ports in the system that support OpenFlow. The request body is empty. The reply body consists of an array of the following:

```
/* Description of a port */
struct ofp_port {
    uint32_t port_no;
    uint8_t pad[4];
    uint8_t hw_addr[OFP_ETH_ALEN];
    uint8_t pad2[2];                  /* Align to 64 bits. */
    char name[OFP_MAX_PORT_NAME_LEN]; /* Null-terminated */

    uint32_t config;        /* Bitmap of OFPPC_* flags. */
    uint32_t state;         /* Bitmap of OFPPS_* flags. */

    /* Bitmaps of OFPPF_* that describe features.  All bits zeroed if
     * unsupported or unavailable. */
    uint32_t curr;          /* Current features. */
    uint32_t advertised;    /* Features being advertised by the port. */
    uint32_t supported;     /* Features supported by the port. */
    uint32_t peer;          /* Features advertised by peer. */

    uint32_t curr_speed;    /* Current port bitrate in kbps. */
    uint32_t max_speed;     /* Max port bitrate in kbps */
};
OFP_ASSERT(sizeof(struct ofp_port) == 64);
```

This structure is described in section 7.2.1.

7.3.5.8 Queue Statistics

The OFPMP_QUEUE multipart request message provides queue statistics for one or more ports and one or more queues. The request body contains a port_no field identifying the OpenFlow port for which statistics are requested, or OFPP_ANY to refer to all ports. The queue_id field identifies one of the priority queues, or OFPQ_ALL to refer to all queues configured at the specified port. OFPQ_ALL is 0xffffffff.

```
struct ofp_queue_stats_request {
    uint32_t port_no;        /* All ports if OFPP_ANY. */
    uint32_t queue_id;       /* All queues if OFPQ_ALL. */
};
OFP_ASSERT(sizeof(struct ofp_queue_stats_request) == 8);
```

The body of the reply consists of an array of the following structure:

```
struct ofp_queue_stats {
    uint32_t port_no;
    uint32_t queue_id;       /* Queue i.d */
    uint64_t tx_bytes;       /* Number of transmitted bytes. */
    uint64_t tx_packets;     /* Number of transmitted packets. */
    uint64_t tx_errors;      /* Number of packets dropped due to overrun. */
    uint32_t duration_sec;   /* Time queue has been alive in seconds. */
    uint32_t duration_nsec;  /* Time queue has been alive in nanoseconds beyond
                                duration_sec. */
};
OFP_ASSERT(sizeof(struct ofp_queue_stats) == 40);
```

The duration_sec and duration_nsec fields indicate the elapsed time the queue has been installed in the switch. The total duration in nanoseconds can be computed as $duration\_sec \times 10^9 + duration\_nsec$. Implementations are required to provide second precision; higher precision is encouraged where available.

7.3.5.9 Group Statistics

The OFPMP_GROUP multipart request message provides statistics for one or more groups. The request body consists of a group_id field, which can be set to OFPG_ALL to refer to all groups on the switch.

```
/* Body of OFPMP_GROUP request. */
struct ofp_group_stats_request {
    uint32_t group_id;       /* All groups if OFPG_ALL. */
    uint8_t pad[4];          /* Align to 64 bits. */
};
OFP_ASSERT(sizeof(struct ofp_group_stats_request) == 8);
```

The body of the reply consists of an array of the following structure:


```
/* Body of reply to OFPMP_GROUP request. */
struct ofp_group_stats {
    uint16_t length;         /* Length of this entry. */
    uint8_t pad[2];          /* Align to 64 bits. */
    uint32_t group_id;       /* Group identifier. */
    uint32_t ref_count;      /* Number of flows or groups that directly forward
                                to this group. */
    uint8_t pad2[4];         /* Align to 64 bits. */
    uint64_t packet_count;   /* Number of packets processed by group. */
    uint64_t byte_count;     /* Number of bytes processed by group. */
    uint32_t duration_sec;   /* Time group has been alive in seconds. */
    uint32_t duration_nsec;  /* Time group has been alive in nanoseconds beyond
                                duration_sec. */
    struct ofp_bucket_counter bucket_stats[0]; /* One counter set per bucket. */
};
OFP_ASSERT(sizeof(struct ofp_group_stats) == 40);
```

The fields consist of those provided in the `group_mod` that created the group, plus the `ref_count` counting the number of flows referencing directly the group, the `packet_count`, and the `byte_count` counting all packets processed by the group.

The `duration_sec` and `duration_nsec` fields indicate the elapsed time the group has been installed in the switch. The total duration in nanoseconds can be computed as $duration\_sec \times 10^9 + duration\_nsec$. Implementations are required to provide second precision; higher precision is encouraged where available.

The `bucket_stats` field consists of an array of `ofp_bucket_counter` structs:

```
/* Used in group stats replies. */
struct ofp_bucket_counter {
    uint64_t packet_count;   /* Number of packets processed by bucket. */
    uint64_t byte_count;     /* Number of bytes processed by bucket. */
};
OFP_ASSERT(sizeof(struct ofp_bucket_counter) == 16);
```

7.3.5.10 Group Description

The `OFPMP_GROUP_DESC` multipart request message provides a way to list the set of groups on a switch, along with their corresponding bucket actions. The request body is empty, while the reply body is an array of the following structure:

```
/* Body of reply to OFPMP_GROUP_DESC request. */
struct ofp_group_desc {
    uint16_t length;              /* Length of this entry. */
    uint8_t type;                 /* One of OFPGT_*. */
    uint8_t pad;                  /* Pad to 64 bits. */
    uint32_t group_id;            /* Group identifier. */
    struct ofp_bucket buckets[0];   /* List of buckets - 0 or more. */
};
OFP_ASSERT(sizeof(struct ofp_group_desc) == 8);
```

Fields for group description are the same as those used with the `ofp_group_mod` struct (see 7.3.4.2).

7.3.5.11 Group Features

The OFPMP_GROUP_FEATURES multipart request message provides a way to list the capabilities of groups on a switch. The request body is empty, while the reply body is the following structure:

```
/* Body of reply to OFPMP_GROUP_FEATURES request. Group features. */
struct ofp_group_features {
    uint32_t  types;           /* Bitmap of OFPGT_* values supported. */
    uint32_t  capabilities;    /* Bitmap of OFPGFC_* capability supported. */
    uint32_t  max_groups[4];   /* Maximum number of groups for each type. */
    uint32_t  actions[4];      /* Bitmaps of OFPAT_* that are supported. */
};
OFP_ASSERT(sizeof(struct ofp_group_features) == 40);
```

The max_groups field is the maximum number of groups for each type of groups. The actions is the supported actions for each type of groups. The capabilities uses a combination of the following flags:

```
/* Group configuration flags */
enum ofp_group_capabilities {
    OFPGFC_SELECT_WEIGHT   = 1 << 0,  /* Support weight for select groups */
    OFPGFC_SELECT_LIVENESS = 1 << 1,  /* Support liveness for select groups */
    OFPGFC_CHAINING        = 1 << 2,  /* Support chaining groups */
    OFPGFC_CHAINING_CHECKS = 1 << 3,  /* Check chaining for loops and delete */
};
```

7.3.5.12 Meter Statistics

The OFPMT_METER stats request message provides statistics for one or more meters. The request body consists of a meter_id field, which can be set to OFPM_ALL to refer to all meters on the switch.

```
/* Body of OFPMP_METER and OFPMP_METER_CONFIG requests. */
struct ofp_meter_multipart_request {
    uint32_t meter_id;       /* Meter instance, or OFPM_ALL. */
    uint8_t pad[4];          /* Align to 64 bits. */
};
OFP_ASSERT(sizeof(struct ofp_meter_multipart_request) == 8);
```

The body of the reply consists of an array of the following structure:

```
/* Body of reply to OFPMP_METER request. Meter statistics. */
struct ofp_meter_stats {
    uint32_t        meter_id;         /* Meter instance. */
    uint16_t        len;              /* Length in bytes of this stats. */
    uint8_t         pad[6];
    uint32_t        flow_count;       /* Number of flows bound to meter. */
    uint64_t        packet_in_count;  /* Number of packets in input. */
    uint64_t        byte_in_count;    /* Number of bytes in input. */
    uint32_t   duration_sec;  /* Time meter has been alive in seconds. */
    uint32_t   duration_nsec; /* Time meter has been alive in nanoseconds beyond
```



```
                                duration_sec. */
    struct ofp_meter_band_stats band_stats[0]; /* The band_stats length is
                                          inferred from the length field. */
};
OFP_ASSERT(sizeof(struct ofp_meter_stats) == 40);
```

The `packet_in_count`, and the `byte_in_count` count all packets processed by the meter.

The `duration_sec` and `duration_nsec` fields indicate the elapsed time the meter has been installed in the switch. The total duration in nanoseconds can be computed as $duration\_sec \times 10^9 + duration\_nsec$. Implementations are required to provide second precision; higher precision is encouraged where available.

The `band_stats` field consists of an array of `ofp_meter_band_stats` structs:

```
/* Statistics for each meter band */
struct ofp_meter_band_stats {
    uint64_t        packet_band_count;   /* Number of packets in band. */
    uint64_t        byte_band_count;     /* Number of bytes in band. */
};
OFP_ASSERT(sizeof(struct ofp_meter_band_stats) == 16);
```

The `packet_band_count`, and the `byte_band_count` count all packets processed by the band.

The order of the band statistics must be the same as in the `OFPMT_METER_CONFIG` stats reply.

7.3.5.13 Meter Configuration Statistics

The `OFPMT_METER_CONFIG` stats request message provides configuration for one or more meter. The request body consists of a `meter_id` field, which can be set to `OFPM_ALL` to refer to all meters on the switch.

```
/* Body of OFPMP_METER and OFPMP_METER_CONFIG requests. */
struct ofp_meter_multipart_request {
    uint32_t meter_id;       /* Meter instance, or OFPM_ALL. */
    uint8_t pad[4];          /* Align to 64 bits. */
};
OFP_ASSERT(sizeof(struct ofp_meter_multipart_request) == 8);
```

The body of the reply consists of an array of the following structure:

```
/* Body of reply to OFPMP_METER_CONFIG request. Meter configuration. */
struct ofp_meter_config {
    uint16_t        length;          /* Length of this entry. */
    uint16_t        flags;           /* All OFPMC_* that apply. */
    uint32_t        meter_id;        /* Meter instance. */
    struct ofp_meter_band_header bands[0]; /* The bands length is
                                          inferred from the length field. */
};
OFP_ASSERT(sizeof(struct ofp_meter_config) == 8);
```

The fields are the same fields used for configuring the meter (see 7.3.4.4).

7.3.5.14 Meter Features Statistics

The OFPMT_METER_FEATURES stats request message provides the set of features of the metering subsystem. The request body is empty, and the body of the reply consists of the following structure:

```
/* Body of reply to OFPMP_METER_FEATURES request. Meter features. */
struct ofp_meter_features {
    uint32_t    max_meter;    /* Maximum number of meters. */
    uint32_t    band_types;   /* Bitmaps of OFPMBT_* values supported. */
    uint32_t    capabilities; /* Bitmaps of "ofp_meter_flags". */
    uint8_t     max_bands;    /* Maximum bands per meters */
    uint8_t     max_color;    /* Maximum color value */
    uint8_t     pad[2];
};
OFP_ASSERT(sizeof(struct ofp_meter_features) == 16);
```

7.3.5.15 Experimenter Multipart

Experimenter-specific multipart messages are requested with the OFPMP_EXPERIMENTER multipart type. The first bytes of the request and reply bodies are the following structure:

```
/* Body for ofp_multipart_request/reply of type OFPMP_EXPERIMENTER. */
struct ofp_experimenter_multipart_header {
    uint32_t experimenter;    /* Experimenter ID which takes the same form
                                 as in struct ofp_experimenter_header. */
    uint32_t exp_type;        /* Experimenter defined. */
    /* Experimenter-defined arbitrary additional data. */
};
OFP_ASSERT(sizeof(struct ofp_experimenter_multipart_header) == 8);
```

The rest of the request and reply bodies are experimenter-defined.

The experimenter field is the Experimenter ID, which takes the same form as in struct ofp_experimenter (see 7.5.4).

7.3.6 Queue Configuration Messages

Queue configuration takes place outside the OpenFlow protocol, either through a command line tool or through an external dedicated configuration protocol.

The controller can query the switch for configured queues on a port using the following structure:

```
/* Query for port queue configuration. */
struct ofp_queue_get_config_request {
    struct ofp_header header;
    uint32_t port;         /* Port to be queried. Should refer
                              to a valid physical port (i.e. < OFPP_MAX),
                              or OFPP_ANY to request all configured
                              queues.*/
    uint8_t pad[4];
};
OFP_ASSERT(sizeof(struct ofp_queue_get_config_request) == 16);
```


The switch replies back with an ofp_queue_get_config_reply command, containing a list of configured queues.

```
/* Queue configuration for a given port. */
struct ofp_queue_get_config_reply {
    struct ofp_header header;
    uint32_t port;
    uint8_t pad[4];
    struct ofp_packet_queue queues[0]; /* List of configured queues. */
};
OFP_ASSERT(sizeof(struct ofp_queue_get_config_reply) == 16);
```

7.3.7 Packet-Out Message

When the controller wishes to send a packet out through the datapath, it uses the OFPT_PACKET_OUT message:

```
/* Send packet (controller -> datapath). */
struct ofp_packet_out {
    struct ofp_header header;
    uint32_t buffer_id;           /* ID assigned by datapath (OFP_NO_BUFFER
                                     if none). */
    uint32_t in_port;             /* Packet's input port or OFPP_CONTROLLER. */
    uint16_t actions_len;         /* Size of action array in bytes. */
    uint8_t pad[6];
    struct ofp_action_header actions[0]; /* Action list - 0 or more. */
    /* The variable size action list is optionally followed by packet data.
     * This data is only present and meaningful if buffer_id == -1.
    /* uint8_t data[0]; */        /* Packet data.  The length is inferred
                                     from the length field in the header. */
};
OFP_ASSERT(sizeof(struct ofp_packet_out) == 24);
```

The buffer_id is the same given in the ofp_packet_in message. If the buffer_id is OFP_NO_BUFFER, then the packet data is included in the data array, and the packet encapsulated in the message is processed by the actions of the message. OFP_NO_BUFFER is 0xffffffff. If buffer_id is valid, the corresponding packet is removed from the buffer and processed by the actions of the message.

The in_port field is the ingress port that must be associated with the packet for OpenFlow processing. It must be set to either a valid standard switch port (see 4.2) or OFPP_CONTROLLER. For example, this field is used when processing the packet using OFPP_TABLE, OFPP_IN_PORT, OFPP_ALL and groups.

The action field is an action list defining how the packet should be processed by the switch. It may include packet modification, group processing and an output port. The action list of an OFPT_PACKET_OUT message can also specify the OFPP_TABLE reserved port as an output action to process the packet through the OpenFlow pipeline, starting at the first flow table (see 4.5). If OFPP_TABLE is specified, the in_port field is used as the ingress port in the flow table lookup.

In some cases, packets sent to OFPP_TABLE may be forwarded back to the controller as the result of a flow entry action or table miss. Detecting and taking action for such controller-to-switch loops is outside the scope of this specification. In general, OpenFlow messages are not guaranteed to be processed in order,

therefore if a OFPT_PACKET_OUT message using OFPP_TABLE depends on a flow that was recently sent to the switch (with a OFPT_FLOW_MOD message), a OFPT_BARRIER_REQUEST message may be required prior to the OFPT_PACKET_OUT message to make sure the flow entry was committed to the flow table prior to execution of OFPP_TABLE.

7.3.8 Barrier Message

When the controller wants to ensure message dependencies have been met or wants to receive notifications for completed operations, it may use an OFPT_BARRIER_REQUEST message. This message has no body. Upon receipt, the switch must finish processing all previously-received messages, including sending corresponding reply or error messages, before executing any messages beyond the Barrier Request. When such processing is complete, the switch must send an OFPT_BARRIER_REPLY message with the xid of the original request.

7.3.9 Role Request Message

When the controller wants to change its role, it uses the OFPT_ROLE_REQUEST message with the following structure:

```
/* Role request and reply message. */
struct ofp_role_request {
    struct ofp_header header;   /* Type OFPT_ROLE_REQUEST/OFPT_ROLE_REPLY. */
    uint32_t role;              /* One of OFPCR_ROLE_*. */
    uint8_t pad[4];             /* Align to 64 bits. */
    uint64_t generation_id;     /* Master Election Generation Id */
};
OFP_ASSERT(sizeof(struct ofp_role_request) == 24);
```

The field role is the new role that the controller wants to assume, and can have the following values:

```
/* Controller roles. */
enum ofp_controller_role {
    OFPCR_ROLE_NOCHANGE = 0,    /* Don't change current role. */
    OFPCR_ROLE_EQUAL    = 1,    /* Default role, full access. */
    OFPCR_ROLE_MASTER   = 2,    /* Full access, at most one master. */
    OFPCR_ROLE_SLAVE    = 3,    /* Read-only access. */
};
```

If the role value in the message is OFPCR_ROLE_MASTER or OFPCR_ROLE_SLAVE, the switch must validate generation_id to check for stale messages (see 6.3.4). If the validation fails, the switch must discard the role request and return an error message with type OFPET_ROLE_REQUEST_FAILED and code OFPRRFC_STALE.

If the role value is OFPCR_ROLE_MASTER, all other controllers whose role was OFPCR_ROLE_MASTER are changed to OFPCR_ROLE_SLAVE (see 6.3.4). If the role value is OFPCR_ROLE_NOCHANGE, the current role of the controller is not changed ; this enables a controller to query its current role without changing it.

Upon receipt of a OFPT_ROLE_REQUEST message, if there is no error, the switch must return a OFPT_ROLE_REPLY message. The structure of this message is exactly the same as the OFPT_ROLE_REQUEST message, and the field role is the current role of the controller. The field generation_id is set to the current generation_id (the generation_id associated with the last successful role request with role OFPCR_ROLE_MASTER or OFPCR_ROLE_SLAVE), if the current generation_id was never set by a controller, the field generation_id in the reply must be set to the maximum field value (the unsigned equivalent of -1).

7.3.10 Set Asynchronous Configuration Message

The controller is able to set and query the asynchronous messages that it wants to receive (other than error messages) on a given OpenFlow channel with the OFPT_SET_ASYNC and OFPT_GET_ASYNC_REQUEST messages, respectively. The switch responds to a OFPT_GET_ASYNC_REQUEST message with an OFPT_GET_ASYNC_REPLY message; it does not reply to a request to set the configuration.

There is no body for OFPT_GET_ASYNC_REQUEST beyond the OpenFlow header. The OFPT_SET_ASYNC and OFPT_GET_ASYNC_REPLY messages have the following format:

```
/* Asynchronous message configuration. */
struct ofp_async_config {
    struct ofp_header header;     /* OFPT_GET_ASYNC_REPLY or OFPT_SET_ASYNC. */
    uint32_t packet_in_mask[2];   /* Bitmasks of OFPR_* values. */
    uint32_t port_status_mask[2]; /* Bitmasks of OFPPR_* values. */
    uint32_t flow_removed_mask[2];/* Bitmasks of OFPRR_* values. */
};
OFP_ASSERT(sizeof(struct ofp_async_config) == 32);
```

struct ofp_async_config contains three 2-element arrays. Each array controls whether the controller receives asynchronous messages with a specific enum ofp_type. Within each array, element 0 specifies messages of interest when the controller has a OFPCR_ROLE_EQUAL or OFPCR_ROLE_MASTER role; element 1, when the controller has a OFPCR_ROLE_SLAVE role. Each array element is a bit-mask in which a 0-bit disables receiving a message sent with the reason code corresponding to the bit index and a 1-bit enables receiving it. For example, the bit with value $2^2 = 4$ in port_status_mask[1] determines whether the controller will receive OFPT_PORT_STATUS messages with reason OFPPR_MODIFY (value 2) when the controller has role OFPCR_ROLE_SLAVE.

OFPT_SET_ASYNC sets whether a controller should receive a given asynchronous message that is generated by the switch. Other OpenFlow features control whether a given message is generated; for example, the OFPFF_SEND_FLOW_REM flag controls whether the switch generates OFPT_FLOW_REMOVED a message when a flow entry is removed.

A switch configuration, for example using the OpenFlow Configuration Protocol, may set the initial configuration of asynchronous messages when an OpenFlow connection is established. In the absence of such switch configuration, the initial configuration shall be:

- In the "master" or "equal" role, enable all OFPT_PACKET_IN messages, except those with reason OFPR_INVALID_TTL, and enable all OFPT_PORT_STATUS and OFPT_FLOW_REMOVED messages.

- In the "slave" role, enable all OFPT_PORT_STATUS messages and disable all OFPT_PACKET_IN and OFPT_FLOW_REMOVED messages.

The configuration set with OFPT_SET_ASYNC is specific to a particular OpenFlow channel. It does not affect any other OpenFlow channel, whether currently established or to be established in the future.

The configuration set with OFPT_SET_ASYNC does not filter or otherwise affect error messages.

7.4 Asynchronous Messages

7.4.1 Packet-In Message

When packets are received by the datapath and sent to the controller, they use the OFPT_PACKET_IN message:

```
/* Packet received on port (datapath -> controller). */
struct ofp_packet_in {
    struct ofp_header header;
    uint32_t buffer_id;     /* ID assigned by datapath. */
    uint16_t total_len;     /* Full length of frame. */
    uint8_t reason;         /* Reason packet is being sent (one of OFPR_*) */
    uint8_t table_id;       /* ID of the table that was looked up */
    uint64_t cookie;        /* Cookie of the flow entry that was looked up. */
    struct ofp_match match; /* Packet metadata. Variable size. */
    /* The variable size and padded match is always followed by:
     *   - Exactly 2 all-zero padding bytes, then
     *   - An Ethernet frame whose length is inferred from header.length.
     * The padding bytes preceding the Ethernet frame ensure that the IP
     * header (if any) following the Ethernet header is 32-bit aligned.
     */
    //uint8_t pad[2];       /* Align to 64 bit + 16 bit */
    //uint8_t data[0];      /* Ethernet frame */
};
OFP_ASSERT(sizeof(struct ofp_packet_in) == 32);
```

The buffer_id is an opaque value used by the datapath to identify a buffered packet. When a packet is buffered, some number of bytes from the message will be included in the data portion of the message. If the packet is sent because of a "send to controller" action, then max_len bytes from the ofp_action_output of the flow setup request are sent. If the packet is sent for other reasons, such as an invalid TTL, then at least miss_send_len bytes from the OFPT_SET_CONFIG message are sent. The default miss_send_len is 128 bytes. If the packet is not buffered - either because of no available buffers, or because of explicitly requested via OFPCML_NO_BUFFER - the entire packet is included in the data portion, and the buffer_id is OFP_NO_BUFFER.

Switches that implement buffering are expected to expose, through documentation, both the amount of available buffering, and the length of time before buffers may be reused. A switch must gracefully handle the case where a buffered packet_in message yields no response from the controller. A switch should prevent a buffer from being reused until it has been handled by the controller, or some amount of time (indicated in documentation) has passed.

The data field contains the packet itself, or a fraction of the packet if the packet is buffered. The packet header reflect any changes applied to the packet in previous processing.

The reason field can be any of these values:

```
/* Why is this packet being sent to the controller? */
enum ofp_packet_in_reason {
    OFPR_NO_MATCH    = 0,   /* No matching flow (table-miss flow entry). */
    OFPR_ACTION      = 1,   /* Action explicitly output to controller. */
    OFPR_INVALID_TTL = 2,   /* Packet has invalid TTL */
};
```

OFPR_INVALID_TTL indicates that a packets with an invalid IP TTL or MPLS TTL was rejected by the OpenFlow pipeline and passed to the controller. Checking for invalid TTL does not need to be done for every packet, but it must be done at a minimum every time a OFPAT_DEC_MPLS_TTL or OFPAT_DEC_NW_TTL action is applied to a packet.

The cookie field contains the cookie of the flow entry that caused the packet to be sent to the controller. This field must be set to -1 (0xffffffffffffffff) if a cookie cannot be associated with a particular flow. For example, if the packet-in was generated in a group bucket or from the action set.

The match field reflect the packet's headers and context when the event that triggers the packet-in message occurred and contains a set of OXM TLVs. This context includes any changes applied to the packet in previous processing, including actions already executed, if any, but not any changes in the action set. The OXM TLVs must include context fields, that is, fields whose values cannot be determined from the packet data. The standard context fields are OFPXMT_OFB_IN_PORT, OFPXMT_OFB_IN_PHY_PORT, OFPXMT_OFB_METADATA and OFPXMT_OFB_TUNNEL_ID. Fields whose values are all-bits-zero should be omitted. Optionally, the OXM TLVs may also include packet header fields that were previously extracted from the packet, including any modifications of those in the course of the processing.

When a packet is received directly on a physical port and not processed by a logical port, OFPXMT_OFB_IN_PORT and OFPXMT_OFB_IN_PHY_PORT have the same value, the OpenFlow port_no of this physical port. OFPXMT_OFB_IN_PHY_PORT should be omitted if it has the same value as OFPXMT_OFB_IN_PORT.

When a packet is received on a logical port by way of a physical port, OFPXMT_OFB_IN_PORT is the logical port's port_no and OFPXMT_OFB_IN_PHY_PORT is the physical port's port_no. For example, consider a packet received on a tunnel interface defined over a link aggregation group (LAG) with two physical port members. If the tunnel interface is the logical port bound to OpenFlow, then OFPXMT_OFB_IN_PORT is the tunnel port_no and OFPXMT_OFB_IN_PHY_PORT is the physical port_no member of the LAG on which the tunnel is configured.

The port referenced by the OFPXMT_OFB_IN_PORT TLV must be the port used for matching flow entries (see 5.3) and must be available to OpenFlow processing (i.e. OpenFlow can forward packet to this port, depending on port flags). OFPXMT_OFB_IN_PHY_PORT need not be available for matching or OpenFlow processing.

7.4.2 Flow Removed Message

If the controller has requested to be notified when flow entries time out or are deleted from tables (see 5.5), the datapath does this with the `OFPT_FLOW_REMOVED` message:

```
/* Flow removed (datapath -> controller). */
struct ofp_flow_removed {
    struct ofp_header header;
    uint64_t cookie;          /* Opaque controller-issued identifier. */

    uint16_t priority;        /* Priority level of flow entry. */
    uint8_t reason;           /* One of OFPRR_*. */
    uint8_t table_id;         /* ID of the table */

    uint32_t duration_sec;    /* Time flow was alive in seconds. */
    uint32_t duration_nsec;   /* Time flow was alive in nanoseconds beyond
                                 duration_sec. */
    uint16_t idle_timeout;    /* Idle timeout from original flow mod. */
    uint16_t hard_timeout;    /* Hard timeout from original flow mod. */
    uint64_t packet_count;
    uint64_t byte_count;
    struct ofp_match match;   /* Description of fields. Variable size. */
};
OFP_ASSERT(sizeof(struct ofp_flow_removed) == 56);
```

The `match`, `cookie`, and `priority` fields are the same as those used in the flow mod request.

The `reason` field is one of the following:

```
/* Why was this flow removed? */
enum ofp_flow_removed_reason {
    OFPRR_IDLE_TIMEOUT = 0,     /* Flow idle time exceeded idle_timeout. */
    OFPRR_HARD_TIMEOUT = 1,     /* Time exceeded hard_timeout. */
    OFPRR_DELETE       = 2,     /* Evicted by a DELETE flow mod. */
    OFPRR_GROUP_DELETE = 3,     /* Group was removed. */
};
```

The `duration_sec` and `duration_nsec` fields are described in Section 7.3.5.2.

The `idle_timeout` and `hard_timeout` fields are directly copied from the flow mod that created this entry.

With the above three fields, one can find both the amount of time the flow entry was active, as well as the amount of time the flow entry received traffic.

The `packet_count` and `byte_count` indicate the number of packets and bytes that were associated with this flow entry, respectively. Those counters should behave like other statistics counters (see 7.3.5) ; they are unsigned and should be set to the maximum field value if not available.

7.4.3 Port Status Message

As ports are added, modified, and removed from the datapath, the controller needs to be informed with the OFPT_PORT_STATUS message:

```
/* A physical port has changed in the datapath */
struct ofp_port_status {
    struct ofp_header header;
    uint8_t reason;          /* One of OFPPR_*. */
    uint8_t pad[7];          /* Align to 64-bits. */
    struct ofp_port desc;
};
OFP_ASSERT(sizeof(struct ofp_port_status) == 80);
```

The reason can be one of the following values:

```
/* What changed about the physical port */
enum ofp_port_reason {
    OFPPR_ADD     = 0,         /* The port was added. */
    OFPPR_DELETE  = 1,         /* The port was removed. */
    OFPPR_MODIFY  = 2,         /* Some attribute of the port has changed. */
};
```

7.4.4 Error Message

There are times that the switch needs to notify the controller of a problem. This is done with the OFPT_ERROR_MSG message:

```
/* OFPT_ERROR: Error message (datapath -> controller). */
struct ofp_error_msg {
    struct ofp_header header;

    uint16_t type;
    uint16_t code;
    uint8_t data[0];          /* Variable-length data.  Interpreted based
                                 on the type and code.  No padding. */
};
OFP_ASSERT(sizeof(struct ofp_error_msg) == 12);
```

The type value indicates the high-level type of error. The code value is interpreted based on the type. The data is variable length and interpreted based on the type and code. Unless specified otherwise, the data field contains at least 64 bytes of the failed request that caused the error message to be generated, if the failed request is shorter than 64 bytes it should be the full request without any padding.

If the error message is in response to a specific message from the controller, e.g., OFPET_BAD_REQUEST, OFPET_BAD_ACTION, OFPET_BAD_INSTRUCTION, OFPET_BAD_MATCH, or OFPET_FLOW_MOD_FAILED, then the xid field of the header must match that of the offending message.

Error codes ending in _EPERM correspond to a permissions error generated by, for example, an OpenFlow hypervisor interposing between a controller and switch.

Currently defined error types are:


```
/* Values for 'type' in ofp_error_message.  These values are immutable: they
 * will not change in future versions of the protocol (although new values may
 * be added). */
enum ofp_error_type {
    OFPET_HELLO_FAILED         = 0,  /* Hello protocol failed. */
    OFPET_BAD_REQUEST          = 1,  /* Request was not understood. */
    OFPET_BAD_ACTION           = 2,  /* Error in action description. */
    OFPET_BAD_INSTRUCTION      = 3,  /* Error in instruction list. */
    OFPET_BAD_MATCH            = 4,  /* Error in match. */
    OFPET_FLOW_MOD_FAILED      = 5,  /* Problem modifying flow entry. */
    OFPET_GROUP_MOD_FAILED     = 6,  /* Problem modifying group entry. */
    OFPET_PORT_MOD_FAILED      = 7,  /* Port mod request failed. */
    OFPET_TABLE_MOD_FAILED     = 8,  /* Table mod request failed. */
    OFPET_QUEUE_OP_FAILED      = 9,  /* Queue operation failed. */
    OFPET_SWITCH_CONFIG_FAILED = 10, /* Switch config request failed. */
    OFPET_ROLE_REQUEST_FAILED  = 11, /* Controller Role request failed. */
    OFPET_METER_MOD_FAILED     = 12, /* Error in meter. */
    OFPET_TABLE_FEATURES_FAILED = 13, /* Setting table features failed. */
    OFPET_EXPERIMENTER = 0xffff      /* Experimenter error messages. */
};
```

For the OFPET_HELLO_FAILED error type, the following codes are currently defined:

```
/* ofp_error_msg 'code' values for OFPET_HELLO_FAILED.  'data' contains an
 * ASCII text string that may give failure details. */
enum ofp_hello_failed_code {
    OFPHFC_INCOMPATIBLE = 0,    /* No compatible version. */
    OFPHFC_EPERM        = 1,    /* Permissions error. */
};
```

The data field contains an ASCII text string that adds detail on why the error occurred.

For the OFPET_BAD_REQUEST error type, the following codes are currently defined:

```
/* ofp_error_msg 'code' values for OFPET_BAD_REQUEST.  'data' contains at least
 * the first 64 bytes of the failed request. */
enum ofp_bad_request_code {
    OFPBRC_BAD_VERSION      = 0,  /* ofp_header.version not supported. */
    OFPBRC_BAD_TYPE         = 1,  /* ofp_header.type not supported. */
    OFPBRC_BAD_MULTIPART    = 2,  /* ofp_multipart_request.type not supported. */
    OFPBRC_BAD_EXPERIMENTER = 3,  /* Experimenter id not supported
                                   * (in ofp_experimenter_header or
                                   * ofp_multipart_request or
                                   * ofp_multipart_reply). */
    OFPBRC_BAD_EXP_TYPE     = 4,  /* Experimenter type not supported. */
    OFPBRC_EPERM            = 5,  /* Permissions error. */
    OFPBRC_BAD_LEN          = 6,  /* Wrong request length for type. */
    OFPBRC_BUFFER_EMPTY     = 7,  /* Specified buffer has already been used. */
    OFPBRC_BUFFER_UNKNOWN   = 8,  /* Specified buffer does not exist. */
    OFPBRC_BAD_TABLE_ID     = 9,  /* Specified table-id invalid or does not
                                   * exist. */
    OFPBRC_IS_SLAVE         = 10, /* Denied because controller is slave. */
    OFPBRC_BAD_PORT         = 11, /* Invalid port. */
    OFPBRC_BAD_PACKET       = 12, /* Invalid packet in packet-out. */
    OFPBRC_MULTIPART_BUFFER_OVERFLOW    = 13, /* ofp_multipart_request
```



```
                                    overflowed the assigned buffer. */
};
```

For the OFPET_BAD_ACTION error type, the following codes are currently defined:

```
/* ofp_error_msg 'code' values for OFPET_BAD_ACTION.  'data' contains at least
 * the first 64 bytes of the failed request. */
enum ofp_bad_action_code {
    OFPBAC_BAD_TYPE           = 0,  /* Unknown action type. */
    OFPBAC_BAD_LEN            = 1,  /* Length problem in actions. */
    OFPBAC_BAD_EXPERIMENTER   = 2,  /* Unknown experimenter id specified. */
    OFPBAC_BAD_EXP_TYPE       = 3,  /* Unknown action for experimenter id. */
    OFPBAC_BAD_OUT_PORT       = 4,  /* Problem validating output port. */
    OFPBAC_BAD_ARGUMENT       = 5,  /* Bad action argument. */
    OFPBAC_EPERM              = 6,  /* Permissions error. */
    OFPBAC_TOO_MANY           = 7,  /* Can't handle this many actions. */
    OFPBAC_BAD_QUEUE          = 8,  /* Problem validating output queue. */
    OFPBAC_BAD_OUT_GROUP      = 9,  /* Invalid group id in forward action. */
    OFPBAC_MATCH_INCONSISTENT = 10, /* Action can't apply for this match,
                                       or Set-Field missing prerequisite. */
    OFPBAC_UNSUPPORTED_ORDER  = 11, /* Action order is unsupported for the
                                     action list in an Apply-Actions instruction */
    OFPBAC_BAD_TAG            = 12, /* Actions uses an unsupported
                                       tag/encap. */
    OFPBAC_BAD_SET_TYPE       = 13, /* Unsupported type in SET_FIELD action. */
    OFPBAC_BAD_SET_LEN        = 14, /* Length problem in SET_FIELD action. */
    OFPBAC_BAD_SET_ARGUMENT   = 15, /* Bad argument in SET_FIELD action. */
};
```

For the OFPET_BAD_INSTRUCTION error type, the following codes are currently defined:

```
/* ofp_error_msg 'code' values for OFPET_BAD_INSTRUCTION.  'data' contains at least
 * the first 64 bytes of the failed request. */
enum ofp_bad_instruction_code {
    OFPBIC_UNKNOWN_INST     = 0, /* Unknown instruction. */
    OFPBIC_UNSUP_INST       = 1, /* Switch or table does not support the
                                    instruction. */
    OFPBIC_BAD_TABLE_ID     = 2, /* Invalid Table-ID specified. */
    OFPBIC_UNSUP_METADATA   = 3, /* Metadata value unsupported by datapath. */
    OFPBIC_UNSUP_METADATA_MASK = 4, /* Metadata mask value unsupported by
                                       datapath. */
    OFPBIC_BAD_EXPERIMENTER = 5, /* Unknown experimenter id specified. */
    OFPBIC_BAD_EXP_TYPE     = 6, /* Unknown instruction for experimenter id. */
    OFPBIC_BAD_LEN          = 7, /* Length problem in instructions. */
    OFPBIC_EPERM            = 8, /* Permissions error. */
};
```

For the OFPET_BAD_MATCH error type, the following codes are currently defined:

```
/* ofp_error_msg 'code' values for OFPET_BAD_MATCH.  'data' contains at least
 * the first 64 bytes of the failed request. */
enum ofp_bad_match_code {
    OFPBMC_BAD_TYPE         = 0,  /* Unsupported match type specified by the
```



```
                                   match */
    OFPBMC_BAD_LEN          = 1,  /* Length problem in match. */
    OFPBMC_BAD_TAG          = 2,  /* Match uses an unsupported tag/encap. */
    OFPBMC_BAD_DL_ADDR_MASK = 3,  /* Unsupported datalink addr mask - switch
                                     does not support arbitrary datalink
                                     address mask. */
    OFPBMC_BAD_NW_ADDR_MASK = 4,  /* Unsupported network addr mask - switch
                                     does not support arbitrary network
                                     address mask. */
    OFPBMC_BAD_WILDCARDS    = 5,  /* Unsupported combination of fields masked
                                     or omitted in the match. */
    OFPBMC_BAD_FIELD        = 6,  /* Unsupported field type in the match. */
    OFPBMC_BAD_VALUE        = 7,  /* Unsupported value in a match field. */
    OFPBMC_BAD_MASK         = 8,  /* Unsupported mask specified in the match,
                                     field is not dl-address or nw-address. */
    OFPBMC_BAD_PREREQ       = 9,  /* A prerequisite was not met. */
    OFPBMC_DUP_FIELD        = 10, /* A field type was duplicated. */
    OFPBMC_EPERM            = 11, /* Permissions error. */
};
```

For the OFPET_FLOW_MOD_FAILED error type, the following codes are currently defined:

```
/* ofp_error_msg 'code' values for OFPET_FLOW_MOD_FAILED.  'data' contains
 * at least the first 64 bytes of the failed request. */
enum ofp_flow_mod_failed_code {
    OFPFMFC_UNKNOWN      = 0,   /* Unspecified error. */
    OFPFMFC_TABLE_FULL   = 1,   /* Flow not added because table was full. */
    OFPFMFC_BAD_TABLE_ID = 2,   /* Table does not exist */
    OFPFMFC_OVERLAP      = 3,   /* Attempted to add overlapping flow with
                                   CHECK_OVERLAP flag set. */
    OFPFMFC_EPERM        = 4,   /* Permissions error. */
    OFPFMFC_BAD_TIMEOUT  = 5,   /* Flow not added because of unsupported
                                   idle/hard timeout. */
    OFPFMFC_BAD_COMMAND  = 6,   /* Unsupported or unknown command. */
    OFPFMFC_BAD_FLAGS    = 7,   /* Unsupported or unknown flags. */
};
```

For the OFPET_GROUP_MOD_FAILED error type, the following codes are currently defined:

```
/* ofp_error_msg 'code' values for OFPET_GROUP_MOD_FAILED.  'data' contains
 * at least the first 64 bytes of the failed request. */
enum ofp_group_mod_failed_code {
    OFPGMFC_GROUP_EXISTS         = 0,  /* Group not added because a group ADD
                                          attempted to replace an
                                          already-present group. */
    OFPGMFC_INVALID_GROUP        = 1,  /* Group not added because Group
                                          specified is invalid. */
    OFPGMFC_WEIGHT_UNSUPPORTED   = 2,  /* Switch does not support unequal load
                                          sharing with select groups. */
    OFPGMFC_OUT_OF_GROUPS        = 3,  /* The group table is full. */
    OFPGMFC_OUT_OF_BUCKETS       = 4,  /* The maximum number of action buckets
                                          for a group has been exceeded. */
    OFPGMFC_CHAINING_UNSUPPORTED = 5,  /* Switch does not support groups that
                                          forward to groups. */
```



```
    OFPGMFC_WATCH_UNSUPPORTED    = 6,  /* This group cannot watch the watch_port
                                          or watch_group specified. */
    OFPGMFC_LOOP                 = 7,  /* Group entry would cause a loop. */
    OFPGMFC_UNKNOWN_GROUP        = 8,  /* Group not modified because a group
                                          MODIFY attempted to modify a
                                          non-existent group. */
    OFPGMFC_CHAINED_GROUP        = 9,  /* Group not deleted because another
                                          group is forwarding to it. */
    OFPGMFC_BAD_TYPE             = 10, /* Unsupported or unknown group type. */
    OFPGMFC_BAD_COMMAND          = 11, /* Unsupported or unknown command. */
    OFPGMFC_BAD_BUCKET           = 12, /* Error in bucket. */
    OFPGMFC_BAD_WATCH            = 13, /* Error in watch port/group. */
    OFPGMFC_EPERM                = 14, /* Permissions error. */
};
```

For the OFPET_PORT_MOD_FAILED error type, the following codes are currently defined:

```
/* ofp_error_msg 'code' values for OFPET_PORT_MOD_FAILED.  'data' contains
 * at least the first 64 bytes of the failed request. */
enum ofp_port_mod_failed_code {
    OFPPMFC_BAD_PORT      = 0,   /* Specified port number does not exist. */
    OFPPMFC_BAD_HW_ADDR   = 1,   /* Specified hardware address does not
                                  * match the port number. */
    OFPPMFC_BAD_CONFIG    = 2,   /* Specified config is invalid. */
    OFPPMFC_BAD_ADVERTISE = 3,   /* Specified advertise is invalid. */
    OFPPMFC_EPERM         = 4,   /* Permissions error. */
};
```

For the OFPET_TABLE_MOD_FAILED error type, the following codes are currently defined:

```
/* ofp_error_msg 'code' values for OFPET_TABLE_MOD_FAILED.  'data' contains
 * at least the first 64 bytes of the failed request. */
enum ofp_table_mod_failed_code {
    OFPTMFC_BAD_TABLE  = 0,      /* Specified table does not exist. */
    OFPTMFC_BAD_CONFIG = 1,      /* Specified config is invalid. */
    OFPTMFC_EPERM      = 2,      /* Permissions error. */
};
```

For the OFPET_QUEUE_OP_FAILED error type, the following codes are currently defined:

```
/* ofp_error msg 'code' values for OFPET_QUEUE_OP_FAILED. 'data' contains
 * at least the first 64 bytes of the failed request */
enum ofp_queue_op_failed_code {
    OFPQOFC_BAD_PORT   = 0,     /* Invalid port (or port does not exist). */
    OFPQOFC_BAD_QUEUE  = 1,     /* Queue does not exist. */
    OFPQOFC_EPERM      = 2,     /* Permissions error. */
};
```

For the OFPET_SWITCH_CONFIG_FAILED error type, the following codes are currently defined:


```
/* ofp_error_msg 'code' values for OFPET_SWITCH_CONFIG_FAILED. 'data' contains
 * at least the first 64 bytes of the failed request. */
enum ofp_switch_config_failed_code {
    OFPSCFC_BAD_FLAGS  = 0,      /* Specified flags is invalid. */
    OFPSCFC_BAD_LEN    = 1,      /* Specified len is invalid. */
    OFPSCFC_EPERM      = 2,      /* Permissions error. */
};
```

For the OFPET_ROLE_REQUEST_FAILED error type, the following codes are currently defined:

```
/* ofp_error_msg 'code' values for OFPET_ROLE_REQUEST_FAILED. 'data' contains
 * at least the first 64 bytes of the failed request. */
enum ofp_role_request_failed_code {
    OFPRRFC_STALE      = 0,      /* Stale Message: old generation_id. */
    OFPRRFC_UNSUP      = 1,      /* Controller role change unsupported. */
    OFPRRFC_BAD_ROLE   = 2,      /* Invalid role. */
};
```

For the OFPET_METER_MOD_FAILED error type, the following codes are currently defined:

```
/* ofp_error_msg 'code' values for OFPET_METER_MOD_FAILED.  'data' contains
 * at least the first 64 bytes of the failed request. */
enum ofp_meter_mod_failed_code {
    OFPMMFC_UNKNOWN       = 0,  /* Unspecified error. */
    OFPMMFC_METER_EXISTS  = 1,  /* Meter not added because a Meter ADD
                                 * attempted to replace an existing Meter. */
    OFPMMFC_INVALID_METER = 2,      /* Meter not added because Meter specified
                                 * is invalid. */
    OFPMMFC_UNKNOWN_METER = 3,  /* Meter not modified because a Meter
                                   MODIFY attempted to modify a non-existent
                                   Meter. */
    OFPMMFC_BAD_COMMAND   = 4,  /* Unsupported or unknown command. */
    OFPMMFC_BAD_FLAGS     = 5,  /* Flag configuration unsupported. */
    OFPMMFC_BAD_RATE      = 6,  /* Rate unsupported. */
    OFPMMFC_BAD_BURST     = 7,  /* Burst size unsupported. */
    OFPMMFC_BAD_BAND      = 8,  /* Band unsupported. */
    OFPMMFC_BAD_BAND_VALUE = 9, /* Band value unsupported. */
    OFPMMFC_OUT_OF_METERS = 10, /* No more meters available. */
    OFPMMFC_OUT_OF_BANDS  = 11, /* The maximum number of properties
                                 * for a meter has been exceeded. */
};
```

For the OFPET_TABLE_FEATURES_FAILED error type, the following codes are currently defined:

```
/* ofp_error_msg 'code' values for OFPET_TABLE_FEATURES_FAILED. 'data' contains
 * at least the first 64 bytes of the failed request. */
enum ofp_table_features_failed_code {
    OFPTFFC_BAD_TABLE    = 0,      /* Specified table does not exist. */
    OFPTFFC_BAD_METADATA = 1,      /* Invalid metadata mask. */
    OFPTFFC_BAD_TYPE     = 2,      /* Unknown property type. */
    OFPTFFC_BAD_LEN      = 3,      /* Length problem in properties. */
    OFPTFFC_BAD_ARGUMENT = 4,      /* Unsupported property value. */
    OFPTFFC_EPERM        = 5,      /* Permissions error. */
};
```


For the OFPET_EXPERIMENTER error type, the error message is defined by the following structure and fields, followed by experimenter defined data:

```
/* OFPET_EXPERIMENTER: Error message (datapath -> controller). */
struct ofp_error_experimenter_msg {
    struct ofp_header header;

    uint16_t type;            /* OFPET_EXPERIMENTER. */
    uint16_t exp_type;        /* Experimenter defined. */
    uint32_t experimenter;    /* Experimenter ID which takes the same form
                                 as in struct ofp_experimenter_header. */
    uint8_t data[0];          /* Variable-length data.  Interpreted based
                                 on the type and code.  No padding. */
};
OFP_ASSERT(sizeof(struct ofp_error_experimenter_msg) == 16);
```

The experimenter field is the Experimenter ID, which takes the same form as in struct ofp_experimenter (see 7.5.4).

7.5 Symmetric Messages

7.5.1 Hello

The OFPT_HELLO message consists of an OpenFlow header plus a set of variable size hello elements.

```
/* OFPT_HELLO.  This message includes zero or more hello elements having
 * variable size. Unknown elements types must be ignored/skipped, to allow
 * for future extensions. */
struct ofp_hello {
    struct ofp_header header;

    /* Hello element list */
    struct ofp_hello_elem_header elements[0]; /* List of elements - 0 or more */
};
OFP_ASSERT(sizeof(struct ofp_hello) == 8);
```

The version field part of the header field (see 7.1) must be set to the highest OpenFlow protocol version supported by the sender (see 6.3.1).

The elements field is a set of hello elements, containing optional data to inform the initial handshake of the connection. Implementations must ignore (skip) all elements of a Hello message that they do not support. The list of hello elements types that are currently defined are:

```
/* Hello elements types.
 */
enum ofp_hello_elem_type {
    OFPHET_VERSIONBITMAP          = 1,  /* Bitmap of version supported. */
};
```

An element definition contains the element type, length, and any associated data:


```
/* Common header for all Hello Elements */
struct ofp_hello_elem_header {
    uint16_t         type;    /* One of OFPHET_*. */
    uint16_t         length;  /* Length in bytes of this element. */
};
OFP_ASSERT(sizeof(struct ofp_hello_elem_header) == 4);
```

The `OFPHET_VERSIONBITMAP` element use the following structure and fields:

```
/* Version bitmap Hello Element */
struct ofp_hello_elem_versionbitmap {
    uint16_t         type;    /* OFPHET_VERSIONBITMAP. */
    uint16_t         length;  /* Length in bytes of this element. */
    /* Followed by:
     *   - Exactly (length - 4) bytes containing the bitmaps, then
     *   - Exactly (length + 7)/8*8 - (length) (between 0 and 7)
     *     bytes of all-zero bytes */
    uint32_t         bitmaps[0];   /* List of bitmaps - supported versions */
};
OFP_ASSERT(sizeof(struct ofp_hello_elem_versionbitmap) == 4);
```

The `bitmaps` field indicates the set of versions of the OpenFlow switch protocol a device supports, and may be used during version negotiation (see 6.3.1). The bits of the set of bitmaps are indexed by the *ofp_version* number of the protocol ; if the bit identified by the number of left bitshift equal to a *ofp_version* number is set, this OpenFlow version is supported. The number of bitmaps included in the field depend on the highest version number supported : *ofp_versions* 0 to 31 are encoded in the first bitmap, *ofp_versions* 32 to 63 are encoded in the second bitmap and so on. For example, if a switch supports only version 1.0 (*ofp_version*=0x01) and version 1.3 (*ofp_version*=0x04), the first bitmap would be set to 0x00000012.

7.5.2 Echo Request

An Echo Request message consists of an OpenFlow header plus an arbitrary-length data field. The data field might be a message timestamp to check latency, various lengths to measure bandwidth, or zero-size to verify liveness between the switch and controller.

7.5.3 Echo Reply

An Echo Reply message consists of an OpenFlow header plus the unmodified data field of an echo request message.

In an OpenFlow protocol implementation divided into multiple layers, the echo request/reply logic should be implemented in the "deepest" practical layer. For example, in the OpenFlow reference implementation that includes a userspace process that relays to a kernel module, echo request/reply is implemented in the kernel module. Receiving a correctly formatted echo reply then shows a greater likelihood of correct end-to-end functionality than if the echo request/reply were implemented in the userspace process, as well as providing more accurate end-to-end latency timing.

7.5.4 Experimenter

The Experimenter message is defined as follows:

```
/* Experimenter extension. */
struct ofp_experimenter_header {
    struct ofp_header header;   /* Type OFPT_EXPERIMENTER. */
    uint32_t experimenter;      /* Experimenter ID:
                                 * - MSB 0: low-order bytes are IEEE OUI.
                                 * - MSB != 0: defined by ONF. */
    uint32_t exp_type;          /* Experimenter defined. */
    /* Experimenter-defined arbitrary additional data. */
};
OFP_ASSERT(sizeof(struct ofp_experimenter_header) == 16);
```

The `experimenter` field is a 32-bit value that uniquely identifies the experimenter. If the most significant byte is zero, the next three bytes are the experimenter's IEEE OUI. If the most significant byte is not zero, it is a value allocated by the Open Networking Foundation. If experimenter does not have (or wish to use) their OUI, they should contact the Open Networking Foundation to obtain a unique experimenter ID.

The rest of the body is uninterpreted by standard OpenFlow processing and is arbitrarily defined by the corresponding experimenter.

If a switch does not understand a experimenter extension, it must send an `OFPT_ERROR` message with a `OFPBRC_BAD_EXPERIMENTER` error code and `OFPET_BAD_REQUEST` error type.

Appendix A Release Notes

This section contains release notes highlighting the main changes between the main versions of the OpenFlow protocol.

The text of the release notes is informative and historical, and should not be considred normative. Many items of the release notes refer to features and text that has been removed, replaced or updated in subsequent versions of this specification, and therefore does not necessarily match the actual specification.

A.1 OpenFlow version 0.2.0

Release date : March 28,2008
Wire Protocol : 1

A.2 OpenFlow version 0.2.1

Release date : March 28,2008
Wire Protocol : 1
No protocol change.

A.3 OpenFlow version 0.8.0

Release date : May 5, 2008
Wire Protocol : 0x83

- Reorganise OpenFlow message types
- Add `OFPP_TABLE` virtual port to send packet-out packet to the tables
- Add global flag `OFPC_SEND_FLOW_EXP` to configure flow expired messages generation
- Add flow priority
- Remove flow Group-ID (experimental QoS support)
- Add Error messages
- Make stat request and stat reply more generic, with a generic header and stat specific body
- Change fragmentation strategy for stats reply, use explicit flag `OFPSF_REPLY_MORE` instead of empty packet
- Add table stats and port stats mesages

A.4 OpenFlow version 0.8.1

Release date : May 20, 2008
Wire Protocol : 0x83
No protocol change.

A.5 OpenFlow version 0.8.2

Release date : October 17, 2008
Wire Protocol : 0x85

- Add Echo Request and Echo Reply messages
- Make all message 64 bits aligned

A.6 OpenFlow version 0.8.9

Release date : December 2, 2008
Wire Protocol : 0x97

A.6.1 IP Netmasks

It is now possible for flow entries to contain IP subnet masks. This is done by changes to the wildcards field, which has been expanded to 32-bits:

```
/* Flow wildcards. */
enum ofp_flow_wildcards {
OFPFW_IN_PORT  = 1 << 0,  /* Switch input port. */
OFPFW_DL_VLAN  = 1 << 1,  /* VLAN. */
OFPFW_DL_SRC   = 1 << 2,  /* Ethernet source address. */
OFPFW_DL_DST   = 1 << 3,  /* Ethernet destination address. */
OFPFW_DL_TYPE  = 1 << 4,  /* Ethernet frame type. */
OFPFW_NW_PROTO = 1 << 5,  /* IP protocol. */
OFPFW_TP_SRC   = 1 << 6,  /* TCP/UDP source port. */
OFPFW_TP_DST   = 1 << 7,  /* TCP/UDP destination port. */

/* IP source address wildcard bit count.  0 is exact match, 1 ignores the
 * LSB, 2 ignores the 2 least-significant bits, ..., 32 and higher wildcard
 * the entire field.  This is the *opposite* of the usual convention where
 * e.g. /24 indicates that 8 bits (not 24 bits) are wildcarded. */
OFPFW_NW_SRC_SHIFT = 8,
OFPFW_NW_SRC_BITS = 6,
OFPFW_NW_SRC_MASK = ((1 << OFPFW_NW_SRC_BITS) - 1) << OFPFW_NW_SRC_SHIFT,
OFPFW_NW_SRC_ALL = 32 << OFPFW_NW_SRC_SHIFT,

/* IP destination address wildcard bit count.  Same format as source. */
OFPFW_NW_DST_SHIFT = 14,
OFPFW_NW_DST_BITS = 6,
OFPFW_NW_DST_MASK = ((1 << OFPFW_NW_DST_BITS) - 1) << OFPFW_NW_DST_SHIFT,
OFPFW_NW_DST_ALL = 32 << OFPFW_NW_DST_SHIFT,

/* Wildcard all fields. */
OFPFW_ALL = ((1 << 20) - 1)
};
```

The source and destination netmasks are each specified with a 6-bit number in the wildcard description. It is interpreted similar to the CIDR suffix, but with the opposite meaning, since this is being used to indicate which bits in the IP address should be treated as "wild". For example, a CIDR suffix of "24" means to use a netmask of "255.255.255.0". However, a wildcard mask value of "24" means that the least-significant 24-bits are wild, so it forms a netmask of "255.0.0.0".

A.6.2 New Physical Port Stats

The `ofp_port_stats` message has been expanded to return more information. If a switch does not support a particular field, it should set the value to have all bits enabled (i.e., a "-1" if the value were treated as signed). This is the new format:

```
/* Body of reply to OFPST_PORT request. If a counter is unsupported, set
 * the field to all ones. */
struct ofp_port_stats {
    uint16_t port_no;
```



```
    uint8_t pad[6];          /* Align to 64-bits. */
    uint64_t rx_packets;     /* Number of received packets. */
    uint64_t tx_packets;     /* Number of transmitted packets. */
    uint64_t rx_bytes;       /* Number of received bytes. */
    uint64_t tx_bytes;       /* Number of transmitted bytes. */
    uint64_t rx_dropped;     /* Number of packets dropped by RX. */
    uint64_t tx_dropped;     /* Number of packets dropped by TX. */
    uint64_t rx_errors;      /* Number of receive errors.  This is a super-set
                                of receive errors and should be great than or
                                equal to the sum of al rx_*_err values. */
    uint64_t tx_errors;      /* Number of transmit errors.  This is a super-set
                                of transmit errors. */
    uint64_t rx_frame_err;   /* Number of frame alignment errors. */
    uint64_t rx_over_err;    /* Number of packets with RX overrun. */
    uint64_t rx_crc_err;     /* Number of CRC errors. */
    uint64_t collisions;     /* Number of collisions. */
};
```

A.6.3 IN_PORT Virtual Port

The behavior of sending out the incoming port was not clearly defined in earlier versions of the specification. It is now forbidden unless the output port is explicitly set to `OFPP_IN_PORT` virtual port (0xfff8) is set. The primary place where this is used is for wireless links, where a packet is received over the wireless interface and needs to be sent to another host through the same interface. For example, if a packet needed to be sent to **all** interfaces on the switch, two actions would need to be specified: "actions=output:ALL,output:IN_PORT".

A.6.4 Port and Link Status and Configuration

The switch should inform the controller of changes to port and link status. This is done with a new flag in `ofp_port_config`:

- `OFPPC_PORT_DOWN` - The port has been configured "down".

... and a new flag in `ofp_port_state`:

- `OFPPS_LINK_DOWN` - There is no physical link present.

The switch should support enabling and disabling a physical port by modifying the `OFPPFL_PORT_DOWN` flag (and mask bit) in the `ofp_port_mod` message. Note that this is **not** the same as adding or removing the interface from the list of OpenFlow monitored ports; it is equivalent to `"ifconfig eth0 down"` on Unix systems.

A.6.5 Echo Request/Reply Messages

The switch and controller can verify proper connectivity through the OpenFlow protocol with the new echo request (`OFPT_ECHO_REQUEST`) and reply (`OFPT_ECHO_REPLY`) messages. The body of the message is undefined and simply contains uninterpreted data that is to be echoed back to the requester. The requester matches the reply with the transaction id from the OpenFlow header.

A.6.6 Vendor Extensions

Vendors are now able to add their own extensions, while still being OpenFlow compliant. The primary way to do this is with the new `OFPT_VENDOR` message type. The message body is of the form:

```
/* Vendor extension. */
struct ofp_vendor {
    struct ofp_header header;   /* Type OFPT_VENDOR. */
    uint32_t vendor;            /* Vendor ID:
                                 * - MSB 0: low-order bytes are IEEE OUI.
                                 * - MSB != 0: defined by OpenFlow
                                 *   consortium. */
    /* Vendor-defined arbitrary additional data. */
};
```

The *vendor* field is a 32-bit value that uniquely identifies the vendor. If the most significant byte is zero, the next three bytes are the vendor's IEEE OUI. If vendor does not have (or wish to use) their OUI, they should contact the OpenFlow consortium to obtain one. The rest of the body is uninterpreted.

It is also possible to add vendor extensions for stats messages with the `OFPST_VENDOR` stats type. The first four bytes of the message are the vendor identifier as described earlier. The rest of the body is vendor-defined.

To indicate that a switch does not understand a vendor extension, a `OFPBRC_BAD_VENDOR` error code has been defined under the `OFPET_BAD_REQUEST` error type.

Vendor-defined actions are described below in the "Variable Length and Vendor Actions" section.

A.6.7 Explicit Handling of IP Fragments

In previous versions of the specification, handling of IP fragments was not clearly defined. The switch is now able to tell the controller whether it is able to reassemble fragments. This is done with the following `capabilities` flag passed in the `ofp_switch` features message:

```
OFPC_IP_REASM     = 1 << 5  /* Can reassemble IP fragments. */
```

The controller can configure fragment handling in the switch through the setting the following new `ofp_config_flags` in the `ofp_switch_config` message:

```
/* Handling of IP fragments. */
OFPC_FRAG_NORMAL   = 0 << 1,  /* No special handling for fragments. */
OFPC_FRAG_DROP     = 1 << 1,  /* Drop fragments. */
OFPC_FRAG_REASM    = 2 << 1,  /* Reassemble (only if OFPC_IP_REASM set). */
OFPC_FRAG_MASK     = 3 << 1
```

"Normal" handling of fragments means that an attempt should be made to pass the fragments through the OpenFlow tables. If any field is not present (e.g., the TCP/UDP ports didn't fit), then the packet should not match any entry that has that field set.

A.6.8 802.1D Spanning Tree

OpenFlow now has a way to configure and view results of on-switch implementations of 802.1D Spanning Tree Protocol.

A switch that implements STP must set the new OFPC_STP bit in the 'capabilities' field of its OFPT_FEATURES_REPLY message. A switch that implements STP at all must make it available on all of its physical ports, but it need not implement it on virtual ports (e.g. OFPP_LOCAL).

Several port configuration flags are associated with STP. The complete set of port configuration flags are:

```
enum ofp_port_config {
   OFPPC_PORT_DOWN    = 1 << 0,  /* Port is administratively down. */
   OFPPC_NO_STP       = 1 << 1,  /* Disable 802.1D spanning tree on port. */
   OFPPC_NO_RECV      = 1 << 2,  /* Drop most packets received on port. */
   OFPPC_NO_RECV_STP  = 1 << 3,  /* Drop received 802.1D STP packets. */
   OFPPC_NO_FLOOD     = 1 << 4,  /* Do not include this port when flooding. */
   OFPPC_NO_FWD       = 1 << 5,  /* Drop packets forwarded to port. */
   OFPPC_NO_PACKET_IN = 1 << 6   /* Do not send packet-in msgs for port. */
};
```

The controller may set OFPPFL_NO_STP to 0 to enable STP on a port or to 1 to disable STP on a port. (The latter corresponds to the Disabled STP port state.) The default is switch implementation-defined; the OpenFlow reference implementation by default sets this bit to 0 (enabling STP).

When OFPPFL_NO_STP is 0, STP controls the OFPPFL_NO_FLOOD and OFPPFL_STP_* bits directly. OFPPFL_NO_FLOOD is set to 0 when the STP port state is Forwarding, otherwise to 1. The bits in OFPPFL_STP_MASK are set to one of the other OFPPFL_STP_* values according to the current STP port state.

When the port flags are changed by STP, the switch sends an OFPT_PORT_STATUS message to notify the controller of the change. The OFPPFL_NO_RECV, OFPPFL_NO_RECV_STP, OFPPFL_NO_FWD, and OFPPFL_NO_PACKET_IN bits in the OpenFlow port flags may be useful for the controller to implement STP, although they interact poorly with in-band control.

A.6.9 Modify Actions in Existing Flow Entries

New ofp_flow_mod commands have been added to support modifying the actions of existing entries: OFPFC_MODIFY and OFPFC_MODIFY_STRICT. They use the match field to describe the entries that should be modified with the supplied actions. OFPFC_MODIFY is similar to OFPFC_DELETE, in that wildcards are "active". OFPFC_MODIFY_STRICT is similar to OFPFC_DELETE_STRICT, in that wildcards are not "active", so both the wildcards and priority must match an entry. When a matching flow is found, only its actions are modified–information such as counters and timers are not reset.

If the controller uses the OFPFC_ADD command to add an entry that already exists, then the new entry replaces the old and all counters and timers are reset.

A.6.10 More Flexible Description of Tables

Previous versions of OpenFlow had very limited abilities to describe the tables supported by the switch. The `n_exact`, `n_compression`, and `n_general` fields in `ofp_switch_features` have been replaced with `n_tables`, which lists the number of tables in the switch.

The behavior of the `OFPST_TABLE` stat reply has been modified slightly. The `ofp_table_stats` body now contains a wildcards field, which indicates the fields for which that particular table supports wildcarding. For example, a direct look-up hash table would have that field set to zero, while a sequentially searched table would have it set to `OFPFW_ALL`. The `ofp_table_stats` entries are returned in the order that packets traverse the tables.

When the controller and switch first communicate, the controller will find out how many tables the switch supports from the Features Reply. If it wishes to understand the size, types, and order in which tables are consulted, the controller sends a `OFPST_TABLE` stats request.

A.6.11 Lookup Count in Tables

Table stats returned `ofp_table_stats` structures now return the number of packets that have been looked up in the table–whether they hit or not. This is stored in the `lookup_count` field.

A.6.12 Modifying Flags in Port-Mod More Explicit

The `ofp_port_mod` is used to modify characteristics of a switch's ports. A supplied `ofp_phy_port` structure describes the behavior of the switch through its `flags` field. However, it's possible that the controller wishes to change a particular flag and may not know the current status of all flags. A `mask` field has been added which has a bit set for each flag that should be changed on the switch.

The new `ofp_port_mod` message looks like the following:

```
/* Modify behavior of the physical port */
struct ofp_port_mod {
    struct ofp_header header;
    uint32_t mask;         /* Bitmap of "ofp_port_flags" that should be
                              changed. */
    struct ofp_phy_port desc;
};
```

A.6.13 New Packet-Out Message Format

The previous version's `packet-out` message treated the variable-length array differently depending on whether the `buffer_id` was set or not. If set, the array consisted of actions to be executed and the `out_port` was ignored. If not, the array consisted of the actual packet that should be placed on the wire through the `out_port` interface. This was a bit ugly, and it meant that in order for a non-buffered packet to have multiple actions executed on it, that a new flow entry be created just to match that entry.

A new format is now used, which cleans the message up a bit. The packet always contains a list of actions. An additional variable-length array follows the list of actions with the contents of the packet if `buffer_id` is not set. This is the new format:

```
struct ofp_packet_out {
    struct ofp_header header;
    uint32_t buffer_id;           /* ID assigned by datapath (-1 if none). */
    uint16_t in_port;             /* Packet's input port (OFPP_NONE if none). */
    uint16_t n_actions;           /* Number of actions. */
    struct ofp_action actions[0]; /* Actions. */
    /* uint8_t data[0]; */        /* Packet data.  The length is inferred
                                     from the length field in the header.
                                     (Only meaningful if buffer_id == -1.) */
};
```

A.6.14 Hard Timeout for Flow Entries

A hard timeout value has been added to flow entries. If set, then the entry must be expired in the specified number of seconds regardless of whether or not packets are hitting the entry. A `hard_timeout` field has been added to the `flow_mod` message to support this. The `max_idle` field has been renamed `idle_timeout`. A value of zero means that a timeout has not been set. If both `idle_timeout` and `hard_timeout` are zero, then the flow is permanent and should not be deleted without an explicit deletion.

The new `ofp_flow_mod` format looks like this:

```
struct ofp_flow_mod {
    struct ofp_header header;
    struct ofp_match match;      /* Fields to match */

    /* Flow actions. */
    uint16_t command;             /* One of OFPFC_*. */
    uint16_t idle_timeout;        /* Idle time before discarding (seconds). */
    uint16_t hard_timeout;        /* Max time before discarding (seconds). */
    uint16_t priority;            /* Priority level of flow entry. */
    uint32_t buffer_id;           /* Buffered packet to apply to (or -1).
                                     Not meaningful for OFPFC_DELETE*. */
    uint32_t reserved;            /* Reserved for future use. */
    struct ofp_action actions[0]; /* The number of actions is inferred from
                                     the length field in the header. */
};
```

Since flow entries can now be expired due to idle or hard timeouts, a `reason` field has been added to the `ofp_flow_expired` message. A value of 0 indicates an idle timeout and 1 indicates a hard timeout:

```
enum ofp_flow_expired_reason {
    OFPER_IDLE_TIMEOUT,         /* Flow idle time exceeded idle_timeout. */
    OFPER_HARD_TIMEOUT          /* Time exceeded hard_timeout. */
};
```

The new `ofp_flow_expired` message looks like the following:


```
struct ofp_flow_expired {
    struct ofp_header header;
    struct ofp_match match;   /* Description of fields */

    uint16_t priority;        /* Priority level of flow entry. */
    uint8_t reason;           /* One of OFPER_*. */
    uint8_t pad[1];           /* Align to 32-bits. */

    uint32_t duration;        /* Time flow was alive in seconds. */
    uint8_t pad2[4];          /* Align to 64-bits. */
    uint64_t packet_count;
    uint64_t byte_count;
};
```

A.6.15 Reworked initial handshake to support backwards compatibility

OpenFlow now includes a basic "version negotiation" capability. When an OpenFlow connection is established, each side of the connection should immediately send an `OFPT_HELLO` message as its first OpenFlow message. The 'version' field in the hello message should be the highest OpenFlow protocol version supported by the sender. Upon receipt of this message, the recipient may calculate the OpenFlow protocol version to be used as the smaller of the version number that it sent and the one that it received.

If the negotiated version is supported by the recipient, then the connection proceeds. Otherwise, the recipient must reply with a message of `OFPT_ERROR` with a 'type' value of `OFPET_HELLO_FAILED`, a 'code' of `OFPHFC_COMPATIBLE`, and optionally an ASCII string explaining the situation in 'data', and then terminate the connection.

The `OFPT_HELLO` message has no body; that is, it consists only of an OpenFlow header. Implementations must be prepared to receive a hello message that includes a body, ignoring its contents, to allow for later extensions.

A.6.16 Description of Switch Stat

The `OFPST_DESC` stat has been added to describe the hardware and software running on the switch:

```
#define DESC_STR_LEN   256
#define SERIAL_NUM_LEN 32
/* Body of reply to OFPST_DESC request.  Each entry is a NULL-terminated
 * ASCII string. */
struct ofp_desc_stats {
    char mfr_desc[DESC_STR_LEN];       /* Manufacturer description. */
    char hw_desc[DESC_STR_LEN];        /* Hardware description. */
    char sw_desc[DESC_STR_LEN];        /* Software description. */
    char serial_num[SERIAL_NUM_LEN];   /* Serial number. */
};
```

It contains a 256 character ASCII description of the manufacturer, hardware type, and software version. It also contains a 32 character ASCII serial number. Each entry is padded on the right with 0 bytes.

A.6.17 Variable Length and Vendor Actions

Vendor-defined actions have been added to OpenFlow. To enable more versatility, actions have switched from fixed-length to variable. All actions have the following header:

```
struct ofp_action_header {
    uint16_t type;                  /* One of OFPAT_*. */
    uint16_t len;                   /* Length of action, including this
                                       header.  This is the length of action,
                                       including any padding to make it
                                       64-bit aligned. */
    uint8_t pad[4];
};
```

The length for actions must always be a multiple of eight to aid in 64-bit alignment. The action types are as follows:

```
enum ofp_action_type {
    OFPAT_OUTPUT,           /* Output to switch port. */
    OFPAT_SET_VLAN_VID,     /* Set the 802.1q VLAN id. */
    OFPAT_SET_VLAN_PCP,     /* Set the 802.1q priority. */
    OFPAT_STRIP_VLAN,       /* Strip the 802.1q header. */
    OFPAT_SET_DL_SRC,       /* Ethernet source address. */
    OFPAT_SET_DL_DST,       /* Ethernet destination address. */
    OFPAT_SET_NW_SRC,       /* IP source address. */
    OFPAT_SET_NW_DST,       /* IP destination address. */
    OFPAT_SET_TP_SRC,       /* TCP/UDP source port. */
    OFPAT_SET_TP_DST,       /* TCP/UDP destination port. */
    OFPAT_VENDOR = 0xffff
};
```

The vendor-defined action header looks like the following:

```
struct ofp_action_vendor_header {
    uint16_t type;                  /* OFPAT_VENDOR. */
    uint16_t len;                   /* Length is 8. */
    uint32_t vendor;                /* Vendor ID, which takes the same form
                                       as in "struct ofp_vendor". */
};
```

The `vendor` field uses the same vendor identifier described earlier in the "Vendor Extensions" section. Beyond using the `ofp_action_vendor` header and the 64-bit alignment requirement, vendors are free to use whatever body for the message they like.

A.6.18 VLAN Action Changes

It is now possible to set the priority field in VLAN tags and stripping VLAN tags is now a separate action. The OFPAT_SET_VLAN_VID action behaves like the former OFPAT_SET_DL_VLAN action, but no longer accepts a special value that causes it to strip the VLAN tag. The OFPAT_SET_VLAN_PCP action modifies the 3-bit priority field in the VLAN tag. For existing tags, both actions only modify the bits associated with the field being updated. If a new VLAN tag needs to be added, the value of all other fields is zero.

The OFPAT_SET_VLAN_VID action looks like the following:

```
struct ofp_action_vlan_vid {
    uint16_t type;                  /* OFPAT_SET_VLAN_VID. */
    uint16_t len;                   /* Length is 8. */
    uint16_t vlan_vid;              /* VLAN id. */
    uint8_t pad[2];
};
```

The OFPAT_SET_VLAN_PCP action looks like the following:

```
struct ofp_action_vlan_pcp {
    uint16_t type;                  /* OFPAT_SET_VLAN_PCP. */
    uint16_t len;                   /* Length is 8. */
    uint8_t vlan_pcp;               /* VLAN priority. */
    uint8_t pad[3];
};
```

The OFPAT_STRIP_VLAN action takes no argument and strips the VLAN tag if one is present.

A.6.19 Max Supported Ports Set to 65280

What: Increase maximum number of ports to support large vendor switches; was previously 256, chosen arbitrarily.

Why: The HP 5412 chassis supports 288 ports of Ethernet, and some Cisco switches go much higher. The current limit (OFPP_MAX) is 255, set to equal the maximum number of ports in a bridge segment in the 1998 STP spec. The RSTP spec from 2004 supports up to 4096 (12 bits) of ports.

How: Change OFPP_MAX to 65280. (However, out of the box, the reference switch implementation supports at most 256 ports.)

A.6.20 Send Error Message When Flow Not Added Due To Full Tables

The switch now sends an error message when a flow is added, but cannot because all the tables are full. The message has an error type of OFPET_FLOW_MOD_FAILED and code of OFPFMFC_ALL_TABLES_FULL. If the Flow-Mod command references a buffered packet, then actions are not performed on the packet. If the controller wishes the packet to be sent regardless of whether or not a flow entry is added, then it should use a Packet-Out directly.

A.6.21 Behavior Defined When Controller Connection Lost

What: Ensure that all switches have at least one common behavior when the controller connection is lost.

Why: When the connection to the controller is lost, the switch should behave in a well-defined way. Reasonable behaviors include 'do nothing - let flows naturally timeout', 'freeze timeouts', 'become learning switch', and 'attempt connection to other controller'. Switches may implement one or more of these, and network admins may want to ensure that if the controller goes out, they know what the network can do.

The first is the simplest: ensure that every switch implements a default of 'do nothing - let flows timeout naturally'. Changes must be done via vendor-specific command line interface or vendor extension OpenFlow messages.

The second may help ensure that a single controller can work with switches from multiple vendors. The different failure behaviors, plus 'other', could be feature bits set for the switch. A switch would still only have to support the default.

The worry here is that we may not be able to enumerate in advance the full range of failure behaviors, which argues for the first approach.

How: Added text to spec: "In the case that the switch loses contact with the controller, the default behavior must be to do nothing - to let flows timeout naturally. Other behaviors can be implemented via vendor-specific command line interface or vendor extension OpenFlow messages."

A.6.22 ICMP Type and Code Fields Now Matchable

What: Allow matching ICMP traffic based on type or code.

Why: We can't distinguish between different types of ICMP traffic (e.g., echo replies vs echo requests vs redirects).

How: Changed spec to allow matching on these fields.

As for implementation: The type and code are each a single byte, so they easily fit in our existing flow structure. Overload the `tp_src` field to ICMP type and `tp_dst` to ICMP code. Since they are only a single byte, they will reside in the low-byte of these two byte fields (stored in network-byte order). This will allow a controller to use the existing wildcard bits to wildcard these ICMP fields.

A.6.23 Output Port Filtering for Delete*, Flow Stats and Aggregate Stats

Add support for listing and deleting entries based on an output port.

To support this, an `out_port` field has been added to the `ofp_flow_mod`, `ofp_flow_stats_request`, and `ofp_aggregate_stats_request` messages. If an `out_port` contains a value other than `OFPP_NONE`, it introduces a constraint when matching. This constraint is that the rule must contain an output action directed at that port. Other constraints such as `ofp_match` structs and priorities are still used; this is purely an *additional* constraint. Note that to get previous behavior, though, `out_port` must be set

to `OFPP_NONE`, since "0" is a valid port id. This only applies to the `delete` and `delete_strict` flow mod commands; the field is ignored by `add`, `modify`, and `modify_strict`.

A.7 OpenFlow version 0.9

Release date : July 20, 2009
Wire Protocol : 0x98

A.7.1 Failover

The reference implementation now includes a simple failover mechanism. A switch can be configured with a list of controllers. If the first controller fails, it will automatically switch over to the second controller on the list.

A.7.2 Emergency Flow Cache

The protocol and reference implementation have been extended to allow insertion and management of emergency flow entries.

Emergency-specific flow entries are inactive until a switch loses connectivity from the controller. If this happens, the switch invalidates all normal flow table entries and copies all emergency flows into the normal flow table.

Upon connecting to a controller again, all entries in the flow cache stay active. The controller then has the option of resetting the flow cache if needed.

A.7.3 Barrier Command

The Barrier Command is a mechanism to get notified when an OpenFlow message has finished executing on the switch. When a switch receives a Barrier message it must first complete all commands sent before the Barrier message before executing any commands after it. When all commands before the Barrier message have completed, it must send a Barrier Reply message back to the controller.

A.7.4 Match on VLAN Priority Bits

There is an optional new feature that allows matching on priority VLAN fields. Pre 0.9, the VLAN id is a field used in identifying a flow, but the priority bits in the VLAN tag are not. In this release we include the priority bits as a separate field to identify flows. Matching is possible as either an exact match on the 3 priority bits, or as a wildcard for the entire 3 bits.

A.7.5 Selective Flow Expirations

Flow expiration messages can now be requested on a per-flow, rather than per-switch granularity.

A.7.6 Flow Mod Behavior

There now is a `CHECK_OVERLAP` flag to flow mods which requires the switch to do the (potentially more costly) check that there doesn't already exist a conflicting flow with the same priority. If there is one, the mod fails and an error code is returned. Support for this flag is required in an OpenFlow switch.

A.7.7 Flow Expiration Duration

The meaning of the "duration" field in the Flow Expiration message has been changed slightly. Previously there were conflicting definitions of this in the spec. In 0.9 the value returned will be the time that the flow was active and not include the timeout period.

A.7.8 Notification for Flow Deletes

If a controller deletes a flow it now receives a notification if the notification bit is set. In previous releases only flow expirations but not delete actions would trigger notifications.

A.7.9 Rewrite DSCP in IP ToS header

There is now an added Flow action to rewrite the DiffServ CodePoint bits part of the IP ToS field in the IP header. This enables basic support for basic QoS with OpenFlow in some switches. A more complete QoS framework is planned for a future OpenFlow release.

A.7.10 Port Enumeration now starts at 1

Previous releases of OpenFlow had port numbers start at 0, release 0.9 changes them to start at 1.

A.7.11 Other changes to the Specification

- 6633/TCP is now the recommended default OpenFlow Port. Long term the goal is to get a IANA approved port for OpenFlow.
- The use of "Type 1" and "Type 0" has been depreciated and references to it have been removed.
- Clarified Matching Behavior for Flow Modification and Stats
- Made explicit that packets received on ports that are disabled by spanning tree must follow the normal flow table processing path.
- Clarified that transaction ID in header should match offending message for `OFPET_BAD_REQUEST`, `OFPET_BAD_ACTION`, `OFPET_FLOW_MOD_FAILED`.
- Clarified the format for the Strip VLAN Action
- Clarify behavior for packets that are buffered on the switch while switch is waiting for a reply from controller
- Added the new `EPERM` Error Type
- Fixed Flow Table Matching Diagram
- Clarified datapath ID 64 bits, up from 48 bits
- Clarified `miss-send-len` and `max-len` of output action

A.8 OpenFlow version 1.0

Release date : December 31, 2009
Wire Protocol : 0x01

A.8.1 Slicing

OpenFlow now supports multiple queues per output port. Queues support the ability to provide minimum bandwidth guarantees; the bandwidth allocated to each queue is configurable. The name slicing is derived from the ability to provide a slice of the available network bandwidth to each queue.

A.8.2 Flow cookies

Flows have been extended to include an opaque identifier, referred to as a cookie. The cookie is specified by the controller when the flow is installed; the cookie will be returned as part of each flow stats and flow expired message.

A.8.3 User-specifiable datapath description

The OFPST_DESC (switch description) reply now includes a datapath description field. This is a user-specifiable field that allows a switch to return a string specified by the switch owner to describe the switch.

A.8.4 Match on IP fields in ARP packets

The reference implementation can now match on IP fields inside ARP packets. The source and destination protocol address are mapped to the nw_src and nw_dst fields respecitively, and the opcode is mapped to the nw_proto field.

A.8.5 Match on IP ToS/DSCP bits

OpenFlow now supports matching on the IP ToS/DSCP bits.

A.8.6 Querying port stats for individual ports

Port stat request messages include a `port_no` field to allow stats for individual ports to be queried. Port stats for all ports can still be requested by specifying `OFPP_NONE` as the port number.

A.8.7 Improved flow duration resolution in stats/expiry messages

Flow durations in stats and expiry messages are now expressed with nanosecond resolution. Note that the accuracy of flow durations in the reference implementation is on the order of milliseconds. (The actual accuracy is in part dependent upon kernel parameters.)

A.8.8 Other changes to the Specification

- remove `multi_phy_tx` spec text and capability bit
- clarify execution order of actions
- replace SSL refs with TLS
- resolve overlap ambiguity
- clarify flow mod to non-existing port
- clarify port definition
- update packet flow diagram
- update header parsing diagram for ICMP
- fix English ambiguity for flow-removed messages
- fix async message English ambiguity
- note that multiple controller support is undefined
- clarify that byte equals octet
- note counter wrap-around
- removed warning not to build a switch from this specification

A.9 OpenFlow version 1.1

Release date : February 28, 2011
Wire Protocol : 0x02

A.9.1 Multiple Tables

Prior versions of the OpenFlow specification did expose to the controller the abstraction of a single table. The OpenFlow pipeline could internally be mapped to multiple tables, such as having a separate wildcard and exact match table, but those tables would always act logically as a single table.

OpenFlow 1.1 introduces a more flexible pipeline with multiple tables. Exposing multiple tables has many advantages. The first advantage is that many hardware have multiple tables internally (for example L2 table, L3 table, multiple TCAM lookups), and the multiple table support of OpenFlow may enable to expose this hardware with greater efficiency and flexibility. The second advantage is that many network deployments combine orthogonal processing of packets (for example ACL, QoS and routing), forcing all those processing in a single table creates huge ruleset due to the cross product of individual rules, multiple tables may decouple properly those processing.

The new OpenFlow pipeline with multiple table is quite different from the simple pipeline of prior OpenFlow versions. The new OpenFlow pipeline expose a set of completely generic tables, supporting the full match and full set of actions. It's difficult to build a pipeline abstraction that represent accurately all possible hardware, therefore OpenFlow 1.1 is based on a generic and flexible pipeline that may be

mapped to the hardware. Some limited table capabilities are available to denote what each table is capable of supporting.

Packets are processed through the pipeline, they are matched and processed in the first table, and may be matched and processed in other tables. As it goes through the pipeline, a packet is associated with an action set, accumulating action, and a generic metadata register. The action set is resolved at the end of the pipeline and applied to the packet. The metadata can be matched and written at each table and enables to carry state between tables.

OpenFlow introduces a new protocol object called instruction to control pipeline processing. Actions which were directly attached to flows in previous versions are now encapsulated in instructions, instructions may apply those actions between tables or accumulate them in the packet action set. Instructions can also change the metadata, or direct packet to another table.

- The switch now expose a pipeline with multiple tables
- Flow entry have instructions to control pipeline processing
- Controller can choose packet traversal of tables via goto instruction
- Metadata field (64 bits) can be set and match in tables
- Packet actions can be merged in packet action set
- Packet action set is executed at the end of pipeline
- Packet actions can be applied between table stages
- Table miss can send to controller, continue to next table or drop
- Rudimentary table capability and configuration

A.9.2 Groups

The new group abstraction enables OpenFlow to represent a set of ports as a single entity for forwarding packets. Different types of groups are provided, to represent different abstractions such as multicasting or multipathing. Each group is composed of a set group buckets, each group bucket contains the set of actions to be applied before forwarding to the port. Groups buckets can also forward to other groups, enabling to chain groups together.

- Group indirection to represent a set of ports
- Group table with 4 types of groups :
  - All - used for multicast and flooding
  - Select - used for multipath
  - Indirect - simple indirection
  - Fast Failover - use first live port
- Group action to direct a flow to a group
- Group buckets contains actions related to the individual port

A.9.3 Tags : MPLS & VLAN

Prior versions of the OpenFlow specification had limited VLAN support, it only supported a single level of VLAN tagging with ambiguous semantic. The new tagging support has explicit actions to add, modify and remove VLAN tags, and can support multiple level of VLAN tagging. It also adds similar support the MPLS shim headers.

- Support for VLAN and QinQ, adding, modifying and removing VLAN headers
- Support for MPLS, adding, modifying and removing MPLS shim headers

A.9.4 Virtual ports

Prior versions of the OpenFlow specification assumed that all the ports of the OpenFlow switch were physical ports. This version of the specification add support for virtual ports, which can represent complex forwarding abstractions such as LAGs or tunnels.

- Make port number 32 bits, enable larger number of ports
- Enable switch to provide virtual ports as OpenFlow ports
- Augment packet-in to report both virtual and physical ports

A.9.5 Controller connection failure

Prior versions of the OpenFlow specification introduced the emergency flow cache as a way to deal with the loss of connectivity with the controller. The emergency flow cache feature was removed in this version of the specification, due to the lack of adoption, the complexity to implement it and other issues with the feature semantic.

This version of the specification adds two simpler modes to deal with the loss of connectivity with the controller. In fail secure mode, the switch continues operating in OpenFlow mode, until it reconnects to a controller. In fail standalone mode, the switch revert to using normal processing (Ethernet switching).

- Remove Emergency Flow Cache from spec
- Connection interruption triggers fail secure or fail standalone mode

A.9.6 Other changes

- Remove 802.1d-specific text from the specification
- Cookie Enhancements Proposal - cookie mask for filtering
- Set_queue action (unbundled from output port action)
- Maskable DL and NW address match fields
- Add TTL decrement, set and copy actions for IPv4 and MPLS
- SCTP header matching and rewriting support
- Set ECN action
- Define message handling : no loss, may reorder if no barrier
- Rename `VENDOR` APIs to `EXPERIMENTER` APIs
- Many other bug fixes, rewording and clarifications

A.10 OpenFlow version 1.2

Release date : December 5, 2011
Wire Protocol : 0x03

Please refers to the bug tracking ID for more details on each change

A.10.1 Extensible match support

Prior versions of the OpenFlow specification used a static fixed length structure to specify `ofp_match`, which prevents flexible expression of matches and prevents inclusion of new match fields. The `ofp_match` has been changed to a TLV structure, called OpenFlow Extensible Match (OXM), which dramatically increases flexibility.

The match fields themselves have been reorganised. In the previous static structure, many fields were overloaded ; for example `tcp.src_port`, `udp.src_port`, and `icmp.code` were using the same field entry. Now, every logical field has its own unique type.

List of features for OpenFlow Extensible Match :

- Flexible and compact TLV structure called OXM (EXT-1)
- Enable flexible expression of match, and flexible bitmasking (EXT-1)
- Pre-requisite system to insure consistency of match (EXT-1)
- Give every match field a unique type, remove overloading (EXT-1)
- Modify VLAN matching to be more flexible (EXT-26)
- Add vendor classes and experimenter matches (EXT-42)
- Allow switches to override match requirements (EXT-56, EXT-33)

A.10.2 Extensible 'set_field' packet rewriting support

Prior versions of the OpenFlow specification were using hand-crafted actions to rewrite header fields. The Extensible `set_field` action reuses the OXM encoding defined for matches, and enables to rewrite any header field in a single action (EXT-13). This allows any new match field, including experimenter fields, to be available for rewrite. This makes the specification cleaner and eases cost of introducing new fields.

- Deprecate most header rewrite actions
- Introduce generic `set-field` action (EXT-13)
- Reuse match TLV structure (OXM) in `set-field` action

A.10.3 Extensible context expression in 'packet-in'

The `packet-in` message did include some of the packet context (ingress port), but not all (metadata), preventing the controller from figuring how match did happen in the table and which flow entries would match or not match. Rather than introduce a hard coded field in the `packet-in` message, the flexible OXM encoding is used to carry packet context.

- Reuse match TLV structure (OXM) to describe metadata in packet-in (EXT-6)
- Include the 'metadata' field in packet-in
- Move ingress port and physical port from static field to OXM encoding
- Allow to optionally include packet header fields in TLV structure

A.10.4 Extensible Error messages via experimenter error type

An experimenter error code has been added, enabling experimenter functionality to generate custom error messages (EXT-2). The format is identical to other experimenter APIs.

A.10.5 IPv6 support added

Basic support for IPv6 match and header rewrite has been added, via the Flexible match support.

- Added support for matching on IPv6 source address, destination address, protocol number, traffic class, ICMPv6 type, ICMPv6 code and IPv6 neighbor discovery header fields (EXT-1)
- Added support for matching on IPv6 flow label (EXT-36)

A.10.6 Simplified behaviour of flow-mod request

The behaviour of flow-mod request has been simplified (EXT-30).

- MODIFY and MODIFY_STRICT commands never insert new flows in the table
- New flag OFPFF_RESET_COUNTS to control counter reset
- Remove quirky behaviour for cookie field.

A.10.7 Removed packet parsing specification

The OpenFlow specification no longer attempts to define how to parse packets (EXT-3). The match fields are only defined logically.

- OpenFlow does not mandate how to parse packets
- Parsing consistency acheived via OXM pre-requisite

A.10.8 Controller role change mechanism

The controller role change mechanism is a simple mechanism to support multiple controllers for failover (EXT-39). This scheme is entirely driven by the controllers ; the switch only need to remember the role of each controller to help the controller election mechanism.

- Simple mechanism to support multiple controllers for failover
- Switches may now connect to multiple controllers in parallel
- Enable each controller to change its roles to equal, master or slave

A.10.9 Other changes

- Per-table metadata bitmask capabilities (EXT-34)
- Rudimentary group capabilities (EXT-61)
- Add hard timeout info in flow-removed messages (OFP-283)
- Add ability for controller to detect STP support(OFP-285)
- Turn off packet buffering with OFPCML_NO_BUFFER (EXT-45)
- Added ability to query all queues (EXT-15)
- Added experimenter queue property (EXT-16)
- Added max-rate queue property (EXT-21)
- Enable deleting flow in all tables (EXT-10)
- Enable switch to check chaining when deleting groups (EXT-12)
- Enable controller to disable buffering (EXT-45)
- Virtual ports renamed logical ports (EXT-78)
- New error messages (EXT-1, EXT-2, EXT-12, EXT-13, EXT-39, EXT-74 and EXT-82)
- Include release notes into the specification document
- Many other bug fixes, rewording and clarifications

A.11 OpenFlow version 1.3

Release date : April 13, 2012
Wire Protocol : 0x04

Please refers to the bug tracking ID for more details on each change

A.11.1 Refactor capabilities negotiation

Prior versions of the OpenFlow specification included limited expression of the capabilities of an OpenFlow switch. OpenFlow 1.3 include a more flexible framework to express capabilities (EXT-123).

The main change is the improved description of table capabilities. Those capabilities have been moved out of the table statistics structure in its own request/reply message, and encoded using a flexible TLV format. This enables the additions of next-table capabilities, table-miss flow entry capabilities and experimenter capabilities.

Other changes include renaming the 'stats' framework into the 'multipart' framework to reflect the fact that it is now used for both statistics and capabilities, and the move of port descriptions into its own multipart message to enable support of a greater number of ports.

List of features for Refactor capabilities negotiation :

- Rename 'stats' framework into the 'multipart' framework.
- Enable 'multipart' requests (requests spanning multiple messages).
- Move port list description to its own multipart request/reply.
- Move table capabilities to its own multipart request/reply.
- Create flexible property structure to express table capabilities.
- Enable to express experimenter capabilities.
- Add capabilities for table-miss flow entries.

- Add next-table (i.e. goto) capabilities

A.11.2 More flexible table miss support

Prior versions of the OpenFlow specification included table configuration flags to select one of three 3 behaviour for handling table-misses (packet not matching any flows in the table). OpenFlow 1.3 replace those limited flags with the table-miss flow entry, a special flow entry describing the behaviour on table miss (EXT-108).

The table-miss flow entry uses standard OpenFlow instructions and actions to process table-miss packets, this enables to use the full flexibility of OpenFlow in processing those packets. All previous behaviour expressed by the table-miss config flags can be expressed using the table-miss flow entry. Many new way of handling table-miss, such as processing table-miss with normal, can now trivially be described by the OpenFlow protocol.

- Remove table-miss config flags (EXT-108).
- Define table-miss flow entry as the all wildcard, lowest priority flow entry (EXT-108).
- Mandate support of the table-miss flow entry in every table to process table-miss packets (EXT-108).
- Add capabilities to describe the table-miss flow entry (EXT-123).
- Change table-miss default to drop packets (EXT-119).

A.11.3 IPv6 Extension Header handling support

Add the ability of match the presence of common IPv6 extension headers, and some anomalous conditions in IPv6 extension headers (EXT-38). A new OXM pseudo header field `OXM_OF_IPV6_EXTHDR` enables to match the following conditions :

- Hop-by-hop IPv6 extension header is present.
- Router IPv6 extension header is present.
- Fragmentation IPv6 extension header is present.
- Destination options IPv6 extension headers is present.
- Authentication IPv6 extension header is present.
- Encrypted Security Payload IPv6 extension header is present.
- No Next Header IPv6 extension header is present.
- IPv6 extension headers out of preferred order.
- Unexpected IPv6 extension header encountered.

A.11.4 Per flow meters

Add support for per-flow meters (EXT-14). Per-flow meters can be attached to flow entries and can measure and control the rate of packets. One of the main applications of per-flow meters is to rate limit packets sent to the controller.

The per-flow meter feature is based on a new flexible meter framework, which includes the ability to describe complex meters through the use of multiple metering bands, metering statistics and capabilities. Currently, only simple rate-limiter meters are defined over this framework. Support for color-aware

meters, which support Diff-Serv style operation and are tightly integrated in the pipeline, was postponed to a later release.

- Flexible meter framework based on per-flow meters and meter bands.
- Meter statistics, including per band statistics.
- Enable to attach meters flexibly to flow entries.
- Simple rate-limiter support (drop packets).

A.11.5 Per connection event filtering

Previous version of the specification introduced the ability for a switch to connect to multiple controllers for fault tolerance and load balancing. Per connection event filtering improves the multi-controller support by enabling each controller to filter events from the switch it does not want (EXT-120).

A new set of OpenFlow messages enables a controller to configure an event filter on its own connection to the switch. Asynchronous messages can be filtered by type and reason. This event filter comes in addition to other existing mechanisms that enable or disable asynchronous messages, for example the generation of flow-removed events can be configured per flow. Each controller can have a separate filter for the slave role and the master/equal role.

- Add asynchronous message filter for each controller connection.
- Controller message to set/get the asynchronous message filter.
- Set default filter value to match OpenFlow 1.2 behaviour.
- Remove `OFPC_INVALID_TTL_TO_CONTROLLER` config flag.

A.11.6 Auxiliary connections

In previous version of the specification, the channel between the switch and the controller is exclusively made of a single TCP connection, which does not allow to exploit the parallelism available in most switch implementations. OpenFlow 1.3 enables a switch to create auxiliary connections to supplement the main connection between the switch and the controller (EXT-114). Auxiliary connections are mostly useful to carry packet-in and packet-out messages.

- Enable switch to create auxiliary connections to the controller.
- Mandate that auxiliary connection can not exist when main connection is not alive.
- Add auxiliary-id to the protocol to disambiguate the type of connection.
- Enable auxiliary connection over UDP and DTLS.

A.11.7 MPLS BoS matching

A new OXM field `OXM_OF_MPLS_BOS` has been added to match the Bottom of Stack bit (BoS) from the MPLS header (EXT-85). The BoS bit indicates if other MPLS shim header are in the payload of the present MPLS packet, and matching this bit can help to disambiguate case where the MPLS label is reused across levels of MPLS encapsulation.

A.11.8 Provider Backbone Bridging tagging

Add support for tagging packet using Provider Backbone Bridging (PBB) encapsulation (EXT-105). This support enables OpenFlow to support various network deployment based on PBB, such as regular PBB and PBB-TE.

- Push and Pop operation to add PBB header as a tag.
- New OXM field to match I-SID for the PBB header.

A.11.9 Rework tag order

In previous version of the specification, the final order of tags in a packet was statically specified. For example, a MPLS shim header was always inserted after all VLAN tags in the packet. OpenFlow 1.3 removes this restriction, the final order of tags in a packet is dictated by the order of the tagging operations, each tagging operation adds its tag in the outermost position (EXT-121).

- Remove defined order of tags in packet from the specification.
- Tags are now always added in the outermost possible position.
- Action-list can add tags in arbitrary order.
- Tag order is predefined for tagging in the action-set.

A.11.10 Tunnel-ID metadata

The logical port abstraction enables OpenFlow to support a wide variety of encapsulations. The tunnel-id metadata `OXM_OF_TUNNEL_ID` is a new OXM field that expose to the OpenFlow pipeline metadata associated with the logical port, most commonly the demultiplexing field from the encapsulation header (EXT-107).

For example, if the logical port perform GRE encapsulation, the tunnel-id field would map to the GRE key field from the GRE header. After decapsulation, OpenFlow would be able to match the GRE key in the tunnel-id match field. Similarly, by setting the tunnel-id, OpenFlow would be able to set the GRE key in an encapsulated packet.

A.11.11 Cookies in packet-in

A cookie field was added to the packet-in message (EXT-7). This field takes its value from the flow the sends the packet to the controller. If the packet was not sent by a flow, this field is set to 0xffffffffffffffff.

Having the cookie in the packet-in enables the controller to more efficiently classify packet-in, rather than having to match the packet against the full flow table.

A.11.12 Duration for stats

A duration field was added to most statistics, including port statistics, group statistics, queue statistics and meter statistics (EXT-102). The duration field enables to more accurately calculate packet and byte rate from the counters included in those statistics.

A.11.13 On demand flow counters

New flow-mod flags have been added to disable packet and byte counters on a per-flow basis. Disabling such counters may improve flow handling performance in the switch.

A.11.14 Other changes

- Fix a bug describing VLAN matching (EXT-145).
- Flow entry description now mention priority (EXT-115).
- Flow entry description now mention timeout and cookies (EXT-147).
- Unavailable counters must now be set to all 1 (EXT-130).
- Correctly refer to flow entry instead of rule (EXT-132).
- Many other bug fixes, rewording and clarifications.

A.12 OpenFlow version 1.3.1

Release date : September 06, 2012
Wire Protocol : 0x04

Please refers to the bug tracking ID for more details on each change

A.12.1 Improved version negotiation

Prior versions of the OpenFlow specification included a simple scheme for version negotiation, picking the lowest of the highest version supported by each side. Unfortunately this scheme does not work properly in all cases, if both implementations don't implement all versions up to their highest version, the scheme can fail to negotiate a version they have in common (EXT-157).

The main change is adding a bitmap of version numbers in the Hello messages using during negotiation. By having the full list of version numbers, negotiation can always negotiate the appropriate version if one is available. This version bitmap is encoded in a flexible TLV format to retain future extensibility of the Hello message.

List of features for Improved version negotiation :

- Hello Elements, new flexible TLV format for Hello message
- Optional version bitmap in Hello messages.
- Improve version negotiation using optional version bitmaps.

A.12.2 Other changes

- Mandate that table-miss flow entry support drop and controller (EXT-158).
- Clarify the mapping of encapsulation data in `OXM_OF_TUNNEL_ID` (EXT-161).
- Rules and restrictions for UDP connections (EXT-162).
- Clarify virtual meters (EXT-165).
- Remove reference to switch fragmentation - confusing (EXT-172).
- Fix meter constant names to always be multipart (`OFPST_ => OFPMT_`) (EXT-184).

- Add `OFPG_*` definitions to spec (EXT-198).
- Add `ofp_instruction` and `ofp_table_feature_prop_header` in spec text (EXT-200).
- Bad error code in connection setup, must be `OFPHFC_INCOMPATIBLE` (EXT-201).
- Instructions must be a multiple of 8 bytes in length (EXT-203).
- Port status includes a reason, not a status (EXT-204).
- Clarify usage of table config field (EXT-205).
- Clarify that required match fields don't need to be supported in every flow table (EXT-206).
- Clarify that prerequisite does not require full match field support (EXT-206).
- Include in the spec missing definitions from openflow.h (EXT-207).
- Fix invalid error code `OFPQCFC_EPERM -> OFPSCFC_EPERM` (EXT-208).
- Clarify PBB language about B-VLAN (EXT-215)
- Fix inversion between source and destination ethernet addresses (EXT-215)
- Clarify how to reorder group buckets, and associated group bucket clarifications (EXT-217).
- Add disclaimer that release notes may not match specification (EXT-218)
- Figure 1 still says "Secure Channel" (EXT-222).
- OpenFlow version must be calculated (EXT-223).
- Meter band drop precedence should be increased, not reduced (EXT-225)
- Fix ambiguous uses of may/can/should/must (EXT-227)
- Fix typos (EXT-228)
- Many typos (EXT-231)

A.13 OpenFlow version 1.3.2

Release date : April 25, 2013
Wire Protocol : 0x04

Please refers to the bug tracking ID for more details on each change

A.13.1 Changes

- Mandate in OXM that 0-bits in mask must be 0-bits in value (EXT-238).
- Allow connection initiated from one of the controllers (EXT-252).
- Add clause on frame misordering to spec (EXT-259).
- Set table features doesn't generate flow removed messages (EXT-266).
- Fix description of set table features error response (EXT-267).
- Define use of `generation_id` in role reply messages (EXT-272).
- Switch with only one flow table are not mandated to implement goto (EXT-280).

A.13.2 Clarifications

- Clarify that MPLS Pop action uses Ethertype regardless of BOS bit (EXT-194).
- Controller message priorities using auxiliary connections (EXT-240).
- Clarify padding rules and variable size arrays (EXT-251).
- Better description buffer-id in flow mod (EXT-257).
- Semantic of `OFPPS_LIVE` (EXT-258).
- Improve multipart introduction (EXT-263).

- Clarify set table features description (EXT-266).
- Clarify meter flags and burst fields (EXT-270).
- Clarify slave access rights (EXT-271).
- Clarify that a switch can't change a controller role (EXT-276).
- Clarify roles of coexisting master and equal controllers (EXT-277).
- Various typos and rewording (EXT-282, EXT-288, EXT-290)

Appendix B Credits

Spec contributions, in alphabetical order:

Anders Nygren, Ben Pfaff, Bob Lantz, Brandon Heller, Casey Barker, Curt Beckmann, Dan Cohn, Dan Talayco, David Erickson, David McDysan, David Ward, Edward Crabbe, Fabian Schneider, Glen Gibb, Guido Appenzeller, Jean Tourrilhes, Johann Tonsing, Justin Pettit, KK Yap, Leon Poutievski, Lorenzo Vicisano, Martin Casado, Masahiko Takahashi, Masayoshi Kobayashi, Navindra Yadav, Nick McKeown, Nico dHeureuse, Peter Balland, Rajiv Ramanathan, Reid Price, Rob Sherwood, Saurav Das, Shashidhar Gandham, Tatsuya Yabe, Yiannis Yiakoumis, Zoltán Lajos Kis.

Pitfalls for ISP-friendly P2P design

Michael Piatek* Harsha V. Madhyastha† John P. John* Arvind Krishnamurthy* Thomas Anderson*

ABSTRACT

Peer-to-peer file sharing applications have become enormously popular over the past few years, coming to represent a large fraction of wide-area Internet traffic. A side effect of this explosive growth has been an emerging tussle between users, who want fast downloads, and ISPs, whose flat-rate pricing business model is threatened by the extreme volume of P2P traffic. Because ISP costs scale with usage while their prices do not, many ISPs have attempted to throttle or shut down P2P systems. Recently, several researchers have proposed that this tussle is unnecessary, that small changes in client and/or protocol behavior can lead to a "win-win" solution of better performance for end-users with less wide-area traffic for ISPs. Using a very large scale trace measurement of BitTorrent usage, we find evidence that such a win-win outcome is unlikely for at least one very popular P2P protocol.

1. INTRODUCTION

Peer-to-peer (P2P) networks have emerged as a powerful tool for building robust and scalable systems. Many P2P systems are unmanaged, yet achieve high levels of performance, scalability, and robustness through the use of *randomness* in their communication pattern. Randomization increases path diversity and avoids bottlenecks, but this robustness is not without cost. For example, the most popular use of P2P today is for file sharing in networks such as BitTorrent and eDonkey. Because files in these networks are shared with a random selection of peers spread across the globe, data being transferred often traverses multiple ISPs. The combination of the popularity of P2P services and their inefficiency means that file sharing accounts for a large portion of backbone ISP traffic [11, 21].

Network oblivious sharing increases costs for ISPs. As demand increases on transit links, backbone ISPs are forced to invest in increasing capacity. These costs are passed on to edge ISPs that pay transit costs proportional to the amount of interdomain traffic they generate. Many customer facing ISPs, however, offer flat-rate pricing, forcing them to absorb the costs externalized by P2P file sharing. In response, some ISPs have elected to rate-limit or simply block these "problem" protocols [4, 14], in turn leading developers into an arms race to evade restrictions.

Recent research has suggested that this emerging tussle between ISPs and their users can largely be avoided through small changes in client and/or protocol behavior. We call this class of techniques *ISP-friendly* in that they reduce the burden of P2P applications on providers. For example, the Ono system [3] proposes that the BitTorrent client be modified to prefer local peers, and the P4P project [20] provides a way for ISPs to notify clients which peers are preferred. Both claim that their designs are "win-win" for both users and ISPs: download speeds will improve and interdomain traffic will be reduced.

To validate these claims and quantitatively compare ISP-friendly design strategies, we conducted a very large scale measurement of BitTorrent usage. Our goal is to develop data to guide system and protocol designers in shaping the tussle space between users and their ISPs. Our measurements span almost twenty thousand swarms (groups of peers downloading the same file) encompassing nearly fifteen million unique IP addresses in total. By using simultaneous measurements of the Internet topology at scale, we can determine how often data in each swarm would transit the boundaries between ISPs. And indeed, we find that, in an idealized setting, most of the interdomain P2P traffic in our trace is unnecessary.

Our principal result, however, is that this benefit is difficult to achieve in practice. Specifically, we find the following pitfalls to adapting BitTorrent to be ISP-friendly:

- **Limited impact:** Contrary to the published literature, client-only optimizations to BitTorrent yield neither better performance nor less interdomain traffic in the common case. Our traces show that BitTorrent clients usually have too few peers to find many in the same ISP. To confirm this in the wild, we show that Ono reduces interdomain traffic by less than 1% when connecting to live swarms through a large residential ISP.
- **Reduced performance and robustness:** Optimizing for locality alone degrades the structural robustness of overlay topologies and, for many users, performance. In constrast to random topologies, those optimized for locality rely on fewer interdomain links to connect clusters of local peers. Also, local peers are not always faster, particularly for users in regions where asymmetric bandwidth capacities are typical, e.g., the US.
- **Conflicting interests:** Reducing interdomain traffic reduces costs for some ISPs, while it reduces revenue for others. We present trace data demonstrating that the set of tier-1 ISPs have a strong incentive to strategically manipulate BitTorrent peering relationships, creating longer paths than necessary and potentially setting up an arms race between ISPs.

In sum, the tussle over P2P traffic between users and ISPs, and between ISPs themselves, is likely to continue. We hope that these challenges will serve to motivate the community to revisit the issue of ISP-friendly P2P design

*Univ. of Washington
†Univ. of California, San Diego

from a holistic perspective, taking into account the interests of content providers, network operators, and users.

2. MEASUREMENT OVERVIEW

We are interested in understanding the interaction between file sharing and ISPs from an *Internet wide* perspective. An Internet wide perspective should include a characterization of the many objects and the many networks involved in distribution. Thus, we use large-scale measurements of both 1) membership in file sharing networks, i.e., which IPs are participating in which swarms, and 2) the AS paths between those peers. For the latter, we use measurements from the iPlane project [12], which refreshes its atlas of the Internet topology daily. We next describe our measurements of BitTorrent.

BitTorrent distributes large files by splitting them into blocks and distributing blocks out-of-order from the data source. Peers bootstrap into the overlay by contacting one or more central servers, called trackers, which maintain a list of active peers and provide a random subset of these upon client request. We focus our attention on BitTorrent because it is among the most popular P2P networks today and represents a significant volume of Internet traffic.

To collect a sample of BitTorrent peers, we crawled a set of well-known websites that aggregate .torrent metadata, downloading the set of 18,370 target swarms hosted by those websites. Crawled swarms range in popularity from new, popular swarms with tens of thousands of users to those with few participants. Swarm metadata includes the total size of the set of files to be downloaded, providing us with the demand (in bytes) of each user in each swarm. From a cluster of ten machines at the UW, we contacted the trackers associated with each swarm repeatedly over a one month period and requested membership information. Because many BitTorrent trackers return only a small subset of total available peers ($\sim$50), we made multiple requests per swarm, one from each of our measurement nodes, and repeated this query every 15 minutes (membership in each swarm was queried every ninety seconds, on average). Over the course of a month, we observed 14,380,622 distinct IPs, with many occurring in multiple swarms.

3. POTENTIAL REDUCTION IN INTER-DOMAIN TRAFFIC

We first use our traces to analyze locality optimization with cooperative users and complete topology information. This is the best case for reducing interdomain traffic; peers select paths to reduce traffic without regard to performance, and each client has complete knowledge of *all* other peers and AS-level paths to these peers. Of course, individual network operators may have more complex traffic engineering priorities, but reducing interdomain traffic is a broadly shared goal. We consider four approaches for peer selection at clients:

- Random: Matching with random existing users.

		Percentage reduction		
Method	Lifetimes	Small	Large	Overall
Shortest path	8 Hours	11.2	35.6	27.3
	6 Days	30.9	51.6	43.6
Same-AS	8 Hours	4.1	23.6	17.0
	6 Days	17.8	41.9	32.6
Latency	8 Hours	4.7	22.1	16.2
	6 Days	18.2	36.9	29.7

Table 1: Percentage reduction in interdomain traffic relative to random peer matching.

- Latency: Matching users with least delays.
- Same-AS: Matching with users from the same AS when possible. Otherwise, random selection.
- Shortest path: Matching with peers in order of ascending AS path length.

Random peering is the default behavior of BitTorrent trackers, and it serves as our basis for comparison. Some BitTorrent clients have been extended to use latency-based heuristics for choosing among the random set of peers available locally (e.g., [10]). We evaluate the application of a simple latency-based policy globally. Same-AS reflects ISP self-interest with respect to minimizing interdomain traffic, but without distinguishing between short interdomain paths and those that are lengthy. Shortest path attempts to minimize the use of interdomain links overall, representing—in some sense—the common good.

We apply each heuristic during playback of our trace. For each peer join event, we use our measurements to predict AS paths and latencies between the new user and existing peers. Candidate peers are rank ordered according to the given metric and selected in order until the new peer has either satisfied a connection requirement of 30 peers or has exhausted existing peers. To make our analysis tractable, we restrict our consideration to 1,000 randomly sampled swarms from our overall trace for a week long period. Subsequent trace analysis refers to this sample.

The main factor that controls locality in the BitTorrent sharing workload is choice. Only when users have many available sources from which to request data can they take advantage of locality-aware peer matching. When there are few available peers, users will exchange with anyone that can provide needed data, regardless of locality. Choice in a given swarm is determined by two properties: 1) the swarm's *fundamental* popularity and 2) peer lifetimes. Fundamental popularity refers to the total number of users interested in a data object over its lifetime. If these users are also persistent, i.e., they continue to share after a download completes, many choices will be available.

We distinguish among these four cases in our analysis as each has different traffic and locality properties. Specifically, we perform the same trace playback using each different peer matching strategy and with peers having either a 6 day or an 8 hour lifetime. Note that our trace method-

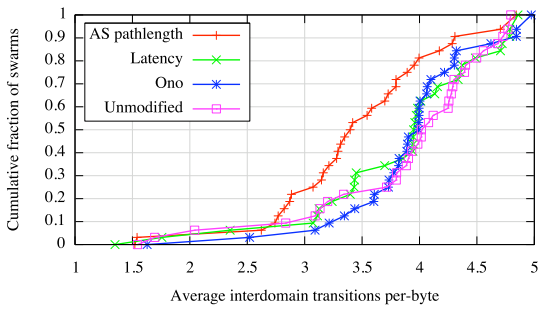


Figure 1: The cumulative fraction of swarms with a given average of interdomain transitions per-byte.

ology is unable to measure the actual peer lifetime, but other measurements indicated that most BitTorrent clients disconnect within a few hours after completing a download. Hence the 6 day lifetime is designed to illustrate what would happen if clients were incentivized to continue sharing well past download completion. We further separate these results into those for large, popular swarms (the top 10%) and remaining smaller swarms.

Results for each of these trials are summarized in Table 1. These results show across-the-board reductions in interdomain traffic. The greatest benefit is achieved when many choices are available; i.e., for popular swarms with long-lived peers. Moreover, locality optimization is most effective when using information about the underlying network topology; latency is a poor predictor of the number of interdomain crossings.

4. REDUCING INTERDOMAIN TRAFFIC IN PRACTICE

The encouraging results of our trace replay are consistent with published literature attesting to the benefits of locality-aware peer selection. In practice, however, we found these benefits difficult to achieve. In this section, we report results of a real-world comparison of three methods of reducing interdomain traffic.

From a measurement node connected via Comcast, a popular US residential cable ISP, we joined a set of 32 popular, recently created candidate swarms drawn from a popular BitTorrent aggregation website and performed back-to-back downloads with instrumented BitTorrent clients. Each download was performed four times with four different peer selection strategies: 1) shortest AS path length, 2) minimum latency, 3) the Ono client plugin, and 4) unmodified BitTorrent. To select peers with minimal AS path length or latency, we use path predictions from iPlane [12]. Ono estimates whether two peers are local by measuring the overlap in CDN replicas to which each node is directed [3], and peers with high CDN replica overlap are considered local. Because Ono is a plugin for the Azureus BitTorrent client that we use for all trials, we provide a direct, apples-to-apples comparison. To limit the time between the first and last downloads of a given swarm, we downloaded only the first 30 MB of each file.

Figure 1 compares the average interdomain transits per-byte obtained from downloading candidate swarms with each peer selection strategy. These results show the limited real world reduction in interdomain traffic realized by a single locality-aware client today. The median value is reduced from the unmodified baseline of 4.01 to 3.39 for shortest AS path, a reduction of just 15% with a negligible difference in performance (not shown in graph). The median ratio of download times between a client using shortest AS path and unmodified peer selection is 0.98. This is by design; we swap distant for local peers only when the switch is expected to maintain or improve performance. Our assumption is that users are unlikely to adopt a locality-aware client that reduces performance, which is a risk when optimizing for locality alone, a topic we return to in the next section.

The impact of Ono on both performance and locality is negligible. With respect to performance, the median ratio of download times between Ono and an unmodified Azureus BitTorrent client is 1.02. Figure 1 summarizes the impact on interdomain traffic. The median weighted AS path length for Ono is 3.99, versus 4.02 for the unmodified client. Although these results might seem contradictory given previously published measurements of Ono, the difference is simply one of presentation. While 33% *of Ono-recommended peers* are within a single AS and download rates increase by 31% *for recommended paths*, Ono's end-to-end benefit is limited by the vanishingly small fraction of peers it recommends as "local", even when applied to new, popular swarms.

These results expose many pitfalls in adapting BitTorrent to be ISP-friendly.

- Client-only ISP-friendly designs suffer from the lack of complete information regarding concurrent downloaders. Maximizing efficiency depends on peer matching with a global perspective, e.g., at the tracker.
- Download-and-depart behavior limits the potential for reducing interdomain traffic. Increasing exploitable locality depends on users continuing to share even after downloads complete. But, BitTorrent includes no incentives to do so.
- Even if local replicas exist, a client is likely to prefer non-local peers that provide higher download rates. In the next section, we consider the performance implications of optimizing for locality in isolation.

5. DOWNLOAD PERFORMANCE

Locality optimization exposes a performance tradeoff. Without any effort to remain performance neutral, preferential exchange with local peers may reduce performance for some users. The bandwidth capacity of BitTorrent users is highly skewed, and the majority of total capacity comes from a small minority of high capacity peers [7]. But, these peers are not uniformly distributed; clustering peers globally on the basis of locality also tends to clus-

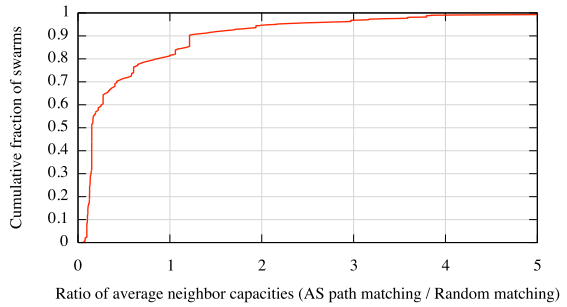


Figure 2: The ratio of the capacity of each user's set of local peers when matched to minimize AS path length length and randomly.

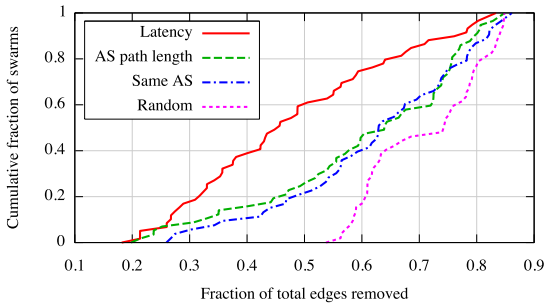


Figure 3: The cumulative fraction of swarms as a function of the fraction of edges removed to disconnect at least half of peers in the overlay topology.

ter them by capacity.[1] Although the total amount of capacity remains unchanged, clustering high capacity peers increases download rates for the high capacity minority while reducing the overall average download rate per-user.

To make these issues concrete, we consider the potential change in download rates resulting from an idealized shift to shortest AS path matching. We consider the 100 most popular swarms from our trace. For the set of observed peers from each of these swarms, we simulated a tracker that selected either 50 users at random or selected 50 users based on minimal AS path length. We assign capacities on a prefix level, using bandwidth measurements of more than 100,000 BitTorrent users collected from popular swarms in 2006 [7].

To express the change in per-user performance, we compute the ratio of the average capacity of each client's peers when matched to minimize AS path length and when matched randomly. A ratio greater than 1 implies that the average capacity of peers per-user increases when using shortest AS path matching and a ratio less than 1 implies that average capacity decreases. The distribution of these ratios is shown in Figure 2. These results show that for the majority of peers in the majority of swarms, the total capacity of their peers is greater under random matching than under shortest AS path matching. One might expect the median ratio of average download rate to be 1; i.e., for each peer in a swarm, some nearby peers will be slower, but others will faster. Instead, the median ratio is 0.15. This is because most BitTorrent *peers* from popular swarms in our trace come from the United States, while most *capacity* comes from comparatively high bandwidth peers in Europe.

6. STRUCTURAL ROBUSTNESS

To what extent does locality-aware peer selection degrade the resilience of the overlay graph? Unfortunately, there is no standard metric for quantifying the robustness of a network topology. We apply the following heuristic. For each peer in a swarm, we compute the shortest path in the overlay graph to all other nodes. Next, edges are ordered by their popularity among the set of shortest paths from all nodes. We then remove the most popular 1% of these edges and repeat this process until at least half the peers are disconnected from the largest connected component. This metric measures the extent to which the robustness of the overlay topology depends on a small minority of crucial connections that have relatively low redundancy.

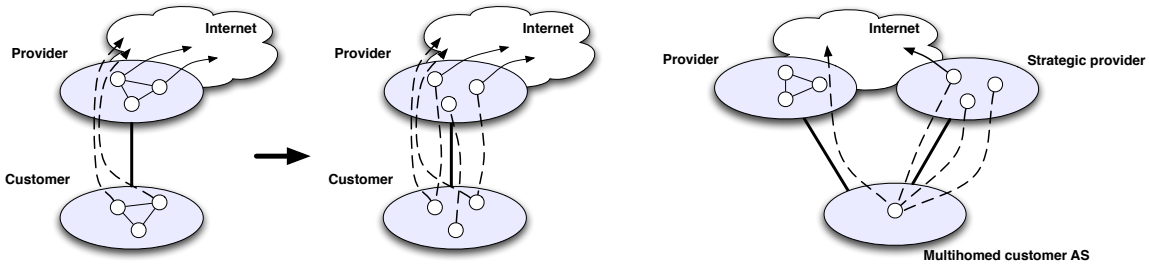
Figure 3 summarizes the results, showing the cumulative distribution of the fraction of edge removals required to disconnect half the overlay peers from the largest connected component. The median fraction of removals required decreases from the random baseline of 0.75 to 0.45 for a latency-based overlay. Using either same-AS or shortest path preferences results in more resilience but still falls short of the robustness of a randomly constructed overlay. This data reflects the varying impact of locality-aware selection depending on the type of swarm. Each selection strategy sometimes constructs overlays much more easily disconnected than random pairing (y-axis ≤ 0.1). These generally correspond to very popular swarms with significant exploitable locality amenable to a particular selection strategy. Also, for very unpopular swarms (y-axis ≥ 0.9), the selection strategy has little influence on robustness since choice among peers is so limited.

7. STRATEGIC ISPS

So far, we have considered ISPs as cooperating to achieve a common goal: reducing interdomain traffic. In practice, ISP-friendliness is not well-defined. Individual ISPs may have specific traffic engineering goals within their network that are not taken into account by our analysis. And, while minimizing interdomain traffic may represent the common good, individual ISPs derive little benefit from minimizing AS path length once traffic exits their network.

More fundamentally, what is friendly to one ISP may be unfriendly to another, as ISPs themselves have commercial relationships with each other that are strongly affected by user choices. Tier-1 ASes would prefer *more* interdomain traffic, not less, since their customers pay for transit traffic. For ASes with customers that pay based on peak usage or per-byte, an increase in locality means a likely reduction in revenue. This raises the question: can an ISP *increase* its revenue by influencing the file sharing choices

[1]BitTorrent's tit-for-tat policy attempts to achieve bandwidth matching *locally* by choosing among a subset of peers, but its effectiveness is limited by a partial view of potential peers, slow convergence, and churn [15].



(a) Single-homed customers (b) Multi-homed customers

Figure 4: Strategic peer matching. Circles denote P2P users and dashed lines represent monetized paths.

made by its clients, and how much of an impact would that have on global efficiency?

We next examine the following questions: 1) What mechanisms can a strategic ISP use to influence sharing among BitTorrent users? and 2) what is the impact of strategic ISP behavior on efficiency? We find that opportunities for subverting locality to increase revenue are frequent. Strategic behavior increases average path lengths relative to shortest AS path matching by 72.6%.

Matching strategy: Generally, a strategic AS would prefer that its users connect to other P2P users in other ASes according to its default BGP policy: with its 1) customers, 2) other users in its AS or peer ASes, or 3) users reachable through its provider(s), in decreasing order of preference. If the strategic AS is not a tier-1, clearly any match which requires transit on its provider links (case 3) should be avoided if possible. Alternatively, if the strategic AS can induce P2P users in its customer ASes to send and receive additional traffic through other customers (case 1), such matches are preferable as they may generate revenue. If such profitable matches cannot be found, matches that are made should prefer P2P users within the strategic AS or its peer ASes (case 2) and avoid the provider link. How might a strategic ISP implement this policy?

- In the case of BitTorrent, if the ISP hosts the tracker or can mandate the use of a particular client, it can implement a strategic policy directly.
- If a P2P network supports ISP-provided hints about which paths are ISP-friendly, as suggested by P4P [20] or the recently proposed ALTO/BGP IETF draft [16], a strategic ISP can explicitly mark peers on revenue generating paths as ISP-friendly regardless of locality.
- If the ISP can shape traffic on its network, it can indirectly induce profitable sharing by making revenue generating paths fast and expensive paths slow. By design, BitTorrent attempts to discover (and use) fast paths.

Although the strategic policy that we apply will never cause a strategic AS to lose revenue, it does not guarantee that each connection with a customer results in revenue gain. This depends on the circumstances of the customer. Figure 4 shows two typical cases: (a) when the strategic AS is the customer's only provider and default route, and (b) when the customer AS is multi-homed.

Single-homed: In (a), we compare strategic and locality aware matching. At left, P2P users in both provider and customer prefer local peers to minimize path length and interdomain traffic. Although several connections transit the provider, more revenue can potentially be generated by inducing local peers to connect to remote ones, as shown at the right of (a). In this case, P2P users in the provider AS are directed to connect preferentially to users of its customer AS. In this case, the monetized paths result in a net traffic gain. However, because most demand is unlikely to be satisfied by intradomain users, the potential for gain is limited. Most of the P2P traffic will traverse the provider regardless.

Multi-homed: For multi-homed customers (b), a strategic provider benefits by competing for transit P2P traffic from its customer, increasing the chance that a monetized path will generate billable traffic. Each customer connection represents billable data that may have been routed through another provider. In this case, a strategic ISP should try to induce users in a customer AS to communicate with its P2P users rather than those of an another provider.

In general, strategic ASes may not know detailed information about the routing policies of their customers, and so should be strategic with respect to all customers whether singly or multiply homed.

Efficiency loss: The strategic policy we apply avoids local peers, leading to longer AS path lengths on average. We quantify the extent of this increase by replaying our BitTorrent trace and choosing a target, strategic AS. For comparison, we record path lengths using the previously described shortest AS path and latency matching methods as well as random matching. When making strategic matches, we apply AS relationship data from CAIDA to predicted paths. While we report results for a single, large AS, similar trials for other ASes yield similar results.

Table 2 gives the overall efficiency loss for each method relative to shortest AS path and the overall efficiency gain relative to random matching. On the whole, strategic behavior results in an increase in interdomain paths by 72.6% relative to shortest path matching, with most of that increase resulting in revenue for the strategic ISP.

8. RELATED WORK

There have been various measurement studies of the traffic generated by P2P systems and evaluations of ways

	% reduction relative to random	% increase relative to shortest AS path
Shortest AS-path	51.4	0.0
Latency	40.7	21.9
Strategic	16.1	72.6

Table 2: Percentage change in AS path lengths relative to random and shortest AS path based peer matching.

to mitigate the resulting load. Saroiu et al. [17] and Gummadi et al. [6] examine the Gnutella and Kazaa workloads, document the increasing popularity of P2P systems, study the impact of caching and the potential for bandwidth savings of a locality aware mechanism. Sen and Wang [18] perform trace analysis of P2P traffic along the border routers of a single ISP and provide data that suggests that application-level traffic engineering might help.

Other researchers have studied the interactions between P2P systems and ISPs. Karagiannis et al. [8] study the impact of peer-assisted content distribution on ISPs. Also related to our work are efforts that examine whether ISPs and P2P systems can work together to perform traffic engineering and propose various solutions to achieve the necessary cooperation. For instance, Keralapura et al. [9] show that P2P systems could have an adverse impact on the stability of traffic engineering techniques currently used by ISPs in the absence of cooperation. Aggarwal et al. [1] and Bindal et al. [2] propose that ISPs use "oracles" to recommend peerings that are locality preserving.

The notion of ISPs manipulating existing protocols to accomplish traffic engineering goals or for strategic benefit has also received attention by researchers. Wang et al. report widespread use of path prepending to influence routing [19]. Mahajan et al. suggest additional protocol mechanisms by which ISPs coordinate their actions to overcome common inefficiencies in interdomain routing [13], but efficient outcomes depend on mutual trust between ISPs. More recently, Goldberg et al. [5] examine the incentives for ISPs to manipulate routing announcements to attract generic revenue-generating traffic and find that ensuring honesty likely requires substantial restriction in policy freedom. We apply similar ideas to the interaction between ISPs and P2P applications and quantify the potential for increasing revenue with measured workloads.

9. CONCLUSION

P2P systems and ISP operators currently have an adversarial relationship. The random matching of senders and receivers typical of current P2P file sharing networks generates significant amounts of interdomain traffic that increases costs for ISPs. In this paper, we have reported measurements of BitTorrent file sharing and network-level paths, examining the potential for locality-awareness to align the interests of users and ISPs. We find that while locality exists, simple heuristics are not sufficient to fully exploit it and may hamper network robustness. Further, large ISPs that provide transit service derive revenue from today's P2P file sharing patterns suggesting that systems designed to discover locality should not expect universal cooperation from ISPs.

10. REFERENCES

[1] V. Aggarwal, A. Feldmann, and C. Scheideler. Can ISPs and P2P systems cooperate for improved performance? *ACM CCR*, 2007.

[2] R. Bindal, P. Cao, W. Chan, J. Medval, G. Suwala, T. Bates, and A. Zhang. Improving Traffic Locality in BitTorrent via Biased Neighbor Selection. In *Proc. of IEEE ICDCS*, 2006.

[3] D. R. Choffnes and F. E. Bustamante. Taming the Torrent: A practical approach to reducing cross-ISP traffic in P2P systems. In *SIGCOMM*, 2008.

[4] M. Dischinger, A. Mislove, A. Haeberlen, and K. P. Gummadi. Detecting BitTorrent blocking. In *IMC*, 2008.

[5] S. Goldberg, S. Halevi, A. D. Jaggard, V. Ramachandran, and R. N. Wright. Rationality and traffic attraction: Incentives for honest path announcements in BGP. In *SIGCOMM*, 2008.

[6] K. P. Gummadi, R. Dunn, S. Saroiu, S. Gribble, H. Levy, and J. Zahorjan. Measurement, Modeling and Analysis of a Peer-to-Peer File-Sharing Workload. In *SOSP*, 2003.

[7] T. Isdal, M. Piatek, A. Krishnamurthy, and T. Anderson. Leveraging BitTorrent for end host measurements. In *Proc. of PAM*, 2007.

[8] T. Karagiannis, P. Rodriguez, and K. Papagiannaki. Should Internet Service Providers fear Peer-Assisted Content Distribution? In *IMC*, 2005.

[9] R. Keralapura, N. Taft, C.-N. Chuah, and G. Iannaccone. Can ISPs take the heat from overlay networks? In *HotNets*, 2004.

[10] J. Ledlie, P. Gardner, and M. Seltzer. Network Coordinates in the Wild. In *NSDI*, 2007.

[11] Lightreading.com. P2P plagues service providers. http://www.lightreading.com/document.asp?doc_id=31767, Apr. 2003.

[12] H. V. Madhyastha, T. Isdal, M. Piatek, C. Dixon, T. Anderson, A. Krishnamurthy, and A. Venkataramani. iPlane: An Information Plane for Distributed Services. In *OSDI*, 2006.

[13] R. Mahajan, D. Wetherall, and T. Anderson. Mutually controlled routing with independent ISPs. In *NSDI*, 2007.

[14] Packet Forgery by ISPs: A Report on the Comcast Affair. http://www.eff.org/wp/packet-forgery-isps-report-comcast-affair.

[15] M. Piatek, T. Isdal, A. Krishnamurthy, and T. Anderson. Do incentives build robustness in BitTorrent? In *NSDI*, 2007.

[16] P. Racz and Z. Despotovic. An ALTO service based on BGP routing information. http://www.ietf.org/id/draft-racz-bgp-based-alto-service-00.txt.

[17] S. Saroiu, K. P. Gummadi, R. Dunn, S. Gribble, and H. Levy. An Analysis of Internet Content Delivery Systems. In *OSDI*, 2002.

[18] S. Sen and J. Wang. Analyzing peer-to-peer traffic across large networks. *IEEE/ACM Trans. on Netw.*, 2004.

[19] H. Wang, R. K. C, C. Dah, M. Chiu, and J. C. S. Lui. Characterizing the performance and stability issues of the as path prepending method: taxonomy, measurement study and analysis. In *SIGCOMM Asia Workshop*, 2005.

[20] H. Xie, R. Yang, A. Krishnamurthy, Y. Liu, and A. Silberschatz. P4P: Provider portal for P2P applications. In *SIGCOMM*, 2008.

[21] ZDNet. ISPs see costs of file sharing rise. http://news.zdnet.com/2100-9584_22-1009456.html, 2003.

A Survey of Software-Defined Networking: Past, Present, and Future of Programmable Networks

Bruno Astuto A. Nunes, Marc Mendonca, Xuan-Nam Nguyen, Katia Obraczka, and Thierry Turletti

***Abstract*—The idea of *programmable networks* has recently re-gained considerable momentum due to the emergence of the Software-Defined Networking (SDN) paradigm. SDN, often referred to as a "radical new idea in networking", promises to dramatically simplify network management and enable innovation through network programmability. This paper surveys the state-of-the-art in programmable networks with an emphasis on SDN. We provide a historic perspective of programmable networks from early ideas to recent developments. Then we present the SDN architecture and the OpenFlow standard in particular, discuss current alternatives for implementation and testing of SDN-based protocols and services, examine current and future SDN applications, and explore promising research directions based on the SDN paradigm.**

***Index Terms*—Software-Defined Networking, programmable networks, survey, data plane, control plane, virtualization.**

I. Introduction

COMPUTER networks are typically built from a large number of network devices such as routers, switches and numerous types of middleboxes (i.e., devices that manipulate traffic for purposes other than packet forwarding, such as a firewall) with many complex protocols implemented on them. Network operators are responsible for configuring policies to respond to a wide range of network events and applications. They have to manually transform these high level-policies into low-level configuration commands while adapting to changing network conditions. And often they need to accomplish these very complex tasks with access to very limited tools. As a result, network management and performance tuning is quite challenging and thus error-prone. The fact that network devices are usually vertically-integrated *black boxes* exacerbates the challenge network operators and administrators face.

Another almost unsurmountable challenge network practitioners and researchers face has been referred to as "Internet ossification". Because of its huge deployment base and the fact it is considered part of our society's critical infrastructure (just like transportation and power grids), the Internet has become extremely difficult to evolve both in terms of its physical infrastructure as well as its protocols and performance. However, as current and emerging Internet applications and services become increasingly more complex and demanding, it is imperative that the Internet be able to evolve to address these new challenges.

Bruno Astuto A. Nunes, Xuan-Nam Nguyen and Thierry Turletti are with INRIA, France, {bruno.astuto-arouche-nunes, xuan-nam.nguyen, thierry.turletti}@inria.fr

Marc Mendonca and Katia Obraczka are with UC Santa Cruz, {msm, katia}@soe.ucsc.edu

The idea of "programmable networks" has been proposed as a way to facilitate network evolution. In particular, Software Defined Networking (SDN) is a new networking paradigm in which the forwarding hardware is decoupled from control decisions. It promises to dramatically simplify network management and enable innovation and evolution. The main idea is to allow software developers to rely on network resources in the same easy manner as they do on storage and computing resources. In SDN, the network intelligence is logically centralized in software-based controllers (the control plane), and network devices become simple packet forwarding devices (the data plane) that can be programmed via an open interface (e.g., ForCES [1], OpenFlow [2], etc).

SDN is currently attracting significant attention from both academia and industry. A group of network operators, service providers, and vendors have recently created the Open Network Foundation [3], an industrial-driven organization, to promote SDN and standardize the OpenFlow protocol [2]. On the academic side, the OpenFlow Network Research Center [4] has been created with a focus on SDN research. There have also been standardization efforts on SDN at the IETF and IRTF and other standards producing organizations.

The field of software defined networking is quite recent, yet growing at a very fast pace. Still, there are important research challenges to be addressed. In this paper, we survey the state-of-the-art in programmable networks by providing a historic perspective of the field and also describing in detail the SDN paradigm and architecture. The paper is organized as follows: in Section II, it begins by describing early efforts focusing on programmable networks. Section III provides an overview of SDN and its architecture. It also describes the OpenFlow protocol. Section IV describes existing platforms for developing and testing SDN solutions including emulation and simulation tools, SDN controller implementations, as well as verification and debugging tools. In Section V, we discuss several SDN applications in areas such as data centers and wireless networking. Finally, Section VI discusses research challenges and future directions.

II. Early Programmable Networks

SDN has great potential to change the way networks operate, and OpenFlow in particular has been touted as a "radical new idea in networking" [5]. The proposed benefits range from centralized control, simplified algorithms, commoditizing network hardware, eliminating middleboxes, to enabling the design and deployment of third-party 'apps'.

While OpenFlow has received considerable attention from industry, it is worth noting that the idea of programmable

networks and decoupled control logic has been around for many years. In this section, we provide an overview of early programmable networking efforts, precursors to the current SDN paradigm that laid the foundation for many of the ideas we are seeing today.

a) **Open Signaling:** The Open Signaling (OPENSIG) working group began in 1995 with a series of workshops dedicated to "making ATM, Internet and mobile networks more open, extensible, and programmable" [6]. They believed that a separation between the communication hardware and control software was necessary but challenging to realize; this is mainly due to vertically integrated switches and routers, whose closed nature made the rapid deployment of new network services and environments impossible. The core of their proposal was to provide access to the network hardware via open, programmable network interfaces; this would allow the deployment of new services through a distributed programming environment.

Motivated by these ideas, an IETF working group was created, which led to the specification of the General Switch Management Protocol (GSMP) [7], a general purpose protocol to control a label switch. GSMP allows a controller to establish and release connections across the switch, add and delete leaves on a multicast connection, manage switch ports, request configuration information, request and delete reservation of switch resources, and request statistics. The working group is officially concluded and the latest standards proposal, GSMPv3, was published in June 2002.

b) **Active Networking:** Also in the mid 1990s, the Active Networking [8], [9] initiative proposed the idea of a network infrastructure that would be programmable for customized services. There were two main approaches being considered, namely: (1) user-programmable switches, with in-band data transfer and out-of-band management channels; and (2) capsules, which were program fragments that could be carried in user messages; program fragments would then be interpreted and executed by routers. Despite considerable activity it motivated, Active Networking never gathered critical mass and transferred to widespread use and industry deployment, mainly due to practical security and performance concerns [10].

c) **DCAN:** Another initiative that took place in the mid 1990s is the Devolved Control of ATM Networks (DCAN) [11]. The aim of this project was to design and develop the necessary infrastructure for scalable control and management of ATM networks. The premise is that control and management functions of the many devices (ATM switches in the case of DCAN) should be decoupled from the devices themselves and delegated to external entities dedicated to that purpose, which is basically the concept behind SDNs. DCAN assumes a minimalist protocol between the manager and the network, in the lines of what happens today in proposals such as OpenFlow. More on the DCAN project can be found at [12].

Still in the lines of SDNs and the proposed decoupling of control and data plane over ATM networks, amongst others, in the work proposed in [13] multiple heterogeneous control architectures are allowed to run simultaneously over single physical ATM network by partitioning the resources of that switch between those controllers.

d) **4D Project:** Starting in 2004, the 4D project [14], [15], [16] advocated a clean slate design that emphasized separation between the routing decision logic and the protocols governing the interaction between network elements. It proposed giving the "decision" plane a global view of the network, serviced by a "dissemination" and "discovery" plane, for control of a "data" plane for forwarding traffic. These ideas provided direct inspiration for later works such as NOX [17], which proposed an "operating system for networks" in the context of an OpenFlow-enabled network.

e) **NETCONF:** In 2006, the IETF Network Configuration Working Group proposed NETCONF [18] as a management protocol for modifying the configuration of network devices. The protocol allowed network devices to expose an API through which extensible configuration data could be sent and retrieved.

Another management protocol, widely deployed in the past and used until today, is the SNMP [19]. SNMP was proposed in the late 80's and proved to be a very popular network management protocol, which uses the Structured Management Interface (SMI) to fetch data contained in the Management Information Base (MIB). It could be used as well to change variables in the MIB in order to modify configuration settings. It later became apparent that in spite of what it was originally intended for, SNMP was not being used to configure network equipment, but rather as a performance and fault monitoring tool. Moreover, multiple shortcomings were detected in the conception of SNMP, the most notable of which was its lack of strong security. This was addressed in a later version of the protocol.

NETCONF, at the time it was proposed by IETF, was seen by many as a new approach for network management that would fix the aforementioned shortcomings in SNMP. Although the NETCONF protocol accomplishes the goal of simplifying device (re)configuration and acts as a building block for management, there is no separation between data and control planes. The same can be stated about SNMP. A network with NETCONF should not be regarded as fully programmable as any new functionality would have to be implemented at both the network device and the manager so that any new functionality can be provided; furthermore, it is designed primarily to aid automated configuration and not for enabling direct control of state nor enabling quick deployment of innovative services and applications. Nevertheless, both NETCONF and SNMP are useful management tools that may be used in parallel on hybrid switches supporting other solutions that enable programmable networking.

The NETCONF working group is currently active and the latest proposed standard was published in June 2011.

f) **Ethane:** The immediate predecessor to OpenFlow was the SANE / Ethane project [20], which, in 2006, defined a new architecture for enterprise networks. Ethane's focus was on using a centralized controller to manage policy and security in a network. A notable example is providing identity-based access control. Similar to SDN, Ethane employed two components: a *controller* to decide if a packet should be forwarded, and an *Ethane switch* consisting of a flow table

and a secure channel to the controller.

Ethane laid the foundation for what would become Software-Defined Networking. To put Ethane in the context of today's SDN paradigm, Ethane's identity-based access control would likely be implemented as an application on top of an SDN controller such as NOX [17], Maestro [21], Beacon [22], SNAC [23], Helios [24], etc.

III. Software-Defined Networking Architecture

Data communication networks typically consist of end-user devices, or hosts interconnected by the network infrastructure. This infrastructure is shared by hosts and employs switching elements such as routers and switches as well as communication links to carry data between hosts. Routers and switches are usually "closed" systems, often with limited- and vendor-specific control interfaces. Therefore, once deployed and in production, it is quite difficult for current network infrastructure to evolve; in other words, deploying new versions of existing protocols (e.g., IPv6), not to mention deploying completely new protocols and services is an almost insurmountable obstacle in current networks. The Internet, being a network of networks, is no exception.

As mentioned previously, the so-called Internet "ossification" [2] is largely attributed to the tight coupling between the data– and control planes which means that decisions about data flowing through the network are made on-board each network element. In this type of environment, the deployment of new network applications or functionality is decidedly non-trivial, as they would need to be implemented directly into the infrastructure. Even straightforward tasks such as configuration or policy enforcement may require a good amount of effort due to the lack of a common control interface to the various network devices. Alternatively, workarounds such as using "middleboxes" (e.g., firewalls, Intrusion Detection Systems, Network Address Translators, etc.) overlayed atop the underlying network infrastructure have been proposed and deployed as a way to circumvent the network ossification effect. Content Delivery Networks (CDNs) [25] are a good example.

Software-Defined Networking was developed to facilitate innovation and enable simple programmatic control of the network data-path. As visualized in Figure 1, the separation of the forwarding hardware from the control logic allows easier deployment of new protocols and applications, straightforward network visualization and management, and consolidation of various middleboxes into software control. Instead of enforcing policies and running protocols on a convolution of scattered devices, the network is reduced to "simple" forwarding hardware and the decision-making network controller(s).

A. Current SDN Architectures

In this section, we review two well-known SDN architectures, namely ForCES [1] and Openflow [2]. Both OpenFlow and ForCES follow the basic SDN principle of separation between the control and data planes; and both standardize information exchange between planes. However, they are technically very different in terms of design, architecture, forwarding model, and protocol interface.

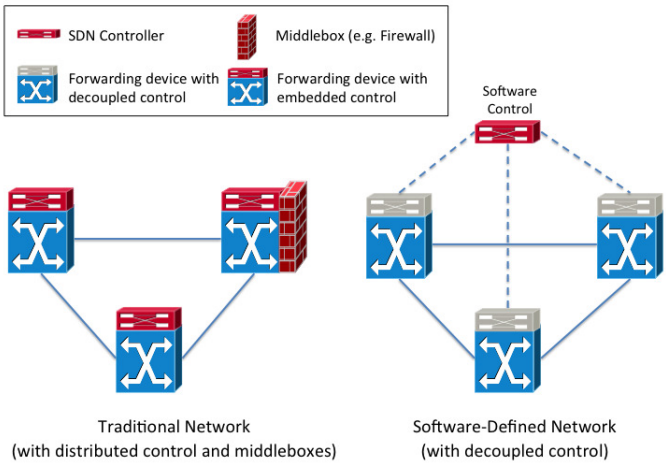


Fig. 1. The SDN architecture decouples control logic from the forwarding hardware, and enables the consolidation of middleboxes, simpler policy management, and new functionalities. The solid lines define the data-plane links and the dashed lines the control-plane links.

1) **ForCES:** The approach proposed by the IETF ForCES (Forwarding and Control Element Separation) Working Group, redefines the network device's internal architecture having the control element separated from the forwarding element. However, the network device is still represented as a single entity. The driving use case provided by the working group considers the desire to combine new forwarding hardware with third-party control within a single network device. Thus, the control and data planes are kept within close proximity (e.g., same box or room). In contrast, the control plane is ripped entirely from the network device in "OpenFlow-like" SDN systems.

ForCES defines two logic entities called the Forwarding Element (FE) and the Control Element (CE), both of which implement the ForCES protocol to communicate. The FE is responsible for using the underlying hardware to provide per-packet handling. The CE executes control and signaling functions and employs the ForCES protocol to instruct FEs on how to handle packets. The protocol works based on a master-slave model, where FEs are slaves and CEs are masters.

An important building block of the ForCES architecture is the LFB (Logical Function Block). The LFB is a well-defined functional block residing on the FEs that is controlled by CEs via the ForCES protocol. The LFB enables the CEs to control the FEs' configuration and how FEs process packets.

ForCES has been undergoing standardization since 2003, and the working group has published a variety of documents including: an applicability statement, an architectural framework defining the entities and their interactions, a modeling language defining the logical functions within a forwarding element, and the protocol for communication between the control and forwarding elements within a network element. The working group is currently active.

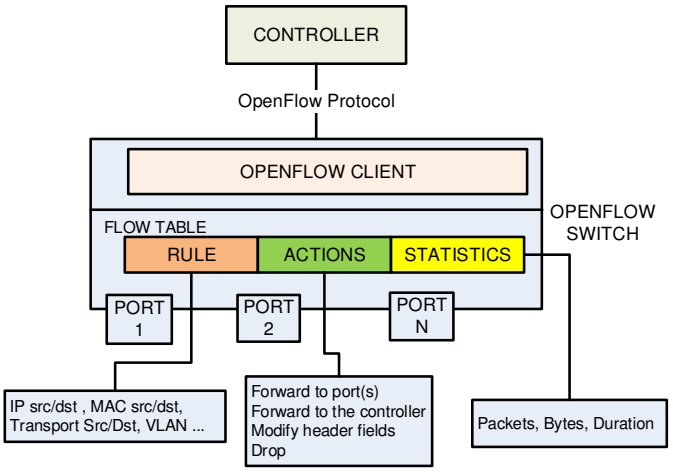


Fig. 2. Communication between the controller and the forwarding devices happens via OpenFlow protocol. The flow tables are composed by matching rules, actions to be taken when the flow matches the rules, and counters for collecting flow statistics.

2) **OpenFlow:** Driven by the SDN principle of decoupling the control and data forwarding planes, OpenFlow [2], like ForCES, standardizes information exchange between the two planes.

In the OpenFlow architecture, illustrated in Figure 2, the forwarding device, or OpenFlow switch, contains one or more *flow tables* and an abstraction layer that securely communicates with a *controller* via OpenFlow protocol. Flow tables consist of flow entries, each of which determines how packets belonging to a flow will be processed and forwarded. Flow entries typically consist of: (1) *match fields*, or matching rules, used to match incoming packets; match fields may contain information found in the packet header, ingress port, and metadata; (2) *counters*, used to collect statistics for the particular flow, such as number of received packets, number of bytes and duration of the flow; and (3) a *set of instructions*, or *actions*, to be applied upon a match; they dictate how to handle matching packets.

Upon a packet arrival at an OpenFlow switch, packet header fields are extracted and matched against the matching fields portion of the flow table entries. If a matching entry is found, the switch applies the appropriate set of instructions, or actions, associated with the matched flow entry. If the flow table look-up procedure does not result on a match, the action taken by the switch will depend on the instructions defined by the *table-miss* flow entry. Every flow table must contain a table-miss entry in order to handle table misses. This particular entry specifies a set of actions to be performed when no match is found for an incoming packet, such as dropping the packet, continue the matching process on the next flow table, or forward the packet to the controller over the OpenFlow channel. It is worth noting that from version 1.1 OpenFlow supports multiple tables and pipeline processing. Another possibility, in the case of *hybrid switches*, i.e., switches that have both OpenFlow– and non-OpenFlow ports, is to forward non-matching packets using regular IP forwarding schemes.

The communication between controller and switch happens via OpenFlow protocol, which defines a set of messages that can be exchanged between these entities over a secure channel. Using the OpenFlow protocol a remote controller can, for example, add, update, or delete flow entries from the switch's flow tables. That can happen *reactively* (in response to a packet arrival) or *proactively*.

3) **Discussion:** In [26], the similarities and differences between ForCES and OpenFlow are discussed. Among the differences, they highlight the fact that the forwarding model used by ForCES relies on the Logical Function Blocks (LFBs), while OpenFlow uses flow tables. They point out that in OpenFlow actions associated with a flow can be combined to provide greater control and flexibility for the purposes of network management, administration, and development. In ForCES the combination of different LFBs can also be used to achieve the same goal.

We should also re-iterate that ForCES does not follow the same SDN model underpinning OpenFlow, but can be used to achieve the same goals and implement similar functionality [26].

The strong support from industry, research, and academia that the Open Networking Foundation (ONF) and its SDN proposal, OpenFlow, has been able to gather is quite impressive. The resulting critical mass from these different sectors has produced a significant number of deliverables in the form of research papers, reference software implementations, and even hardware. So much so that some argue that OpenFlow's SDN architecture is the current SDN de-facto standard. In line with this trend, the remainder of this section focuses on OpenFlow's SDN model. More specifically, we will describe the different components of the SDN architecture, namely: the switch, the controller, and the interfaces present on the controller for communication with forwarding devices (southbound communication) and network applications (northbound communication). Section IV also has an OpenFlow focus as it describes existing platforms for SDN development and testing, including emulation and simulation tools, SDN controller implementations, as well as verification and debugging tools. Our discussion of future SDN applications and research directions is more general and is SDN architecture agnostic.

B. Forwarding Devices

The underlaying network infrastructure may involve a number of different physical network equipment, or forwarding devices such as routers, switches, virtual switches, wireless access points, to name a few. In a software-defined network, such devices are often represented as basic forwarding hardware accessible via an open interface at an abstraction layer, as the control logic and algorithms are off-loaded to a controller. Such forwarding devices are commonly referred to, in SDN terminology, simply as "switches", as illustrated in Figure 3.

In an OpenFlow network, switches come in two varieties: pure and hybrid. Pure OpenFlow switches have no legacy features or on-board control, and completely rely on a controller for forwarding decisions. Hybrid switches support OpenFlow in addition to traditional operation and protocols. Most commercial switches available today are hybrids.

1) Processing Forwarding Rules: Flow-based SDN architectures such as OpenFlow may utilize additional forwarding table entries, buffer space, and statistical counters that are difficult to implement in traditional ASIC switches. Some recent proposals [27], [28] have advocated adding a general-purpose CPU, either on-switch or nearby, that may be used to supplement or take over certain functions and reduce the complexity of the ASIC design. This would have the added benefit of allowing greater flexibility for on-switch processing as some aspects would be software-defined.

In [29], network processor based acceleration cards were used to perform OpenFlow switching. They proposed and described the design options and reported results that showed a 20% reduction on packet delay. In [30], an architectural design to improve look-up performance of OpenFlow switching in Linux was proposed. Preliminary results reported showed a packet switching throughput increase of up to 25% compared to the throughput of regular software-based OpenFlow switching. Another study on data-plane performance over Linux based Openflow switching was presented in [31], which compared OpenFlow switching, layer-2 Ethernet switching and layer-3 IP routing performance. Fairness, forwarding throughput and packet latency in diverse load conditions were analyzed. In [32], a basic model for the forwarding speed and blocking probability of an OpenFlow switch was derived, while the parameters for the model were drawn from measurements of switching times of current OpenFlow hardware, combined with an OpenFlow controller.

2) Installing Forwarding Rules: Another issue regarding the scalability of an OpenFlow network is memory limitation in forwarding devices. OpenFlow rules are more complex than forwarding rules in traditional IP routers. They support more flexible matchings and matching fields and also different actions to be taken upon packet arrival. A commodity switch normally supports between a few thousand up to tens of thousands forwarding rules [33]. Also, Ternary Content-Addressable Memory (TCAM) has been used to support forwarding rules, which can be expensive and power-hungry. Therefore, the rule space is a bottleneck to the scalability of OpenFlow, and the optimal use of the rule space to serve a scaling number of flow entries while respecting network policies and constraints is a challenging and important topic.

Some proposals address memory limitations in OpenFlow switches. Devoflow [34] is an extension to OpenFlow for high-performance networks. It handles mice flows (i.e. short flows) at the OpenFlow switch and only invokes the controller in order to handle elephant flows (i.e larger flows). The performance evaluation conducted in [34] showed that Devoflow uses 10 to 53 times less flow table space. In DIFANE [35], "ingress" switches redirect packets to "authority" switches that store all the forwarding rules while ingress switches cache flow table rules for future use. The controller is responsible for partitioning rules over authority switches.

Palette [36] and One Big Switch [37] address the rule placement problem. Their goal is to minimize the number of rules that need to be installed in forwarding devices and use end-to-end policies and routing policies as input to a rule placement optimizer. End-to-end policies consist of a set of prioritized rules dictating, for example, access control and load balancing, while viewing the whole network as a single virtual switch. Routing policies, on the other hand, dictate through what paths traffic should flow in the network. The main idea in Palette is to partition end-to-end policies into sub tables and then distribute them over the switches. Their algorithm consists of two steps: determine the number k of tables needed and then partition the rules set over k tables. One Big Switch, on the other hand, solves the rule placement problem separately for each path, choosing the paths based on network metrics (e.g. latency, congestion and bandwidth), and then combining the result to reach a global solution.

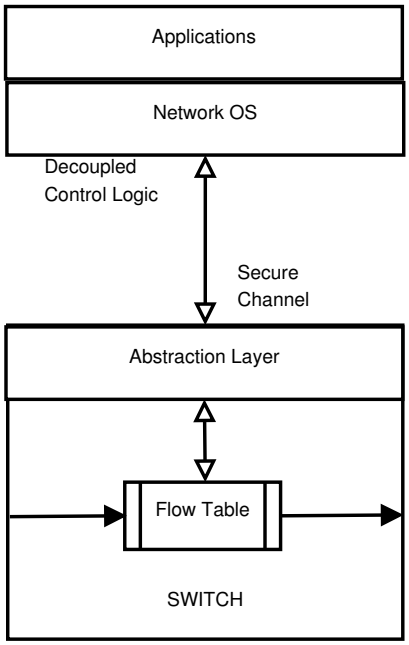


Fig. 3. The separated control logic can be viewed as a network operating system, upon which applications can be built to "program" the network.

C. The Controller

The decoupled system has been compared to an operating system [17], in which the controller provides a programmatic interface to the network. That can be used to implement management tasks and offer new functionalities. A layered view of this model is illustrated in Figure 3. This abstraction assumes the control is centralized and applications are written as if the network is a single system. It enables the SDN model to be applied over a wide range of applications and heterogeneous network technologies and physical media such as wireless (e.g. 802.11 and 802.16), wired (e.g. Ethernet) and optical networks.

As a practical example of the layering abstraction accessible through open application programming interfaces (APIs), Figure 4 illustrates the architecture of an SDN controller based on the OpenFlow protocol. This specific controller is a fork of the Beacon controller [22] called Floodlight [38]. In this figure it is possible to observe the separation between the controller and the application layers. Applications can be written in Java and can interact with the built-in controller modules via a JAVA API. Other applications can be written in different languages

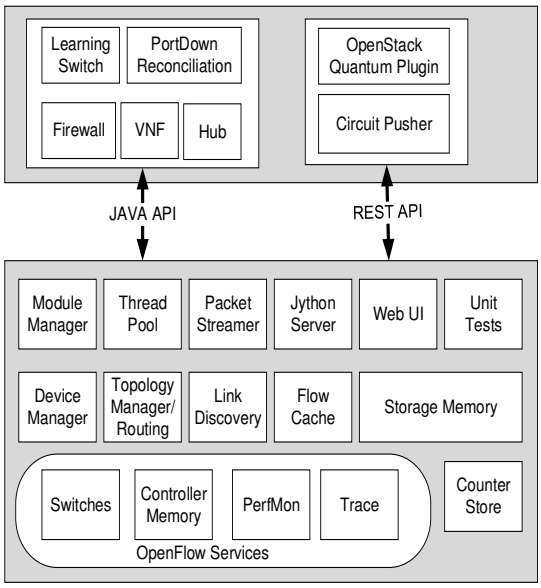


Fig. 4. The Floodlight architecture as an example of an OpenFlow controller.

and interact with the controller modules via the REST API. This particular example of an SDN controller allows the implementation of built-in modules that can communicate with their implementation of the OpenFlow controller (i.e. OpenFlow Services). The controller, on the other hand, can communicate with the forwarding devices via the OpenFlow protocol through the abstraction layer present at the forwarding hardware, illustrated in Figure 3.

While the aforementioned layering abstractions accessible via open APIs allow the simplification of policy enforcement and management tasks, the bindings must be closely maintained between the control and the network forwarding elements. The choices made while implementing such layering architectures can dramatically influence the performance and scalability of the network. In the following, we address some such scalability concerns and go over some proposals that aim on overcoming these challenges. We leave a more detailed discussion on the application layer and the implementation of services and policy enforcement to Section VI-C.

1) Control Scalability: An initial concern that arises when offloading control from the switching hardware is the scalability and performance of the network controller(s). The original Ethane [20] controller, hosted on a commodity desktop machine, was tested to handle up to 11,000 new flow requests per second and responded within 1.5 milliseconds. A more recent study [39] of several OpenFlow controller implementations (NOX-MT, Maestro, Beacon), conducted on a larger emulated network with 100,000 endpoints and up to 256 switches, found that all were able to handle at least 50,000 new flow requests per second in each of the tested scenarios. On an eight-core machine, the multi-threaded NOX-MT implementation handled 1.6 million new flow requests per second with an average response time of 2 milliseconds. As the results show, a single controller is able to handle a surprising number of new flow requests, and should be able to manage all but the largest networks. Furthermore, new controllers under development such as McNettle [40] target powerful multicore servers and are being designed to scale up to large data center workloads (around 20 million flows requests per second and up to 5000 switches). Nonetheless, multiple controllers may be used to reduce latency or increase fault tolerance.

A related concern is the controller placement problem [41], which attempts to determine both the optimal number of controllers and their location within the network topology, often choosing between optimizing for average and worst case latency. The latency of the link used for communication between controller and switch is of great importance when dimensioning a network or evaluating its performance [34]. That was one of the main motivations behind the work in [42] which evaluated how the controller and the network perform with bandwidth and latency issues on the control link. This work concludes that bandwidth in the control link arbitrates how many flows can be processed by the controller, as well as the loss rate when under saturation conditions. The switch-to-control latency on the other hand, has a major impact on the overall behavior of the network, as each switch cannot forward data until it receives the message from the controller that inserts the appropriate rules in the flow table. This interval can grow with the link latency and impact dramatically the performance of network applications.

Also, control modeling greatly impacts the network scalability. Some important scalability issues are presented in [43], along with a discussion about scalability trade-offs in software-defined network design.

2) Control models: In the following, we go over some of these SDN design options and discuss different methods of controlling a software-defined network, many of which are interrelated:

- **Centralized vs. Distributed**
  Although protocols such as OpenFlow specify that a switch is controlled by a controller and therefore appears to imply centralization, software-defined networks may have either a centralized or distributed control-plane. Though controller-to-controller communication is not defined by OpenFlow, it is necessary for any type of distribution or redundancy in the control-plane.
  A physically centralized controller represents a single point of failure for the entire network; therefore, OpenFlow allows the connection of multiple controllers to a switch, which would allow backup controllers to take over in the event of a failure.
  Onix [44] and HyperFlow [45] take the idea further by attempting to maintain a logically centralized but physically distributed control plane. This decreases the look-up overhead by enabling communication with local controllers, while still allowing applications to be written with a simplified central view of the network. The potential downside are trade-offs [46] related to consistency and staleness when distributing state throughout the control plane, which has the potential to cause applications that believe they have an accurate view of the network to act incorrectly.
  A hybrid approach, such as Kandoo [47], can utilize local controllers for local applications and redirect to a global

controller for decisions that require centralized network state. This reduces the load on the global controller by filtering the number of new flow requests, while also providing the data-path with faster responses for requests that can be handled by a local control application.

A software-defined network can also have some level of logical decentralization, with multiple logical controllers. An interesting type of proxy controller, called Flowvisor [48], can be used to add a level of network virtualization to OpenFlow networks and allow multiple controllers to simultaneously control overlapping sets of physical switches. Initially developed to allow experimental research to be conducted on deployed networks alongside production traffic, it also facilitates and demonstrates the ease of deploying new services in SDN environments.

A logically decentralized control plane would be needed in an inter-network spanning multiple administrative domains. Though the domains may not agree to centralized control, a certain level of sharing may be appropriate (e.g., to ensure service level agreements are met for traffic flowing between domains).

- **Control Granularity**

  Traditionally, the basic unit of networking has been the packet. Each packet contains address information necessary for a network switch to make routing decisions. However, most applications send data as a flow of many individual packets. A network that wishes to provide QoS or service guarantees to certain applications may benefit from individual flow-based control. Control can be further abstracted to an aggregated flow-match, rather than individual flows. Flow aggregation may be based on source, destination, application, or any combination thereof.

  In a software-defined network where network elements are controlled remotely, overhead is caused by traffic between the data-plane and control-plane. As such, using packet level granularity would incur additional delay as the controller would have to make a decision for each arriving packet. When controlling individual flows, the decision made for the first packet of the flow can be applied to all subsequent packets of that flow. The overhead may be further reduced by grouping flows together, such as all traffic between two hosts, and performing control decisions on the aggregated flows.

- **Reactive vs. Proactive Policies**

  Under a *reactive* control model, such as the one proposed by Ethane [20], forwarding elements must consult a controller each time a decision must be made, such as when a packet from a new flow reaches a switch. In the case of flow-based control granularity, there will be a small performance delay as the first packet of each new flow is forwarded to the controller for decision (e.g., forward or drop), after which future packets within that flow will travel at line rate within the forwarding hardware. While the delay incurred by the first-packet may be negligible in many cases, it may be a concern if the controller is geographically remote (though this can be mitigated by physically distributing the controller [45]) or if most flows are short-lived, such as single-packet flows. There are also some scalability issues in larger networks, as the controller must be able to handle a larger volume of new flow requests.

  Alternatively, *proactive* control approaches push policy rules from the controller to the switches. A good example of proactive control is DIFANE [35], which partitions rules over a hierarchy of switches, such that the controller rarely needs to be consulted about new flows and traffic is kept within the data-plane. In their experiments, DIFANE reduces first-packet delay from a 10ms average round-trip time (RTT) with a centralized NOX controller to a 0.4ms average RTT for new single-packet flows. It was also shown to increase the new flow throughput, as the tested version of NOX achieved a peak of 50,000 single-packet flows per second while the DIFANE solution achieved 800,000 single-packet flows per second. Interestingly, it was observed that the OpenFlow switch's local controller implementation becomes a bottleneck before the central NOX controller. This was attributed to the fact that commercial OpenFlow switch implementations were limited to sending 60-330 new flows requests per second at the time of their publication (2010).

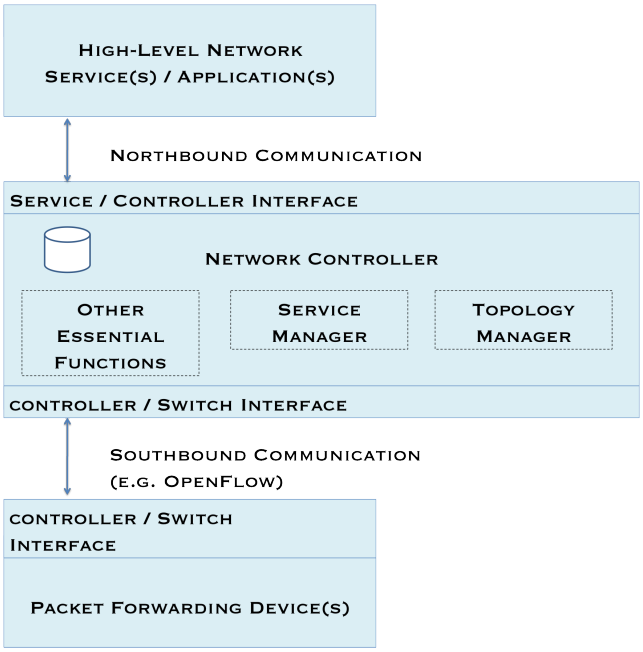


Fig. 5. A controller with a northbound and southbound interface.

As shown in Figure 5, a controller that acts as a network operating system must implement at least two interfaces: a "southbound" interface that allows switches to communicate with the controller and a "northbound" interface that presents an API to network control and high-level applications/services.

D. Southbound Communication: Controller-Switch

An important aspect of SDNs is the link between the data-plane and the control-plane. As forwarding elements are controlled by an open interface, it is important that this link remains available and secure.

The OpenFlow protocol can be viewed as one possible implementation of controller-switch interactions, as it defines the communication between the switching hardware and a network controller. For security, OpenFlow 1.3.0 provides optional support for encrypted Transport Layer Security (TLS) communication and a certificate exchange between the switches and the controller(s); however, the exact implementation and certificate format is not currently specified. Also outside the scope of the current specification are fine-grained security options regarding scenarios with multiple controllers, as there is no method specified to only grant partial access permissions to an authorized controller. We examine OpenFlow controller implementation options in greater detail in Section IV.

E. Northbound Communication: Controller-Service

External management systems or network services may wish to extract information about the underlying network or control an aspect of network behavior or policy. Additionally, controllers may find it necessary to communicate with each other for a variety of reasons. For example, an internal control application may need to reserve resources across multiple domains of control or a "primary" controller may need to share policy information with a backup, etc.

Unlike controller-switch communication, there is no currently accepted standard for northbound interactions and they are more likely to be implemented on an ad hoc basis for particular applications. We discuss this further in Section VI.

F. Standardization Efforts

Recently, several standardization organizations have been turning the spotlights towards SDN. For example, as previously mentioned, the IETF's Forwarding and Control Element Separation (ForCES) Working Group [1] has been working on standardizing mechanisms, interfaces, and protocols aiming at the centralization of network control and abstraction of network infrastructure. The Open Network Foundation (ONF) [3] has been trying to standardize the OpenFlow protocol. As the control plane abstracts network applications from underlying hardware infrastructure, they focus on standardizing the interfaces between: (1) network applications and the controller (i.e. northbound interface) and (2) the controller and the switching infrastructure (i.e., southbound interface) which defines the OpenFlow protocol itself. Some of the Study Groups (SGs) of ITU's Telecommunication Standardization Sector (ITU-T) [49] are currently working towards discussing requirements and creating recommendations for SDNs under different perspectives. For instance, the SG13 focuses on Future Networks, including cloud computing, mobile and next generation networks, and is establishing requirements for network virtualization. Other ITU-T SGs such as the SG11 for protocols and test specifications started, in early 2013, requirements and architecture discussions on SDN signaling. The Software-Defined Networking Research Group (SDNRG) at IRTF [50] is also focusing on SDN under various perspectives with the goal of identifying new approaches that can be defined and deployed, as well as identifying future research challenges. Some of their main areas of interest include solution scalability, abstractions, security and programming languages and paradigms particularly useful in the context of SDN.

These and other working groups perform important work, coordinating efforts to evolve existing standards and proposing new ones. The goal is to facilitate smooth transitions from legacy networking technology to the new protocols and architectures, such as SDN Some of these groups, such as ITU-T's SG13, advocate the establishment of a Joint Coordination Activity on SDN (JCA-SDN) for collaboration and coordination between standardizing efforts and also taking advantage of the work performed by the Open Source Software (OSS) community, such as OpenStack [51] and OpenDayLight [52] as they start developing the building blocks for SDN implementation.

IV. SDN Development Tools

SDN has been proposed to facilitate network evolution and innovation by allowing rapid deployment of new services and protocols. In this section, we provide an overview of currently available tools and environments for developing SDN-based services and protocols.

A. Emulation and Simulation Tools

Mininet [53] allows an entire OpenFlow network to be emulated on a single machine, simplifying the initial development and deployment process. New services, applications and protocols can first be developed and tested on an emulation of the anticipated deployment environment before moving to the actual hardware. By default Mininet supports OpenFlow v1.0, though it may be modified to support a software switch that implements a newer release.

The `ns-3` [54] network simulator supports OpenFlow switches within its environment, though the current version only implements OpenFlow v0.89.

B. Available Software Switch Platforms

There are currently several SDN software switches available that can be used, for example, to run an SDN testbed or when developing services over SDN. Table I presents a list of current software switch implementations with a brief description including implementation language and the OpenFlow standard version that the current implementation supports.

C. Native SDN Switches

One of the main SDN enabling technologies currently being implemented in commodity networking hardware is the OpenFlow standard. In this section we do not intend to present a detailed overview of OpenFlow enabled hardware and makers, but rather provide a list of native SDN switches currently available in the market and provide some information about them, including the version of OpenFlow they implement.

One clear evidence of industry's strong commitment to SDN is the availability of commodity network hardware that are OpenFlow enabled. Table II lists commercial switches that are currently available, their manufacturer, and the version of OpenFlow they implement.

Software Switch	Implementation	Overview	Version
Open vSwitch [55]	C/Python	Open source software switch that aims to implement a switch platform in virtualized server environments. Supports standard management interfaces and enables programmatic extension and control of the forwarding functions. Can be ported into ASIC switches.	v1.0
Pantou/OpenWRT [56]	C	Turns a commercial wireless router or Access Point into an OpenFlow-enabled switch.	v1.0
ofsoftswitch13 [57]	C/C++	OpenFlow 1.3 compatible user-space software switch implementation.	v1.3
Indigo [58]	C	Open source OpenFlow implementation that runs on physical switches and uses the hardware features of Ethernet switch ASICs to run OpenFlow.	v1.0

TABLE I
CURRENT SOFTWARE SWITCH IMPLEMENTATIONS COMPLIANT WITH THE OPENFLOW STANDARD.

Controller	Implementation	Open Source	Developer	Overview
POX [59]	Python	Yes	Nicira	General, open-source SDN controller written in Python.
NOX [17]	Python/C++	Yes	Nicira	The first OpenFlow controller written in Python and C++.
MUL [60]	C	Yes	Kulcloud	OpenFlow controller that has a C-based multi-threaded infrastructure at its core. It supports a multi-level north-bound interface (see Section III-E) for application development.
Maestro [21]	Java	Yes	Rice University	A network operating system based on Java; it provides interfaces for implementing modular network control applications and for them to access and modify network state.
Trema [61]	Ruby/C	Yes	NEC	A framework for developing OpenFlow controllers written in Ruby and C.
Beacon [22]	Java	Yes	Stanford	A cross-platform, modular, Java-based OpenFlow controller that supports event-based and threaded operations.
Jaxon [62]	Java	Yes	Independent Developers	a Java-based OpenFlow controller based on NOX.
Helios [24]	C	No	NEC	An extensible C-based OpenFlow controller that provides a programmatic shell for performing integrated experiments.
Floodlight [38]	Java	Yes	BigSwitch	A Java-based OpenFlow controller (supports v1.3), based on the Beacon implementation, that works with physical- and virtual- OpenFlow switches.
SNAC [23]	C++	No	Nicira	An OpenFlow controller based on NOX-0.4, which uses a web-based, user-friendly policy manager to manage the network, configure devices, and monitor events.
Ryu [63]	Python	Yes	NTT, OSRG group	An SDN operating system that aims to provide logically centralized control and APIs to create new network management and control applications. Ryu fully supports OpenFlow v1.0, v1.2, v1.3, and the Nicira Extensions.
NodeFlow [64]	JavaScript	Yes	Independent Developers	An OpenFlow controller written in JavaScript for Node.JS [65].
ovs-controller [55]	C	Yes	Independent Developers	A simple OpenFlow controller reference implementation with Open vSwitch for managing any number of remote switches through the OpenFlow protocol; as a result the switches function as L2 MAC-learning switches or hubs.
Flowvisor [48]	C	Yes	Stanford/Nicira	Special purpose controller implementation.
RouteFlow [66]	C++	Yes	CPqD	Special purpose controller implementation.

TABLE III
CURRENT CONTROLLER IMPLEMENTATIONS COMPLIANT WITH THE OPENFLOW STANDARD.

Maker	Switch Model	Version
Hewlett-Packard	8200zl, 6600, 6200zl, 5400zl, and 3500/3500yl	v1.0
Brocade	NetIron CES 2000 Series	v1.0
IBM	RackSwitch G8264	v1.0
NEC	PF5240 PF5820	v1.0
Pronto	3290 and 3780	v1.0
Juniper	Junos MX-Series	v1.0
Pica8	P-3290, P-3295, P-3780 and P-3920	v1.2

TABLE II
MAIN CURRENT AVAILABLE COMMODITY SWITCHES BY MAKERS, COMPLIANT WITH THE OPENFLOW STANDARD.

D. Available Controller Platforms

Table III shows a snapshot of current controller implementations. To date, all the controllers in the table support the OpenFlow protocol version 1.0, unless stated otherwise. This table also provides a brief overview of the listed controllers.

Included in Table III are also two special purpose controller implementations: Flowvisor [48], mentioned previously, and RouteFlow [66]. The former acts as a transparent proxy between OpenFlow switches and multiple OpenFlow controllers. It is able to create network slices and can delegate control of each slice to a different controller, also promoting isolation between slices. RouteFlow, on the other hand, is an open source project to provide virtualized IP routing over OpenFlow capable hardware. It is composed of an OpenFlow Controller application, an independent server, and a virtual network environment that reproduces the connectivity of a physical infrastructure and runs IP routing engines. The routing engines generate the forwarding information base (FIB) into the Linux IP tables according to the routing protocols configured (e.g., OSPF, BGP). An extension of RouteFlow is presented in [67], which discusses Routing Control Platforms (RCPs) in the context of OpenFlow/SDN. They proposed a controller-centric networking model along with a prototype implementation of an autonomous-system-wide abstract BGP routing service.

E. Code Verification and Debugging

Verification and debugging tools are vital resources for traditional software development and are no less important for SDN. Indeed, for the idea of portable network "apps" to be successful, network behavior must be thoroughly tested and verified.

NICE [68] is an automated testing tool used to help uncover bugs in OpenFlow programs through model checking and symbolic execution.

Anteater [69] takes a different approach by attempting to check network invariants that exist in the data plane, such as

connectivity or consistency. The main benefit of this approach is that it is protocol-agnostic; it will also catch errors that result from faulty switch firmware or inconsistencies with the control plane communication. VeriFlow [70] has a similar goal, but goes further by proposing a real-time verification tool that resides between the controller and the forwarding elements. This adds the potential benefit of being able to halt bad rules that will cause anomalous behavior before they reach the network.

Other efforts proposed debugging tools that provide insights gleaned from control plane traffic. OFRewind [71] allows network events (control and data) to be recorded at different granularities and later replayed to reproduce a specific scenario, granting the opportunity to localize and troubleshoot the events that caused the network anomaly. *ndb* [72] implements breakpoints and packet-backtraces for SDN. Just as with the popular software debugger *gdb*, users can pinpoint events that lead to error by pausing execution at a breakpoint, or, using a packet backtrace, show the sequence of forwarding actions seen by that packet. STS [73] is a software-defined network troubleshooting simulator. It is written in python and depends on POX. It simulates the devices in a given network allowing for testing cases and identifying the set of inputs that generates a given error.

V. SDN Applications

Software-defined networking has applications in a wide variety of networked environments. By decoupling the control– and data planes, programmable networks enable customized control, an opportunity to eliminate middleboxes, as well as simplified development and deployment of new network services and protocols. Below, we examine different environments for which SDN solutions have been proposed or implemented.

A. Enterprise Networks

Enterprises often run large networks, while also having strict security and performance requirements. Furthermore, different enterprise environments can have very different requirements, characteristics, and user population, For example, University networks can be considered a special case of enterprise networks: in such an environment, many of the connecting devices are temporary and not controlled by the University, further challenging security and resource allocation. Additionally, Universities must often provide support for research testbeds and experimental protocols.

Adequate management is critically important in Enterprise environments, and SDN can be used to programmatically enforce and adjust network policies as well as help monitor network activity and tune network performance.

Additionally, SDN can be used to simplify the network by ridding it from middleboxes and integrating their functionality within the network controller. Some notable examples of middlebox functionality that has been implemented using SDN include NAT, firewalls, load balancers [74] [75], and network access control [76]. In the case of more complex middleboxes with functionalities that cannot be directly implemented without performance degradation (e.g., deep packet inspection), SDN can be used to provide unified control and management[77].

The work presented in [78] addresses the issues related to consistent network updates. Configuration changes are a common source of instability in networks and can lead to outages, security flaws, and performance disruptions. In [78], a set of high-level abstractions are proposed that allow network administrators to update the entire network, guaranteeing that every packet traversing the network is processed by exactly one consistent global network configuration. To support these abstractions, several OpenFlow-based update mechanisms were developed.

As discussed in earlier sections, OpenFlow evolved from Ethane [20], a network architecture designed specifically to address the issues faced by enterprise networks.

B. Data Centers

Data centers have evolved at an amazing pace in recent years, constantly attempting to meet increasingly higher and rapidly changing demand. Careful traffic management and policy enforcement is critical when operating at such large scales, especially when any service disruption or additional delay may lead to massive productivity and/or profit loss. Due to the challenges of engineering networks of this scale and complexity to dynamically adapt to application requirements, it is often the case that data centers are provisioned for peak demand; as a result, they run well below capacity most of the time but are ready to rapidly service higher workloads.

An increasingly important consideration is energy consumption, which has a non-trivial cost in large-scale data centers. Heller et al. [79] indicates that much research has been focused on improved servers and cooling (70% of total energy) through better hardware or software management, but the data center's network infrastructure (which accounts for 10-20% of the total energy cost) still consumed 3 billion kWh in 2006. They proposed ElasticTree, a network-wide power manager that utilizes SDN to find the minimum-power network subset which satisfies current traffic conditions and *turns off* switches that are not needed. As a result, they show energy savings between 25-62% under varying traffic conditions. One can imagine that these savings can be further increased if used in parallel with server management and virtualization; one possibility is the Honeyguide[80] approach to energy optimization which uses virtual machine migration to increase the number of machines and switches that can be shutdown.

However, not all SDN solutions may be appropriate in high performance networks. While simplified traffic management and visibility are useful, it must be sensibly balanced with scalability and performance overhead. Curtis et al. [34] believe that OpenFlow excessively couples central control and complete visibility, when in reality only "significant" flows need to be managed; this may lead to bottlenecks as the control-data communication adds delay to flow setup while switches are overloaded with thousands of flow table entries. Though aggressive use of proactive policies and wild-card rules may

resolve that issue, it may undermine the ability of the controller to have the right granularity to effectively manage traffic and gather statistics. Their framework, DevoFlow, proposes some modest design changes to keep flows in the data plane as much as possible while maintaining enough visibility for effective flow management. This is accomplished by pushing responsibility over most flows back to the switches and adding more efficient statistics collection mechanisms, through which "significant" flows (e.g. long-lived, high-throughput) are identified and managed by the controller. In a load-balancing simulation, their solution had 10-53 times fewer flow table entries and 10-42 times fewer control messages on average over OpenFlow.

A practical example of a real application of the SDN concept and architecture in the context of data centers was presented by Google in early 2012. The company presented at the Open Network Summit [81] a large scale implementation of an SDN-based network connecting its data centers. The work in [82] presents in more detail the design, implementation, and evaluation of B4, a WAN connecting Google's datacenters world wide. This work describes one of the first and largest SDN deployments. The motivation was the need for customized routing and traffic engineering and the fact that the level of scalability, fault tolerance, cost efficiency and control required, could not be achieved by means of a traditional WAN architecture. A customized solution was proposed and an OpenFlow-based SDN architecture was built to control individual switches. After three years in production, B4 is shown to be efficient in the sense that it drives many links at near 100% utilization while splitting flows among multiple paths. Furthermore, the experience reported in the work shows that the bottleneck resulting from control-plane to data-plane communication and overhead in hardware programming are important issues to be considered in future work.

C. Infrastructure-based Wireless Access Networks

Several efforts have focused on ubiquitous connectivity in the context of infrastructure-based wireless access networks, such as cellular and WiFi.

For example, the OpenRoads project [83], [84] envisions a world in which users could freely and seamlessly move across different wireless infrastructures which may be managed by various providers. They proposed the deployment of an SDN-based wireless architecture that is backwards-compatible, yet open and sharable between different service providers. They employ a testbed using OpenFlow-enabled wireless devices such as WiFi APs and WiMAX base stations controlled by NOX– and Flowvisor controllers and show improved performance on handover events. Their vision provided inspiration for subsequent work [85] that attempts to address specific requirements and challenges in deploying a software-defined cellular network.

Odin[86] introduces programmability in enterprise wireless LAN environments. In particular, it builds an access point abstraction on the controller that separates the association state from the physical access point, enabling proactive mobility management and load balancing without changes to the client.

At the other end of the spectrum, OpenRadio [87] focuses on deploying a programmable wireless data plane that provides flexibility at the PHY and MAC layers (as opposed to layer-3 SDN) while meeting strict performance and time deadlines. The system is designed to provide a modular interface that is able to process traffic subsets using different protocols such as WiFi, WiMAX, 3GPP LTE-Advanced, etc. Based on the idea of separation of the decision and forwarding planes, an operator may express decision plane rules and corresponding actions, which are assembled from processing plane modules (e.g., FFT, Viterbi decoding, etc); the end result is a state machine that expresses a fully-functional protocol.

D. Optical Networks

Handling data traffic as flows, allows software-defined networks, and OpenFlow networks in particular, to support and integrate multiple network technologies. As a result, it is possible to provide also technology-agnostic unified control for optical transport networks and facilitating interaction between both packet and circuit-switched networks. According to the Optical Transport Working Group (OTWG) created in 2013 by the Open Network Foundation (ONF), the benefits from applying SDN and the OpenFlow standard in particular to optical transport networks include: improving optical transport network control and management flexibility, enabling deployment of third-party management and control systems, and deploying new services by leveraging virtualization and SDN [88].

There has been several attempts and proposals to control both circuit switched and packet switched networks using the OpenFlow protocol. In [89] a NetFPGA [90] platform is used in the proposal of a packet switching and circuit switched networks architectures based on Wavelength Selective Switching (WSS), using the OpenFlow protocol. Another control plane architecture based on OpenFlow for enabling SDN operations in optical networks was proposed in [91], which discusses specific requirements and describes implementation of OpenFlow protocol extensions to support optical transport networks.

A proof-of-concept demonstration of an OpenFlow-based wavelength path control in transparent optical networks is presented in [92]. In this work, virtual Ethernet interfaces (*veths*) are introduced. These veths, are mapped to physical interfaces of an optical node (e.g. photonic cross-connect - PXC), and enable an SDN controller (e.g. the NOX controller in this case) to operate the optical lightpaths (e.g., via the OpenFlow protocol). In their experimental setup, they quantitatively evaluate network performance metrics, such as the latency of lightpath setup and release, and verify the feasibility of routing and wavelength assignment, and the dynamic control of optical nodes in an OpenFlow-based network composed by four PXCs nodes in a mesh topology.

A Software Defined Optical Network (SDON) architecture is introduced in [93] and a QoS-aware unified control protocol for optical burst switching in OpenFlow-based SDON is developed. The performance of the proposed protocol was evaluated with the conventional GMPLS-based distributed protocol and

the results indicate that SDON offers an infrastructure to support unified control protocols to better optimize network performance and improve capacity.

E. Home and Small Business

Several projects have examined how SDN could be used in smaller networks, such as those found in the home or small businesses. As these environments have become increasingly complex and prevalent with the widespread availability of low-cost network devices, the need for more careful network management and tighter security has correspondingly increased. Poorly secured networks may become unwitting targets or hosts for malware, while outages due to network configuration issues may cause frustration or lost business. Unfortunately, it is not practical to have a dedicated network administrator in every home and office.

Calvert et al. [94] assert that the first step in managing home networks is to know what is actually happening; as such, they proposed instrumenting the network gateway/controller to act as a "Home Network Data Recorder" to create logs that may be utilized for troubleshooting or other purposes.

Feamster [95] proposes that such networks should operate in a "plug in and forget" fashion, namely by outsourcing management to third-party experts, and that this could be accomplished successfully through the remote control of programmable switches and the application of distributed network monitoring and inference algorithms used to detect possible security problems.

In contrast, Mortier et al. [96] believe that users desire greater understanding and control over their networks' behavior; rather than following traditional policies, a home network may be better managed by their users who better understand the dynamics and needs of their environment. Towards this goal, they created a prototype network in which SDN is used to provide users a view into how their network is being utilized while offering a single point of control.

Mehdi et al. [97] argues that an Anomaly Detection System (ADS) implemented within a programmable home network provides a more accurate identification of malicious activity as compared to one deployed at the ISP; additionally, the implementation would be able to operate at line rate with no performance penalty, while, at the same time, offloading the ISP from having to monitor these large number of networks. The ADS algorithm could operate alongside other controller services, such as a HomeOS that may react to suspicious activity and report anomalies to the ISP or local administrator.

VI. Research Challenges and Future Directions

As SDN becomes more widely adopted and protocols such as OpenFlow are further defined, new solutions are proposed and new challenges arise. In this section we discuss various challenges posed by SDN as well as future research directions, namely: (1) controller and switch design, (2) scalability and performance in SDNs, (3) controller-service interfacing, (4) virtualization and cloud service applications, (5) information centric networking, and (6) enabling heterogeneous networking with SDN.

A. Controller and Switch Design

SDN raises significant scalability, performance, robustness, and security challenges. Below we review a number of research efforts focusing on addressing these issues at the switch– and controller design level.

In DIFANE [35], flow entries are proactively pushed to switches in an attempt to reduce the number of requests to the controller. Devoflow [34] proposes to handle "short-lived" flows in switches and "long-lived" flows in the controller to mitigate flow setup delay and controller overhead. The work proposed in [28] advocates replacing counters on ASIC by a stream of rule-matching records and processing them in the CPU to allow efficient access to counters. FLARE [98] is a new network node model focusing on "deeply programmable networks" that provides programmability for the data plane, the control plane, as well as the interface between them. The work presented in [99] discusses important aspects in controller design including hierarchical control, data model, scalability, and extensibility.

As far as performance and scalability, the study presented in [100] showed that one single controller can handle up to 6 million flows per second. A more recent study [101], focusing on the Beacon controller, showed that a controller can handle 12.8 million new flows per second in a 12 cores machine, with an average latency of 24.7 us for each flow. However, for increased scalability and especially for reliability and robustness purposes, it has been recognized that the logically-centralized controller must be physically distributed. Onix [44], Kando [47], and HyperFlow [45] use this approach to achieve robust and scalable control plane. In [46], trade-offs related to control distribution, such as staleness versus optimality and application logic complexity versus robustness to inconsistency are identified and quantified. In [41], the controller placement problem is discussed in terms of the number of controllers needed and where to place them in the network. In more recent work on distributed control, the need for dynamic assignment of switches to controllers is addressed in [102], which proposes an algorithm to increase or decrease the pool of controllers based on controllers' load estimates. They also propose a mechanism to dynamically handover switches from one controller to another as needed.

In [103] an SDN variant inspired by MPLS was proposed along with the notions of *edge controllers* and *fabric controllers*: the former control ingress and egress switches and handle the host-network interface, while the latter handle fabric switches and the operator-network interface.

Although control and measurement are two important components of network management, little thought has gone into designing APIs for measurement. The work presented in [104] proposes a software-defined traffic measurement architecture, which separates the measurement data plane from the control plane.

B. Software-Defined Internetworking

The Internet has revolutionized the way we, as individuals and as a society, live, work, conduct business, socialize, get entertainment, etc. As a result, the Internet is now considered

part of our society's critical infrastructure much like the power, water, and transportation grids.

Scalability and performance requirements from increasingly complex applications have posed a variety of challenges difficult to address with the current Internet architecture. This has led the research community to examine "clean-slate" solutions [105]. As the Internet has grown beyond the point at which a "flag day", such as the one used to "upgrade" the ARPANET with the TCP/IP protocol suite, would be realistic, another considerable challenge is evolving its physical infrastructure and protocols. A notable example is the deployment of IPv6: despite over a decade in the standards track and two worldwide deployment events, IPv4 still makes up the majority of Internet traffic.

Much of the current work on SDN examines or proposes solutions within the context of a single administrative domain which matches quite well SDN's logically centralized control model. However, environments whose administration is inherently decentralized, like the Internet, call for a control plane that is logically distributed. This will allow participating autonomous systems (ASes) to be controlled independently by their own (logically centralized and possibly physically distributed) controller. To-date, a few efforts have explored the idea of a Software-Defined Internet. For example, the work in [106] proposed a software-defined Internet architecture that borrows from MPLS the distinction between network edge and core to split tasks between inter-domain and intra-domain components. As only the boundary routers and their associated controller in each domain are involved in inter-domain tasks, changes to inter-domain service models would be limited to software modifications at the inter-domain controllers rather than the entire infrastructure. Examples of how this architecture could be used to realize new Internet services such as information-centric networking, and middlebox service sharing are explored.

Another approach to inter-AS routing [107] uses NOX and OpenFlow to implement BGP-like functionality. Alternatively, an extensible session protocol [108] supports application-driven configuration of network resources across domains.

C. Controller-Service Interaction

While controller-switch ("southbound") interaction is fairly well defined in protocols such as OpenFlow and ForCES, there is no standard for interactions between controllers and network services or applications ("northbound"). One possible explanation is that the northbound interface is defined entirely in software, while controller-switch interactions must enable hardware implementation.

If we think of the controller as a "network operating system", then there should be a clearly defined interface by which applications can access the underlying hardware (switches), co-exist and interact with other applications, and utilize system services (e.g. topology discovery, forwarding), without requiring the application developer to know the implementation details of the controller. While there are several controllers that exist, their application interfaces are still in the early stages and independent from each other.

Some proposals (e.g., Procera [109], Frenetic [110], FML [111], Nettle [112]) advocate the use of a network configuration language to express policies. For example, Procera [109] builds a policy layer on top of existing controllers to interface with configuration files, GUIs, and external sensors; the proposed policy layer is responsible for converting high-level policies to flow constraints given to be used by the controller. In [113], network configuration and management mechanisms are proposed that focus on enabling changes to network condition and state, supporting network configuration and policy definitions, and providing visibility and control over tasks for network diagnostics and troubleshooting. The specification of a northbound interface via a policy layer and a high level language such as Procera is discussed.

Additionally, the northbound API should allow applications to apply different policies to the same flow (e.g. forwarding by destination and monitoring by source IP). The work in [114] proposed modularization to ensure that rules installed to perform one task do not override other rules. This was accomplished by means of an abstraction layer implemented with a language based on Frenetic.

Until a clear northbound interface standard emerges, SDN applications will continue to be developed in an "ad hoc" fashion and the concept of flexible and portable "network apps" may have to wait.

D. Virtualization and Cloud Services

The demand for virtualization and cloud services has been growing rapidly and attracting considerable interest from industry and academia. The challenges it presents include rapid provisioning, efficient resource management, and scalability which can be addressed using SDN's control model.

For example, FlowVisor [48] and AutoSlice [115] create different slices of network resources (e.g., bandwidth, topology, CPU, forwarding table), delegate them to different controllers, and enforce isolation between slices. Other SDN controllers can be used as a network backend to support virtualization in cloud operating systems, such as Floodlight for OpenStack [38] and NOX for Mirage [116]. FlowN [117] aims to offer a scalable solution for network virtualization by providing an efficient mapping between virtual and physical networks and by leveraging scalable database systems.

In [118], an algorithm for efficient migration with bandwidth guarantees using OpenFlow was proposed. LIME [119] is an SDN-based solution for live migration of Virtual Machines, which handles the network state during migration and automatically configures network devices at new locations. NetGraph [120] provides a set of APIs for customers to access its virtual network functions such as real-time monitoring and diagnostics.

On the context of cloud data centers providing Infrastructure as a Service (IaaS), [121] presents a management framework for resources in cloud data centers and addresses multiple management issues. In this paper, authors proposed a data-centric and event-driven architecture with open management interfaces, that leverages SDN techniques to integrate network resources into datacenter orchestration and service provision-

ing with the aim of improving service-level agreements and faster service delivery.

E. Information-Centric Networking

Information-Centric Networking (ICN) is a new paradigm proposed for the future architecture of the Internet, which aims to increase the efficiency of content delivery and content availability. This new concept has been popularized recently by a number of architecture proposals, such as Content-Centric Networking (CCN), also known as the Named Data Networking (NDN) project [122]. Their driving motivation is that the current Internet is information-driven, yet networking technology is still focused on the idea of location-based addressing and host-to-host communication.By proposing an architecture that addresses *named data* rather than *named hosts*, content distribution is implemented directly into the network fabric rather than relying on the complicated mapping, availability, and security mechanisms currently used to map content to a single location.

The separation between information processing and forwarding in ICN is aligned with the decoupling of the data plane and control plane in SDN. The question then becomes how to combine ICN with SDN towards "Software-Defined Information-Centric Networks". A number of projects [123], [124], [125], [126], [127], [128] have proposed using SDN concepts to implement ICNs. As OpenFlow expands to support customized header matchings, SDN can be employed as a key enabling technology for ICNs.

F. Heterogeneous Network Support

Future networks will become increasingly more heterogeneous, interconnecting users and applications over networks ranging from wired, infrastructure-based wireless (e.g., cellular–based networks, wireless mesh networks), to infrastructure-less wireless networks (e.g. mobile ad-hoc networks, vehicular networks). In the meantime, mobile traffic has been increasing exponentially over the past several years, and is expected to increase 18–fold by 2016, with more mobile-connected devices than the world's population, which is already a reality [129]. As mobile devices with multiple network interfaces become commonplace, users will demand high quality communication service regardless of location or type of network access. Self-organizing networks (e.g., wireless multi-hop ad-hoc networks) may form to extend the range of infrastructure-based networks or handle episodic connectivity disruptions. Self-organizing networks may thus enable a variety of new applications such as cloud-based services, vehicular communication, community services, healthcare delivery, emergency response, and environmental monitoring, to name a few. Efficient content delivery over wireless access networks will become essential, and self–organizing networks may become a prevalent part of the future hybrid Internet.

A major challenge facing future networks is efficient utilization of resources; this is especially the case in wireless multi-hop ad-hoc networks as the available wireless capacity is inherently limited. This is due to a number of factors including the use of shared physical medium compounded, wireless channel impairments, and the absence of managed infrastructure. Though these self–organizing networks can be used to supplement or "fill the gaps" in an overburdened infrastructure [130], their lack of dedicated resources and shifting connectivity makes capacity sharing difficult. The heterogeneous characteristics of the underlying networks (e.g., physical medium, topology, stability) and nodes (e.g., buffer size, power limitations, mobility) also add another important factor when considering routing and resource allocation.

SDN has the potential to facilitate the deployment and management of network applications and services with greater efficiency. However, SDN techniques to–date, such as OpenFlow, largely target infrastructure–based networks. They promote a centralized control mechanism that is ill–suited to the level of decentralization, disruption, and delay present in infrastructure-less environments.

While previous work has examined the use of SDN in wireless environments, the scope has primarily focused on infrastructure-based deployments (e.g., WiMAX, Wi-Fi access points). A notable example is the OpenRoads project [83], which envisioned a world in which users could freely move between wireless infrastructures while also providing support to the network provider. Other studies such as [128], [131], [132] have examined OpenFlow in wireless mesh environments.

VII. Concluding Remarks

In this paper, we provided an overview of *programmable networks* and, in this context, examined the emerging field of Software-Defined Networking (SDN). We look at the history of programmable networks, from early ideas until recent developments. In particular we described the SDN architecture in detail as well as the OpenFlow [2] standard. We presented current SDN implementations and testing platforms and examined network services and applications that have been developed based on the SDN paradigm. We concluded with a discussion of future directions enabled by SDN ranging from support for heterogeneous networks to Information Centric Networking (ICN).

Acknowledgments

The authors would like to thank the reviewers for their careful examination of the manuscript and valuable comments which helped to considerably improve the quality of the paper. This work is partly funded by the Community Associated Team between INRIA and UCSC and the French ANR under the "ANR-13-INFR-013" project, and by NSF grant CNS 1150704.

References

[1] A. Doria, J. Hadi Salim, R. Haas, H. Khosravi, W. Wang, L. Dong, R. Gopal, and J. Halpern. Forwarding and Control Element Separation (ForCES) Protocol Specification. RFC 5810 (Proposed Standard), March 2010.

[2] N. McKeown, T. Anderson, H. Balakrishnan, G. Parulkar, L. Peterson, J. Rexford, S. Shenker, and J. Turner. Openflow: enabling innovation in campus networks. *ACM SIGCOMM Computer Communication Review*, 38(2):69–74, 2008.

[3] Open networking foundation. https://www.opennetworking.org/about.

[4] Open Networking Research Center (ONRC). http://onrc.net.

[5] Thomas A. Limoncelli. Openflow: a radical new idea in networking. *Commun. ACM*, 55(8):42–47, August 2012.

[6] A.T. Campbell, I. Katzela, K. Miki, and J. Vicente. Open signaling for atm, internet and mobile networks (opensig'98). *ACM SIGCOMM Computer Communication Review*, 29(1):97–108, 1999.

[7] A. Doria, F. Hellstrand, K. Sundell, and T. Worster. General Switch Management Protocol (GSMP) V3. RFC 3292 (Proposed Standard), June 2002.

[8] D.L. Tennenhouse, J.M. Smith, W.D. Sincoskie, D.J. Wetherall, and G.J. Minden. A survey of active network research. *Communications Magazine, IEEE*, 35(1):80–86, 1997.

[9] D.L. Tennenhouse and D.J. Wetherall. Towards an active network architecture. In *DARPA Active NEtworks Conference and Exposition, 2002. Proceedings*, pages 2–15. IEEE, 2002.

[10] J.T. Moore and S.M. Nettles. Towards practical programmable packets. In *Proceedings of the 20th Conference on Computer Communications (INFOCOM)*. Citeseer, 2001.

[11] Devolved Control of ATM Networks. http://www.cl.cam.ac.uk/research/srg/netos/old-projects/dcan/#pub.

[12] J. E. Van Der Merwe and I. M. Leslie. Switchlets and dynamic virtual atm networks. In *Proc Integrated Network Management V*, pages 355–368. Chapman and Hall, 1997.

[13] J.E. Van der Merwe, S. Rooney, I. Leslie, and S. Crosby. The tempest-a practical framework for network programmability. *Network, IEEE*, 12(3):20–28, 1998.

[14] J. Rexford, A. Greenberg, G. Hjalmtysson, D.A. Maltz, A. Myers, G. Xie, J. Zhan, and H. Zhang. Network-wide decision making: Toward a wafer-thin control plane. In *Proc. HotNets*, pages 59–64. Citeseer, 2004.

[15] A. Greenberg, G. Hjalmtysson, D.A. Maltz, A. Myers, J. Rexford, G. Xie, H. Yan, J. Zhan, and H. Zhang. A clean slate 4d approach to network control and management. *ACM SIGCOMM Computer Communication Review*, 35(5):41–54, 2005.

[16] M. Caesar, D. Caldwell, N. Feamster, J. Rexford, A. Shaikh, and J. van der Merwe. Design and implementation of a routing control platform. In *Proceedings of the 2nd conference on Symposium on Networked Systems Design & Implementation-Volume 2*, pages 15–28. USENIX Association, 2005.

[17] N. Gude, T. Koponen, J. Pettit, B. Pfaff, M. Casado, N. McKeown, and S. Shenker. Nox: towards an operating system for networks. *ACM SIGCOMM Computer Communication Review*, 38(3):105–110, 2008.

[18] R. Enns. NETCONF Configuration Protocol. RFC 4741 (Proposed Standard), December 2006. Obsoleted by RFC 6241.

[19] J. D. Case, M. Fedor, M. L. Schoffstall, and J. Davin. Simple network management protocol (snmp), rfc1157, 1990.

[20] M. Casado, M.J. Freedman, J. Pettit, J. Luo, N. McKeown, and S. Shenker. Ethane: Taking control of the enterprise. *ACM SIGCOMM Computer Communication Review*, 37(4):1–12, 2007.

[21] Z. Cai, AL Cox, and TSE Ng. Maestro: A system for scalable openflow control. Technical Report TR10-08, Rice University, December 2010.

[22] Beacon. https://openflow.stanford.edu/display/Beacon/Home.

[23] Simple Network Access Control (SNAC). http://www.openflow.org/wp/snac/.

[24] Helios by nec. http://www.nec.com/.

[25] Andrea Passarella. Review: A survey on content-centric technologies for the current internet: Cdn and p2p solutions. *Comput. Commun.*, 35(1):1–32, January 2012.

[26] Zhiliang Wang, Tina Tsou, Jing Huang, Xingang Shi, and Xia Yin. Analysis of Comparisons between OpenFlow and ForCES, March 2012.

[27] Guohan Lu, Rui Miao, Yongqiang Xiong, and Chuanxiong Guo. Using cpu as a traffic co-processing unit in commodity switches. In *Proceedings of the first workshop on Hot topics in software defined networks*, HotSDN '12, pages 31–36, New York, NY, USA, 2012. ACM.

[28] Jeffrey C. Mogul and Paul Congdon. Hey, you darned counters!: get off my asic! In *Proceedings of the first workshop on Hot topics in software defined networks*, HotSDN '12, pages 25–30, New York, NY, USA, 2012. ACM.

[29] Yan Luo, Pablo Cascon, Eric Murray, and Julio Ortega. Accelerating openflow switching with network processors. In *Proceedings of the 5th ACM/IEEE Symposium on Architectures for Networking and Communications Systems*, ANCS '09, pages 70–71, New York, NY, USA, 2009. ACM.

[30] Voravit Tanyingyong, Markus Hidell, and Peter Sjödin. Improving pc-based openflow switching performance. In *Proceedings of the 6th ACM/IEEE Symposium on Architectures for Networking and Communications Systems*, ANCS '10, pages 13:1–13:2, New York, NY, USA, 2010. ACM.

[31] A. Bianco, R. Birke, L. Giraudo, and M. Palacin. Openflow switching: Data plane performance. In *Communications (ICC), 2010 IEEE International Conference on*, pages 1–5, May.

[32] M. Jarschel, S. Oechsner, D. Schlosser, R. Pries, S. Goll, and P. Tran-Gia. Modeling and performance evaluation of an openflow architecture. In *Teletraffic Congress (ITC), 2011 23rd International*, pages 1–7, Sept.

[33] Brent Stephens, Alan Cox, Wes Felter, Colin Dixon, and John Carter. Past: scalable ethernet for data centers. In *Proceedings of the 8th international conference on Emerging networking experiments and technologies*, CoNEXT '12, pages 49–60, New York, NY, USA, 2012. ACM.

[34] Andrew R. Curtis, Jeffrey C. Mogul, Jean Tourrilhes, Praveen Yalagandula, Puneet Sharma, and Sujata Banerjee. Devoflow: scaling flow management for high-performance networks. *SIGCOMM Comput. Commun. Rev.*, 41(4):254–265, August 2011.

[35] M. Yu, J. Rexford, M.J. Freedman, and J. Wang. Scalable flow-based networking with difane. In *Proceedings of the ACM SIGCOMM 2010 conference on SIGCOMM*, pages 351–362. ACM, 2010.

[36] Yossi Kanizo, David Hay, and Isaac Keslassy. Palette: Distributing tables in software-defined networks. In *INFOCOM*, pages 545–549, 2013.

[37] Nanxi Kang, Zhenming Liu, Jennifer Rexford, and David Walker. Optimizing the one big switch abstraction in software-defined networks.

[38] Floodlight, an open sdn controller. http://floodlight.openflowhub.org/.

[39] A. Tootoonchian, S. Gorbunov, Y. Ganjali, M. Casado, and R. Sherwood. On controller performance in software-defined networks. In *USENIX Workshop on Hot Topics in Management of Internet, Cloud, and Enterprise Networks and Services (Hot-ICE)*, 2012.

[40] Andreas Voellmy and Junchang Wang. Scalable software defined network controllers. In *Proceedings of the ACM SIGCOMM 2012 conference on Applications, technologies, architectures, and protocols for computer communication*, SIGCOMM '12, pages 289–290, New York, NY, USA, 2012. ACM.

[41] Brandon Heller, Rob Sherwood, and Nick McKeown. The controller placement problem. In *Proceedings of the first workshop on Hot topics in software defined networks*, HotSDN '12, pages 7–12, New York, NY, USA, 2012. ACM.

[42] K. Phemius and M. Bouet. Openflow: Why latency does matter. In *Integrated Network Management (IM 2013), 2013 IFIP/IEEE International Symposium on*, pages 680–683, 2013.

[43] S.H. Yeganeh, A. Tootoonchian, and Y. Ganjali. On scalability of software-defined networking. *Communications Magazine, IEEE*, 51(2):136–141, February.

[44] T. Koponen, M. Casado, N. Gude, J. Stribling, L. Poutievski, M. Zhu, R. Ramanathan, Y. Iwata, H. Inoue, T. Hama, et al. Onix: A distributed control platform for large-scale production networks. *OSDI, Oct*, 2010.

[45] A. Tootoonchian and Y. Ganjali. Hyperflow: A distributed control plane for openflow. In *Proceedings of the 2010 internet network management conference on Research on enterprise networking*, pages 3–3. USENIX Association, 2010.

[46] Dan Levin, Andreas Wundsam, Brandon Heller, Nikhil Handigol, and Anja Feldmann. Logically centralized?: state distribution trade-offs in software defined networks. In *Proceedings of the first workshop on Hot topics in software defined networks*, HotSDN '12, pages 1–6, New York, NY, USA, 2012. ACM.

[47] Soheil Hassas Yeganeh and Yashar Ganjali. Kandoo: a framework for efficient and scalable offloading of control applications. In *Proceedings of the first workshop on Hot topics in software defined networks*, HotSDN '12, pages 19–24, New York, NY, USA, 2012. ACM.

[48] R. Sherwood, M. Chan, A. Covington, G. Gibb, M. Flajslik, N. Handigol, T.Y. Huang, P. Kazemian, M. Kobayashi, J. Naous, et al. Carving research slices out of your production networks with openflow. *ACM SIGCOMM Computer Communication Review*, 40(1):129–130, 2010.

[49] http://www.itu.int/en/ITU T/sdn/Pages/default.aspx. Itu telecommunication standardization sector's sdn portal, 2013.

[50] http://irtf.org/sdnrg. Software-defined networking research group - sdnrg at irtf, 2013.

[51] http://www.openstack.org/. Openstack, 2013.

[52] http://www.opendaylight.org/. Opendaylight, 2013.

[53] Bob Lantz, Brandon Heller, and Nick McKeown. A network in a laptop: rapid prototyping for software-defined networks. In *Proceedings of the Ninth ACM SIGCOMM Workshop on Hot Topics in Networks*, 2010.

[54] T.R. Henderson, M. Lacage, G.F. Riley, C. Dowell, and JB Kopena. Network simulations with the ns-3 simulator. *SIGCOMM demonstration*, 2008.

[55] Open vswitch and ovs-controller. http://openvswitch.org/.

[56] Pantou: Openflow 1.0 for openwrt. http://www.openflow.org/wk/index.php/ OpenFlow_1.0_for_OpenWRT.

[57] ofsoftswitch13 - cpqd. https://github.com/CPqD/ofsoftswitch13.

[58] Indigo: Open source openflow switches. http://www.openflowhub.org/display/Indigo/.

[59] Pox. http://www.noxrepo.org/pox/about-pox/.

[60] Mul. http://sourceforge.net/p/mul/wiki/Home/.

[61] Trema openflow controller framework. https://github.com/trema/trema.

[62] Jaxon:java-based openflow controller. http://jaxon.onuos.org/.

[63] Ryu. http://osrg.github.com/ryu/.

[64] The nodeflow openflow controller. http://garyberger.net/?p=537.

[65] Node.js. http://nodejs.org/.

[66] Marcelo R. Nascimento, Christian E. Rothenberg, Marcos R. Salvador, Carlos N. A. Corrêa, Sidney C. de Lucena, and Maurício F. Magalhães. Virtual routers as a service: the routeflow approach leveraging software-defined networks. In *Proceedings of the 6th International Conference on Future Internet Technologies*, CFI '11, pages 34–37, New York, NY, USA, 2011. ACM.

[67] Christian Esteve Rothenberg, Marcelo Ribeiro Nascimento, Marcos Rogerio Salvador, Carlos Nilton Araujo Corrêa, Sidney Cunha de Lucena, and Robert Raszuk. Revisiting routing control platforms with the eyes and muscles of software-defined networking. In *Proceedings of the first workshop on Hot topics in software defined networks*, HotSDN '12, pages 13–18, New York, NY, USA, 2012. ACM.

[68] M. Canini, D. Venzano, P. Peresini, D. Kostic, and J. Rexford. A nice way to test openflow applications. *NSDI, Apr*, 2012.

[69] Haohui Mai, Ahmed Khurshid, Rachit Agarwal, Matthew Caesar, P. Brighten Godfrey, and Samuel Talmadge King. Debugging the data plane with anteater. In *Proceedings of the ACM SIGCOMM 2011 conference*, SIGCOMM '11, pages 290–301, New York, NY, USA, 2011. ACM.

[70] Ahmed Khurshid, Wenxuan Zhou, Matthew Caesar, and P. Brighten Godfrey. Veriflow: verifying network-wide invariants in real time. In *Proceedings of the first workshop on Hot topics in software defined networks*, HotSDN '12, pages 49–54, New York, NY, USA, 2012. ACM.

[71] Andreas Wundsam, Dan Levin, Srini Seetharaman, and Anja Feldmann. Ofrewind: enabling record and replay troubleshooting for networks. In *Proceedings of the 2011 USENIX conference on USENIX annual technical conference*, USENIXATC'11, pages 29–29, Berkeley, CA, USA, 2011. USENIX Association.

[72] Nikhil Handigol, Brandon Heller, Vimalkumar Jeyakumar, David Maziéres, and Nick McKeown. Where is the debugger for my software-defined network? In *Proceedings of the first workshop on Hot topics in software defined networks*, HotSDN '12, pages 55–60, New York, NY, USA, 2012. ACM.

[73] Sdn troubleshooting simulator. http://ucb-sts.github.com/sts/.

[74] N. Handigol, S. Seetharaman, M. Flajslik, N. McKeown, and R. Johari. Plug-n-serve: Load-balancing web traffic using openflow. *ACM SIGCOMM Demo*, 2009.

[75] R. Wang, D. Butnariu, and J. Rexford. Openflow-based server load balancing gone wild. In *Workshop of HotICE*, volume 11, 2011.

[76] A.K. Nayak, A. Reimers, N. Feamster, and R. Clark. Resonance: Dynamic access control for enterprise networks. In *Proceedings of the 1st ACM workshop on Research on enterprise networking*, pages 11–18. ACM, 2009.

[77] A. Gember, P. Prabhu, Z. Ghadiyali, and A. Akella. Toward software-defined middlebox networking. 2012.

[78] Mark Reitblatt, Nate Foster, Jennifer Rexford, Cole Schlesinger, and David Walker. Abstractions for network update. In *Proceedings of the ACM SIGCOMM 2012 conference on Applications, technologies, architectures, and protocols for computer communication*, SIGCOMM '12, pages 323–334, New York, NY, USA, 2012. ACM.

[79] B. Heller, S. Seetharaman, P. Mahadevan, Y. Yiakoumis, P. Sharma, S. Banerjee, and N. McKeown. Elastictree: Saving energy in data center networks. In *Proceedings of the 7th USENIX conference on Networked systems design and implementation*, pages 17–17. USENIX Association, 2010.

[80] H. Shirayanagi, H. Yamada, and K. Kono. Honeyguide: A vm migration-aware network topology for saving energy consumption in data center networks. In *Computers and Communications (ISCC), 2012 IEEE Symposium on*, pages 000460–000467. IEEE, 2012.

[81] Inter-datacenter wan with centralized te using sdn and openflow. In *Open Networking Summit*, April 2012.

[82] Sushant Jain, Alok Kumar, Subhasree Mandal, Joon Ong, Leon Poutievski, Arjun Singh, Subbaiah Venkata, Jim Wanderer, Junlan Zhou, Min Zhu, et al. B4: Experience with a globally-deployed software defined wan. In *Proceedings of the ACM SIGCOMM 2013 conference on SIGCOMM*, pages 3–14. ACM, 2013.

[83] K.K. Yap, R. Sherwood, M. Kobayashi, T.Y. Huang, M. Chan, N. Handigol, N. McKeown, and G. Parulkar. Blueprint for introducing innovation into wireless mobile networks. In *Proceedings of the second ACM SIGCOMM workshop on Virtualized infrastructure systems and architectures*, pages 25–32. ACM, 2010.

[84] K.K. Yap, M. Kobayashi, R. Sherwood, T.Y. Huang, M. Chan, N. Handigol, and N. McKeown. Openroads: Empowering research in mobile networks. *ACM SIGCOMM Computer Communication Review*, 40(1):125–126, 2010.

[85] L.E. Li, Z.M. Mao, and J. Rexford. Toward software-defined cellular networks. In *Software Defined Networking (EWSDN), 2012 European Workshop on*, pages 7–12, 2012.

[86] Lalith Suresh, Julius Schulz-Zander, Ruben Merz, Anja Feldmann, and Teresa Vazao. Towards programmable enterprise wlans with odin. In *Proceedings of the first workshop on Hot topics in software defined networks*, HotSDN '12, pages 115–120, New York, NY, USA, 2012. ACM.

[87] M. Bansal, J. Mehlman, S. Katti, and P. Levis. Openradio: a programmable wireless dataplane. In *Proceedings of the first workshop on Hot topics in software defined networks*, pages 109–114. ACM, 2012.

[88] Optical transport working group otwg. In *Open Networking Foundation ONF*, 2013.

[89] V. Gudla, S. Das, A. Shastri, G. Parulkar, N. McKeown, L. Kazovsky, and S. Yamashita. Experimental demonstration of openflow control of packet and circuit switches. In *Optical Fiber Communication (OFC), collocated National Fiber Optic Engineers Conference, 2010 Conference on (OFC/NFOEC)*, pages 1–3, 2010.

[90] Netfpga platform. http://netfpga.org.

[91] Dimitra E. Simeonidou, Reza Nejabati, and Mayur Channegowda. Software defined optical networks technology and infrastructure: Enabling software-defined optical network operations. In *Optical Fiber Communication Conference/National Fiber Optic Engineers Conference 2013*, page OTh1H.3. Optical Society of America, 2013.

[92] Lei Liu, T. Tsuritani, I. Morita, Hongxiang Guo, and Jian Wu. Openflow-based wavelength path control in transparent optical networks: A proof-of-concept demonstration. In *Optical Communication (ECOC), 2011 37th European Conference and Exhibition on*, pages 1–3, 2011.

[93] A.N. Patel, P.N. Ji, and Ting Wang. Qos-aware optical burst switching in openflow based software-defined optical networks. In *Optical Network Design and Modeling (ONDM), 2013 17th International Conference on*, pages 275–280, 2013.

[94] K.L. Calvert, W.K. Edwards, N. Feamster, R.E. Grinter, Y. Deng, and X. Zhou. Instrumenting home networks. *ACM SIGCOMM Computer Communication Review*, 41(1):84–89, 2011.

[95] N. Feamster. Outsourcing home network security. In *Proceedings of the 2010 ACM SIGCOMM workshop on Home networks*, pages 37–42. ACM, 2010.

[96] R. Mortier, T. Rodden, T. Lodge, D. McAuley, C. Rotsos, AW Moore, A. Koliousis, and J. Sventek. Control and understanding: Owning your home network. In *Communication Systems and Networks (COMSNETS), 2012 Fourth International Conference on*, pages 1–10. IEEE, 2012.

[97] S. Mehdi, J. Khalid, and S. Khayam. Revisiting traffic anomaly detection using software defined networking. In *Recent Advances in Intrusion Detection*, pages 161–180. Springer, 2011.

[98] Akihiro Nakao. Flare : Open deeply programmable network node architecture. http://netseminar.stanford.edu/10_18_12.html.

[99] R. Bifulco, R. Canonico, M. Brunner, P. Hasselmeyer, and F. Mir. A practical experience in designing an openflow controller. In *Software Defined Networking (EWSDN), 2012 European Workshop on*, pages 61–66, Oct.

[100] Controller performance comparisons. http://www.openflow.org/wk/index.php/Controller_Performance_Comparisons.

[101] David Erickson. The beacon openflow controller, 2012.

[102] Advait Dixit, Fang Hao, Sarit Mukherjee, T.V. Lakshman, and Ramana Kompella. Towards an elastic distributed sdn controller. In *Proceedings of the second ACM SIGCOMM workshop on Hot topics in software*

defined networking, HotSDN '13, pages 7–12, New York, NY, USA, 2013. ACM.

[103] Martin Casado, Teemu Koponen, Scott Shenker, and Amin Tootoonchian. Fabric: a retrospective on evolving sdn. In *Proceedings of the first workshop on Hot topics in software defined networks*, HotSDN '12, pages 85–90, New York, NY, USA, 2012. ACM.

[104] Minlan Yu, Lavanya Jose, and Rui Miao. Software defined traffic measurement with opensketch. In *Proceedings 10th USENIX Symposium on Networked Systems Design and Implementation, NSDI'13*, 2013.

[105] Anja Feldmann. Internet clean-slate design: what and why? *SIGCOMM Comput. Commun. Rev.*, 37(3):59–64, July 2007.

[106] Barath Raghavan, Martín Casado, Teemu Koponen, Sylvia Ratnasamy, Ali Ghodsi, and Scott Shenker. Software-defined internet architecture: decoupling architecture from infrastructure. In *Proceedings of the 11th ACM Workshop on Hot Topics in Networks*, HotNets-XI, pages 43–48, New York, NY, USA, 2012. ACM.

[107] R. Bennesby, P. Fonseca, E. Mota, and A. Passito. An inter-as routing component for software-defined networks. In *Network Operations and Management Symposium (NOMS), 2012 IEEE*, pages 138–145, 2012.

[108] E. Kissel, G. Fernandes, M. Jaffee, M. Swany, and M. Zhang. Driving software defined networks with xsp. In *SDN12: Workshop on Software Defined Networks*, 2012.

[109] Andreas Voellmy, Hyojoon Kim, and Nick Feamster. Procera: a language for high-level reactive network control. In *Proceedings of the first workshop on Hot topics in software defined networks*, HotSDN '12, pages 43–48, New York, NY, USA, 2012. ACM.

[110] Nate Foster, Rob Harrison, Michael J. Freedman, Christopher Monsanto, Jennifer Rexford, Alec Story, and David Walker. Frenetic: a network programming language. In *Proceedings of the 16th ACM SIGPLAN international conference on Functional programming*, ICFP '11, pages 279–291, New York, NY, USA, 2011. ACM.

[111] Timothy L. Hinrichs, Natasha S. Gude, Martin Casado, John C. Mitchell, and Scott Shenker. Practical declarative network management. In *Proceedings of the 1st ACM workshop on Research on enterprise networking*, WREN '09, pages 1–10, New York, NY, USA, 2009. ACM.

[112] Andreas Voellmy and Paul Hudak. Nettle: taking the sting out of programming network routers. In *Proceedings of the 13th international conference on Practical aspects of declarative languages*, PADL'11, pages 235–249, Berlin, Heidelberg, 2011. Springer-Verlag.

[113] Hyojoon Kim and N. Feamster. Improving network management with software defined networking. *Communications Magazine, IEEE*, 51(2):114–119, February.

[114] Christopher Monsanto, Joshua Reich, Nate Foster, Jennifer Rexford, and David Walker. Composing software-defined networks. In *Proceedings 10th USENIX Symposium on Networked Systems Design and Implementation, NSDI'13*, 2013.

[115] Zdravko Bozakov and Panagiotis Papadimitriou. Autoslice: automated and scalable slicing for software-defined networks. In *Proceedings of the 2012 ACM conference on CoNEXT student workshop*, CoNEXT Student '12, pages 3–4, New York, NY, USA, 2012. ACM.

[116] Connected cloud control: Openflow in mirage. http://www.openmirage.org/blog/announcing-mirage-openflow.

[117] D. Drutskoy, E. Keller, and J. Rexford. Scalable network virtualization in software-defined networks. *Internet Computing, IEEE*, PP(99):1–1.

[118] Soudeh Ghorbani and Matthew Caesar. Walk the line: consistent network updates with bandwidth guarantees. In *Proceedings of the first workshop on Hot topics in software defined networks*, HotSDN '12, pages 67–72, New York, NY, USA, 2012. ACM.

[119] Eric Keller, Soudeh Ghorbani, Matt Caesar, and Jennifer Rexford. Live migration of an entire network (and its hosts). In *Proceedings of the 11th ACM Workshop on Hot Topics in Networks*, HotNets-XI, pages 109–114, New York, NY, USA, 2012. ACM.

[120] Ramya Raghavendra, Jorge Lobo, and Kang-Won Lee. Dynamic graph query primitives for sdn-based cloudnetwork management. In *Proceedings of the first workshop on Hot topics in software defined networks*, HotSDN '12, pages 97–102, New York, NY, USA, 2012. ACM.

[121] Xiang Wang, Zhi Liu, Yaxuan Qi, and Jun Li. Livecloud: A lucid orchestrator for cloud datacenters. In *Cloud Computing Technology and Science (CloudCom), 2012 IEEE 4th International Conference on*, pages 341–348, 2012.

[122] V. Jacobson, D.K. Smetters, J.D. Thornton, M.F. Plass, N.H. Briggs, and R.L. Braynard. Networking named content. In *Proceedings of the 5th international conference on Emerging networking experiments and technologies*, pages 1–12. ACM, 2009.

[123] Xuan-Nam Nguyen, Damien Saucez, and Thierry Turletti. Efficient caching in Content-Centric Networks using OpenFlow. In *2013 IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS)*, pages 67–68. IEEE, April 2013.

[124] L. Veltri, G. Morabito, S. Salsano, N. Blefari-Melazzi, and A. Detti. Supporting information-centric functionality in software defined networks. *IEEE ICC Workshop on Software Defined Networks*, June 2012.

[125] N. Blefari-Melazzi, A. Detti, G. Mazza, G. Morabito, S. Salsano, and L. Veltri. An openflow-based testbed for information centric networking. *Future Network & Mobile Summit*, pages 4–6, 2012.

[126] Junho Suh, Hyogi Jung, Taekyoung Kwon, and Yanghee Choi. C-flow: Content-oriented networking over openflow. In *Open Networking Summit*, April 2012.

[127] D. Syrivelis, G. Parisis, D. Trossen, P. Flegkas, V. Sourlas, T. Korakis, and L. Tassiulas. Pursuing a software defined information-centric network. In *Software Defined Networking (EWSDN), 2012 European Workshop on*, pages 103–108, Oct.

[128] X.N. Nguyen. Software defined networking in wireless mesh network. Msc. thesis, INRIA, UNSA, August 2012.

[129] Cisco visual networking index: Global mobile data traffic forecast update, 2011–2016. Technical report, Cisco, February 2012.

[130] B. Rais, M. Mendonca, T. Turletti, and K. Obraczka. Towards truly heterogeneous internets: Bridging infrastructure-based and infrastructureless networks. In *Communication Systems and Networks (COMSNETS), 2011 Third International Conference on*, pages 1–10. IEEE, 2011.

[131] A. Coyle and H. Nguyen. A frequency control algorithm for a mobile adhoc network. In *Military Communications and Information Systems Conference (MilCIS)*, Canberra, Australia, November 2010.

[132] P. Dely, A. Kassler, and N. Bayer. Openflow for wireless mesh networks. In *Proceedings of 20th International Conference on Computer Communications and Networks (ICCCN)*, pages 1–6. IEEE, 2011.

A Comprehensive and Effective Mechanism for DDoS Detection in SDN

Mauro Conti, Ankit Gangwal
Department of Mathematics
University of Padua, Italy
Email: {conti, ankit.gangwal}@math.unipd.it

Manoj Singh Gaur
Department of Computer Science & Engineering
Malaviya National Institute of Technology, India
Email: gaurms@mnit.ac.in

***Abstract*—DDoS attack is one of the major concerns for network and cloud service providers, due to its substantial impact on revenue/cost and especially on their reputation. Also, network administrators are looking for solutions to manage voluminous data traffic. SDN is an emerging networking paradigm that provides a flexible network management. Hence, SDN is being widely adopted for wired, wireless, and mobile networks. Apart from a single point of failure (the controller), an attacker can target SDN at various levels by DDoS attacks.**

Existing solutions either focus on a particular attack type or require cumbersome alterations in SDN infrastructure. In this paper, we propose a comprehensive, yet effective and lightweight approach to detect various fundamentally different DDoS attacks in SDN. Our approach relies on sequential analysis. We employ a non-parametric change point detection technique called Cumulative Sum (CuSum). Our framework also includes an adaptive threshold scheme that adapts with the changing traffic pattern. Additionally, our framework can be tuned to suffice critical security requirements such as high detection rate and low false alarm rate. We evaluated the effectiveness of our solution using CAIDA Internet traces as well as DARPA intrusion detection evaluation dataset. Our results confirm the effectiveness of our mechanism. In particular, average false alarm rate in our experiments was under 11.64%. On average, our method is able to detect DDoS attacks within 4.15 seconds.

***Index Terms*—Distributed Denial of Service (DDoS), Software Defined Networks (SDN), Network Security**

I. Introduction

In today's Internet, end hosts have almost no control over the quantity or type of traffic forwarded to them. Typically, Internet Service Providers (ISPs) are responsible for regulating the traffic in network through traffic engineering. An ISP must take into account several important factors when performing the traffic engineering tasks, including highly unpredictable and dynamic nature of the Internet traffic, its resources and their capacity, its Service Level Agreements (SLA) with its customers, its policies and agreements with other ISPs. Moreover, an ISP would never want to upset its consumers by dropping their traffic despite the fact that a substantial amount of the traffic may be potentially unwanted by a consumer.

Such quandary of an ISP leads to several significant problems, especially to the catastrophic Distributed Denial of Service (DDoS) attacks that not only affect end hosts but sometimes also affect the ISP itself. Since their inception, DDoS attacks are still one of the biggest threats to the Internet's stability and security. With the increase in capacity of the Internet, the scale of DDoS attacks has also enlarged. As an illustrative example, a hosting company OVH was the victim of a 1 Tbps DDoS attack that hit its servers, which was one of the largest attacks ever seen on the Internet till late 2016[1]. Such attacks have been partially facilitated by user-friendly tools, e.g., Low Orbit Ion Cannon (LOIC) [1], hping3 [2], Stacheldraht [3]. Such tools enable even novice users to launch massive attacks against several targets simultaneously. Furthermore, employment of techniques such as IP spoofing makes it even harder to track and identify the attacker.

Recent developments and innovations in networking assure to change how the future Internet will work. In particular, networking infrastructure along with data plane and control plane witnessed promising improvements. The data plane is typically responsible for packet forwarding while the control plane takes all the routing decisions. Figure 1 depicts a simplified architecture of the conventional network, where the control plane and the data plane are embedded into the same device. In general, the forwarding rules are hardwired into a traditional device, and hence, traditional networks lack flexibility. Traditional networks are largely un-programmable by their owners while the innovations are limited to vendors or their partners. Besides, the devices have longer hardware fabrication cycles, and network management remains complex [4].

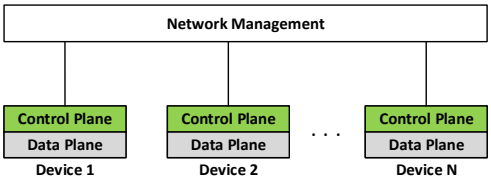


Figure 1: A simplified architecture of the traditional network

Software Defined Network (SDN) is a recently proposed networking architecture that completely separates the control plane from the data plane. All the networking elements in the data plane act as a simple packet forwarding device while all the routing decisions are made by a logically centralized system, i.e., the controller in the control plane [5]. Figure 2 presents a simplified architecture of SDN. The programmability of the control plane enables us to devise resilient routing logics, which can accommodate diverse requirements of various network applications.

[1] http://securityaffairs.co/wordpress/51640/cyber-crime/tbps-ddos-attack.html

978-1-5386-3839-2/17/$31.00 ©2017 IEEE

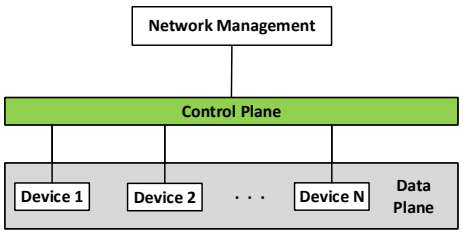


Figure 2: A simplified architecture of SDN

The potential of SDN has gained enormous attention from the research community as well as the industries. As SDN is still an emerging networking concept, it has various concerns related to performance, reliability, management, and security [6]. The performance issues include utilization of switch resources (e.g., bandwidth, flow-table size), efficient handling of new flows, and lookup procedures. The management issues mainly focus on careful management of the control plane and its resources as the control plane is responsible for handling the entire network. The reliability issues comprise link failure, controller failure, and asynchronous update of switches. However, the main objective of our work is to analyze the security issues in SDN; DDoS attacks in particular, and propose an effective DDoS detection framework.

Contributions: In this paper, we introduce a comprehensive, effective, and lightweight approach for detection of DDoS attacks in SDN. The major contributions of our work are listed as follows:

1) We thoroughly analyze how DDoS attacks impact on the different levels of SDN architecture.
2) We propose a simple, yet efficient solution for the detection of DDoS attacks in SDN.
3) We emulated our solution and evaluated its effectiveness using Internet traces provided by CAIDA [7] as well as DARPA intrusion detection evaluation dataset [8].

Organization: The remainder of this paper is organized as follows. Section II thoroughly explains various important aspects of DDoS attacks. Here, we discuss the typical classes of DDoS attacks, DDoS detection techniques, and how DDoS attack can be launched in SDN environment along with its repercussions on SDN. Here, we also present an overview of related works regarding DDoS attacks in SDN. In Section III, we elaborate the threat model. In Section IV, we explain our detection approach with its implementation details. Section V covers the details of our experimental setup while we discuss and analyze our results in Section VI. Finally, Section VII concludes the paper and explores the future directions.

II. Preliminaries and Related Work

DDoS is a cyber-attack where the attacker uses more than one machine (usually compromised) to make network services or resources unavailable to its intended users. A DDoS attack is typically launched in four phases namely: recruit; exploit; infect; and use [9]. In the recruit phase, the attacker scans remote machines for security holes that will help to barge in. In the exploit phase, the discovered loopholes are exploited to break into vulnerable machines. Such machines are then infected with the attack code in the next phase. And finally, the compromised machines are used to launch attack payload. Based on the targeted protocol level, DDoS attacks can be broadly classified into two categories [10, 11]:

1) Transport/network-level attacks: Such attacks use ICMP, TCP, UDP, and DNS protocol packets to launch DDoS. The aim here is to disrupt legitimate users' connectivity by exhausting the bandwidth of victim's network. The attacker can either use direct flooding or reflection-based flooding. In reflection-based flooding, the attacker sends forged requests to a large number of hosts, which in turn reflects massive replies towards the target.
2) Application-level attacks: Such attacks focus on exhausting server's resources such as CPU, memory to interrupt legitimate users' services. In general, the attacker employs request flooding attacks and slow request/response attacks.

The remainder of this section explains the typical DDoS detection techniques, DDoS attack scenario in SDN environment along with its impact on SDN architecture, and state-of-the-art regarding DDoS detection in SDN.

A. DDoS Detection Techniques

DDoS detection techniques can be widely classified into two categories: signature-based detection, and anomaly-based detection.

Signature-based Detection: A signature is a pattern of string that corresponds to a known threat or attack. The signature-based detection relies on string comparison techniques. Such methods compare and search for the current unit of activity such as a packet entry or a log entry in a signature repository. Signature-based detection approaches are efficient to identify only recognized attacks without any complex procedures. On another side, such methods are not capable of identifying variants of known attacks as well as new attacks. Other challenges include keeping an up-to-date signatures repository and proliferating size of the signature database.

Anomaly-based Detection: As opposed to signature-based detection, anomaly-based detection does not require predefined signatures or patterns to classify an activity. Such methods employ statistical features of network traffic to identify attacks. As a representative example, incoming packet rate can serve as a feature. The current network behavior is compared with the observed network behavior, and an alarm is raised when there is a significant variation from the normal course of operation. Such methods are capable of identifying variants of known attacks as well as unknown attacks. Nevertheless, they may create various spurious alarms [12].

B. DDoS in SDN

A DDoS attack in SDN environment can affect the network at various levels. It is worth mentioning that some attack vectors are common to the traditional network while some

threats are unique to SDN. For instance, instead of high-volume traffic flows, an attacker might use low-volume traffic flows to generate a huge number of *Packet_In* messages, which in turn overloads the ingress switch as well as the controller. Figure 3 shows that the attacker can strike on the switch, the controller, and the secure link between the control plane and the data plane.

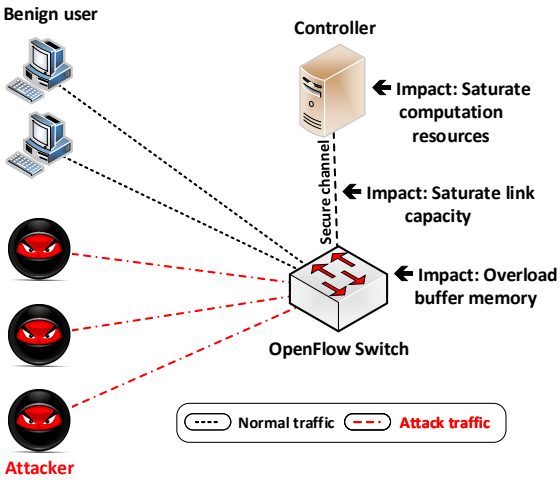


Figure 3: Impact of DDoS attack in SDN

Impact on switches: The objective of the attacker is to drop or at least delay legitimate users' packets to deteriorate the performance of the network and ultimately, ruin users' experience. Normally, when a new flow of packets reaches a switch, a "table-miss" event is raised. Consequently, a *Packet_In* message is forwarded to the controller to obtain an action. For all subsequent packets in the same flow, the switch applies the obtained action without any controller intervention. While the switch awaits for a response from the controller, it buffers the incoming packets in its buffer memory. In case the buffer gets full, the subsequent packets are dropped due to insufficient space in the buffer. In general, the attacker employs IP spoofing to generate a huge number of flows with random headers. Hence, botnets overflow the switching device with supposedly fresh flows of packets. Consequently, packets from legitimate hosts are dropped or at least delayed.

Impact on controller: The logically centralized controller is the single point of failure, and its breakdown can disrupt the entire network. The controller computes an action set for each *Packet_In* request coming from a switch. Calculating the action sets consumes controller's resources such as CPU, memory, I/O bandwidth. The controller can handle a large, but still a limited number of request at a given instance of time. Hence, with a huge number of requests generated by a DDoS attack will saturate the resources of the controller. Eventually, genuine requests are delayed or even dropped.

An attacker may also seek to interfere with controller's functioning through attempts such as buffer overflow, which may lead to the installation of erroneous forwarding rules in the data plane.

Impact on secure channel between control and data plane: The control and the data plane communicate over a secure channel. The secure channel carries periodic as well as sporadic messages. Even minute congestions in the channel may lead to inevitable delays in network functioning. Especially, delaying *Packet_In* messages notably degrades network performance. An enormous number of flows generated by a DDoS attack can saturate the secure channel, eventually ceasing the operation of the entire network [13].

C. DDoS Detection in SDN

Several researchers argued that a sensible solution for DDoS attacks is to enhance the security of every Internet host and prevent the damages from such attacks [14], while others suspect the widespread acceptance of such mechanisms [15]. Other researchers insisted that DDoS attacks are not even a security issue, but they are scalability questions [16]. In support of their claim they say that the attackers will continuously attempt to make their requests indistinct from the benign traffic; hence, they may defeat the detection mechanisms. In this scenario, the final solution is to increase resources in terms of quantity, which is expensive.

Mousavi et al. [17] proposed an entropy-based mechanism to detect a DDoS attack in SDN. Here, in case of an attack, the entropy decreases on the basis of the randomness of incoming packets' destination address. However, the approach assumes that the number of hosts in the network will always remain static and destination IP addresses are always evenly distributed for normal traffic. Mehdi et al. in [18] used maximum entropy estimation to determine benign traffic distribution and anomaly detection in SDN. The solution focuses only on small networks such as office and home networks.

Braga et al. [19] performed cluster analysis to detect DDoS in SDN. In this work, the system continuously collects statistical features of the flows and observe the collected features to identify any unusual activity. However, gathering and observing a large amount of data significantly deteriorates the performance of the controller. YuHunag et al. in [20] proposed a flow monitoring system to identify a DDoS attack. The proposed approach produces spurious alarms in a situation when a benign user starts to generate a large volume of traffic. Dong et al. in [21] introduced a solution to deal with *Packet_In* flooding attack against the controller. Shin et al. in [22] proposed a system called Avant-Guard that identifies DDoS attack induced by a flooding of TCP SYN packets. LineSwitch [23] improves Avant-Guard through a solution based on probability and blacklisting. However, both Avant-Guard and LineSwitch focus only on SYN flooding-based saturation attacks.

Kotani et al. in [24] proposed a *Packet_In* filtering approach for protection of control plane in OpenFlow networks. The solution requires large TCAM space to accommodate pending flow tables. Also, it fails when the datapath cannot parse packets of certain protocols and extract the required information, for instance, payloads in ARP, VLAN ID in 802.1Q headers. To defend against DDoS attacks, SDNShield [25] requires

deployment of specialized software boxes. Wang et al. [26] introduced a DDoS mitigation architecture where the DDoS mitigation strategy relies on a public cloud provider, which takes actions against the threats. However, the authors did not clarify where the suspicious traffic is hosted after it is detected.

Kalliola et al. [27] proposed a machine-learning based approach that integrates traffic learning with external blacklist information for DDoS detection. The defense mechanism works at the IP layer; hence, attacks targeting other layers do not fall within the scope of the proposed defense mechanism. The work presented in [28] employs a multi-controller system to solve the problem of DDoS attacks. However, the approach has several limitations. On one side, it employs random packet transmission delay to protect from scanning attacks, which in fact, affects the data transmission for legitimate users. On another side, synchronization of prolonged route tables among multiple controllers is overlooked.

To summarize, existing solutions either focus on a particular attack type or require alterations in SDN infrastructure and support from external entities such as public cloud provider. Our work is different from the state-of-the-art on various dimensions: (1) it can detect various fundamentally different attacks; (2) it does not require any change to the infrastructure or support from external entities; (3) it does not need any exhaustive training before implementation; and (4) it adapts automatically to changing traffic pattern.

III. Threat Model

The target of the attacker is a server, i.e., the victim server, which provides services to the hosts. The victim's network employs SDN as an underlying network architecture. The attacker has no information about the topology of victim's network, but the attacker knows that the victim's network is using SDN. The attacker and the victim may reside in the same or different networks. The attacker could be an individual user, a group of users, or a group of compromised systems (bots) controlled by the attacker. The attacker uses IP spoofing as a camouflage technique. When the attacker launches a DDoS attack (for attack details, please refer to Section V), it reaches victim's network and can cause damages to the network resources as well as to the victim.

IV. Proposed Approach

In this section, we present our framework for detection of DDoS attacks in SDN. Here, we explain the fundamental principles, followed by its comprehensive implementation details.

A. Cumulative Sum (CuSum)

A non-parametric method, called the CuSum approach, is an anomaly detection technique used for change point detection. CuSum based mechanisms measure the deviation of current observation from a historical (long-term) average of the observations. In our framework, we consider the volume of packets flowing per unit of time as a parameter to CuSum. When the current observation overshoots the historical average of the observations, then the value of CuSum coefficient ascends, and vice versa. Hence, if the value of CuSum coefficient surpasses an implemented threshold, it designates an exaggerating packet arrival rate, which is likely to be due to a DDoS attack [29]. Equation (1) shows the computation of the CuSum coefficient:

$$S(t) = \max\{0, (S(t-1) + N_{pk}(t) - m(t))\}; \quad S(0) = 0, \quad (1)$$

where t represents the time of current observation, $t-1$ represents the time of previous observation, $S(t)$ represents the CuSum coefficient at time t, $N_{pk}(t)$ represents the number of packets arrived between $t-1$ and t, and $m(t)$ represents the long-term average of packets arrived till t. Equation (2) shows the computation of $m(t)$:

$$m(t) = \epsilon * m(t-1) + (1-\epsilon) * N_{pk}(t); \quad m(0) = 0, \quad (2)$$

where the value of ϵ varies from zero to one, i.e., $0 < \epsilon < 1$. It is clear from Eq. (2) that a value of ϵ that is greater than 0.5 indicates dominance of the historical average of packet count; otherwise, current packet count holds more importance.

Interpreting a CuSum graph is straightforward. A segment of the CuSum graph with a positive slope represents a duration when the values tend to be higher than the overall average. Similarly, a segment of the CuSum graph with a negative slope represents a duration when the values tend to be lower than the overall average. An abrupt change in the direction of the CuSum value represents a sudden change or shift in the average. Segments where the CuSum graph observes a relatively straight path represents a period when the overall average did not change much.

Need for an Adaptive Threshold: Our approach incorporates an adaptive threshold system for the following reasons:

1) A static threshold cannot consider the tendencies and recurring conduct of the network traffic. As an example, traffic load during peak hours is expected to remain higher as compared to off-peak hours, which may induce abundant false alarms if a static threshold is engaged.
2) On the contrary, an adaptive threshold can adapt to the trends of traffic.
3) An adaptive threshold can help to reduce false alarms [30].

B. CuSum with Adaptive Threshold

Our method uses an adaptive threshold for CuSum with the following rules:

For the current value of CuSum (C_L) in a window (W_L) of L seconds:

1) **if** $C_L > \mu_{C_L} + k \cdot \sigma_{C_L}$ **then**
   $threshold_{new} = threshold_{old} + \alpha \cdot threshold_{old}$
2) **else if** $C_L < \mu_{C_L} - k \cdot \sigma_{C_L}$ **then**
   $threshold_{new} = \max(threshold_{old} - \beta \cdot threshold_{old}, min\_threshold)$
3) **else**
   $threshold_{new} = threshold_{old}$

where, k is a constant, α and β determines the degree of adjustment in the threshold. The values of α and β can be chosen to create a slow-increase/fast-decrease effect.

Here, μ_{C_L} represents the average of CuSum values in W_L, which is computed using Eq. (3).

$$\mu_{C_L} = \frac{\sum_{i=1}^{L} C_i}{L}. \quad (3)$$

And σ_{C_L} represents the standard deviation of CuSum values in W_L, which is computed using Eq. (4).

$$\sigma_{C_L} = \sqrt{\frac{\sum_{i=1}^{L}(C_i - \mu_{C_L})^2}{(L-1)}}. \quad (4)$$

C. System Model

We implemented our framework as an SDN controller module that works alongside other forwarding modules. The value of CuSum for the server is computed periodically. At the same time, μ_{C_L} and σ_{C_L} of the CuSum values within the window W_L are also computed. The window of observation glides in a sliding window fashion to accommodate new values. The threshold is adjusted according to the rules described in Section IV-B. When the observed CuSum overshoots the threshold, then an attack is identified.

One important aspect in CuSum computation is to enumerate the packets arrived between $t-1$ and t, i.e., $N_{pk}(t)$. The OpenFlow [31] switches maintain counters that include the number of packets, bytes, etc., for each flow entry. In our approach, the controller utilizes the built-in message exchange capabilities of the OpenFlow protocol and proactively interacts with the switches. As a part of the proactive interaction, the controller periodically exchanges common *FLOW_STATS* messages to acquire real-time traffic statistics. Using *FLOW_STATS* replies from the switches, the controller can enumerate $N_{pk}(t)$. Each flow-rule has a *HARD_TIMEOUT* and a relatively shorter *IDLE_TIMEOUT*. A shorter *IDLE_TIMEOUT* ensures rapid eviction of momentary flow-rules.

V. Experiments

We built our test scenario as reliable and realistic as possible, by considering the suggestions in [32]. Figure 4 shows the network topology of our test scenario. The target system resides in Network 1, which is composed of three OpenFlow switches controlled by a POX[2] controller. Here, Open vSwitch[3] serves as an OpenFlow-enabled switch. The botnets that flood DDoS traffic are in Network 2 while legitimate traffic flows from hosts residing in Network 2 and Network 3. Inter-network links are configured with 1 Gbps bandwidth and 25 ms of delay while the links among OpenFlow switches are configured with 1 Gbps bandwidth. All other intra-network links are set to 100 Mbps bandwidth. We emulated our test scenario using the Mininet[4].

[2]POX - `http://github.com/noxrepo/pox/`
[3]Open vSwitch - `http://openvswitch.org/`
[4]Mininet - `http://mininet.org/`

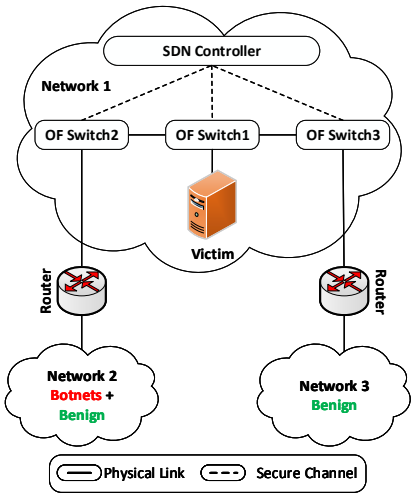


Figure 4: Test scenario's topology

As SDN is still an emerging networking concept, no DDoS attack dataset for SDN was publicly available at the time of evaluation. To evaluate our proposed approach, we orchestrated two different experiments. In the first experiment, we produced synthetic traffic for both legitimate users and botnets using Scapy[5]. To differentiate between the attack traffic and the normal traffic we used traces provide by CAIDA. These network traces are longer than an hour in duration and contain a distribution of traffic for various geographical locations. The information available in CAIDA traces was used to compute the average packet transmission rate for a benign source in a network. After considering the work presented in [33], we constituted a 20% attack traffic where botnet traffic comprised ICMP pings and had a rate higher than the normal traffic.

To assess effectiveness and versatility of our proposed mechanism we set another experiment. Here, we used DARPA intrusion detection evaluation dataset as they contain ample type of attacks, i.e., over 200 instances of more than 50 types of attacks. We downsampled and transformed the packet traces to match our emulation settings. As a representative example, Table I shows some attacks that may overload various components of an SDN environment.

Attacks	Descriptions
Smurf	Victim's source IP is used to broadcast ICMP requests to a network, which creates a reply flood towards the victim.
Neptune	A flood of SYN on one or more TCP ports.
IPsweep	ICMP pings are sent to every address within a subnet, and ping responses help to identify which hosts are listening.
Portscan	A surveillance sweep that scans several ports to identify which services are running on a machine.

Table I: Some attacks that may overload SDN components

It is important to note that these attacks have different working principles and they work at different layers. For example, "IPsweep" works at the network layer while "Neptune" works at the transport layer. Although "Portscan" and "IPsweep" are not considered as DDoS attacks by conventional intrusion

[5]Scapy - `http://www.secdev.org/projects/scapy/`

detection mechanisms, however, they may be used to generate a huge number of traffic flows to overload SDN components.

VI. Results and Analysis

In this section, we present and discuss the results from our experiments. Figures 5, 6, and 7 are plotted with respect to the first experiment mentioned in Section V. The purpose of Figure 5 and Figure 6 is to illustrate the effect of CuSum values on the threshold. Figure 5 depicts the computed value of CuSum and the corresponding threshold under the normal traffic. The initial threshold was computed according to the information available in the CAIDA traces. Although the traffic generator script generates the traffic right from the beginning, it takes a while to transmit the actual traffic flows. Because in the beginning, the hosts exchange initial network message such as ARPs. Once the network stabilizes, the normal traffic exhibits a relatively steady state. Meanwhile, the threshold adapts according to the value of CuSum. Since the threshold remained higher than the CuSum value, the traffic was classified as benign traffic.

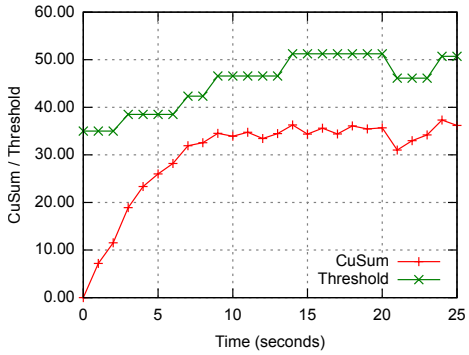


Figure 5: Variation of threshold against CuSum values under normal traffic

Figure 6 depicts the computed value of CuSum and the corresponding threshold under the attack traffic. After the initial exchange of network messages, attack flows generate a huge amount of traffic. The threshold adapts according to the traffic, but the value of CuSum quickly surpasses the threshold. The CuSum value exceeded the threshold around the sixth second and continued to stay above it.

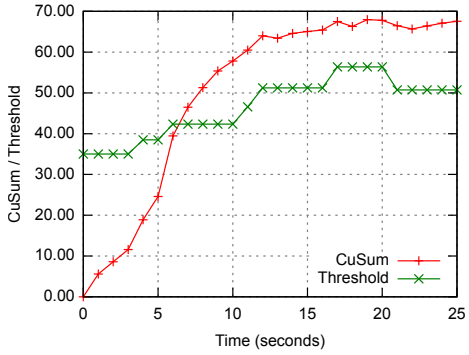


Figure 6: Variation of threshold against CuSum values under attack traffic

Figure 7 depicts the computed value of CuSum and corresponding threshold under the traffic that contains both normal and attack traffic. In this case, two attack sessions were scheduled, the first one starting from the nineteenth second till the twenty-ninth second and other one starting from the fifty-ninth second till the sixty-eighth second. Both the attack sessions were detected within four seconds. Although, there were a few false positives after the attack sessions were over.

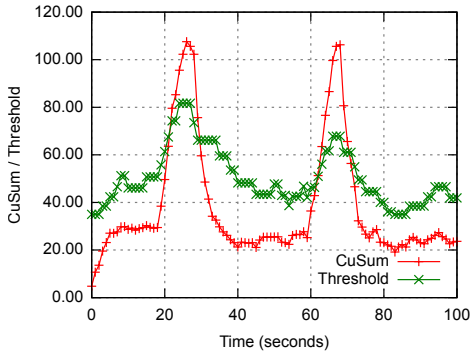


Figure 7: Variation of threshold against CuSum values under the traffic consisting of both normal and attack traffic

Now we discuss the results of the second experiment. Here, we conducted experiments for each combination of k, α, β, and W_L. The chosen values of k were 0.5, 1.0, 2.0 while the values for α, β were 0.100, 0.050, 0.025. For the window size (W_L), the chosen values were from 1 to 10. We used Detection Rate (DR), False Alarm Rate (FAR), Accuracy (ACC) to assess our proposed approach. The confusion matrix [34] as presented in Table II is a standard matrix that is widely used for the assessment of classification methods.

Confusion Matrix		**Predicted Label**	
		Normal	**Attack**
Actual Label	**Normal**	True Negative (TN)	False Positive (FP)
	Attack	False Negative (FN)	True Positive (TP)

Table II: Confusion matrix

DR measures the percentage of correctly identified attacks over all the actual attacks and is computed using Eq. (5).

$$DR\ (\%) = \frac{TP}{TP + FN} * 100. \tag{5}$$

FAR measures the percentage of legitimate traffic incorrectly identified as attack over the entire legitimate traffic and is computed using Eq. (6).

$$FAR\ (\%) = \frac{FP}{FP + TN} * 100. \tag{6}$$

ACC measures the percentage of true detection over the entire traffic trace and is computed using Eq. (7).

$$ACC\ (\%) = \frac{TP + TN}{TP + TN + FP + FN} * 100. \tag{7}$$

To illustrate the effectiveness of our approach, Figure 8, as an illustrative example, shows DR, FAR, and ACC for different window sizes with k set to 1 and α, β set to 0.025. In this case, since our method did not produce any false negative, DR was 100% for all window sizes. While FAR was 11.63% for window size one and two seconds, which improves and reaches 6.98% for wider windows. Since ACC is affected by true as well as false detections; hence, ACC was 90.38% for window size one and two seconds, which improves and reaches 94.23% for wider windows.

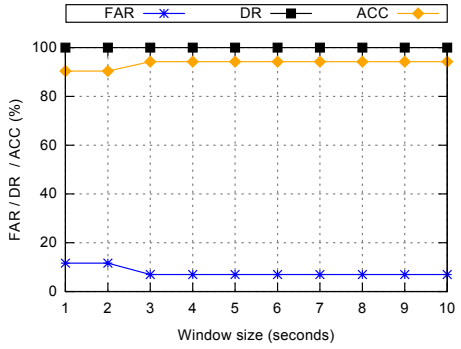


Figure 8: DR, FAR, and ACC for different window sizes where k=1 and α = β = 0.025

We performed ten experiments for each combination of k, α, and β, one for each window size. Due to space limitation, we show the results of the experiments where α was set equal to β. Table III presents the average and the standard deviation of DR, FAR, and ACC computed for results from each window size under every unique combination of k, α, and β. As explained in Section IV-B, the value of α, β decides the degree of modification in the threshold.

k	$\alpha = \beta$	μ_{DR} (%)	σ_{DR} (%)	μ_{FAR} (%)	σ_{FAR} (%)	μ_{ACC} (%)	σ_{ACC} (%)
0.5	0.100	10.00	31.62	1.16	3.68	83.46	2.43
	0.050	74.44	20.98	1.86	3.60	94.04	3.07
	0.025	96.67	5.37	5.58	2.50	94.81	1.82
1.0	0.100	35.56	46.50	2.56	4.83	86.73	5.47
	0.050	94.44	5.86	4.88	3.71	95.00	2.43
	0.025	100.00	0.00	7.91	1.96	93.46	1.62
2.0	0.100	100.00	0.00	9.07	2.99	92.50	2.47
	0.050	100.00	0.00	11.16	0.98	90.77	0.81
	0.025	100.00	0.00	11.63	0.00	90.38	0.00

Table III: Average value and standard deviation of DR, FAR, and ACC for different values of k, α, β

DR degrades with increasing value of α, β because a larger value of α, β modifies the threshold more as compare to smaller values. Consequently, attacks are misclassified. While FAR increases with decreasing value of α, β because the threshold does not appropriately adapt for a smaller value of α, β. An increasing value of k improves the DR, while the FAR is also increased. ACC observes no direct relation with α, β, or k as it relies on both true and false detections.

Detection Time: Another important evaluation criteria for any detection method is the time it takes to detect the attack. The detection time in our approach improves with increasing window size. The detection time was under six and a half seconds for all the experiments, and the average detection time considering all the experiments was 4.15 seconds with a standard deviation of 1.92 seconds.

Overhead: In our solution, the controller utilizes commonly exchanged *FLOW_STATS* messages to obtain real-time traffic statistics (in particular, to compute $N_{pk}(t)$) for DDoS detection. The induced overhead of our solution majorly depends on the frequency of message exchange. To assess the overhead, we configured the controller to exchange messages every second on a system with Intel Core i5-7200U CPU @ 2.50 GHz x 4 processor. The measured CPU overhead due to message exchange was nearly 11%. We believe that the reported overhead is not an issue since in a real-network scenario, the controller runs on a dedicated resource-rich server grade system.

VII. Conclusion and Future Work

SDN provides a simpler network administration with more flexibility as compared to the traditional networks. There are several security-related concerns in SDN, which are still required to be solved. In this work, we have proposed a framework that is capable of detecting various fundamentally different DDoS attacks in SDN. As shown by the results, the proposed approach is not only effective, but it can also be tuned on various parameters to fit vast security requirements, e.g., high DR, low FAR.

In the future, we will extend our approach to find a feasible way to mitigate an attack after its detection. We would also extend our approach to multi-domain networks containing more than one SDN controller. It would be interesting to investigate the detection and mitigation of DDoS attacks where multiple SDN controllers can communicate with each other over dedicated (e.g., EAST/WEST bound) interfaces.

Acknowledgment

Mauro Conti is supported by a Marie Curie Fellowship funded by the European Commission (agreement PCIG11-GA-2012-321980). Ankit Gangwal is pursuing his Ph.D. with a fellowship for international students funded by Fondazione Cassa di Risparmio di Padova e Rovigo (CARIPARO). This work is partially supported by the EU TagItSmart! Project (agreement H2020-ICT30-2015-688061). This work is partially supported by the grant n. 2017-166478 (3696) from Cisco University Research Program Fund and Silicon Valley Community Foundation, and by the grant "Scalable IoT Management and Key Security Aspects in 5G Systems" from Intel. This work is also partially funded by the project CNR-MOST/Taiwan 2016-17 "Verifiable Data Structure Streaming".

References

[1] A. M. Batishchev, "Low Orbit Ion Cannon (LOIC)," https://sourceforge.net/projects/loic, 2012.
[2] S. Sanfilippo, "hping3," http://www.hping.org/, 2006.
[3] D. Dittrich, "The stacheldraht DDoS tool," http://staff.washington.edu/dittrich/misc/stacheldraht.analysis.txt.

[4] F. Hu, Q. Hao, and K. Bao, "A survey on software defined network and OpenFlow: From concept to implementation," *IEEE Communications Surveys & Tutorials*, vol. 16, no. 4, pp. 2181–2206, 2014.

[5] A. Akhunzada, E. Ahmed, A. Gani, M. Khan, M. Imran, and S. Guizani, "Securing software defined networks: Taxonomy, requirements, and open issues," *IEEE Communications Magazine*, vol. 53, no. 4, pp. 36–44, 2015.

[6] Y. Jarraya, T. Madi, and M. Debbabi, "A survey and a layered taxonomy of software defined networking," *IEEE Communications Surveys & Tutorials*, vol. 16, no. 4, pp. 1955–1980, 2014.

[7] The CAIDA UCSD Statistical information for the CAIDA Anonymized Internet Traces, http://www.caida.org/data/passive/passive_trace_statistics.xml, 2016.

[8] MIT Lincoln Laboratory, "Intrusion detection attacks database," http://www.ll.mit.edu/ideval/docs/attackDB.html.

[9] M. Jelena and R. Peter, "A taxonomy of DDoS attack and DDoS defense mechanisms," *ACM SIGCOMM Computer Communication Review*, vol. 34, no. 2, pp. 39–53, 2004.

[10] S. T. Zargar, J. Joshi, and D. Tipper, "A survey of defense mechanisms against Distributed Denial of Service (DDoS) flooding attacks," *IEEE Communications Surveys & Tutorials*, vol. 15, no. 4, pp. 2046–2069, 2013.

[11] S. Ranjan, R. Swaminathan, M. Uysal, and E. W. Knightly, "DDoS-resilient scheduling to counter application layer attacks under imperfect detection," in *IEEE INFOCOM*, 2006, pp. 1–13.

[12] H. J. Liao, C. H. R. Lin, Y. C. Lin, and K. Y. Tung, "Intrusion detection system: A comprehensive review," *Journal of Network and Computer Applications*, vol. 36, no. 1, pp. 16–24, 2013.

[13] M. Dabbagh, B. Hamdaoui, M. Guizani, and A. Rayes, "Software defined networking security: Pros and cons," *IEEE Communications Magazine*, vol. 53, no. 6, pp. 73–79, 2015.

[14] F. Kargl, J. Maier, and M. Weber, "Protecting web servers from distributed denial of service attacks," in *WWW*, 2001, pp. 514–524.

[15] M. Sachdeva, G. Singh, and K. Kumar, "Deployment of distributed defense against DDoS attacks in ISP domain," *International Journal of Computer Applications*, vol. 15, no. 2, pp. 25–31, 2011.

[16] Y. Chung, "Distributed denial of service is a scalability problem," *ACM SIGCOMM Computer Communication Review*, vol. 42, no. 1, pp. 69–71, 2012.

[17] S. M. Mousavi and M. St-Hilaire, "Early detection of DDoS attacks against SDN controllers," in *IEEE ICNC*, 2015, pp. 77–81.

[18] S. A. Mehdi, J. Khalid, and S. A. Khayam, "Revisiting traffic anomaly detection using software defined networking," in *RAID*. Springer, 2011, pp. 161–180.

[19] R. Braga, E. Mota, and A. Passito, "Lightweight DDoS flooding attack detection using NOX/OpenFlow," in *IEEE LCN*, 2010, pp. 408–415.

[20] C. YuHunag, T. MinChi, C. YaoTing, C. YuChieh, and C. YanRen, "A novel design for future on-demand service and security," in *IEEE ICCT*, 2010, pp. 385–388.

[21] P. Dong, X. Du, H. Zhang, and T. Xu, "A detection method for a novel DDoS attack against SDN controllers by vast new low-traffic flows," in *IEEE ICC*, 2016, pp. 1–6.

[22] S. Shin, V. Yegneswaran, P. Porras, and G. Gu, "AvantGuard: Scalable and vigilant switch flow management in software defined networks," in *ACM CCS*, 2013, pp. 413–424.

[23] M. Ambrosin, M. Conti, F. De Gaspari, and R. Poovendran, "LineSwitch: Efficiently managing switch flow in software defined networking while effectively tackling DoS attacks," in *ACM ASIACCS*, 2015, pp. 639–644.

[24] D. Kotani and Y. Okabe, "A Packet-In message filtering mechanism for protection of control plane in OpenFlow networks," in *ACM/IEEE ANCS*, 2014, pp. 29–40.

[25] K. Chen, A. R. Junuthula, I. K. Siddhrau, Y. Xu, and H. J. Chao, "SDNShield: Towards more comprehensive defense against DDoS attacks on SDN control plane," in *IEEE CNS*, 2016, pp. 28–36.

[26] B. Wang, Y. Zheng, W. Lou, and Y. T. Hou, "DDoS attack protection in the era of cloud computing and software defined networking," *Elsevier Computer Networks*, vol. 81, pp. 308–319, 2015.

[27] A. Kalliola, K. Lee, H. Lee, and T. Aura, "Flooding DDoS mitigation and traffic management with software defined networking," in *IEEE CloudNet*, 2015, pp. 248–254.

[28] D. Ma, Z. Xu, and D. Lin, "Defending blind DDoS attack on SDN based on moving target defense," in *SecureComm*. Springer, 2014, pp. 463–480.

[29] I. Ozcelik, Y. Fu, and R. R. Brooks, "DoS detection is easier now," in *2nd GENI Research and Educational Experiment Workshop*, 2013, pp. 50–55.

[30] V. A. Siris and F. Papagalou, "Application of anomaly detection algorithms for detecting SYN flooding attacks," *Elsevier Computer communications*, vol. 29, no. 9, pp. 1433–1442, 2006.

[31] N. McKeown, T. Anderson, H. Balakrishnan, G. Parulkar, L. Peterson, J. Rexford, S. Shenker, and J. Turner, "OpenFlow: Enabling innovation in campus networks," *ACM SIGCOMM Computer Communication Review*, vol. 38, no. 2, pp. 69–74, 2008.

[32] J. Mirkovic, S. Fahmy, P. Reiher, and R. K. Thomas, "How to test DoS defenses," in *Cybersecurity Applications & Technology Conference for Homeland Security*, 2009, pp. 103–117.

[33] J. Sommers, V. Yegneswaran, and P. Barford, "Toward comprehensive traffic generation for online IDS evaluation," *Technical Report, University of Wisconsin*, 2005.

[34] P. Dokas, L. Ertoz, V. Kumar, A. Lazarevic, J. Srivastava, and P. N. Tan, "Data mining for network intrusion detection," in *NSF Workshop on Next Generation Data Mining*, 2002, pp. 21–30.

ELBA: Efficient Layer Based routing Algorithm in SDN

Ankit Gangwal*, Megha Gupta*, Manoj Singh Gaur*, Vijay Laxmi*, Mauro Conti†
* Department of Computer Science & Engineering, Malaviya National Institute of Technology, India.
† Department of Mathematics, University of Padua, Italy.
Email: {2014pcp5290, 2014pcp5026, gaurms, vlaxmi}@mnit.ac.in, conti@math.unipd.it

***Abstract*—Adaptive streaming dynamically adapts video quality level according to the perceived device status and network conditions. It requires several representations of the same content, each encoded at different quality rates. As a representative example, H.264/SVC eliminates the requirement of redundant representations, improving the efficiency of caching and storage infrastructure. SVC video consists of a "Base Layer" and one or more of "Enhancement Layers". These layers have inter-dependencies and different QoS requirements. On another side, SDN allows forwarding tables to be adjusted dynamically, enabling us to route every individual flow differently. In this paper, we propose ELBA, an algorithm for scalable video streaming over SDN. ELBA utilizes the dynamic re-routing capability of SDN, to stream different layers of SVC coded video over possibly different suitable paths. In the proposed video streaming system, we use a novel mechanism to exchange information between the control plane and streaming servers. We have compared the performance of our approach with traditional Internet routing technique. Our evaluation results show that our proposal is not only feasible but in particular, it significantly outperforms the traditional Internet routing approach in terms of QoE.**

***Index Terms*—H.264/SVC, QoE, SDN, Video Streaming**

I. Introduction

Over the past decade, the demand for multimedia services over the Internet has witnessed a tremendous growth. Internet video traffic is projected to be 80% of entire Internet traffic by 2019 [1]. Web video streaming, mobile TV, real-time video conference and many other streaming media applications require steady network resources with no or little variations. These specific requirements cannot always be effectively met by the best-effort Internet. Also, there may exist more than one route (path) between network entities, where each path may have different properties, e.g., one path may provide lower error rate while the other may offer higher bandwidth. For a reliable transmission of video streams and reuse of existing infrastructure adaptive streaming techniques are gaining popularity.

In adaptive streaming, a video is split into segments, and different quality representations are used to encode these segments. Traditionally, H.264 Advanced Video Coding (H.264/AVC) [2] is used to encode video segments. However, AVC lacks scalability, restricting it to meet the diverse requirements of different users having varying displays sizes and connected through differing network links. To overcome this limitation, a scalable extension of H.264/AVC was proposed, which is known as H.264 Scalable Video Coding (H.264/SVC)

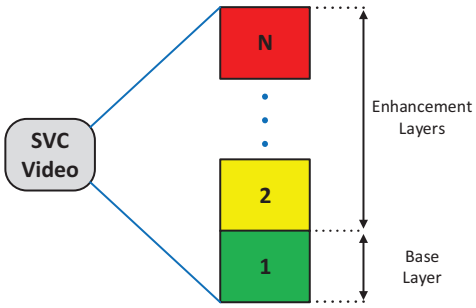


Figure 1: Illustration of SVC video layers

[3]. SVC encodes a video in a base layer and one or more enhancement layers, as shown in Figure 1. The base layer provides basic (standard) video quality while adding enhancement layer to the base layer enhances quality.

Software-defined networking (SDN) [4, 5] is a recently emerging paradigm. The key concept of SDN is to decouple data plane and control plane. The data plane provides actual forwarding functionality. The data plane is controlled by the control plane. The control plane is programmable. The programmability of control plane enables us to devise flexible routing algorithms, which can adapt according to the diverse requirements of various network applications.

The aim of developing a new routing algorithm is to enhance the quality of experience (QoE) for the end user. There are various proposals for materializing new streaming technique for video applications working over SDN. But one commonly practiced assumption in SDN literature is that the control plane always has prior information about the characteristics (e.g. bit-rate, etc.) of the video traffic. One of the biggest challenges in SDN environment is to efficiently exchange traffic's characteristics information between hosts and the control plane.

In this paper, we propose ELBA (Efficient Layer Based routing Algorithm), an algorithm that exploits the dynamic re-routing capability of SDN, to stream different layers of SVC coded video over distinctly suitable paths. It intends to improve the delivered video quality without affecting rest of the network traffic and also to improve utilization of network resources. We also propose a novel and feasible mechanism to exchange information between the control plane and streaming servers. To the best of our knowledge, a mechanism to exchange information about network traffic's characteristics

978-1-5090-2279-3/16/$31.00 ©2016 IEEE

between the control plane and data plane elements has not been proposed previously. While in this paper we focus particularly on scalable videos, we believe that the approach used in this context can be extended to other network services as well.

The remainder of this paper is organized as follows. Section II presents an overview of related work regarding multimedia communications over SDN. Section III elaborates the proposed video streaming system and routing algorithm. The details of prototype implementation and results are discussed in Section IV. In Section V, the conclusions are given, followed by the references.

II. Related Work

In SDN environment, the key concept is to partition a network into data plane and control plane. The data plane consists of many forwarding devices that provide actual forwarding of data packets. The data plane is controlled by the control plane. The control plane consists of at least one decision-making entity called the controller, which has a global view of the network of its domain. The controller communicates with the forwarding devices to acquire traffic information in real-time. Utilizing topology information and the real-time traffic statistics, the controller can decide the route of data packets in the network. Hence, SDN allows the network operators to develop application-specific route decision strategies considering the network topology information and the obtained traffic statistics.

The controller and the switches communicate via OpenFlow [6] protocol. It creates a secure communication channel between the controller and switches. The controller instructs the switches by simply updating their 'flow table' via the OpenFlow protocol. Apart from adding flow table entries, the controller can also delete/modify existing entries. It can also ask for real-time traffic statistics using the OpenFlow protocol.

In [7] and [8], authors have addressed that overall network throughput can be significantly affected by Quality-of-Service (QoS) routing. A routing algorithm striving to optimize the QoS traffic cannot ignore the presence of best-effort traffic. As the majority of the Internet traffic is best-effort, any performance enhancing approach concerned with QoS traffic must also account for best-effort traffic and prevent its congestion. To attain better performance, Peter et al. in [7] suggest that QoS traffic should avoid heavily loaded best-effort shortest-paths. The work presented in [9] proposes a model of an intermediary adaptation node, where an estimation of the available bandwidth on the client's link by media gateway removes identified SVC streams. Schierl et al. in [10] demonstrate a graceful degradation of quality in SVC streaming when the network load increases.

In [11], a flow rate shaper for SDN is proposed, which adapts the transmission pace according to the flow rate of distinct applications. Based on SDN framework, a network caching service along with load balancing for video on demand (VoD) applications is proposed in [12]. According to service negotiations, the work in [13] suggests to assign appropriate flows to the users. Patrick et al. in [14] propose to assure a predictable quality of service level. In the proposed work, an IPTV service operates in SDN system, and there are two paths between the client and server. The secondary path is chosen for video streaming whenever a problem is discovered on the primary path. In this work the method of primary path selection is not defined. In [15], the controller changes the quality of the streamed video according to the client's download capacity. In this study, client's download capacity information is acquired from the switches, but path selection method is not discussed.

A non-layered codec requires several representations of the same video content, each encoded at different quality rates. In contrast, layered codec such as H.264/SVC offers higher storage and transmission efficiency. SVC video contains a "Base Layer" and one or more of "Enhancement Layers". These layers have inter-layer decoding dependencies. The video plays at the lowest quality when the client gets only the base layer. Adding an enhancement layer enhances the quality [16].

Civanlar et al. in [17] propose to route the base layer over a lossless path. After considering the available bandwidth and congestion, the controller selects a path having adequate bandwidth to stream the base layer. In [18], the paths are allotted by analyzing the length and capacity of the path. The work in [19] proposes amendments to [18] for multi-domain SDN networks. The work in [20] considers the priorities of the base layer and enhancement layer packets to define different flow rules for these layers. Each video layer is streamed through different TCP port. The controller identifies each video layer by inspecting the TCP source port value of the packets. In this study, the process of informing the controller about this mapping of TCP port value to the video layer streamed through it is not discussed. Similar to this approach, another path selection technique for streaming the base and enhancement layers separately is discussed in [21]. In [22], a learning based approach is proposed. The controller considers the available bandwidth for route selection and performs periodic quality adaption by instructing the streaming server to add/remove one or more video layers. The controller signals the video server via its northbound API. However, the method of intimating the controller about the bit-rate information of each video layer is not addressed in this study. Similarly, the procedure of informing the controller about the characteristics of video layers is not discussed in [17, 18, 19, 21].

In this paper, we propose an efficient video streaming technique to enhance the quality of the delivered video. It uses a combination of the dynamic routing capabilities of SDN and the layered characteristics of SVC video to stream different layers of SVC coded video over possibly different suitable paths. We also propose a novel and feasible mechanism to convey information about the characteristics of video layers between the control plane and data plane elements.

III. ELBA: Proposed Scalable Video Streaming Technique

In this section, we present ELBA, our solution for scalable video streaming over SDN. Here, we elucidate the underlying principles and overview of ELBA, followed by its comprehensive implementation details.

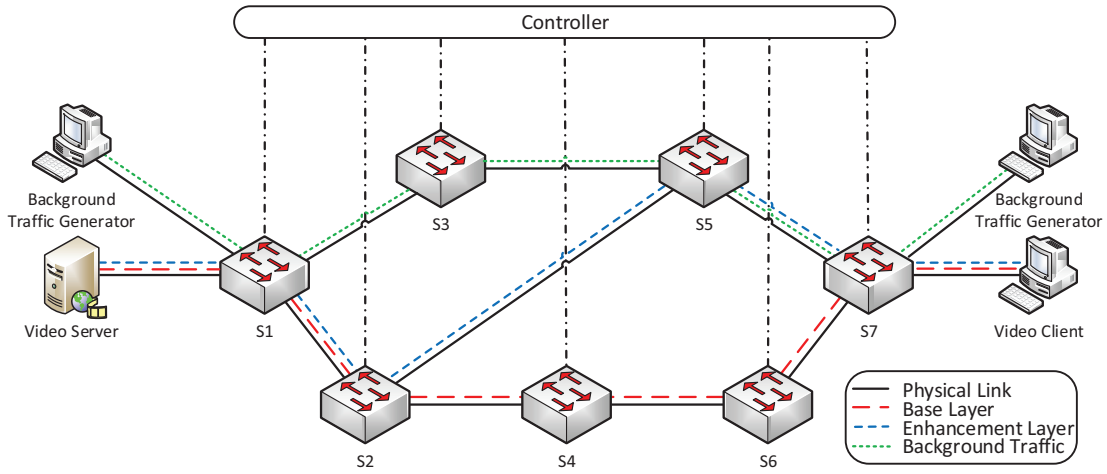


Figure 2: ELBA: an illustrative example for video streaming

A. Algorithm Design

According to the tenet of SVC, an upper layer of video becomes useless if any of the corresponding lower layers is absent at the decoding time. Hence, the base layer packets are the most important as the absence of these packet makes all enhancement layer packets useless and the video starts to freeze. In real-time streaming, retransmitting lost packets is usually not suitable. On the other side, if an enhancement layer packet is lost, the video continues to play with degraded quality. So, the traffic stream in the network can be categorized into following distinct streams:

1) As SVC base layer must be streamed without any packet loss, hence it can be defined as lossless QoS (i.e., any packet loss is intolerable) traffic.
2) All SVC enhancement layers are defined as lossy QoS (i.e., few packet losses are tolerable) traffic.
3) Background traffic can be treated as best-effort traffic.

The quality of received video at the client depends greatly upon two factors, loss and delay. Due to the loss, the video packets might reach the client either as corrupted packets or they do not reach the destination at all. While due to the delay, the video packets might not reach the destination on time. Moreover, the effect of these consequences varies with the position of a layer in the SVC layer hierarchy.

To optimize the video delivery, based on the routing decision the controller in our approach may assign distinct routes for each video layer. Paths for lossless and lossy QoS traffic is calculated using a slightly modified version of Dijkstra algorithm [23], where the respective edge's weight is considered according to Eq. 3. The dominating best-effort traffic is traditionally accommodated on a hop-based shortest-path. While in our approach, it is accommodated on a bottleneck shortest-path (maximum bandwidth) path [24], where the available bandwidth on a link serves as the capacity of the edge. Figure 2 depicts an abstract working of the proposed algorithm for a specific (and simple) topology. At S1, the controller decides to forward both base and enhancement layers over the same link until S2. At S2, both the layers take a separate path to reach the client. While the background traffic is forwarded via a different path from S1, which meets enhancement layer at S5. Influence of the link loss and link delay is elaborated in the remaining section.

1) Influence of the loss: Due to packet losses, exact frames cannot be decoded at the receiver's side. Furthermore, an undecoded lower layer makes all the corresponding upper layers useless, and the video starts to freeze. For this reason, higher priority is given to lower video layers, and lower layers are streamed over a path having a higher probability of delivering these layers. To accomplish this behavior, the loss component W_{loss} of the edge weight is weighted by a priority factor v_i for every i^{th} video layer. The value of v_i is inversely proportional to the priority of an i^{th} video layer. The loss component W_{loss} of the edge weight is given by Eq. 1, where L is the loss probability on the link.

$$W_{loss} = 2^{-v_i} \cdot L \tag{1}$$

2) Influence of the delay: Due to path delay, the video packets might not reach the destination on time. Too late frame packets are discarded. While early frame packets are saved in the buffer and they wait for their time to decode and display. For this reason, higher priority is given to upper video layers, and upper layers are streamed over paths offering quicker delivery of these layers. Consequently, upper layers become available when the base layer reaches the client. To accomplish this behavior, the delay component W_{delay} of the edge weight is also weighted by the priority factor v_i for every i^{th} video layer. The delay component W_{delay} of the edge weight is given by Eq. 2, where D is the delay introduced by the link.

$$W_{delay} = (1 - 2^{-v_i}) \cdot D \tag{2}$$

3) Total edge weight: The total edge weight W of a link can be described as the weighted arithmetic mean of the loss component W_{loss} and the delay component W_{delay}, as defined in Eq. 3.

$$\begin{aligned} W &= (1-\beta) \cdot W_{loss} + \beta \cdot W_{delay} \\ &= (1-\beta) \cdot 2^{-v_i} \cdot L + \beta \cdot (1 - 2^{-v_i}) \cdot D \end{aligned} \tag{3}$$

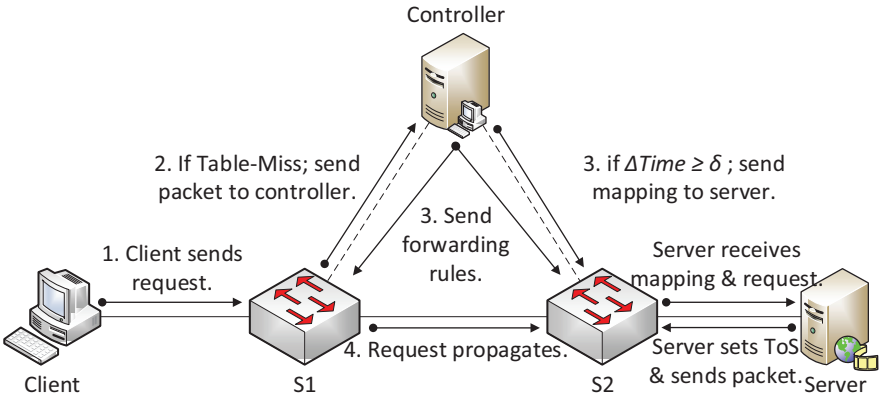


Figure 3: Conveying ToS mapping to a server

B. Influence of Flow-Timeouts

Every flow entry has a *HARD_TIMEOUT* and an *IDLE_TIMEOUT* associated with it. The switch notes the flow entry's arrival time, as it may require to remove the entry later. A non-zero *HARD_TIMEOUT* evicts a flow entry after the given number of seconds, irrespective of how many packets it has matched. A non-zero *IDLE_TIMEOUT* evicts a flow entry when it has not matched any packet in the given number of seconds [6]. In our proposed system, timeouts for flow entries are installed in the following manner:

1) Flow entries related to the video layers has only *IDLE_TIMEOUT* set.
2) Flow entries related to the background traffic have both *IDLE_TIMEOUT* and *HARD_TIMEOUT* set.

The existence of only *IDLE_TIMEOUT* for video related flow entries enables such entries not to be removed as long as the video transmission is active. It helps in avoiding the inevitable delay that incurs if *HARD_TIMEOUT* is set for an entry. Because removal of an entry causes a switch to re-consult the controller for an incoming packet that belongs to the expired flow entry.

The absence of *HARD_TIMEOUT* for video related entries causes switches to keep forwarding active video stream over the same path, despite a better path might be available as link's characteristics such as delay, etc., are not invariant. In our approach, when a client's request reaches the controller, it starts a timer for the client. Each time the timer for a client expires, the controller computes new paths for the video layers. If the new path differs from the previous path it sends corresponding forwarding rules to the switches. Otherwise, switches continue to apply previously installed rules to forward video packets.

C. Identifying Video Layers

When a client initiates a service request, it arrives at the switch to which the client is connected. Initially, the switch does not have a matching entry for this request. Hence, it sends a PACKET_IN message to the controller. The controller finds a path between the client and server and it instructs the corresponding switches to forward the request on an appropriate port. At the same time, the controller sends a mapping to the server which contains a layer identifier and corresponding IP Type-of-Service (ToS) field value for video layers. This mapping is called ToS mapping. While streaming, the server sets ToS field of the video packets according to the received ToS mapping. Figure 3 depicts the process of conveying ToS mapping to a server. Now, the controller can distinctly identify video layers by inspecting the ToS field value of incoming packets. Table I shows an illustrative example of the layer identifier and corresponding ToS field value for various video layers.

Table I: An illustrative example of ToS mapping

Video Layer	Layer Identifier	IP ToS Field Value
Base Layer	0	0x64
Enhancement Layer 1	1	0x6E
Enhancement Layer 2	2	0x78
Enhancement Layer 3	3	0x82
.	.	.
.	.	.
.	.	.

One of the major benefits of this approach is that the controller does not require to perform complex procedures, e.g., deep packet inspection of packet payload to identify video packets. Also, ToS field consumes only one byte in the packet header.

It is important to note that the server is not bound to receive the mapping. As a result, the server may miss it as well. One solution is to retransmit this mapping continuously. However, this would congest the controller-to-switch link. On the other side, several clients may send a request to the server at the same time. Sending mapping for each request will also congest the controller-to-switch link. In our proposed system, the controller records the time when it sends the mapping to a server. When a client's request comes to the controller, it checks when it last sent the mapping to the requested server. The controller sends the mapping to the requested server only if:

$$Time_{Current} - Time^{M}_{Last\_Sent} \geq \delta \tag{4}$$

M is the MAC address of requested server, δ defines the time duration between two consecutive transmissions of the mapping.

D. System Architecture

The implementation details of our approach are given in this section. Initially, the switches contain no entry in the flow tables. A packet from a host arrives at the switch to which the host is connected. The switch obtains the packet and sends a PACKET_IN message to the controller because initially its flow table has no matching entry. The controller determines an appropriate path for the packet and then sends forwarding rules to the corresponding forwarding elements on the computed path. The controller uses FLOW_MOD messages to send forwarding rules to the switches. Since a feature with a broad range of values can dominate the routing decision, the controller uses normalized values of link loss and delay while calculating routes. The controller interacts with the switches both in reactive and proactive manner. Whenever it receives a PACKET_IN message from a switch asking for rules, it reactively sends the forwarding rules to the switch. As a part of the proactive interaction, the controller periodically sends PORT_STATS and FLOW_STATS messages to the switches to acquire real-time traffic volume statistics. It also injects probe packets to calculate delay on the links, as described in [25]. The controller computes loss and available bandwidth on the links by utilizing the received traffic statistics.

IV. Evaluation

The details of experiment setup, performance metrics used and results are discussed in this section.

A. Experiment Setup

The evaluated network topology is illustrated in Figure 4. It has a POX [26] controller and four OpenFlow switches, each of the which is logically linked to the controller. The topology contains loops, providing multiple paths between server and client. Mininet [27, 28] is used to create the topology and perform the simulations. The video packets are sent by Real-time Transport Protocol (RTP) over User Datagram Protocol (UDP). Hence, we observe packet losses in every simulation. The value of δ and timer for each client is set to 10 seconds by the controller. Iperf [29, 30] client and server are used to generate Transmission Control Protocol (TCP) data stream which serves as the background traffic.

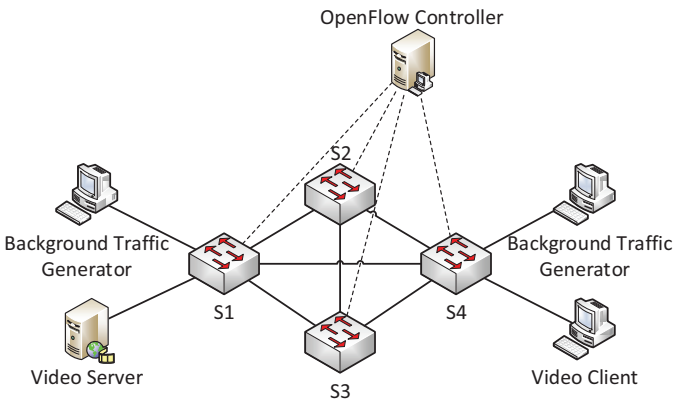


Figure 4: Evaluated network topology

The test video sequence used in the experiments is Foreman [31, 32]. It is in YUV CIF (352 x 288) format, and it consists of 300 frames. This video sequence is encoded by Joint Scalable Video Model (JSVM, version 9.19) [33] encoder with only temporal scalability enabled. The parameters of the resulting video are summarized in Table II.

Table II: Parameters of Foreman video

Layer	Resolution	Frame Rate (Fps)	Bit Rate (Kbps)	(DId, TId, QId)
0	352 x 288	7.5	514.10	(0,0,0)
1	352 x 288	15.0	548.70	(0,1,0)
2	352 x 288	30.0	588.10	(0,2,0)

The dependency id (DId) is used to define the spatial scalability inter-layer coding structure. The temporal id (TId) denotes the temporal scalability hierarchically. The quality id (QId) indicates the quality scalability structure.

B. Performance Metrics

We have compared the algorithms on the basis of average Peak Signal-to-Noise Ratio (PSNR), average frame loss rate and average throughput.

1) Average PSNR: PSNR provides an approximate measure of the quality as subjectively perceived by human observers. It is widely used because it has clear physical meanings. PSNR value for both luminance (Y-PSNR) and chrominance (U-PSNR and V-PSNR) components of the received video is calculated. A higher PSNR value corresponds to a better image quality.

2) Average Frame Loss Rate: A video frame is fragmented into multiple packets. The client cannot reconstruct the frame if any of these packets is lost. Therefore, we measure the percentage of lost frames instead of the percentage of lost packets as it expresses the perceived QoE more precisely.

3) Average Throughput: The video bit-rate received at the client is also measured. Improvement in the quality of the delivered video must not affect the rest of the network traffic. For this purpose average throughput for the Iperf client is also calculated.

C. Results

For a fair comparison and evaluation of the proposed algorithm, we performed ten runs of every algorithm. Averaged results are obtained and shown in the graphs. Figure 5, 6, 7 illustrate average PSNR value of Y, U and V components respectively for different algorithms. ELBA improves the Y, U and V components by 23.11%, 8.32%, 10.59% respectively when compared to shortest-path routing. Since the human eye is more sensitive to brightness (luminance) than color (chrominance), Y-PSNR typically holds more importance. Figure 8 shows the average frame loss rate for video traffic. The lower layer packets experience lesser losses because ELBA streams these packets over a less error-prone path. Successful delivery of lower layer packets leads to correct decoding of the frames. ELBA reduces frame loss rate by 39.56%. Figure 9 depicts the average bit-rate received for the video traffic while the average throughput for the background traffic is shown in Figure 10.

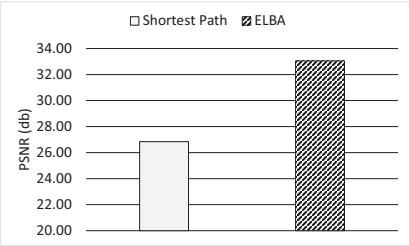


Figure 5: Average PSNR values for Y component

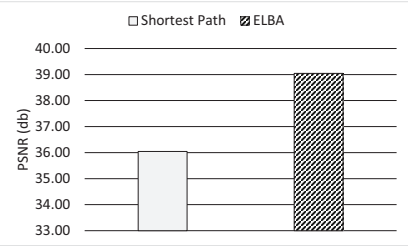


Figure 6: Average PSNR values for U component

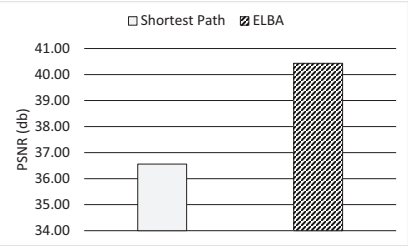


Figure 7: Average PSNR values for V component

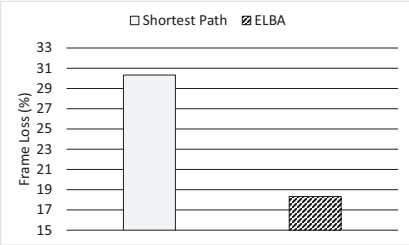


Figure 8: Average frame loss rate for video traffic

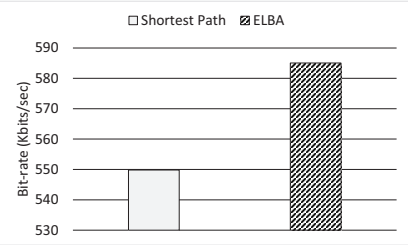


Figure 9: Average bit-rate received for video traffic

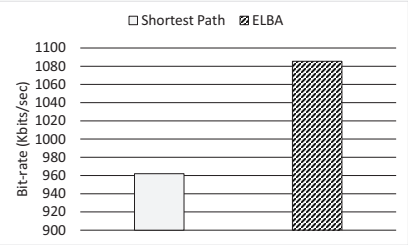


Figure 10: Average throughput for background traffic

As expected, shortest-path routing assigns the same path for each video layer as well as for the background traffic. In our approach, different video layers and the background traffic are streamed over their suitable paths. The received average video bit-rate is improved by 6.41% while the average throughput for the background traffic is improved by 12.83%.

Additionally, to illustrate how an end user perceives the difference in performance, we randomly chose and took snapshots of three consecutive frames from the received videos for each approach. Figure 11 illustrates a corresponding visual comparison in which frame number 94, 95 and 96 are snapshotted using a RAW video sequence player, i.e., PYUV [34]. By comparing the frames in Figure 11, it is evident that the video quality delivered using ELBA is better than conventional shortest-path routing approach.

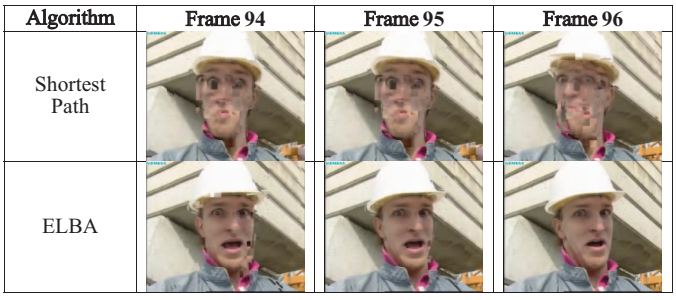


Figure 11: Visual comparison based on simulation

V. Conclusion and Future Work

In this paper, we have presented an algorithm for enhancing the quality of SVC-based scalable video streaming. SDN enables the forwarding devices to be updated dynamically, which allows us to stream different layers of SVC coded video over distinct network routes. As shown by the comparisons, the proposed algorithm significantly outperforms traditional shortest-path routing not only in terms of PSNR and frame loss rate, but it also delivers better throughput for both video traffic and background traffic.

In future, We shall explore how the inclusion of other metrics such as jitter and congestion improves the video quality. We also hope to present a comprehensive mathematical model and validate it using an exhaustive emulated environment. We shall also extend our approach to multi-domain networks containing more than one SDN controller.

VI. Acknowledgments

The work of Ankit Gangwal, Megha Gupta, Manoj Singh Gaur, and Vijay Laxmi was partially supported by Department of Electronics and Information Technology, Government of India project grant “Information Security Education and Awareness” (ISEA Phase- II) for Malaviya National Institute of Technology Jaipur as Resource Centre. Mauro Conti is supported by a Marie Curie Fellowship funded by the European Commission (agreement PCIG11-GA-2012-321980). This work is also partially supported by the EU TagItSmart! Project (agreement H2020-ICT30-2015-688061), the EU-India REACH Project (agreement ICI+/2014/342-896), the Italian MIUR-PRIN TENACE Project (agreement 20103P34XC), and by the projects “Tackling Mobile Malware with Innovative Machine Learning Techniques”, “Physical-Layer Security for Wireless Communication”, and “Content Centric Networking: Security and Privacy Issues” funded by the University of Padua.

References

[1] I. Cisco, “Cisco Visual Networking Index: Forecast and Methodology, 2014–2019,” *CISCO White paper*, 2015.

[2] T. Wiegand, G. J. Sullivan, G. Bjøntegaard, and A. Luthra, “Overview of the H. 264/AVC video coding standard,” *Circuits and Systems for Video Technology, IEEE Transactions on*, vol. 13, no. 7, pp. 560–576, 2003.

[3] H. Schwarz, D. Marpe, and T. Wiegand, "Overview of the scalable video coding extension of the H. 264/AVC standard," *Circuits and Systems for Video Technology, IEEE Transactions on*, vol. 17, no. 9, pp. 1103–1120, 2007.

[4] Y. Jarraya, T. Madi, and M. Debbabi, "A survey and a layered taxonomy of software-defined networking," *Communications Surveys & Tutorials, IEEE*, vol. 16, no. 4, pp. 1955–1980, 2014.

[5] B. Nunes, M. Mendonca, X. N. Nguyen, K. Obraczka, T. Turletti *et al.*, "A survey of software-defined networking: Past, present, and future of programmable networks," *Communications Surveys & Tutorials, IEEE*, vol. 16, no. 3, pp. 1617–1634, 2014.

[6] N. McKeown, T. Anderson, H. Balakrishnan, G. Parulkar, L. Peterson, J. Rexford, S. Shenker, and J. Turner, "OpenFlow: enabling innovation in campus networks," *ACM SIGCOMM Computer Communication Review*, vol. 38, no. 2, pp. 69–74, 2008.

[7] Q. Ma and P. Steenkiste, "Supporting dynamic inter-class resource sharing: a multi-class QoS routing algorithm," in *INFOCOM'99. Eighteenth Annual Joint Conference of the IEEE Computer and Communications Societies. Proceedings. IEEE*, vol. 2. IEEE, 1999, pp. 649–660.

[8] K. Nahrstedt and S. Chen, "Coexistence of QoS and best-effort flows-routing and scheduling," in *Proceedings of 10th IEEE Tyrrhenian International Workshop on Digital Communications: Multimedia Communications*. Citeseer, 1998.

[9] Y. M. Hsiao, S. W. Yeh, J. S. Chen, and Y. S. Chu, "A design of bandwidth adaptive multimedia gateway for scalable video coding," in *Circuits and Systems (APCCAS), 2010 IEEE Asia Pacific Conference on*. IEEE, 2010, pp. 160–163.

[10] T. Schierl, C. Hellge, S. Mirta, K. Grüneberg, and T. Wiegand, "Using H. 264/AVC-based scalable video coding (SVC) for real time streaming in wireless IP networks," in *Circuits and Systems, 2007. ISCAS 2007. IEEE International Symposium on*. IEEE, 2007, pp. 3455–3458.

[11] M. S. Seddiki, M. Shahbaz, S. Donovan, S. Grover, M. Park, N. Feamster, and Y. Q. Song, "FlowQoS: QoS for the rest of us," in *Proceedings of the third workshop on Hot topics in software defined networking*. ACM, 2014, pp. 207–208.

[12] P. Georgopoulos, M. Broadbent, B. Plattner, and N. Race, "Cache as a service: Leveraging sdn to efficiently and transparently support video-on-demand on the last mile," in *Computer Communication and Networks (ICCCN), 2014 23rd International Conference on*. IEEE, 2014, pp. 1–9.

[13] A. Kassler, L. Skorin-Kapov, O. Dobrijevic, M. Matijasevic, and P. Dely, "Towards QoE-driven multimedia service negotiation and path optimization with software defined networking," in *Software, Telecommunications and Computer Networks (SoftCOM), 2012 20th International Conference on*. IEEE, 2012, pp. 1–5.

[14] P. McDonagh, C. Olariu, A. Hava, and C. Thorpe, "Enabling IPTV service assurance using OpenFlow," in *Advanced Information Networking and Applications Workshops (WAINA), 2013 27th International Conference on*. IEEE, 2013, pp. 1456–1460.

[15] P. Georgopoulos, Y. Elkhatib, M. Broadbent, M. Mu, and N. Race, "Towards network-wide QoE fairness using openflow-assisted adaptive video streaming," in *Proceedings of the 2013 ACM SIGCOMM workshop on Future human-centric multimedia networking*. ACM, 2013, pp. 15–20.

[16] I. Unanue, I. Urteaga, R. Husemann, J. Del Ser, V. Roesler, A. Rodríguez, and P. Sánchez, "A tutorial on H. 264/SVC scalable video coding and its tradeoff between quality, coding efficiency and performance," *Recent Advances on Video Coding*, vol. 13, 2011.

[17] S. Civanlar, M. Parlakisik, A. M. Tekalp, B. Gorkemli, B. Kaytaz, and E. Onem, "A qos-enabled openflow environment for scalable video streaming," in *GLOBECOM Workshops (GC Wkshps), 2010 IEEE*. IEEE, 2010, pp. 351–356.

[18] H. E. Egilmez, B. Gorkemli, S. Civanlar *et al.*, "Scalable video streaming over OpenFlow networks: An optimization framework for QoS routing," in *Image Processing (ICIP), 2011 18th IEEE International Conference on*. IEEE, 2011, pp. 2241–2244.

[19] H. E. Egilmez, S. T. Dane, K. T. Bagci, and A. M. Tekalp, "OpenQoS: An OpenFlow controller design for multimedia delivery with end-to-end Quality of Service over Software-Defined Networks," in *Signal & Information Processing Association Annual Summit and Conference (APSIPA ASC), 2012 Asia-Pacific*. IEEE, 2012, pp. 1–8.

[20] S. Laga, T. Van Cleemput, F. Van Raemdonck, F. Vanhoutte, N. Bouten, M. Claeys, and F. De Turck, "Optimizing scalable video delivery through OpenFlow layer-based routing," in *Network Operations and Management Symposium (NOMS), 2014 IEEE*. IEEE, 2014, pp. 1–4.

[21] H. E. Egilmez and A. M. Tekalp, "Distributed QoS architectures for multimedia streaming over software defined networks," *Multimedia, IEEE Transactions on*, vol. 16, no. 6, pp. 1597–1609, 2014.

[22] T. Uzakgider, C. Cetinkaya, and M. Sayit, "Learning-based approach for layered adaptive video streaming over SDN," *Computer Networks*, vol. 92, pp. 357–368, 2015.

[23] E. W. Dijkstra, "A note on two problems in connexion with graphs," *Numerische mathematik*, vol. 1, no. 1, pp. 269–271, 1959.

[24] V. Kaibel and M. A. Peinhardt, *On the bottleneck shortest path problem*. Konrad-Zuse-Zentrum für Informationstechnik, 2006.

[25] V. N. Gourov, "Network Monitoring with Software Defined Networking: Towards OpenFlow network monitoring," Ph.D. dissertation, TU Delft, Delft University of Technology, 2013.

[26] POX. http://www.noxrepo.org/pox/about-pox/. Last accessed on Mar. 01, 2016.

[27] B. Lantz, B. Heller, and N. McKeown, "A network in a laptop: rapid prototyping for software-defined networks," in *Proceedings of the 9th ACM SIGCOMM Workshop on Hot Topics in Networks*. ACM, 2010, p. 19.

[28] Mininet. http://mininet.org/. Last accessed on Mar. 01, 2016.

[29] A. Tirumala, F. Qin, J. Dugan, J. Ferguson, and K. Gibbs, "Iperf: The TCP/UDP bandwidth measurement tool," *http://dast. nlanr. net/Projects*, 2005.

[30] Iperf. https://iperf.fr/. Last accessed on Mar. 01, 2016.

[31] P. Seeling and M. Reisslein, "Video transport evaluation with H. 264 video traces," *Communications Surveys & Tutorials, IEEE*, vol. 14, no. 4, pp. 1142–1165, 2012.

[32] Video Trace Library. http://trace.eas.asu.edu/. Last accessed on Mar. 01, 2016.

[33] Team, Joint Video, "H. 264/SVC Reference Software (JSVM 9.19) and Manual.(2011)," *Retrieved October*, vol. 8, p. 2014, 2011.

[34] PYUV. http://dsplab.diei.unipg.it/software/pyuv_raw_video_sequence_player. Last accessed on Mar. 01, 2016.

Blocking Intrusions at Border using Software Defined-Internet Exchange Point (SD-IXP)

Mauro Conti
Department of Mathematics
University of Padua, Italy
conti@math.unipd.it

Ankit Gangwal
Department of Mathematics
University of Padua, Italy
ankit.gangwal@math.unipd.it

***Abstract*—Servers in a network are typically assigned a static identity. Static assignment of identities is a cornerstone for adversaries in finding targets. Moving Target Defense (MTD) mutates the environment to increase unpredictability for an attacker. On another side, Software Defined Networks (SDN) facilitate a global view of a network through a central control point. The potential of SDN can not only make network management flexible and convenient, but it can also assist MTD to enhance attack surface obfuscation.**

In this paper, we propose an effective framework for the prevention, detection, and mitigation of flooding-based Denial of Service (DoS) attacks. Our framework includes a light-weight SDN assisted MTD strategy for network reconnaissance protection and an efficient approach for tackling DoS attacks using Software Defined-Internet Exchange Point (SD-IXP). To assess the effectiveness of the MTD strategy and DoS mitigation scheme, we set two different experiments. Our results confirm the effectiveness of our framework. With the MTD strategy in place, at maximum, barely 16% reconnaissance attempts were successful while the DoS attacks were accurately detected with false alarm rate as low as 7.1%.

***Index Terms*—Denial of Service, Software Defined Networks, Moving Target Defense, Network Security**

I. Introduction

The Internet has become an integral part of our everyday life, and it strongly affects the economy, politics, etc. While several network security issues are still open, network-based attacks such as large-scale port scan, DoS attacks are becoming increasingly powerful and frequent. In 2016, on average, 58.3% websites were targeted more than once, and 13.1% were targeted more than ten times[1]. A critical aspect of any attack tackling mechanism is to properly classify attack traffic from legitimate traffic. Misclassification of the traffic may lead to customer dissatisfaction causing a substantial impact on revenue/cost and especially, on the reputation of service providers.

The first step in almost all exploits and attacks (except zero-day vulnerabilities) is to identify vulnerabilities and weakness in the target. Investigating the attack surface may include (but not limited to) finding saturation points through network mapping, finding the next victim for worm propagation through network address scanning, or recognizing the version of software running on the target to exploit version-specific vulnerabilities. MTD has gained significant attention from both security experts and research community, as it can help in degrading the effectiveness of an attack by preventing or at least delaying the network reconnaissance.

On another side, SDN is a recently emerging networking paradigm. SDN offers a flexible network management by giving network operators a direct control over network functioning and allows them to perform a variety of actions. Networking experts believe that SDN will shape the architecture of the future Internet. Since SDN enables matching on multiple header fields, there is growing interest in applying the concepts of SDN in the wide-area network to make its management easier. An Internet Exchange Point (IXP) can be an interesting place to begin because it plays a central role in interconnecting many networks. An IXP is a network location where multiple independently operated networks, also known as Autonomous Systems (ASs), exchange internet traffic with one another. An SD-IXP is an IXP that is governed by the principles of SDN.

Contributions: In this paper, we propose an effective framework for detection and mitigation of flooding-based DoS attacks. Our framework employs an SDN assisted MTD strategy as the first line of defense, which is reinforced by a change-point detection technique for DoS detection and mitigation. In particular, the major contributions of our work are as follows:

1) We propose a framework to tackle DoS attacks using SD-IXP. To the best of our knowledge, our work is the first proposal that explores DoS mitigation using SD-IXP.
2) We implemented a light-weight MTD technique that utilizes the global-view available to the SDN controller.
3) We emulated our solution and evaluated its effectiveness using CAIDA [1] and DARPA intrusion detection evaluation dataset [2].

Organization: The remainder of this paper is organized as follows. Section II thoroughly explains MTD and its typical implementation approaches for network security followed by a summary of related works. Section III elaborates threat model. In Section IV, we give a detailed description of our framework for DoS prevention, detection, and mitigation. Section V elucidates our experimental setup and results. Finally, Section VI concludes the paper.

II. Preliminaries and Related Work

In this section, we elucidate MTD and its standard practices for network security, followed by a brief overview of the main research studies related to denial-of-service attacks in SDN.

[1] http://www.govtech.com/blogs/lohrmann-on-cybersecurity/online-denial-of-service-attacks-a-growing-concern.html

978-1-5386-3285-7/17/$31.00 ©2017 IEEE

A. Moving Target Defense

MTD believes that perfect security is unattainable. MTD intends to morph the target, so the attacker is compelled to learn the target over and over again. Such mechanisms increase the complexity and uncertainty of the system to reduce the window of opportunity for an attacker while increasing the cost of attack attempts. It enables the system to continue a safe operation even in a compromised environment [3]. MTD can be broadly classified into three categories: network-level MTD, host-level MTD, application-level MTD.

1) Network-level MTD: It focuses on changing the topology of the network to deceive the attacker at the network reconnaissance and mapping phase. It includes imitating fake listening hosts, IP hopping/mutation, extra closed/open ports, randomized port numbers, obfuscating port traffic.
2) Host-level MTD: It includes faking information about the host, its OS version/type. Broadly, it focuses on the alteration to the OS and host-level naming, resources, and configuration.
3) Application-level MTD: The primary goal of such techniques is to change the environment in which an application executes. It includes shuffling memory layout of the application, randomly changing application version and type, modifying the source code at each compilation, and/or altering the programming languages and settings to compile the source code [4].

Irrespective of the category, the major idea of any MTD strategy is to mutate the environment to prevent or delay the attacks on the system.

B. Related Work

In this section, we provide a summary of studies related to MTD and DoS attacks in SDN. For SDN, we focus only on the works proposing MTD-based solutions for DoS. Rowe et al. in [5] evaluate the security of an MTD attempt and also quantify its effectiveness based on mitigation costs. Here, the MTD approach includes IP address and memory randomization along with a heavyweight stateful machine for protocols such as DHCP. Dunlop et al. [6] propose Moving Target IPv6 Defense (MT6D) that implements MT6D tunneled packets to rotate and hide IPv6 assignments. In order to make the tunnels, MT6D requires a nonce, a secret key, and the endpoint's interface identifier, which makes it impractical in the existing networks. Yackoski et al. [7] use Linux hypervisor to furnish similar functionality. Colbaugh et al. [8] propose a Game Theory-based solution to model adversary's activity and correspondingly optimize mitigation strategy. Here, the solution assumes that an attacker always optimizes its actions for an extreme percussion, which might not always be correct.

The fundamental principles of SD-IXP are described in [9]. Conti et al. in [10] raise the concern of the possibility for an attacker to obtain critical information about an SDN network. Jafarian et al. in [11] present an OpenFlow-based mutation scheme for SDN that exploits OpenFlow capabilities to protect against network reconnaissance by changing the identity of hosts. The approach in [12] presents a similar idea that adopts random route mutation to optimally randomize the path between a pair of hosts. Kampanakis et al. in [13] focus specifically on network mapping and reconnaissance protection. MacFarland et al. [14] introduce a concept of allowing the defenders to discriminate between untrustworthy and trustworthy clients using a trusted computing base. To provide access control to legitimate clients, it relies on cryptographic MACs, pre-shared keys, or at least embedded passwords. The work presented in [15] employs a multi-controller system to solve the problem of saturation. However, the approach has several limitations. On one side, it uses random packet transmission delay to protect from scanning attacks, which in fact, affects the data transmission for legitimate users. On another side, synchronization of prolonged route tables among multiple controllers is overlooked.

Our work is different from the state-of-the-art on many dimensions: (1) to the best of our knowledge, it is the first proposal that implements DoS mitigation at SD-IXP; (2) since it functions at SD-IXP, it minimizes the operation overheads for MTD; and (3) it does not depend on any additional infrastructure such as trusted computing base.

III. Threat Model

The target of the attacker is a server, i.e., victim server, which provides services to the hosts. The victim and the attacker reside in different ASs. The attacker has no information about the topology of victim's network. But, the attacker knows the IP prefixes that the router of target's AS is announcing. None of the hosts in the entire system is aware of mutations (for mutation details, please refer to Section IV-A). The SD-IXP controller has prior information about the victim server to be protected. When the attacker launches a DoS attack, it certainly passes through SD-IXP switch before reaching victim's AS. To avoid excessive queries to DNS and thus detection, an attacker uses network scanning techniques to scan a whole range of IP addresses in the network.

IV. Proposed Approach

In this section, we present our framework for preventing, detecting, and blocking DoS attacks. Here, we elucidate the fundamental principles of our system, followed by its comprehensive implementation details.

A. SDN-based Random Host-IP Mutation for Reconnaissance Protection

We chose random host-IP mutation as the MTD strategy. The fundamental concept of random host-IP mutation is to regularly change the identity of the hosts in a network. In our system, the controller regularly assigns a fresh random IP address to every network host. A fresh IP address is allotted to a host under the following circumstances:

1) On a predefined interval of time.
2) When a host has received a predefined maximum number of connections.

With an aim to minimize the burden on the controller, we use a scheme of virtual IP addresses and real IP addresses. To reach a host within the same network, the source must use the real IP address of the destination. While to reach a host outside the network, the source must use the current virtual IP address of the destination. As shown in Figure 1, a virtual IP address is translated to the real IP address at the edge of the network. The translation occurs due to the flow-rules installed by the SD-IXP controller in the SD-IXP switch. Any request to the real IP address, from a host outside the network, is dropped at the edge. Mutating host-IP address has following considerations:

1) Preserving integrity of the network configuration,
2) Minimizing operational costs,
3) Preventing disruption of the existing connections while IP addresses are changed.

The controller keeps a mapping of the virtual IP addresses to the real IP addresses. Since the controller has a global view of the network, the mapping is always consistent and updated. The biggest advantage of such address translation scheme is that when the virtual IP address of host changes, then only the SD-IXP switch needs to be updated, which minimizes the operational overheads. At the same time, hosts need not care about the mutations.

We explain management of existing connections with the help of Figure 2. Let host H_A be a server that provides services to clients. The real IP address of H_A is $real\_IP_A$, and at time T_x a virtual IP address vIP_1 is assigned to H_A. When H_B attempts to send a request to H_A using its current virtual IP address, i.e. vIP_1, the SD-IXP switch first sends the packets to the controller to obtain necessary flow-rules. Since the packets from H_B request access to the valid virtual IP address of H_A, the controller sends proper forwarding rules to the switch, where $real\_IP_A$ replaces vIP_1. The flow entries are installed with both HARD_TIMEOUT and IDLE_TIMEOUT so that they expire and are removed from the flow-table of the switch. It also allows individual flows to persist even after a host's IP address changes.

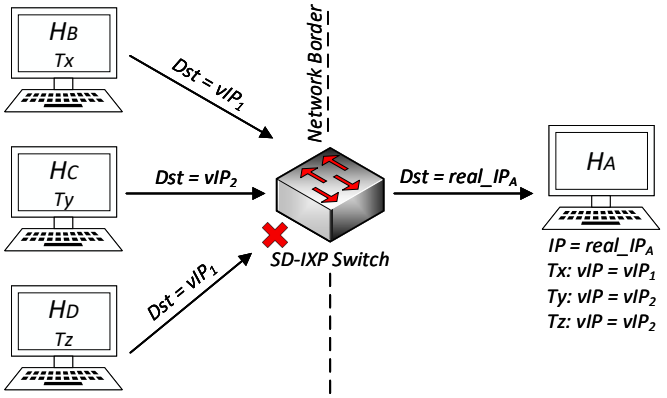


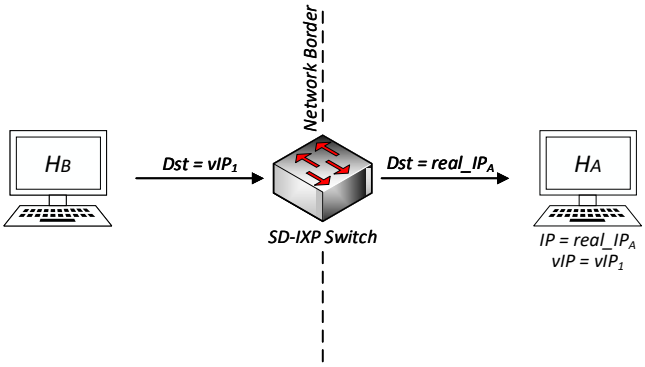
Figure 2: Connection management

Once the virtual IP address of H_A changes at time T_y, any other host will be permitted to connect with H_A only via its new virtual IP address, i.e., vIP_2, e.g., H_C in Figure 2. Importantly, if the flow-table entry in the switch between H_A and H_B that utilizes vIP_1 has not yet expired then H_B's connection would still be valid despite the virtual IP address of H_A has been changed. At this point, if a different host H_D attempts to connect with H_A via the expired virtual IP address, i.e., vIP_1, the controller instructs the switch to drop the traffic from H_D. It is important to note that if the flow-table entry between H_A and H_B has not yet expired, H_B would still be able to reach H_A using the expired vIP_1.

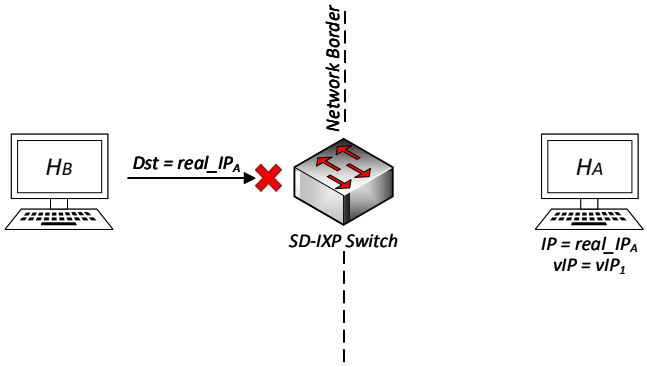
The current implementation of the mutation scheme requires the controller to frequently update Domain Name System (DNS) with the newly generated virtual IP addresses.

B. DoS Detection and Mitigation

A general characteristic of a denial of service attack is that the network observes abrupt changes in the intensity of the traffic when an attack is launched. In the event of such attacks, the statistical properties of network traffic also observe abrupt changes. Hence, the problem of attack detection can



(a) Requests to a valid virtual IP address are translated and forwarded



(b) Requests to the real IP address are denied at the edge

Figure 1: Translation of a virtual IP address to the real IP address at the edge

be formulated as a change-point detection problem [16]. The fundamental idea behind change-point detection approaches is to detect alterations in the statistical properties of the observed parameters with minimal latency and false positive rate.

Statistical Process Control (SPC) techniques have been widely utilized for controlling and monitoring the quality of manufacturing processes. SPC techniques can either be multivariate or univariate. SPC techniques can prominently detect changes in mean shifts (process mean), variance changes (process variance), counter-relationship among multiple variables [17]. Our work focuses on detecting significant changes in the network traffic intensity for DoS detection. The traffic intensity is a single variable that measures the amount of traffic flowing in the network. Hence, we consider only univariate SPC techniques to monitor mean shifts in the traffic intensity to detect possible DoS attacks. CUSUM control charts, Exponentially Weighted Moving Average (EWMA) [18] control charts, and Shewhart control charts are the typical univariate SPC techniques that are widely employed to detect mean shifts. The EMWA control charts are robust to non-normality and are almost perfectly non-parametric (distribution-free) procedures [19]. Since the normality of network traffic cannot be guaranteed, we choose EWMA control charts. A detailed description of EWMA control charts is given in the work [19]. In our work, we compute EWMA of packet arrival rate as shown in Eq. (1).

$$S_t = \begin{cases} N_t^{pk}; & \text{if } t = 1, \\ \alpha * N_t^{pk} + (1-\alpha) * S_{t-1}; & \text{otherwise,} \end{cases} \quad (1)$$

where t represents the time of current observation, $t-1$ represents the time of previous observation, S_t represents EMWA of packet arrival rate at time t, N_t^{pk} represents the number of packets arrived between $t-1$ and t. α is the smoothing factor that varies between zero and one, i.e., $0 < \alpha < 1$. A higher value of α discounts older observations faster. For an N-period moving average system, α is typically calculated as shown in Eq. (2).

$$\alpha = \frac{2}{N+1}. \quad (2)$$

The μ_S and σ_S of S_t are:

$$\mu_S = \mu_{N^{pk}}, \quad (3)$$

$$\sigma_S^2 = \sigma_{N^{pk}}^2 \cdot \left(\frac{\alpha}{2-\alpha}\right). \quad (4)$$

$\mu_{N^{pk}}$ and $\sigma_{N^{pk}}$ can be estimated by observing historical data. The Lower Control Limit (LCL) and Upper Control Limit (UCL) for the EWMA control chart are:

$$UCL_S = \mu_S + L \cdot \sigma_S, \qquad LCL_S = \mu_S - L \cdot \sigma_S. \quad (5)$$

For a 5% significance level, L = 1.96. If S_t drifts outside UCL and LCL then an anomaly is detected, and the controller overrides the forwarding rules for the violating flows with DROP entries. To summarize, the controller has following responsibilities:

1) It coordinates the mutations in the network.
2) It manages connections between hosts by installing appropriate flow-rules in the SD-IXP switch.
3) Using FLOW_STATS messages, it periodically obtains traffic statistics from the SD-IXP switch to detect and mitigate DoS attacks.

A DROP entry has only IDLE_TIMEOUT, which means that the DROP rules for the violating flows persist in the switch until such flows become idle for the specified IDLE_TIMEOUT.

V. Evaluation

In this section, we explain the details of our experiment setup followed by results and their analysis.

A. Experiment Setup

We evaluated our framework through emulation. Considering the suggestion in [20] to build our prototype as reliable and realistic as possible. Figure 3 shows the evaluated network topology. The network consists of three ASs, namely, AS1, AS2, and AS3. Each AS has one router, i.e., A1 in AS1, B1 in AS2, and C1 in AS3. Each AS connects to the SD-IXP switch through its router. The SD-IXP controller governs the SD-IXP switch. The route server is based on ExaBGP[2]. The routers run *bgpd* and *zebra* daemons, Quagga[3] routing suite. The network topology is created using MiniNext[4] emulation tool. MiniNext is an extension of Mininet[5] that allows each node in the network to execute a separate version of routing software. The IP addresses 172.0.*.* refer to the interfaces that the SD-IXP controller/route server and the routers use to connect with one another. The IP prefixes with "/24" indicate the IP prefixes that each router announces to its neighboring ASs using BGP. The target server resides in AS1, the malicious and genuine hosts reside in AS2 and AS3.

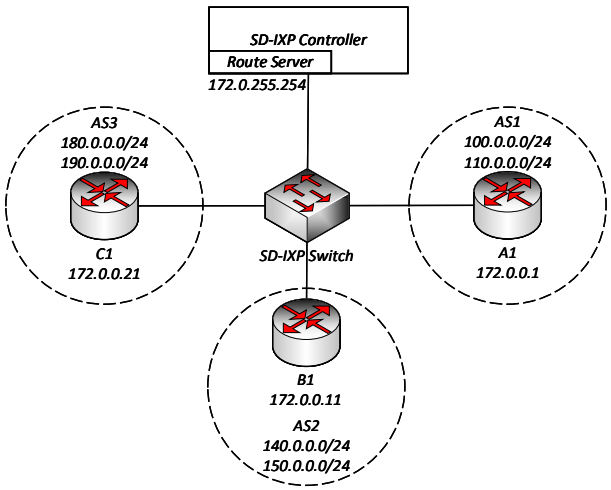


Figure 3: Evaluated network topology

[2]ExaBGP - https://github.com/Exa-Networks/exabgp/
[3]Quagga - http://www.nongnu.org/quagga/
[4]MiniNext - https://github.com/USC-NSL/miniNeXT/
[5]Mininet - http://mininet.org/

Since SDN is a recently emerging concept, no DoS attack dataset for SDN was publicly available at the time of evaluation. To assess the effectiveness of our system we orchestrated two different experiments. In the first experiment, we evaluated the mutation scheme where we used Nmap[6] to imitate an attacker's behavior. Nmap is a free and open-source tool for network mapping and security auditing. Nmap can reveal which hosts are reachable in a network, what services (their name and version) are available on those hosts, which operating system (its name and version) they are running, and several other important information.

To evaluate the versatility and efficiency of our solution for DoS detection and mitigation, we set another experiment. Here, we considered DARPA intrusion detection dataset as well as traffic traces provide by CAIDA. The DARPA dataset contains over 200 instances of more than 50 types of attacks [2]. Table I describes some of the attacks from the DARPA dataset. It is worth mentioning that these attacks are fundamentally different and work at different layers. As a representative example, "Neptune" works at the transport layer while "IPsweep" works at the network layer. On another side, "IPsweep" and "Portscan" can overload an SDN controller by generating a huge number of traffic flows. The CAIDA traffic traces help us to understand the average transmission rate for a genuine source in a network. After following the suggestions in [21], we crafted a 25% attack traffic where the attack traffic has a higher rate than the normal traffic. The network traffic was generated using Scapy[7].

Attacks	Descriptions
Neptune	A flooding of SYN packets on one or many TCP ports.
IPsweep	A surveillance sweep to identify which hosts are listening.
Portscan	A surveillance sweep to discover which services/ TCP-ports are open on the target machine.

Table I: Some attacks from DARPA dataset

B. Results and Analysis

In this section, we present and discuss the results from our experiments. With the SDN-based random host-IP mutation scheme enabled, we performed ten experiments where we ran twenty-five consecutive aggressive (OS detection, version detection, script scanning, and traceroute) Nmap scans against the target network. Figure 4 shows the percentage of correct scan reports against the actual report. It is clear that at maximum barely 16% attempts were successful.

Now we discuss the results from the second experiment. We used Detection Rate (DR) and False Alarm Rate (FAR) to assess our solution. DR measures the percentage of correctly detected attacks over all the real attacks and is computed using Eq. (6).

$$DR\ (\%) = \frac{TP}{TP + FN} * 100. \tag{6}$$

[6]Nmap - https://nmap.org/

[7]Scapy - http://www.secdev.org/projects/scapy/

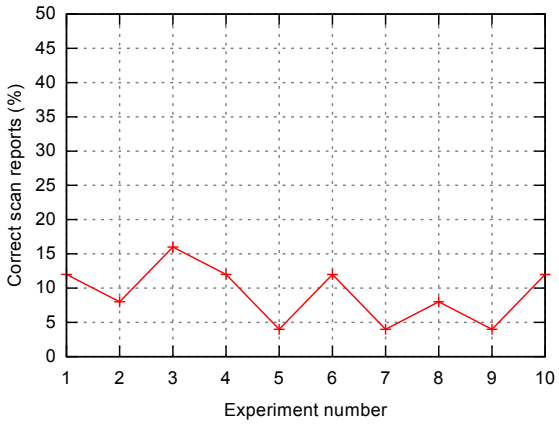


Figure 4: Correctness of consecutive Nmap scans

FAR measures the percentage of benign traffic incorrectly detected as attack over the entire benign traffic and is computed using Eq. (7).

$$FAR\ (\%) = \frac{FP}{FP + TN} * 100. \tag{7}$$

Figure 5 shows DR and FAR for various values of α. A smaller value of α gives more weightage to historical values as compared to the current observation. Hence, the attacks are misclassified leading to lower DR. On the another side, the misclassified attacks still influence the EMWA control chart values, which possibly leads to misclassification of subsequent high-intensity benign traffic. Hence, higher FAR. With the increasing value of α DR and FAR improves. Our results show that our mechanism can perfectly detect the attacks with FAR as low as 7.1%.

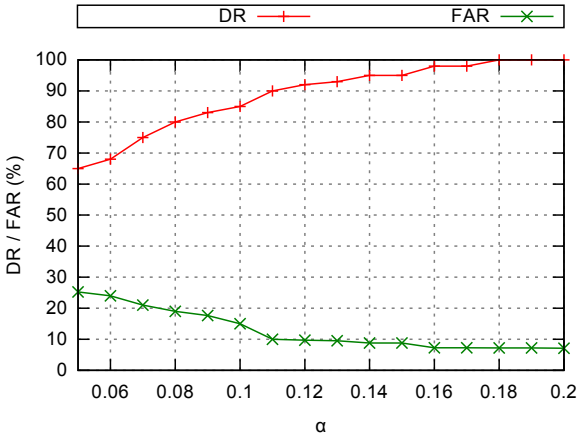


Figure 5: Influence of α on performance

Overheads: We can scrutinize the overheads of our system in terms of the size of address space required for the mutations and CPU usage. To understand the address space requirement, we define two terms M_i and U_i. M_i denotes the rate of mutation for host h_i while U_i defines the time interval during which a virtual IP address must not be reassigned to the same

host h_i. Considering M_i and U_i, the total number of virtual IP addresses required for any host h_i must be at least $\lceil \frac{U_i+M_i}{M_i} \rceil$. Hence, for a system of n hosts, the least size of address space must be $\sum_{i=1}^{n} \lceil \frac{U_i+M_i}{M_i} \rceil$. For CPU usage, we set an experiment where each host has a different M_i and U_i, and the measured CPU overhead for the controller was less than 5% on a system with Intel Core i5-7200U CPU @ 2.50GHz x 4 processor.

VI. Conclusion and Future Work

SDN provides a simple and flexible network management compared to traditional networks. Applying the concepts of SDN at IXP could make some aspects of wide-area network management easier. In this work, we have proposed an effective framework for the prevention, detection, and mitigation of DoS attacks. As shown by the results, our framework is not only effective to prevent network reconnaissance through moving target defense, but it can also efficiently detect and mitigate DoS attacks. Moreover, our framework has subtle operation overheads. In the future, we will extend our framework to detect and mitigate Distributed Denial of Service (DDoS) attacks. We hope to perform a thorough analysis of our extended framework on a physical testbed.

Acknowledgments

Mauro Conti is supported by a Marie Curie Fellowship funded by the European Commission (agreement PCIG11-GA-2012-321980). Ankit Gangwal is pursuing his Ph.D. with a fellowship for international students funded by Fondazione Cassa di Risparmio di Padova e Rovigo (CARIPARO). This work is partially supported by the EU TagItSmart! Project (agreement H2020-ICT30-2015-688061). This work is partially supported by the grant n. 2017-166478 (3696) from Cisco University Research Program Fund and Silicon Valley Community Foundation, and by the grant "Scalable IoT Management and Key Security Aspects in 5G Systems" from Intel. This work is also partially funded by the project CNR-MOST/Taiwan 2016-17 "Verifiable Data Structure Streaming".

References

[1] The CAIDA UCSD Statistical information for the CAIDA Anonymized Internet Traces, http://www.caida.org/data/passive/passive_trace_statistics.xml, 2016.

[2] MIT Lincoln Laboratory, "Intrusion detection attacks database," http://www.ll.mit.edu/ideval/docs/attackDB.html.

[3] S. Venkatesan, M. Albanese, G. Cybenko, and S. Jajodia, "A moving target defense approach to disrupting stealthy botnets," in *Proceedings of the ACM Workshop on Moving Target Defense*, 2016, pp. 37–46.

[4] L. Ge, W. Yu, D. Shen, G. Chen, K. Pham, E. Blasch, and C. Lu, "Toward effectiveness and agility of network security situational awareness using Moving Target Defense (MTD)," *SPIE*, vol. 9085, p. 90850Q, 2014. [Online]. Available: dx.doi.org/10.1117/12.2050782

[5] J. Rowe, K. N. Levitt, T. Demir, and R. Erbacher, "Artificial diversity as maneuvers in a control theoretic moving target defense," in *National Symposium on Moving Target Research*, 2012.

[6] M. Dunlop, S. Groat, R. Marchany, and J. Tront, "Implementing an IPv6 moving target defense on a live network," in *National Symposium on Moving Target Research*, 2012.

[7] J. Yackoski, H. Bullen, X. Yu, and J. Li, "Applying self-shielding dynamics to the network architecture," in *Moving Target Defense II*. Springer, 2013, pp. 97–115.

[8] R. Colbaugh and K. Glass, "Predictive moving target defense," in *National Symposium on Moving Target Research*, 2012.

[9] A. Gupta, L. Vanbever, M. Shahbaz, S. P. Donovan, B. Schlinker, N. Feamster, J. Rexford, S. Shenker, R. Clark, and E. Katz-Bassett, "SDX: A software defined internet exchange," *ACM SIGCOMM Computer Communication Review*, vol. 44, no. 4, pp. 551–562, 2015.

[10] M. Conti, F. De Gaspari, and L. V. Mancini, "Know your enemy: Stealth configuration-information gathering in SDN," in *GPC*, 2017, pp. 386–401.

[11] J. H. Jafarian, E. Al-Shaer, and Q. Duan, "OpenFlow random host mutation: Transparent moving target defense using software defined networking," in *ACM SIGCOMM Workshop on HotSDN*, 2012, pp. 127–132.

[12] J. H. Jafarian, E. AlShaer, and Q. Duan, "On the random route mutation moving target defense," in *National Symposium on Moving Target Research*, 2012.

[13] P. Kampanakis, H. Perros, and T. Beyene, "SDN-based solutions for moving target defense network protection," in *IEEE WoWMoM*, 2014, pp. 1–6.

[14] D. C. MacFarland and C. A. Shue, "The SDN shuffle: Creating a moving target defense using host-based software defined networking," in *the 2nd ACM Workshop on Moving Target Defense*, 2015, pp. 37–41.

[15] D. Ma, Z. Xu, and D. Lin, "Defending blind DDoS attack on SDN based on moving target defense," in *SecureComm*. Springer, 2014, pp. 463–480.

[16] A. G. Tartakovsky, A. S. Polunchenko, and G. Sokolov, "Efficient computer network anomaly detection by changepoint detection methods," *IEEE Journal of Selected Topics in Signal Processing*, vol. 7, no. 1, pp. 4–11, 2013.

[17] T. P. Ryan, "Statistical methods for quality improvement," *John Wiley and Sons, New York*, 1989.

[18] J. S. Hunter, "The exponentially weighted moving average," *Journal of Quality Technology*, vol. 18, no. 4, pp. 203–210, 1986.

[19] D. C. Montgomery, "Introduction to statistical quality control," *John Wiley & Sons, New York*, 1997.

[20] J. Mirkovic, S. Fahmy, P. Reiher, and R. K. Thomas, "How to test DoS defenses," in *Cybersecurity Applications & Technology Conference for Homeland Security*, 2009, pp. 103–117.

[21] J. Sommers, V. Yegneswaran, and P. Barford, "Toward comprehensive traffic generation for online IDS evaluation," *Technical Report, University of Wisconsin*, 2005.

PANORAMA: Real-time Bird's Eye View of an OpenFlow Network

Ankit Gangwal, Mauro Conti
Department of Mathematics,
University of Padua, Italy.
Email: {ankit.gangwal, conti}@math.unipd.it

Manoj Singh Gaur
Department of Computer Science & Engineering,
Malaviya National Institute of Technology, India.
Email: gaurms@mnit.ac.in

***Abstract*—Software Defined Network (SDN) is an emerging networking paradigm that has gained enormous attention from the industries as well as the research community. SDN decouples data plane and control plane. The direct programmability of the control plane allows us to develop routing algorithms, which can accommodate versatile requirements of diverse network applications. On another side, Quality-of-Service provisioning, traffic engineering, etc., require accurate traffic measurements because an up-to-date view of the network facilitates service providers to optimize network performance. Existing approaches of traffic measurements demand either additional resources or alteration to infrastructure.**

In this paper, we present a collection of lightweight mechanisms for obtaining real-time network information in SDN environment. In particular, our mechanisms aim to obtain per-flow and per-port traffic statistics, topology information, data transfer rate for each network link, etc. Since our approach exploits the built-in capabilities of OpenFlow protocol, it does not require any changes to the infrastructure. We also implemented our proposed mechanisms and developed Panorama, a graphical user interface for real-time presentation of the obtained information.

***Index Terms*—SDN, OpenFlow, Real-time, Traffic Statistics, Network Monitoring**

I. INTRODUCTION

With increasing popularity of real-time services such as voice and video, network monitoring has become a significant task in network operation. In order to provide Quality-of-Service assurance for such services and operations like traffic engineering and network security require precise information about network "health". Due to the explosion of traffic volume in IP networks, it has become exceedingly difficult to acquire accurate traffic statistics. Network monitoring has been an active area of research. Current traffic estimation techniques such as flow-based measurements require several precious resources, e.g., processing power and bandwidth while other solutions compel expensive changes to the infrastructure. These limitations of currently available solutions raise demand for an efficient network management technique that is competent to furnish an accurate, precise and real-time view of the network while being inexpensive and simple to implement. In this paper, we propose integrated mechanisms for obtaining real-time network information that includes per-flow and per-port traffic statistics, topology information, data transfer rate for each network link, etc., for OpenFlow-based SDN environment. The collected information can be presented on our developed GUI, Panorama[1].

The remainder of this paper is organized as follows. Section II presents a brief summary of related work. Section III elaborates underlying architecture and working principles of our approach. The details of the prototype implementation and verification are discussed in Section IV. Finally, Section V concludes the paper and explores the future directions related to this work.

II. RELATED WORK

In SDN [1] environment, the key idea is to separate the data and control plane of network devices. The data plane includes several forwarding devices that provide the forwarding of packets. The data plane is controlled by the control plane. The control plane consists of at least one decision-making entity called the controller, which has a global view of the network of its domain. The controller communicates with the forwarding devices, and it decides the route of data packets in the network.

The controller and the switches communicate via OpenFlow [2] protocol. It creates a secure communication channel between the controller and the switches. The controller instructs the switches by simply updating their "flow table" via the OpenFlow protocol. Apart from adding flow table entries, the controller can also delete or modify existing entries. It can also collect logs, which are stored in the form of counters, from the forwarding devices. These logs can be used to calculate various traffic statistics. Utilizing topology information and the real-time traffic statistics, the controller can optimize the route for data packets. Hence, SDN allows network operators to develop application-specific route decision strategies considering the network topology information and the obtained traffic statistics.

Network traffic measurement approaches can be classified into two categories: active and passive. In active measurement approach, traffic flows are continuously monitored by injecting special probe packets. Although an active measurement of traffic flows can fulfill on-demand requests for traffic statistics, however, it involves a significant amount of overhead that may also interfere with critical traffic flows. On another side, in passive measurement approach, the network is probed at a

978-1-5090-4429-0/17/$31.00 ©2017 IEEE

[1]Panorama is available at `spritz.math.unipd.it/projects/pm/`

predefined interval with less overhead compared to the active measurement approach.

Jose et al. [3] have proposed an active measurement mechanism for measuring large-scale traffic aggregation by matching switch rules. However, it requires continuous updates to fetch the matching rules. OpenNetMon [4] adaptively fetches data from the switches where the accuracy increases at the expense of measuring overhead. iSTAMP [5] partitions Ternary Content Addressable Memory (TCAM) for aggregate and de-aggregate traffic. However, it requires an additional mechanism to "stamp" the flows. Zhang et al. [6] have proposed a prediction based algorithm to detect anomalies. However, the approach is restricted only to identified flows. OpenTM [7] constantly polls a switch for collecting flow statistics and produces high accuracy at the cost of high overhead of polling. Payless [8] employs an adaptive algorithm for polling flow statistics with both high- and low-frequency interval. In this approach, accuracy and overhead vary with the length of the polling interval. HONE [9] deploys software agents in every host and a module that interacts with network devices to periodically collect traffic data. The major hindrance in the deployment of this approach is the requirement of installing software agents in every host leading to scalability issues. PLANCK [10] utilizes port mirroring to obtain statistical data rapidly, but in a large network scenario, the traffic could exhaust the capacity of the ports. OpenSample [11] uses sFlow [12] and TCP sequence numbers for achieving high accuracy with low latency. FlowSense [13] uses PacketIn and FlowRemoved messages for low overhead passive measurement. The techniques discussed in [3–10] are active measurement approaches while techniques in [11, 13] are passive measurement approaches.

Existing network monitoring techniques require either expensive changes to the infrastructure or heavy computational resources. In this paper, we address the demand for accurate and precise network monitoring mechanisms, which are not only inexpensive but are easy to deploy.

III. Panorama

Panorama provides a real-time bird's eye view of an OpenFlow Network. Panorama collects and depicts vital information about the network. Panorama is developed as a POX [14] module to work along with any forwarding module, as shown in Figure 1. The forwarding module provides forwarding rules while Panorama collects and streams network information. The benefit of this design approach is that Panorama can also be used as a verification tool to verify the correctness of forwarding functionality developed by OpenFlow developers. The remainder of this section elaborates various type of network information collected by Panorama and their implementation details.

A. Network Configuration

Real-time network topology update is an essential requirement for estimating the "health" of the network. Network topology updates include liveliness report of switches, links, and hosts in the network.

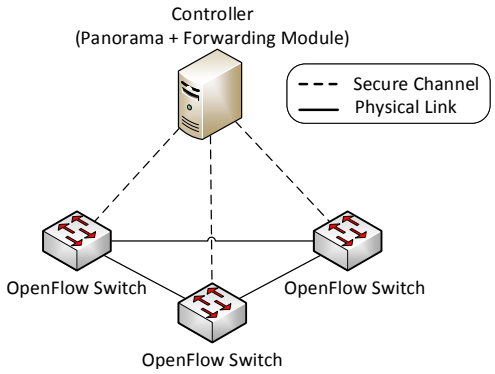


Figure 1: Co-existence of Panorama with forwarding module

1) Switch Discovery: A switch exchanges handshake messages when it connects to the controller. Upon successful connection, the controller learns that a switch with a unique identifier called "DataPath ID" (DPID) is connected to it. Information obtained for each discovered switch includes:

a Serial number.
b Software version.
c Vendor.
d Hardware type.
e DPID.

The connection between switch and controller terminates either because the switch was restarted or it has been turned-off. In such event, the controller learns that the switch has been disconnected.

2) Link Discovery: LLDP (Link Layer Discovery Protocol) is used to discover a switch-to-switch link. OpenFlow controller sends controller-specific LLDP packets to each discovered switch as PACKET_OUT messages. The controller also sends forwarding rule for such packets. The forwarding rule asks the switches to broadcast this packet with their DPID. When the broadcasted packet reaches an adjacent switch, it sends this packet to the controller asking for a forwarding rule, and the controller learns that there is an unidirectional link the two switches. The entire process is illustrated in Figure 2. Information stored for each discovered link includes:

a DPID of switches, between which a link exists.
b Port number of the switches, through which link is established.

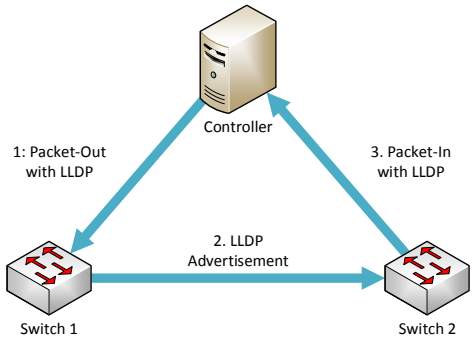


Figure 2: Link discovery in OpenFlow network

When a switch detects that a link to an adjacent switch has been removed or failed, it informs the controller by raising an event.

3) Host Discovery: When a flow of packets from a host reaches to a switch, the switch matches the packets with existing rules. For the first packet from a host, there would be no matching rule. Hence, the switch sends this packet to controller asking for a rule. By inspecting the packet, the controller learns the location of the host. Considering the facts that a host might move from one switch to another (as in the case of wireless connection) or DHCP might reassign IP addresses to the hosts, Panorama maintains a timer for each discovered hosts and a clean up is performed periodically. The duration of this period is usually greater than the expected hard timeouts because removal of a rule would force subsequent packet in the same flow from the host to flow towards the controller again. Information obtained from each discovered host includes:

a IP Address.
b MAC Address.
c Association Switch.
d Switch Port No.
e Time, which defines the time when the host was discovered.

B. Port Stats

OpenFlow PORT_STATS messages are used to obtain per-port statistics for each port on a switch. When a switch receives a PORT_STATS_REQUEST, it replies with a number of transmitted packets, transmitted bytes, transmit errors, packet dropped by TX, collisions, received packets, received bytes, receive errors, packet dropped by RX, packets with RX overrun, frame alignment errors and CRC errors for each of its port. Table I shows the main components of PORT_STAT_REPLY.

port_no	tx_packets	tx_bytes	tx_errors	tx_dropped	collisions	rx_packets	rx_bytes	rx_errors	rx_dropped	rx_over_err	rx_frame_err	rx_crc_err

Table I: Main components of port stats

C. Aggregate Stats

Aggregate statistics for a switch indicate the total number of currently installed flows, the total number of packets and the total number of bytes processed by the switch for these flows, as shown in Table II.

flow_count	packet_count	byte_count

Table II: Main components of aggregate stats

D. Flow Stats

OpenFlow FLOW_STATS messages are used to obtain per-flow statistics for every flow rule installed on a switch. A flow table consists of several flow entries, and each flow entry contains various fields as indicated in Table III.

Match Fields	Priority	Counters	Instructions	Timeouts	Cookie

Table III: Main components of a flow entry

1) **Match Fields:** These are the field against which packet headers are matched. A match field may be exact or may be wild-carded (match any value). Table IV shows the main components of the match fields.

Ingress Port	Ether Src.	Ether Dst.	Ether Type	VLAN Id	VLAN Priority	MPLS Label	MPLS Traffic Class	IP Src.	IP Dst.	IP Proto.	IP ToS	Transp. Src. Port	Transp. Dst. Port

Table IV: Main components of match fields

2) **Priority:** It is an integer number that determines the matching precedence of a flow entry. The higher the number, the higher the priority.
3) **Counters:** Counters are updated when packets are matched. Counters are maintained for each port, flow entry, flow table, etc.
4) **Instructions:** Instructions are executed when a packet matches a flow entry. Instruction may be "Required Instruction" or "Optional Instruction". Instructions either modify pipeline processing or contain a set of actions to be performed for the match.
5) **Timeouts:** A flow entry may have two type of timeouts: hard and idle. Timeouts specify either the maximum amount of time or an idle time before a switch evicts a flow.
6) **Cookies:** Controller usages cookies to filter flow modification, statistics, and deletion.

An OpenFlow switch should essentially have at least one flow table, and it can optionally have more than one flow tables as well. The advantage is that many network deployment demand orthogonal processing of packets (e.g., QoS, ACL, and routing). Using a single flow table to implement all of those processing requirements can create a massive rule-set in the flow table. Matching rules in a single large flow table can be time-consuming. Using multiple tables may properly decouple those processing requirements. Packet processing through multiple flow tables is called the OpenFlow pipeline processing. Panorama reports pipeline processing in the instructions field.

E. Link Data Transfer Rate

One of the important aspects of network monitoring is to estimate the data transfer rate (DTR) of the network links. Panorama calculates uni-directional and bi-directional DTR of a link between a pair of switches using the port statistics received from the switches, as shown in Figure 3. Calculation of uni-directional and bi-directional DTR in bytes per second (Bps) is shown in Eq. 1 and Eq. 2 respectively.

$$DTR^{Uni}_{S1 \rightarrow S2} = \frac{(Cur^{S1}_{tx\_bytes} - Pre^{S1}_{tx\_bytes})}{(Cur^{S1}_{time} - Pre^{S1}_{time})}. \quad (1)$$

Where $Cur^{S}_{tx\_bytes}$ denotes the number of transmitted bytes in the current observation at switch S, and $Pre^{S}_{tx\_bytes}$ refers

to the number of transmitted bytes in the previous observation at switch S. While Cur^{S}_{time} and Pre^{S}_{time} denotes the current time and previous time of observation at switch S.

$$DTR^{Bi}_{S1 \leftrightarrow S2} = \frac{(Cur^{S1}_{(tx+rx)\_bytes} - Pre^{S1}_{(tx+rx)\_bytes})}{(Cur^{S1}_{time} - Pre^{S1}_{time})}. \quad (2)$$

Where $Cur^{S}_{(tx+rx)\_bytes}$ is the total number of transmitted and received bytes in the current observation at switch S, and $Pre^{S}_{(tx+rx)\_bytes}$ is the total number of transmitted and received bytes in the previous observation at switch S.

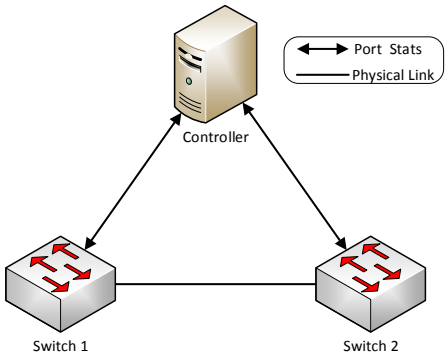


Figure 3: Measuring data transfer rate

F. Link Delay

Delay introduced by a link and link loss play a crucial role in Quality-of-Service provisioning. The work in [15] demonstrates mechanisms to measure link delay and loss. The controller injects probe packets to calculate delay on the links, as shown in Figure 4. At $t0$, it sends UDP packet to switch 1. The controller also installs a new rule instructing switch 1 to send this probe packet to switch 2. At $t1$ the packet flows from switch 1 to switch 2. Since a matching rule is not installed in switch 2, the switch sends the packet to the controller at $t2$. The controller receives this packet at $t3$. Since the controller maintains a constant connection with every switch, hence, it can estimate the delay between itself and the switches, as shown in Eq. 3. This enables the controller to estimate the delay between every pair of switches, as shown in Eq. 4.

$$t1 = t3 = 0.5 * (t_b - t_a). \quad (3)$$

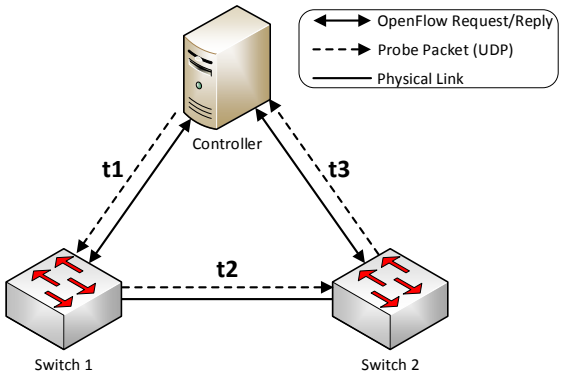


Figure 4: Measuring link delay

$$Time_{total} = t1 + t2 + t3 \Rightarrow t2 = Time_{total} - t1 - t3. \quad (4)$$

Where t_a is the time when an OpenFlow request message is sent, and t_b is the time when an OpenFlow reply message is received.

G. Link Loss

When a flow arrives at a switch and a matching rule does not exist in the switch, the first packet of the flow is sent towards the controller. The controller installs corresponding instruction in the flow table of the switches comprising the path chosen by the controller. As soon as the flow expires, the switch indicates this event to the controller. The entire process is illustrated in Figure 5. The flow is installed at time $t1$ using FLOW_MOD messages. After the flow rule expires at time $t2$ and $t3$, the controller receives FLOW_REMOVED messages from switch 1 and switch 2 respectively. Those messages include specific statistics for the flow such as the number of packets, bytes. The packet-loss rate ($Loss_{\%}$) can be measured on the basis of those statistics, as shown in Eq. 5.

$$Loss_{\%} = (1 - packets_{Switch2}/packets_{Switch1}) * 100. \quad (5)$$

Where $packets_{Switch1}$ is the number of packets transmitted from switch 1, and $packets_{Switch2}$ is the number of packets received at switch 2.

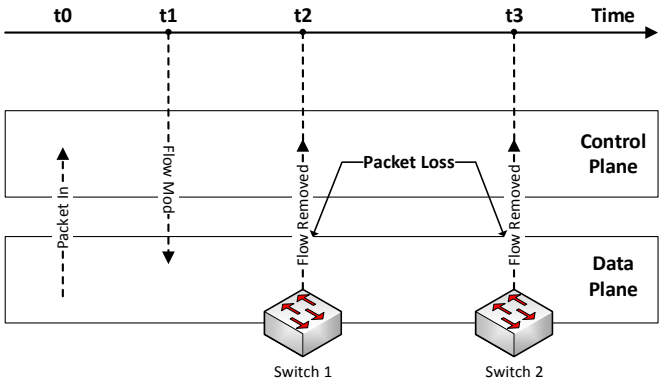


Figure 5: Measuring link loss

IV. Experiment and Verification

Panorama is tested on a physical testbed as well as in an emulation environment. The physical testbed comprises of an HP Aruba 5406R zl2 switch with Panorama running on a PC with 4th Gen Intel Core i3 processor with Intel HD Graphics, 4 GB of RAM, 500 GB HDD. Here, the physical switch is connected to two PC host.The emulation was done using Mininet[2] as the network emulator on a laptop with AMD A4-5000 1.5 GHz Quad Core APU with Radeon HD Graphics, 4 GB of RAM, 500 GB HDD. Here, Open vSwitch[3] serves as an OpenFlow-enabled switch. The emulated network topology is shown in Figure 6, here every edge switch is connected to three hosts. Iperf[4] is used to generate data traffic. To verify port, aggregate, and flow statistics `dpctl`[5] utility is used.

[2]Mininet - `http://mininet.org/`
[3]Open vSwitch - `http://openvswitch.org/`
[4]Iperf - `https://iperf.fr/`
[5]dpctl - `https://github.com/CPqD/ofsoftswitch13/wiki/`

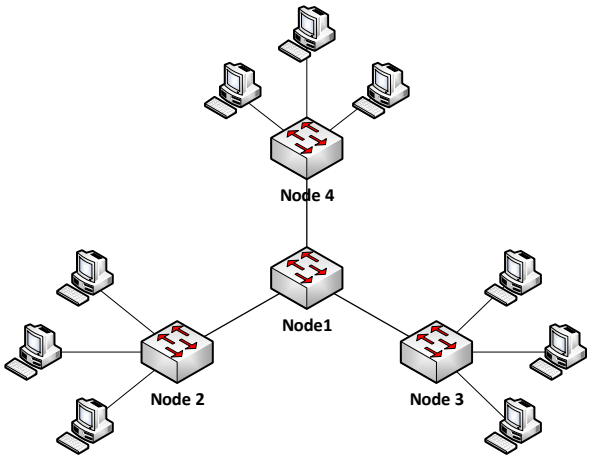


Figure 6: Evaluated network topology

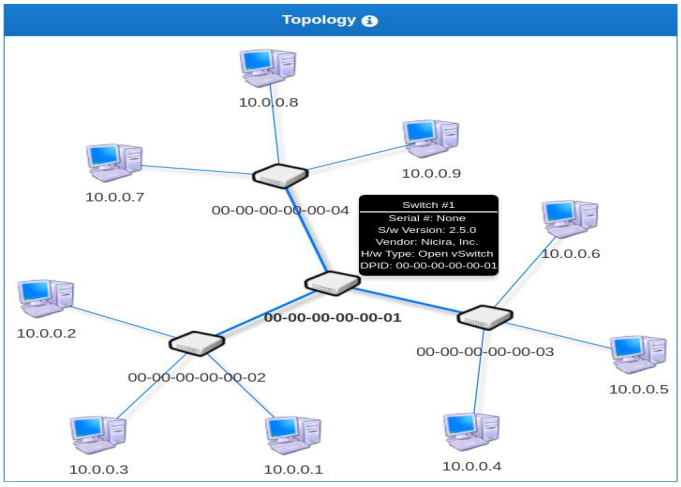


Figure 7: Discovered network topology

Figure 7 depicts the discovered network topology, Panorama labels each switch with its DPID and each host with its IP address. A comparison between Figure 6 and Figure 7 verifies the correctness of the discovered network topology. Due to space limitation, subsequent results are shown for only Node 1, which is referred as "s1" in the emulation. Figure 9 shows the computed port, aggregate, and flow statistics. The results can be verified by comparing Figure 9 with `dpctl` output for switch "s1", shown in Figure 8. Figure 10 illustrates the estimated data transfer rate of each link when each host pings every other host once.

Our proposed approach is lightweight as it relies on regularly exchanged OpenFlow messages. At the same time, it is inexpensive as it requires neither alteration to infrastructure nor injection of probe packets. On another side, it is also easy to deploy since it is a controller based measurement scheme.

```
mininet> dpctl dump-ports
*** s1 ------------------------------------------------------------------------
OFPST_PORT reply (xid=0x2): 4 ports
  port LOCAL: rx pkts=0, bytes=0, drop=125, errs=0, frame=0, over=0, crc=0
           tx pkts=0, bytes=0, drop=0, errs=0, coll=0
  port  1: rx pkts=89, bytes=9071, drop=0, errs=0, frame=0, over=0, crc=0
           tx pkts=154, bytes=17196, drop=0, errs=0, coll=0
  port  2: rx pkts=87, bytes=8940, drop=0, errs=0, frame=0, over=0, crc=0
           tx pkts=153, bytes=16880, drop=0, errs=0, coll=0
  port  3: rx pkts=92, bytes=9317, drop=0, errs=0, frame=0, over=0, crc=0
           tx pkts=154, bytes=17553, drop=0, errs=0, coll=0
```

(a) Port statistics

```
mininet> dpctl dump-aggregate
*** s1 ------------------------------------------------------------------------
NXST_AGGREGATE reply (xid=0x4): packet_count=166 byte_count=6806 flow_count=1
```

(b) Aggregate statistics

```
mininet> dpctl dump-flows
*** s1 ------------------------------------------------------------------------
NXST_FLOW reply (xid=0x4):
 cookie=0x0, duration=79.706s, table=0, n_packets=43, n_bytes=1763, idle_age=2,
priority=65000,dl_dst=01:23:20:00:00:01,dl_type=0x88cc actions=CONTROLLER:65535
```

(c) Flow statistics

Figure 8: `dpctl` output for switch "s1"

4 — 00-00-00-00-00-01

port_no	tx_packets	tx_bytes	tx_errors	tx_dropped	collisions	rx_packets	rx_bytes	rx_errors	rx_dropped	rx_over_err	rx_frame_err	rx_crc_err
65534	0	0	0	0	0	0	0	0	125	0	0	0
1	154	17196	0	0	0	89	9071	0	0	0	0	0
2	153	16880	0	0	0	87	8940	0	0	0	0	0
3	154	17553	0	0	0	92	9317	0	0	0	0	0

(a) Port statistics

1 — 00-00-00-00-00-01

flow_count	packet_count	byte_count
1	166	6806

(b) Aggregate statistics

4 — 00-00-00-00-00-01

match											actions	priority	timeout		duration		count		table_id	cookie
dl_src	dl_dst	dl_type	dl_vlan	nw_src	nw_dst	nw_proto	nw_tos	tp_src	tp_dst	in_port			idle_timeout	hard_timeout	duration_sec	duration_nsec	packet_count	byte_count		
*	01:23:20:00:00:01	LLDP	*	*	*	*	*	*	*	*	max_len : 65535 port : OFPP_CONTROLLER type : OFPAT_OUTPUT	65000	0	0	79	5.32e+8	43	1763	0	0

(c) Flow statistics

Figure 9: Computed statistics for switch "s1"

Uni-Directional (bps)				
Switch	00-00-00-00-00-01	00-00-00-00-00-02	00-00-00-00-00-03	00-00-00-00-00-04
00-00-00-00-00-01	N/A	8544.46	14003.40	12449.31
00-00-00-00-00-02	8600.66	N/A	N/A	N/A
00-00-00-00-00-03	12767.29	N/A	N/A	N/A
00-00-00-00-00-04	13469.73	N/A	N/A	N/A

(a) Uni-directional DTR

Bi-Directional (bps)				
Switch	00-00-00-00-00-01	00-00-00-00-00-02	00-00-00-00-00-03	00-00-00-00-00-04
00-00-00-00-00-01	N/A	17084.75	27222.60	25670.87
00-00-00-00-00-02	17197.32	N/A	N/A	N/A
00-00-00-00-00-03	25528.73	N/A	N/A	N/A
00-00-00-00-00-04	26140.47	N/A	N/A	N/A

(b) Bi-directional DTR

Figure 10: Estimated data transfer rate in response to `pingall` command

V. Conclusion and Future Work

In this paper, we have presented light-weight network monitoring mechanisms for SDN environment. These mechanisms employ the built-in capabilities of OpenFlow protocol. The obtained information can be presented in real-time on our developed GUI, Panorama. In future, we shall explore how other metrics such as congestion, jitter can be calculated in an OpenFlow-based SDN environment. We would also like to extend the GUI to manifest link loss and delay. We also hope to develop Panorama as a controller independent module.

VI. Acknowledgments

Mauro Conti is supported by a Marie Curie Fellowship funded by the European Commission (agreement PCIG11-GA-2012-321980). This work is also partially supported by the EU TagItSmart! Project (agreement H2020-ICT30-2015-688061), the EU-India REACH Project (agreement ICI+/2014/342-896). This work is partially supported by the grant n. 2017-166478 (3696) from Cisco University Research Program Fund and Silicon Valley Community Foundation.

References

[1] Y. Jarraya, T. Madi, and M. Debbabi, "A survey and a layered taxonomy of software-defined networking," *Communications Surveys & Tutorials, IEEE*, vol. 16, no. 4, pp. 1955–1980, 2014.

[2] N. McKeown, T. Anderson, H. Balakrishnan, G. Parulkar, L. Peterson, J. Rexford, S. Shenker, and J. Turner, "OpenFlow: enabling innovation in campus networks," *ACM SIGCOMM Computer Communication Review*, vol. 38, no. 2, pp. 69–74, 2008.

[3] L. Jose, M. Yu, and J. Rexford, "Online Measurement of Large Traffic Aggregates on Commodity Switches," in *Hot-ICE*, 2011.

[4] N. L. Van Adrichem, C. Doerr, and F. A. Kuipers, "Opennetmon: Network monitoring in openflow software-defined networks," in *IEEE NOMS*, 2014, pp. 1–8.

[5] M. Malboubi, L. Wang, C. Chuah, and P. Sharma, "Intelligent sdn based traffic (de) aggregation and measurement paradigm (istamp)," in *IEEE INFOCOM Conference on Computer Communications*, 2014, pp. 934–942.

[6] Y. Zhang, "An adaptive flow counting method for anomaly detection in SDN," in *Proceedings of the ninth ACM conference on Emerging networking experiments and technologies*. ACM, 2013, pp. 25–30.

[7] A. Tootoonchian, M. Ghobadi, and Y. Ganjali, "OpenTM: traffic matrix estimator for OpenFlow networks," in *International Conference on Passive and Active Network Measurement*. Springer, 2010, pp. 201–210.

[8] S. R. Chowdhury, M. F. Bari, R. Ahmed, and R. Boutaba, "Payless: A low cost network monitoring framework for software defined networks," in *IEEE NOMS*, 2014, pp. 1–9.

[9] P. Sun, M. Yu, M. J. Freedman, J. Rexford, and D. Walker, "Hone: Joint host-network traffic management in software-defined networks," *Journal of Network and Systems Management*, vol. 23, no. 2, pp. 374–399, 2015.

[10] J. Rasley, B. Stephens, C. Dixon, E. Rozner, W. Felter, K. Agarwal, J. Carter, and R. Fonseca, "Planck: Millisecond-scale monitoring and control for commodity networks," *ACM SIGCOMM Computer Communication Review*, vol. 44, no. 4, pp. 407–418, 2015.

[11] C. Dixon, W. Felter, and J. Carter, "OpenSample: A Low-latency, Sampling-based Measurement Platform for SDN," *IBM Research Division*, 2014.

[12] sFlow. http://www.sflow.org/.

[13] C. Yu, C. Lumezanu, Y. Zhang, V. Singh, G. Jiang, and H. V. Madhyastha, "Flowsense: Monitoring network utilization with zero measurement cost," in *International Conference on Passive and Active Network Measurement*. Springer, 2013, pp. 31–41.

[14] POX. http://github.com/noxrepo/pox/.

[15] V. N. Gourov, "Network Monitoring with Software Defined Networking: Towards OpenFlow network monitoring," Ph.D. dissertation, TU Delft, Delft University of Technology, 2013.

An Analysis of the Skype Peer-to-Peer Internet Telephony Protocol

Salman A. Baset and Henning G. Schulzrinne
Department of Computer Science
Columbia University, New York NY 10027
{salman,hgs}@cs.columbia.edu

Abstract—Skype is a peer-to-peer VoIP client developed in 2003 by the organization that created Kazaa. Skype claims that it can work almost seamlessly across NATs and firewalls and has better voice quality than other VoIP clients. It encrypts calls end-to-end, and stores user information in a decentralized fashion. Skype also supports instant messaging and conferencing.
This paper analyzes key Skype functions such as login, NAT and firewall traversal, call establishment, media transfer, codecs, and conferencing under three different network setups. Analysis is performed by careful study of the Skype network traffic and by intercepting the shared library and system calls of Skype. We draw a map of super nodes to which Skype establishes a TCP connection at login.

I. INTRODUCTION

Skype [1] is a peer-to-peer (p2p) VoIP client developed by the organization that created Kazaa [2]. Skype allows its users to place voice calls and send text messages to other users of Skype clients. In essence, it is very similar to the MSN and Yahoo IM applications, as it has capabilities for voice-calls, instant messaging, audio conferencing, and buddy lists. However, the underlying protocols and techniques it employs are quite different.

Like its file sharing predecessor Kazaa, Skype uses an overlay peer-to-peer network. There are two types of nodes in this overlay network, ordinary hosts and super nodes (SN). An ordinary host is a Skype application that can be used to place voice calls and send text messages. A super node is an ordinary host's end-point on the Skype network. Any node with a public IP address having sufficient CPU, memory, and network bandwidth is a candidate to become a super node. An ordinary host must connect to a super node and must authenticate itself with the Skype login server. Although not a Skype node itself, the Skype login server is an important entity in the Skype network as user names and passwords are stored at the login server. This server ensures that Skype login names are unique across the Skype name space. Starting with Skype version 1.2, the buddy list is also stored on the login server. Figure 1 illustrates the relationship between ordinary hosts, super nodes and the login server.

Apart from the login server, there are SkypeOut [3] and SkypeIn [4] servers which provide PC-to-PSTN and PSTN-to-PC bridging. SkypeOut and SkypeIn servers do not play a role in PC-to-PC call establishment and hence we do not consider them to be a part of the Skype peer-to-peer network. Thus, we consider the login server to be the only central component in the Skype p2p network. Online and offline user information is stored and propagated in a decentralized fashion.

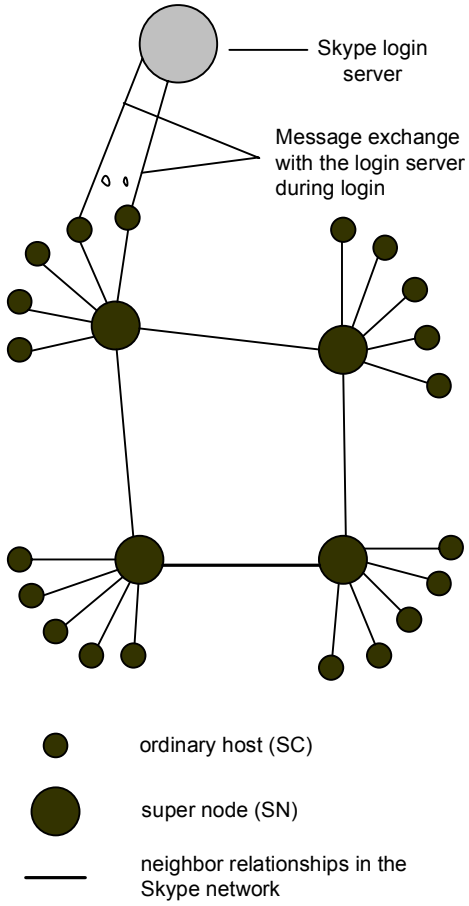


Figure 1. Skype Network. There are three main entities: supernodes, ordinary nodes, and the login server.

We believe that each Skype node uses a variant of the STUN [5] protocol to determine the type of NAT and firewall it is behind. We also believe that there is no global NAT and firewall traversal server because if there was one, the Skype node would have exchanged traffic with it during the login and call establishment phases in the many experiments we performed.

The Skype network is an overlay network and thus each Skype client (SC) needs to build and refresh a table of reachable nodes. In Skype, this table is called host cache (HC) and it contains IP address and port number of super nodes. Starting with Skype v1.0, the HC is stored in an XML file.

Skype claims to have implemented a '3G P2P' or 'Global Index' [6] technology, which is guaranteed to find a user if that user has logged in the Skype network in the last 72 hours.

Skype uses wideband codecs which allows it to maintain reasonable call quality at an available bandwidth of 32 kb/s. It uses TCP for signaling, and both UDP and TCP for transporting media traffic.

The rest of this paper is organized as follows. Section II describes key components of the Skype software and the Skype network. Section III describes the experimental setup we used for reverse-engineering the Skype protocol. Section IV discusses key Skype functions like startup, login, user search, call establishment, media transfer and codecs, and presence timers. Flow diagrams based on actual network traffic have been included to elaborate on the details. Section V discusses conferencing. Section VI discusses other experiments and compares aspects of Skype with Yahoo, MSN and Google Talk IM applications. A world map of SNs to which a SC establishes a TCP connection at login is also drawn.

II. Key Components of the Skype Software

A Skype client listens on particular ports for incoming calls, maintains a table of other Skype nodes called a host cache, uses wideband codecs, maintains a buddy list, encrypts messages end-to-end, and determines if it is behind a NAT or a firewall. This section discusses these components and functionalities in detail.

A. Ports

A Skype client (SC) opens a TCP and a UDP listening port at the port number configured in its connection dialog box. SC randomly chooses the port number upon installation. In addition, SC also opens TCP listening ports at port number 80 and 443 which, otherwise, are used to listen for incoming HTTP and HTTP-over-TLS requests. Unlike many Internet protocols like SIP [9] and HTTP [10], there is no default TCP or UDP listening port. Figure 14 shows a snapshot of the Skype (v1.4) connection dialog box. This figure shows the ports on which a SC listens for incoming connections.

B. Host Cache

The host cache (HC) is a list of super node IP address and port pairs that SC builds and refreshes regularly. It is a critical part to the Skype operation. In SC v0.97, at least one valid entry must be present in the HC. A valid entry is an IP address and port number of an online Skype node. At login time, a SC v0.97 tried to establish a TCP connection and exchange information with any HC entry. If it was unable to do so, it reported a login failure. In Skype v1.2 and onwards, if a SC is unable to establish a TCP connection with any HC entry, it tries to establish a TCP connection and exchange information with one of the seven bootstrap IP address and port pairs hard-coded in the Skype executable. A SC for Windows XP stores the host cache as a XML file 'shared.xml' in C:\Documents and Settings\<XP User>\Application Data\Skype. A SC for Linux stores the HC as a XML file 'shared.xml' at $(HOMEDIR)/.Skype. After running a SC for two days, we observed that HC contained a maximum of 200 entries. Host and peer caches are not new to Skype. Chord [20], another peer-to-peer protocol, has a finger table, which it uses to quickly find a node.

C. Codecs

During our experiments, we observed that Skype uses the iLBC [12], iSAC [13], and iPCM [14] codecs. These codecs have been developed by GlobalIPSound [15]. For SC v1.4 we measured that the Skype codecs allow frequencies between 50-8,000 Hz to pass through. This frequency range is the characteristic of a wideband codec.

D. Buddy List[1]

In Windows XP, Skype stores its buddy information in an XML file 'config.xml' at C:\Documents and Settings\<XP user>\Application Data\Skype\<skype user id>. In Linux, Skype stores the 'config.xml' file in $(HOMEDIR)/.Skype/<skype user id>. Starting with Skype v1.2 for Windows XP, the buddy list is also stored on a central Skype server whose IP address is 212.72.49.142. The buddy list is stored unencrypted on a computer. Figure 2 shows a fragment of the config.xml file.

```
<CentralStorage>
  <LastBackoff>0</LastBackoff>
  <LastFailure>0</LastFailure>
  <LastSync>1135714076</LastSync>
  <NeedSync>0</NeedSync>
  <SyncSet>
    <u>
    <skypebuddy1>2f1b8360:2</skypebuddy1>
    <skypebuddy2>d0450f12:2</skypebuddy2
```

Figure 2. A fragment of the config.xml file for a SC. It shows two Skype buddies and an four-byte number for each buddy. If two SCs have the same buddy, their corresponding config.xml files have a different four-byte number for the same buddy.

E. Encryption

The Skype website [18] explains: "Skype uses AES (Advanced Encryption Standard), also known as Rijndael, which is used by U.S. Government organizations to protect sensitive, information. Skype uses 256-bit encryption, which has a total of 1.1×10^{77} possible keys, in order to actively encrypt the data in each Skype call or instant message. Skype uses 1024 bit RSA to negotiate symmetric AES keys. User public keys are certified by the Skype server at login using 1536 or 2048-bit RSA certificates."

F. NAT and Firewall

We conjecture that SC uses a variation of the STUN [5] and TURN [19] protocols to determine the type of NAT and firewall it is behind. We also conjecture that SC refreshes this information periodically. This information is also stored in the shared.xml file.

Unlike its file sharing counter part Kazaa, a Skype client cannot prevent itself from becoming a super node.

III. Experimental Setup

Experiments were performed for the Windows Skype version 1.4.0.84 and for the Linux Skype version 1.2.0.18. We used traffic analysis, shared library and system call interception techniques to analyze various aspects of the Skype protocol. Tools like memgrp [21] can be used to perform a runtime analysis of the Skype memory. We have used this tool sparingly as it requires an extensive effort and trial and error to 'decipher' the memory dumps. Therefore, we do not present any results from using that tool. Tools by MaxMind [26] were

[1] Buddy list is an AOL trademark.

used to perform reverse country, city, and ISP lookups for an IP address when dig failed to return a DNS PTR record.

Below, we explain the experimental setup for experiments performed on different versions of the Skype client.

A. *Skype version 1.4.0.84.*

This version was available for Windows. Traffic analysis was the primary mechanism for experiments performed for this version. A SC was installed on two Windows XP machines. Each machine had a 3 GHz Pentium 4 CPU with 1 GB of RAM. Each machine had a 10/100 Mb/s Ethernet card and was connected to a 100 Mb/s network.

We performed experiments under three different network setups. In the first setup, both Skype users were on machines with public IP addresses; in the second setup, one Skype user was behind a port-restricted[1] NAT; in the third setup, both Skype users were behind a port-restricted NAT and UDP-restricted firewall. The NAT and firewall machines ran Mandriva Linux 10.2 and were connected to 100 Mb/s Ethernet network. The NAT was configured using Linux 'iptables'.

Ethereal [7] and NetPeeker [8] were used to monitor and control network traffic, respectively. NetPeeker was used to tune the bandwidth so as to analyze the Skype operation under network congestion.

B. *Skype version 1.2.0.18*

This version is available for Linux. We used shared library and system call redirection techniques to gain more insights into the Skype protocol. In Linux, at program startup, dynamic linking allows to load a shared library pointed by LD_PRELOAD environment variable before any other shared library. This makes it possible to overload a library function such as strcpy() or send(). When LD_PRELOAD is set to a library containing an overloaded strcpy() function, and the program which contains strcpy() calls is executed, the overloaded strcpy() is called. The parameters passed to this overloaded strcpy() function can be displayed or any appropriate action can be taken. Also, the overloaded strcpy() function can then call the libc strcpy() function. Austin Godber [22] provides a nice tutorial on this technique and Linux function interception.

In our experiments, we exported the display of two Linux machines using X-Win32 [23]. Thus, we were able to run different instances of a Skype client on the same host machine. However, the sound device cannot be accessed when the display is exported. To overcome this problem, we overrode the open(), close(), select(), and ioctl() calls using the technique described above. Each of these calls called the namesake libc function from within. In Skype, the socket and sound descriptors are polled by a select() system call. When Skype requests to open a sound device, our overloaded open() system call returns a fake descriptor. Skype then requests this descriptor to be polled by select(); however since this is a fake descriptor, we must not pass this descriptor to the actual select() system call. Therefore, our overloaded select() clears this fake descriptor from the read descriptor list before calling the actual select() function.

[1] A port-restricted NAT allows an external host, with source IP address X and source port P, to send a packet to the internal host only if the internal host had previously sent a packet to IP address X and port P.

An actual sound device (microphone) will have periodic data to read after it is open. However, since ours is a dummy sound device, select() will not return periodically on this device. To solve this issue, we created a select() timer in the actual select() system call with an interval of 20 ms. When the select() returns on a timer event, we add to the select() read descriptor list which is passed to the overloaded select() the fake sound device descriptor. Skype then issues a read() on this fake descriptor. Since read() is overloaded, our read() function is called. The overloaded read() then returns a dummy sound buffer to the Skype. We observed that Skype requested to read 960 bytes from the sound device on each read request.

All experiments were performed between November and December, 2005.

In the subsequent sections, any reference to function overloading in experiments implies that Linux Skype version 1.2.0.18 was used. Otherwise, Windows Skype version 1.4.0.84 was under test.

IV. SKYPE FUNCTIONS

Skype functions can be classified into startup, login, user search, call establishment and tear down, media transfer, and presence messages. This section discusses each of them in detail.

A. *Startup*

When SC v1.4 was run for the first time after installation, it sent a HTTP 1.1 GET request to the Skype server (skype.com). The first line of this request contained the keyword 'installed'.

The complete startup messages for Skype v0.97 are reported in the technical report [11].

B. *Login*

Login is perhaps the most critical function to the Skype operation. It is during this process a SC authenticates its user name and password with the login server, advertises its presence to other peers and its buddies, determines the type of NAT and firewall it is behind, discovers online Skype nodes with public IP addresses, and checks the availability of latest Skype version.

1) *Login Process*

Using the library function call overloading technique described in section III.B, we overrode the connect(), and sendto() calls such that these calls always returned with a failure. However, we permitted a TCP connection to localhost since Skype refuses to run if cannot establish this connection. The system time was printed whenever the connect() and sendto() functions were called to accurately profile the time at which Skype sends its login messages. Also, before running the Skype we deleted the HC XML file. Then we ran the SC, and made a login attempt. We observed that the SC first sent a UDP packet of length 18 bytes to each of the seven bootstrap SN IP address and port 33033. If there was no response after five seconds, SC tried to establish a TCP connection with each of these seven default SNs IP address on port 33033. If the connection attempts failed, it repeated the whole process after six seconds. We ran this experiment for 15 minutes, and strangely Skype never reported a login failure. Figure 3 shows these login attempts as a flow chart.

In the same experiment conducted in July 2005 for Skype Linux v1.0, we had observed that Skype tried to establish a

connection with each of the SN IP address on port 80 and port 443. Most firewalls are configured to allow outgoing TCP traffic to port 80 (HTTP port) and port 443 (HTTP-over-TLS port). However, we did not observe such attempts for Skype Linux v1.2.

Since the HC file had been deleted, and since we saw the same bootstrap IP address and port pairs in subsequent failed login attempts, we conclude that these IP address and port pairs are hard-coded in the Skype executable.

We have observed that a SC must establish a TCP connection with a SN in order to connect to the Skype network. If it cannot connect to a super node, it will report a login failure.

In another experiment, we filled the SC HC with an invalid IP address and port pair. Initially, SC was unable to establish a TCP connection with this invalid entry; however, after some time, it established a TCP connection with one of the bootstrap SNs. Since IP address and port number of any bootstrap SN was not present in the HC, it gives more credence to our belief that some SN IP address and port number pairs are hard-coded in the Skype executable.

In order to see the minimal set of messages a SC exchanges with other entities for a successful login, we performed the following experiment. We deleted the HC and permitted inbound and outbound UDP and TCP traffic. A SC was started and a login attempt was made. The login attempt succeeded. We then repeated this experiment for the same Skype user id two more times. Figure 4 shows the set of messages exchanged between SC, bootstrap SN, SN, and the login server in a condensed form.

In these experiments we observed that the first and the

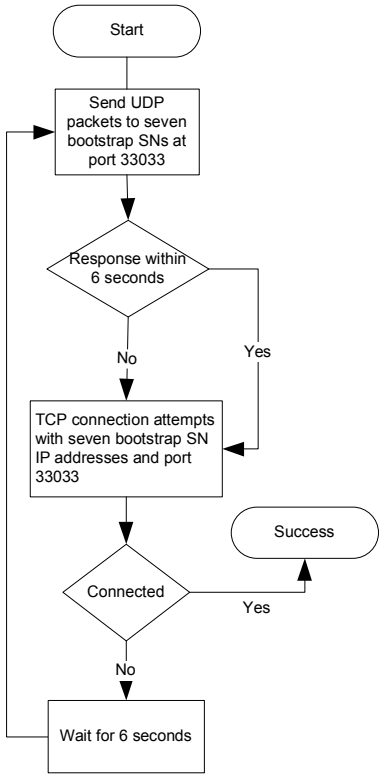


Figure 3. Skype login process. SC sends UDP packets of length 18 bytes to all bootstrap SNs. After 5s, it attempts TCP connections with the seven bootstrap SN IP address and ports 33033. Authentication with the login server is not shown.

second messages exchanged with the login server were always the same across multiple login attempts even for different skype user ids. The decimal representation of message (1) is 22 3 1 0 0 and decimal representation of message (2) is 23 3 1 0 0. In most of our experiments, only four messages were exchanged between SC and the login server. The length of these messages was almost the same in subsequent experiments. Messages (3) and (4) were different for each login attempt. However, message (3) and (4) shared a four byte common header across different experiments. The decimal equivalent of first four bytes of these common headers is the same as message (1) and (2), respectively. The decimal equivalent of the fifth byte in message (3) was 205. On inspection, we found the header '23 3 1 0' at that location [header+205] and another length field after that header whose value was 198. The decimal equivalent of the fifth byte in message (4) was '217' which appears to be the length of the message.

Note that in SSL messages, the first byte indicates the message type and the next two bytes indicate the SSL version. The value 22 (0x16) corresponds to the SSL message type client_key_exchange and the value 3 0 corresponds to the SSL version 3.0. Since the messages a SC sends to the login server contains the header 22 3 1 0, it indicates that Skype is using part of SSL header for its login messages.

Using the same setup that was used for the experiment described in the above paragraph, a login attempt was made with an invalid password. The length of the messages (1), (2), and (3) exchanged with the login server remained the same. The length of the message (4) returned by the login server was 18 bytes indicating a login failure. The decimal equivalent of the fifth byte in message (4) was 13 which indicated the length of this message after a four byte header.

To see if it is possible to block Skype, we performed the following experiment. A successful login attempt was made. Then, the SC was shut down. We overrode connect() such that it returned with an error when a connection attempt was made with the login server IP addresses. SC was then started and a

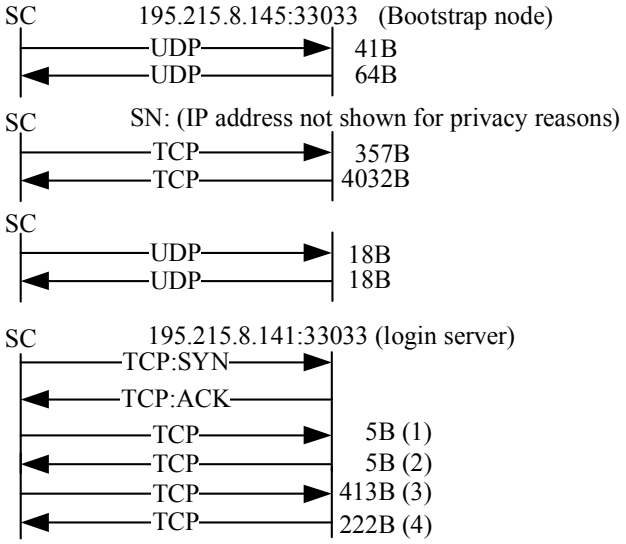


Figure 4. Minimal set of messages exchanged with the bootstrap SN, the SN, and the login server. Messages exchanged with the bootstrap SN, and SN have been aggregated. Message size for messages exchanged with the bootstrap node and SN correspond to the cumulative size. Messages sent after the exchange with the login server is completed are not shown.

TABLE I
SKYPE (VER 1.4) LOGIN EXPERIMENT SUMMARY

Skype on a Machine with/behind	Data Exchanged	Time to Login
Public IP address	10 KB	3-7 s
Port-restricted NAT	11 KB	3-7 s
UDP-restricted firewall	7 KB	35 s

login attempt was made. Strangely, the login attempt succeeded. We noted the IP address of the node to which the initial login message having decimal representation 22 3 1 0 0 was sent. In our overloaded connect(), we blocked connection attempts to this IP address. We then started Skype, and attempted a login. However, Skype was still able to login successfully. We then kept on blocking IP addresses in connect() to which login messages were sent in the previous login attempt. In all, we ended up blocking six IP addresses in connect(). However, Skype was still able to login successfully. From this experiment, we conclude that Skype routes login messages through SNs. This is a change from version 0.97 where it was possible to block Skype by simply blocking the login server IP address.

Next, we overrode the send() call such that it always returned with an error when it saw a message whose first four bytes were 22 3 1 0. Note that these are the first four bytes of message (1) and (3) shown in Figure 4. Skype was then started and a login attempt was made. Skype was unable to login despite multiple login attempts for different Skype user ids. Thus, it is possible to block Skype by dropping all the packets whose first four bytes of payload are 22 3 1 0. However, care should be taken to ensure that any such rule at the firewall does not result in blocking legitimate traffic.

For Skype v1.4, we performed experiments to understand the Skype login behavior for the three network setups described in section III.A. For these experiments, a previous copy of SC was uninstalled and Windows registry was cleared of old Skype entries. Then, a new copy of SC was installed. Table I summarizes the results of these experiments. Detailed message flows for these login attempts for v0.97 are available in the technical report [11].

In most of the login attempts, we observed that a SC sent ICMP messages to the following IP addresses: 204.152.* (USA), 130.244.* (Sweden), 202.139.* (Australia), 202.232.* (Japan). The reason for sending these messages is not clear. The reverse lookup done using MaxMind [26] suggests that each of these IP addresses is in countries located in different continents.

For the first two experimental setups, the SC sent messages to about 22 nodes and received responses from them after authenticating itself with the login server.

2) Login Server

After a SC is connected to a SN, the SC must authenticate the user name and password with the Skype login server. The login server is the only central component in the Skype p2p network. It stores Skype user names and passwords and ensures that Skype user names are unique across the Skype name space. SC must authenticate itself with the login server for a successful login. During our experiments we observed that SC always exchanged data over TCP with a node whose IP address was either 212.72.49.141 or 195.215.8.141. We believe that these nodes are the login servers. A reverse lookup of these two IP addresses did not retrieve a NS record. The first hostname returned in the authority section of the reverse lookup query (dig) was ns07.customer.eu.level3.net and ns3.DK.net respectively. Country lookup done using MaxMind tools suggests that 212.72.49.141 is in Netherlands and 195.215.8.141 is in Denmark. The buddy list is hosted on a server whose IP address is 212.72.49.142. We consider it to be a part of the login server.

TABLE II
BOOTSTRAP SN IP ADDRESS AND HOSTNAMES OBTAINED BY A REVERSE LOOKUP

IP address:port	Reverse Lookup Result	Authority Section
66.235.180.9:33033	sss1.skype.net	ns1.hopone.net
66.235.181.9:33033	No PTR result	ns1.hopone.net
212.72.49.143:33033	No PTR result	ns07.customer.eu.level3.net
195.215.8.145:33033	No PTR result	ns3.DK.net
64.246.49.60:33033	rs-64-246-49-60.ev1.net	ns2.ev1.net
64.246.49.61:33033	rs-64-246-49-61.ev1.net	ns2.ev1.net
64.246.48.23:33033	ev1s-64-246-48-23.ev1servers.net	ns1.ev1.net

3) Bootstrap Super Nodes

We list the IP address and port numbers of the seven default SNs observed during a failed login attempt. The corresponding hostnames and the first entry of the authority section returned by reverse lookup query (dig) are given in Table II.

From the reverse lookup, it appears that one SN is maintained by Skype itself.

4) NAT and Firewall Determination

We conjecture that a SC is able to determine at login if it is behind a NAT and a firewall. We guess that there are at least two ways in which a SC can determine this information. One possibility is that it can determine this information by exchanging messages with its SN using a variant of the STUN [5] protocol. The other possibility is that during login, a SC sends and possibly receives data from some nodes after it has established a TCP connection with the SN. We conjecture that at this point, SC uses its variation of STUN [5] protocol to determine the type of NAT or firewall it is behind. Once determined, the SC stores this information in the shared.xml file. We also conjecture that SC refreshes this information periodically. We are not clear on how often a SC refreshes this information since Skype messages are encrypted.

5) Skype Latest Version

During login, a SC sent a HTTP 1.1 GET request to the Skype server (skype.com) to determine if a new version was available. The first line of this request contained the keyword 'getlatestversion'. Along with the HTTP request sent at first time startup, these are the only text-based messages sent by Skype.

6) Login Process Time

We measured the time to login on the Skype network for the three different network setups described in section III. For this experiment, the HC already contained the maximum of two hundred entries. The SC with a public IP address and the SC behind a port-restricted NAT took about 3-7 seconds to complete the login procedures. The SC behind a UDP-restricted firewall took about 35 seconds to complete the login process. For SC behind a UDP-restricted firewall, we observed that it sent UDP packets to its twenty HC entries. At that point it concluded that it is behind UDP-restricted firewall. It then tried to establish a TCP connection with the HC entries and

was ultimately able to connect to a SN. Also, a SC behind a UDP-restricted firewall and port-restricted NAT took 5-10 seconds for immediate subsequent logins. This shows that a SC stores its last connectivity information in a file.

C. User Search

Skype uses its Global Index (GI) [6] technology to search for a user. Skype claims that search is distributed and is guaranteed to find a user if it exists and has logged in during the last 72 hours. Extensive testing suggests that Skype was always able to locate users who logged in using a public or private IP address in the last 72 hours.

Skype is a not an open protocol and its messages are encrypted. Whereas for login, we were able to form a reasonably precise opinion about the different entities involved, it is not possible to do so in search, since we cannot trace the Skype messages beyond a SN. Also, we were unable to force a SC to connect to a particular SN. Nevertheless, we have observed and present search message flows for the three different network setups.

A SC has a search dialog box. After entering the Skype user id and pressing the find button, SC starts its search for a particular user. For SC on a public IP address, SC sent a TCP packet to its SN. It appears that the SN gave SC the IP address and port number of eight nodes to query, since after that exchange with SN, SC sent UDP packets to eight nodes. If it could not find the user, it informed the SN over TCP. It appears that the SN now asked it to contact sixteen different nodes, since the SC then sent UDP packets to sixteen different nodes. This process continued until the SC found the user or it determined that the user did not exist. On average, SC contacted more than 24 nodes. The search took three to four seconds.

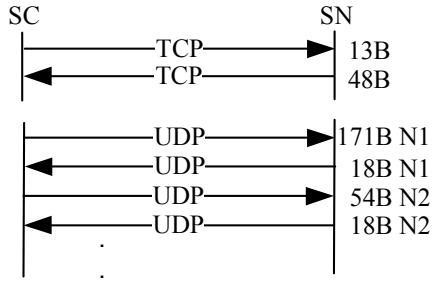


Figure 5. Message flow for a successful user search when SC v1.4 has a public IP address. 'B' stands for bytes and 'N' stands for node. Message sizes correspond to payload size of TCP or UDP packets. Not all messages are shown.

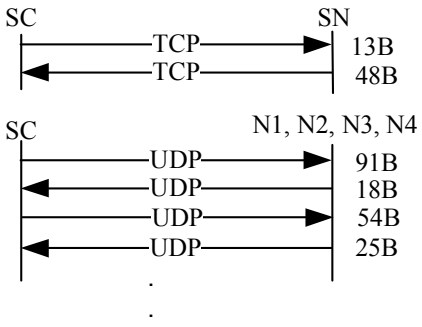


Figure 6. Message flow for a successful user search when SC v1.4 is behind a port-restricted NAT. 'B' stands for bytes and 'N' stands for node. UDP packets were sent to N1, N2, N3, and N4 during login process and responses were received from them. Message size corresponds to payload size of TCP or UDP packets. Not all messages are shown.

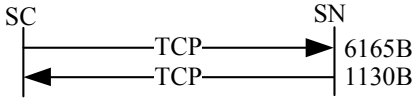


Figure 7. User search by a SC v1.4 behind a UDP-restricted firewall. 'B' stands for bytes. Messages have been aggregated for space. Data is exchanged with SN only. Message size corresponds to the approximate cumulative size of messages exchanged between SC and a SN and vice versa.

A SC behind a port-restricted NAT exchanged data between SN and some of the nodes which responded to its UDP request during login process. The message flow is shown in Figure 6. The search took about five to six seconds.

A SC behind a port-restricted NAT and UDP-restricted firewall sent the search request over TCP to its SN. We believe that SN then performed the search query and informed SC of the search results. Unlike a user search by SC on a public IP address, SC did not contact any other nodes. This suggests that SC knew that it was behind a UDP-restricted firewall. The aggregated message flow is shown Figure 7. The search took about 10-15 seconds.

In some successful searches, we saw the SC exchanging information with the login server. It appears that Skype is using the login server as a fall back option in case the search is unsuccessful. For a non-existent Skype name, a SC always contacted the login server.

We are not clear on how SC terminates the search if it is unable to find a user.

1) Search Result Caching

To observe if search results are cached at intermediate nodes, we performed the following experiment for SC v1.4. User A was behind a port-restricted NAT and UDP-restricted firewall and logged on the Skype network. User B logged in using a SC running on machine B, which was on a public IP address. User B (on a machine with a public IP address) searched for user A, who was behind a port-restricted NAT and UDP-restricted firewall. We observed that search took about 10-11 seconds. Next, SC on machine B was uninstalled, and the Skype registry cleared so as to remove any local caches. SC was reinstalled on machine B and user B searched for user A. The search took about 3-4 seconds. This experiment was repeated four times on different days and similar results were obtained.

From the above discussion we infer that the SC performs user information caching at intermediate nodes.

Skype allows the user to perform wildcard searches of different Skype user ids. To see if the same wildcard search query executed on two instances of SC retrieved the same result, we performed the following experiment. We started two instances of a Skype client on two different machines and executed the same wildcard search query on them. The retrieved results were not completely identical. In all the wildcard searches we performed, the retrieved results were never completely identical.

D. Call Establishment and Teardown

We consider call establishment for SC v1.4 for the three network setups described in section III. Further, for each setup,

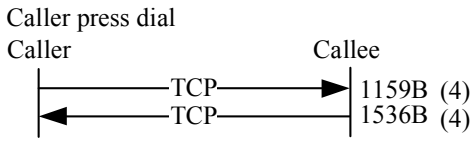


Figure 8. Message flow for call establishment when caller and callee SC v1.4 are on machines with public IP addresses and the callee is present in the buddy list of the caller. 'B' stands for bytes. Messages have been aggregated for space. Message size corresponds to the approximate cumulative size of messages exchanged between caller and a callee and vice versa. The number in paranthesis shows the total number of messages sent in that direction.

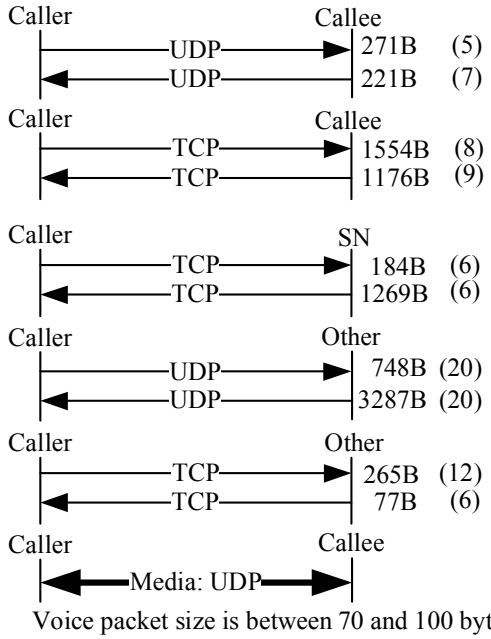


Figure 9. Message flow for call establishment when caller SC is behind a port-restricted NAT and callee SC is on a public IP address. 'B' stands for bytes. Not all messages are shown. Messages have been aggregated for space. Message size corresponds to the approximate cumulative size of messages exchanged between caller, callee, SN, other nodes and vice versa. The number in paranthesis shows the total number of messages sent in that direction.

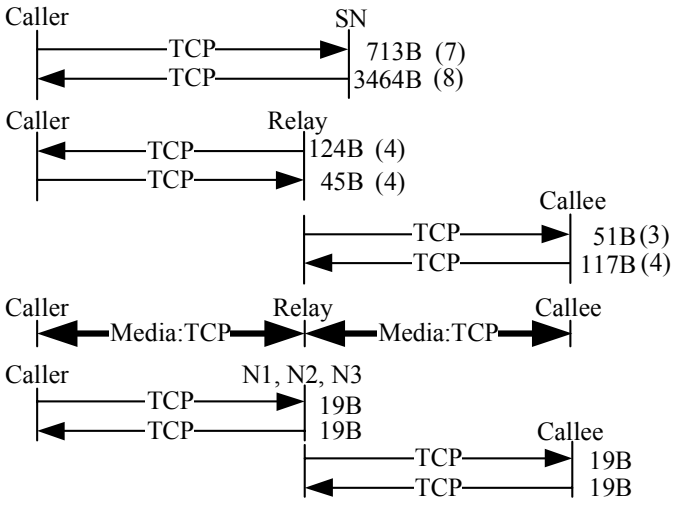


Figure 10. Message flow for call establishment when caller and callee SC are behind a port-restricted NAT and UDP-restricted firewall. 'B' stands for bytes and 'N' stands for a node. Not all messages are shown. Messages have been aggregated for space. Message size corresponds to the approximate cumulative size of messages exchanged between caller, callee, other nodes and vice versa. Voice traffic flows over TCP. The number in paranthesis shows the total number of messages sent in that direction.

we consider call establishment for users that are in the buddy list of caller and for users that are not present in the buddy list. It is important to note that call signaling is always carried over TCP.

For users that are not present in the buddy list, call placement is equal to user search plus call signaling. Thus, we discuss call establishment for the case where the callee is in the buddy list of the caller.

If both users were on machines with public IP addresses, online and were in the buddy list of each other, then upon pressing the call button, the caller SC established a TCP connection with the callee SC. Signaling information was exchanged over TCP. The aggregated message flow between caller and callee is shown in Figure 8.

The initial exchange of messages between caller and callee indicates the existence of a challenge-response mechanism. The caller also sent some messages (not shown in Figure 8) over UDP to alternate Skype nodes. For this scenario, approximately six kilobytes of data was exchanged.

In the second network setup, where the caller was behind a port-restricted NAT and the callee was on a public IP address, signaling information did not flow directly between caller and callee initially. Instead, the caller sent signaling information over TCP to an online Skype node which forwarded it to callee over TCP. After a call had been established, the media flowed directly between caller and callee over UDP. The message flow is shown in Figure 9. For this scenario, approximately eight kilobytes of data was exchanged.

For the third setup, in which both users were behind port-restricted NAT and UDP-restricted firewall, both caller and callee SC exchanged signaling information over TCP with another online Skype node. Caller SC sent media over TCP to an online node, which forwarded it to callee SC over TCP and vice versa. The message flow is shown in Figure 10. For this scenario, approximately eight kilobytes of data was exchanged.

There are certain advantages of having a node route the voice packets from caller to callee and vice versa. First, it provides a mechanism for users behind NATs and firewalls to talk to each other. Second, if users behind NATs or firewalls want to participate in a conference, and some users on public IP address also want to join the conference, this node serves as a mixer and broadcasts the conferencing traffic to the participants. The negative side is that there will be a lot of traffic flowing across this node. Users generally do not like the fact that arbitrary traffic could flow across their machines.

During call tear-down, signaling information is exchanged over TCP between caller and callee if they are both on public IP addresses, or between caller, callee and their respective SNs. The messages observed for call tear down between caller and callee on public IP addresses are shown in Figure 11.

For the second and third network setups, call tear down signaling is also sent over TCP. We, however, do not present these message flows, as they do not provide any interesting information.

For SC v1.4, we performed experiments to determine if the call signaling goes end-to-end when caller and callee SC are on

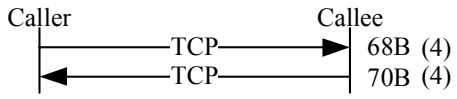


Figure 11. Call tear down message flow for caller and callee with public IP addresses. Messages have been aggregated for space. The number in paranthesis shows the total number of messages sent in that direction.

machines with public IP addresses and are in the buddy list of each other. Two instances of SC were started on two Windows machines with pubic IP addresses. Each instance had the other Skype user in its buddy list. After successful login, we waited

until each instance was aware of the presence of other instance. This was shown by a buddy item changing color to green in the buddy list. We ensured that there was no TCP connection between the two machines. Using NetPeeker, we blocked all outgoing and incoming connections except for the one destined for the callee machine and vice versa. A call attempt was made which succeeded.

We then blocked all TCP connection attempts between these two machines and attempted to make a call. The call attempt failed.

This experiment shows that for the scenario described above, the call signaling does go end-to-end. Also, call signaling is carried over TCP.

E. Media Transfer and Codecs

If both Skype clients (v1.4) were on machines with public IP addresses, then media traffic flowed directly between them over UDP. The media traffic flowed to and from the UDP port configured in the options dialog box. The voice packet size varied between 40 and 120 bytes. For two users connected to Internet over 100 Mb/s Ethernet with almost no congestion in the network, roughly 85 voice packets were exchanged both ways in one second. The total uplink and downlink bandwidth used for voice traffic was 5 kilobytes/s. This bandwidth usage agrees with the Skype claim of 3-16 kilobytes/s.

If either caller or callee or both were behind port-restricted NAT, they sent voice traffic to each other. The voice packet size varied between 40 and 110 bytes, which is the size of UDP payload. The bandwidth used was about 5 kilobytes/s.

If both users were behind port-restricted NAT and UDP-restricted firewall, then caller and callee sent and received voice traffic over TCP from another online Skype node. The TCP packet payload size for voice traffic varied between 30 and 90 bytes. The total uplink and downlink bandwidth used for voice traffic was about 5.5 kilobytes/s. For media traffic, SC used TCP with retransmissions.

In all three cases, the codec used was iSAC [13].

The Skype protocol seems to prefer the use of UDP for voice transmission. The SC will use UDP for voice transmission if it is behind a NAT or firewall that allows UDP packets to flow across.

1) Silence Suppression

No silence suppression is supported in Skype. We observed that when neither caller nor callee was speaking, voice packets were still flowing between them. While this increases the bandwidth usage, transmitting these silence packets has two advantages. First, it maintains the UDP bindings at NAT and second, these packets can be used to play some background noise at the peer. In the case where media traffic flowed over TCP between caller and callee, silence packets were still sent. The purpose is to avoid the drop in TCP congestion window size, which takes some RTT to reach the maximum level again.

2) Putting a Call on Hold

Skype allows peers to hold a call. Since a SC can operate behind NATs, it must ensure that UDP bindings are valid at a NAT box. On average, a SC sent one UDP packet every three seconds to the call peer, SN, or the online Skype node acting as a media proxy when a call is put on hold. We also observed that in addition to UDP messages, the SC also sent periodic messages over TCP to the peer, SN, or online Skype node acting as a media proxy during a call hold.

3) Codec Frequency Range

We performed experiments to determine the range of frequencies Skype codecs allow to pass through. A call was established between two Skype clients (v1.4). Tones of different frequencies were generated using the NCH Tone Generator [16] on the caller SC and output was observed on the callee SC and vice versa. We observed that the minimum and maximum audible frequency Skype codecs allowed to pass through were 50 Hz and 8,000 Hz respectively.

Using Net Peeker [8], we reduced the uplink and downlink bandwidth available to Skype application to 1,500 bytes/s, respectively. We observed that the minimum and maximum audible frequencies Skype codecs allowed to pass through remained unchanged, i.e., 50 Hz and 8,000 Hz, respectively.

4) Congestion

We checked Skype call quality in a low bandwidth environment by using Net Peeker [8] to tune the upload and download bandwidth available for a call. We observed that uplink and downlink bandwidth of 2 kilobytes/s each was necessary for bare minimum audible call quality. The voice was almost unintelligible at an uplink and downlink bandwidth of 1.5 kilobytes/s.

F. Keep-alive Messages

We observed for three different network setups that the SC v1.4 sent a refresh message to its SN over TCP. When SC was on a machine with a public IP address, a refresh message was sent every 120s.

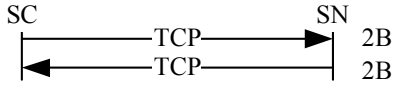


Figure 12. Skype refresh message to SN when SC was on a machine with public IP address.

V. CONFERENCING

We observed the Skype conferencing features for a three-user conference for the three network setups discussed in section III.A for Skype v1.4. We use the term user and machine interchangeably. Let us name the three users or machines as A, B, and C. Machine A was a 1.6 GHz Pentium 4 laptop with 512 MB RAM while machine B and C had a 3 GHz Pentium 4 CPU with 1 GB of RAM. In the first setup, the three machines had public IP addresses. A call was established between A and B. Then B decided to include C in the conference. From the ethereal dump, we observed that B and C were sending their voice traffic over UDP to SC on machine A, which was acting as a mixer. It mixed its own packets with those of B and sent them to C over UDP and vice versa as shown in Figure 13.

TABLE III
SKYPE, YAHOO, MSN AND GOOGLE TALK COMPARISON

	Application version	Memory Usage before call (caller, callee)	Memory Usage during call (caller, callee)	Process priority before call	Process priority during call	Mouth-to-ear latency	Latency Standard Deviation
Skype	1.4.0.84	19 MB, 19 MB	21 MB, 27 MB	Normal	High	96 ms	4
Yahoo	7.0.0.437	38 MB, 34 MB	43 MB, 42 MB	Normal	Normal	152 ms	12
MSN	7.5	25 MB, 22 MB	34 MB, 31 MB	Normal	Normal	184 ms	16
G-Talk	1.0.0.80	9 MB, 9 MB	13 MB, 13 MB	Normal	Normal	109 ms	10

In the second setup, B and C were behind port-restricted NAT, and A was on the public Internet. Initially, user A and B established the call. Both A and B were sending media to each other over UDP. User A then put B on hold and established a call with C. It then started a conference with B and C. We observed that both B and C were now sending their packets to A over UDP, which mixed its own packets with those coming from B and C, and forwarded it to them appropriately.

In the third setup, B and C were behind port-restricted NAT and UDP-restricted firewall and A was on the public Internet. User A started the conference with B and C. We observed that both B and C were sending their voice packets to A over TCP. A mixed its own voice packets with those coming from B and C and forwarded them to B and C appropriately.

If user B was in a call with user C using a relay D and if user B initiated a conference with user A, relay D was still being used between user B and C.

In the same experiments for Skype v0.97, we had observed that the most powerful machine always got elected as a conference host.

For a three party conference, Skype does not do full mesh conferencing [17].

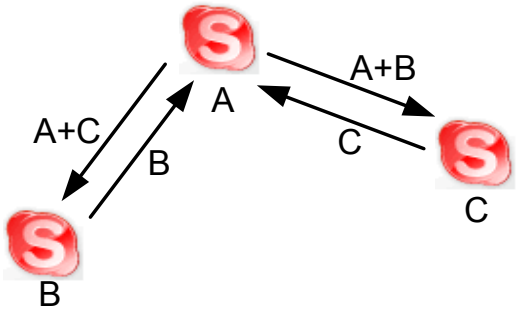


Figure 13. Skype three user conferencing

VI. OTHER EXPERIMENTS

Unlike MSN and Yahoo Messenger, which signs out a user if that user logs in on another machine, Skype allows a user to log in from multiple machines simultaneously. The calls intended for that user are routed to all locations. Upon picking a call at one location, the call is immediately cancelled at other locations. Similarly, instant messages for a user who is logged in at multiple machines are delivered to all the locations.

The SN is selected by the Skype protocol based on a number of factors like CPU and available bandwidth. It is not possible to arbitrarily select a SN by filling the HC with the IP address of an online SC. This conclusion was drawn from the following experiment. Consider two online Skype nodes A and B. A is connected to the Skype network and has only one entry in its HC. We call super node of A as SN_A. Now we modify the HC of SC on machine B, such that it only contains the IP address and port number of SC running at A. When B logged onto the Skype network, we observed that it connected to another super node rather than connecting to A.

To see if a different encryption key is embedded within each Skype executable, we compared the Skype setup files downloaded at randomly chosen times during a week. There were no differences between the setup files.

If two Skype users were behind the same NAT and established a call with each other, voice traffic flowed directly between them over the private network. If two Skype users were behind different NATs, then some ARP messages were seen on the wire. This seems to indicate that Skype was trying to determine network connectivity.

A. Comparison with Yahoo, MSN and Google Talk IM/Voice Applications

We measured memory usage and process priority before and during calls, and mouth-to-ear latency for the Skype, Yahoo, MSN, and Google Talk applications.

For our experiments, mouth-to-ear latency is defined as the difference between the time the words are spoken on one voice client, and the time they are heard at the other voice client given the two voice clients are already in a voice session. If both the original voice signal and the signal that traveled over the network can be recorded in a stereo format, then the delay or relative shift between these two signals can be calculated by computing a correlation between these two signals using a fast fourier transform (FFT). adelay [24] is a tool developed by Hao Huang in our IRT Lab that computes the mouth-to-ear latency using the technique described above.

For this experiment, the input signal was a pre-recorded Sun .au file of 24s. Skype, MSN, Yahoo and Google Talk applications were started on two separate laptops running Windows XP service pack 2 (SP2) and having identical hardware configuration. Each laptop had a Pentium (M) 1.7 GHz processor with 1 GB of RAM. Both machines had public IP addresses and were connected to a 100 Mb/s LAN. A voice session was established between respective voice clients. Using Ethereal we checked that both caller and callee Skype, Yahoo, MSN, and Google Talk clients were sending audio packets directly to each other. The pre-recorded audio file was played on a separate machine and the audio signal was provided as an input to the caller machine. The original signal and the signal received over the network were given as an input to the LineIn jack of another machine which ran the Cool Edit Pro version 2.1 [25] software. Using Cool Edit Pro, the two signals were recorded in Sun .au stereo format sampled at 8,000 Hz with 16 bit signed linear encoding. The sampling time for adelay was two seconds. For each voice client, the experiment was repeated four times. The results of these experiments are summarized in Table III. The mouth-to-ear latency is an average of four experiments for each IM client. The round-trip delay between the caller and callee machines, measured using ping, was less than one second.

We compared the memory usage and process priority for the three clients under test. Unlike Yahoo, MSN and Google Talk clients, Skype changes its priority to High priority, when a call is established.

TABLE IV
SN DISTRIBUTION PER DAY FOR 894 UNIQUE SN'S

	Unique SNs per day	Cumulative Unique SNs	Common SNs between previous and current day
Day1	224	224	
Day2	371	553	42
Day3	202	699	98
Day4	246	898	103

TABLE V
SN DISTRIBUTION PER COUNTRY (NON-US

Countries	Cumulative SNs
Taiwan	256
Israel	194
Japan	168
France	75
Switzerland	55

TABLE VI
SN DISTRIBUTION PER UNIVERSITY

Countries	Cumulative SNs
Harvard	367
Columbia	366
UNL	350
UPenn	245
BU	179

B. Skype Super Node Map

We performed experiments to get insights into the Skype super node selection mechanism. We know that at login, a SC must always connect to a SN. We take advantage of the fact that if a SC is behind a NAT, it will never become a SN and it will most likely connect to only one super node. Also, in a subsequent immediate login, a SC does not always reconnect to the same super node it connected last time. By having Skype repeatedly login onto the Skype network for an extended period of time, we can get a partial snapshot of the Skype super nodes.

For this experiment, we used AutoIt [27] to automatically start the Skype application, have it login on the Skype network, and then terminate it. AutoIt is a scripting tool for automating Windows tasks such as GUI input. There was a gap of 30 seconds between Skype application startup and termination and a gap of 10 seconds between the termination and the next startup. We ran the experiment for 96 hours (four days). Using netstat, we noted the IP address and port number to which the test machine had established a TCP connection. Since there were no applications running on the machine which established TCP connections except Skype, and since Skype must establish a TCP connection with a SN at login, the IP address and port number of the established TCP connection reported by netstat are indeed the IP address and port number of the SN to which Skype connects.

Theoretically, over a period of four days 8,640 login attempts are possible since the runtime of one iteration is 40 seconds. However, we recorded 8,175 login attempts and the lesser number is attributed to the script execution and Windows overhead. After removing the datasets that contained multiple TCP connections in ESTABLISHED state in the netstat data, we found 898 unique super nodes in 8,163 successful login attempts.

Using MaxMind tools, we determined the latitude, longitude, country and city of each IP address. They have been plotted on the map shown in Figure 16. MaxMind tools were unable to determine the country for four IP addresses, which were only used for 10 connections. Thus, our data set size was reduced to 8,153 successful login attempts and 894 unique super nodes.

The processing of data revealed the following:

- SN IP addresses distribution: US 83.7%, Asia 8.9%, Europe 7.1%.
- In 8153 login attempts, 2855 (35%) hostnames had a '.edu' suffix and belonged to 102 universities.
- Out of the 894 unique SNs, the top 20 nodes received 43.8% of the total connections and top 100 nodes received 70.5% connections.

Table IV shows the number of unique super nodes per 24 hours. It also shows the number of unique SNs that were common between the last day and the current day. Note that there were a total of 894 unique SNs. Table V shows the top five countries excluding US that received the most number of connections. Table VI shows the top five universities that received the highest number of connections.

VII. CONCLUSION

In this paper, we have tried to analyze various aspects of the Skype protocol by analyzing the Skype network traffic and by intercepting the shared library and system calls of Skype. We observed that Skype can work almost seamlessly behind NATs and firewalls. We believe that Skype client uses its version of STUN [5] protocol to determine the type of NAT or firewall it is behind. The NAT and firewall traversal techniques of Skype are similar to many existing applications such as network games. It is by the random selection of sender and listener ports, the use of TCP as voice streaming protocol, and the peer-to-peer nature of the Skype network, that not only a SC traverses NATs and firewalls but it does so without any explicit NAT or firewall traversal server. Skype uses TCP for signaling. It uses wide band codecs and has licensed them from GlobalIPSound [15]. Skype communication is encrypted.

The underlying search technique that Skype uses for user search is still not clear. Our guess is that it uses a combination of hashing and periodic controlled flooding to gain information about the online Skype users. Skype search mechanism falls back to the login server for all unsuccessful and some successful searches.

Skype has a central login server which stores the login name, password and buddy list of each user. Since Skype packets are encrypted, it is not possible to say with certainty what other information is stored on the login server. However, during our experiments we did not observe any subsequent exchange of information with the login server after a user logged onto the Skype network.

Compared to Yahoo, MSN, and Google Talk applications, Skype reported the best mouth-to-ear latency.

Skype is a selfish application and it tries to obtain the best available network and CPU resources for its execution. It changes its application priority to high priority in Windows during the time call is established. It evades blocking by routing its login messages over SNs. This also implies that Skype is relying on SNs, who can misbehave, to route login messages to the login server. Skype does not allow a user to prevent its machine from becoming a SN although it is possible to prevent Skype from becoming a SN by putting a bandwidth limiter on the Skype application when no call is in progress. Theoretically speaking, if all Skype users decided to put bandwidth limiter on their application, the Skype network can possibly collapse since the SNs hosted by Skype may not have enough bandwidth to relay all calls.

From our experience of analyzing the Skype protocol, we gather that packet intercept and blocking can be used for protocol reverse engineering. Classical packet sniffing tools

such as Ethereal are less useful when packet content is encrypted. Shared library and system call interception techniques can be used to manipulate the network traffic of a black box executable.

ACKNOWLEDGMENTS

The authors would like to thank Faisal Ghias Mir, Antonio Altamare, and Ricardo Barrato for the insightful discussions on Linux function call interception and various aspects of Skype. The authors would also like to thank David Chaum, Alok Agrawal, Eunsoo Shim, Sarah Baker, Balwant Rathore and numerous others who gave comments on our Skype technical report.

REFERENCES

[1] Skype. http://www.skype.com

[2] Kazaa. http://www.kazaa.com

[3] SkypeOut. http://www.skype.com/products/skypeout/

[4] SkypeIn. http://www.skype.com/products/skypein/

[5] J. Rosenberg, J. Weinberger, C. Huitema, and R. Mahy. STUN: simple traversal of user datagram protocol (UDP) through network address translators (NATs). RFC 3489, IETF, Mar. 2003.

[6] Global Index (GI): http://www.skype.com/skype_p2pexplained.html

[7] Ethereal. http://www.ethereal.com

[8] Net Peeker. http://www.net-peeker.com

[9] J. Rosenberg, H. Schulzrinne, G. Camarillo, A. R. Johnston, J. Peterson, R. Sparks, M. Handley, and E. Schooler. SIP: session initiation protocol. RFC 3261, IETF, June 2002.

[10] R. Fielding, J. Gettys, J. Mogul, H. Frystyk, L. Masinter, P. Leach, T. Berners-Lee. HTTP: hyper text transfer protocol. RFC 2616, IETF, June 1999.

[11] S. Baset and H. Schulzrinne. An analysis of the skype peer-to-peer Internet telephony protocol. Columbia University Technical Report CUCS-039-04, September 2004.

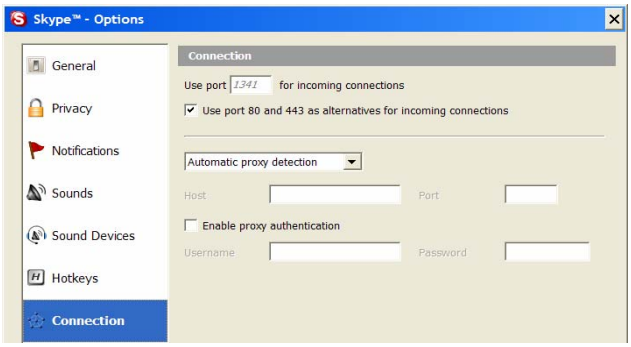


Figure 14. Skype (v1.4) connection tab. It shows the port on which Skype listens for incoming connections.

[12] iLBC codec. http://www.globalipsound.com/datasheets/iLBC.pdf

[13] iSAC codec. http://www.globalipsound.com/datasheets/iSAC.pdf

[14] iPCM codec. http://www.globalipsound.com/datasheets/iPCM-wb.pdf

[15] Global IP Sound. http://www.globalipsound.com/

[16] NCH Tone Generator. http://www.nch.com.au/tonegen/

[17] J. Lennox and H. Schulzrinne. A protocol for reliable decentralized conferencing. ACM International Workshop on Network and Operating Systems Support for Digital Audio and Video (NOSSDAV), Monterrey, California, June 2003.

[18] Skype FAQ. http://support.skype.com/index.php?_a=knowledgebase&_j=questiondetails&_i=145

[19] J. Rosenberg, R. Mahy, C. Huitema. TURN: traversal using relay NAT. Internet draft, Internet Engineering Task Force, September 2005. Work in progress.

[20] I. Stoica, R. Morris, D. Karger, M. F. Kaashoek, H. Balakrishnan. Chord: A scalable peer-to-peer lookup service for internet applications. In Proc. ACM SIGCOMM (San Diego, 2001).

[21] memgrp. http://www.hick.org/code/skape/memgrep/

[22] Linux Function Interception. http://uberhip.com/godber/interception/

[23] X-Win32. http://www.xwin32.com/

[24] adelay. Measure the delay between two audio channels. http://www1.cs.columbia.edu/IRT/software/adelay/adelay.html

[25] Cool Edit Pro v2.1. http://www.softpedia.com/progDownload/Cool-Edit-Pro-Download-2076.html

[26] MaxMind. http://www.maxmind.com

[27] AutoIt. http://www.hiddensoft.com

```
<?xml version="1.0" ?>
- <config version="1.0" serial="6625" timestamp="1135714201.11">
  - <Lib>
    + <Account>
    + <BCM>
    - <Connection>
      - <Bandwidth>
          <CurSlotLength>6008</CurSlotLength>
          <LastRtTestTime>1135714068</LastRtTestTime>
          <OutHistory>7974</OutHistory>
        </Bandwidth>
        <DisablePort80>0</DisablePort80>
      + <EventServers>
      - <Firewall>
          <TcpInHistory>-1431655768</TcpInHistory>
          <UdpInHistory>-1431655768</UdpInHistory>
          <UdpOutHistory>1431655807</UdpOutHistory>
        </Firewall>
      - <HostCache>
          <_1>140.115.23.23:62601</_1>
          <_10>87.69.48.254:1586</_10>
          <_100>140.121.135.224:3256</_100>
          <_101>217.199.108.68:35749</_101>
          <_102>217.199.108.67:59107</_102>
```

Figure 15. Skype (v1.4) host cache list (shared.xml)

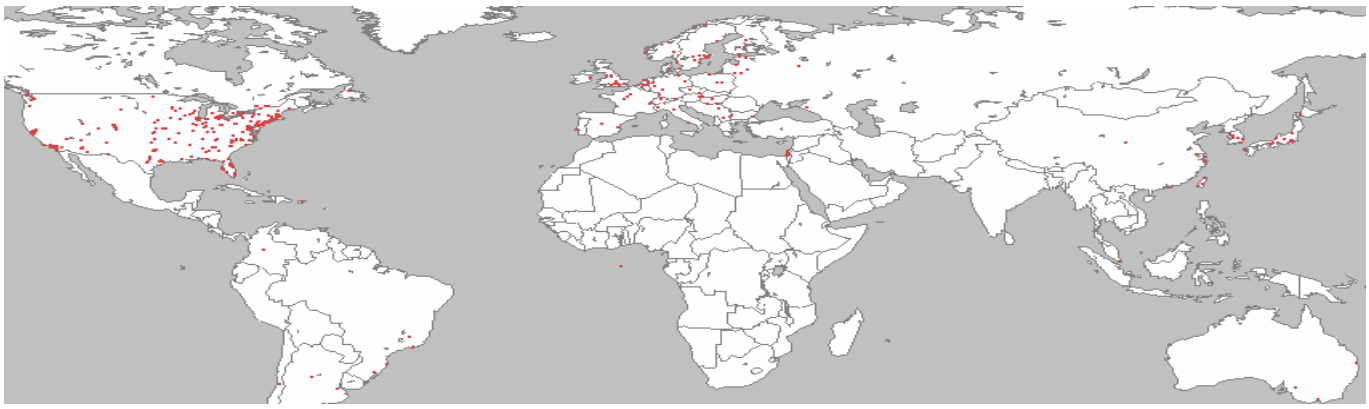


Figure 16. Worldmap of super nodes to which Skype establishes a TCP connection at login.