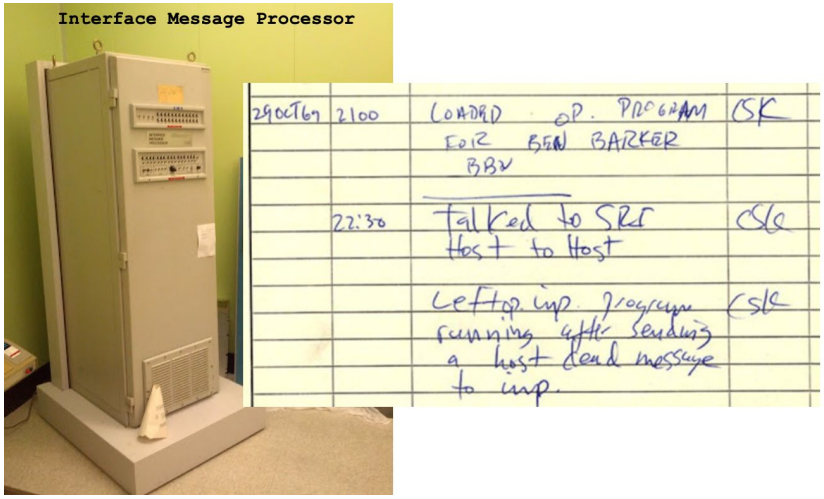
# Table of contents



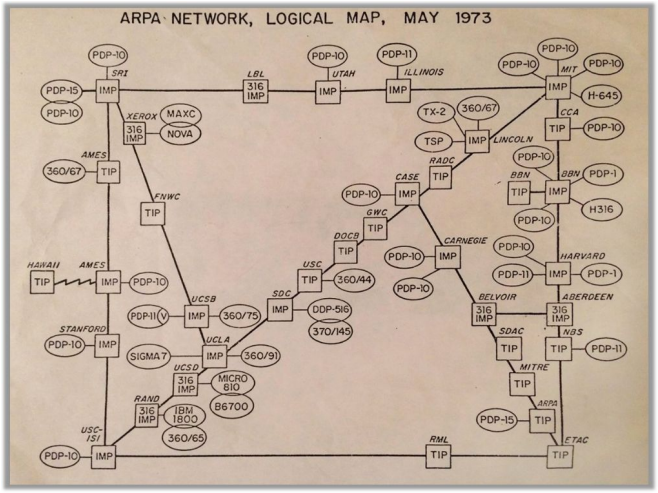


# Origins of Internet



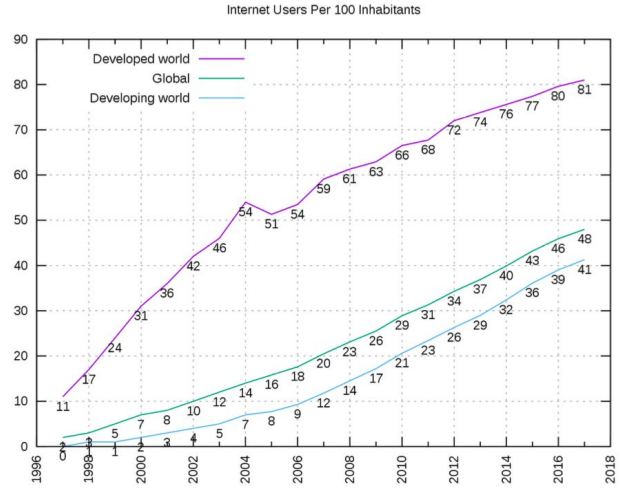



# Origins of Internet






# Origins of Internet



## Impacts of Internet

- Design of Internet
  - Is the reason for its impact?



## Impacts of Internet

- Design of Internet
  - Supports growth and fosters innovation?



## Impacts of Internet

- Internet is a tense place

Search **Bloomberg**

Cybersecurity

**Cyber-Attack Hits U.S. Health Agency Amid Covid-19 Outbreak**

By Shira Stein and Jennifer Jacobs
March 16, 2020, 8:37 AM EDT *Updated on March 16, 2020, 4:35 PM EDT*

- NSC tweet on disinformation Sunday was connected to attack
- Cyber intrusion comes as U.S. battles the coronavirus pandemic

**2018:**




## Impacts of Internet

- Internet is a tense place

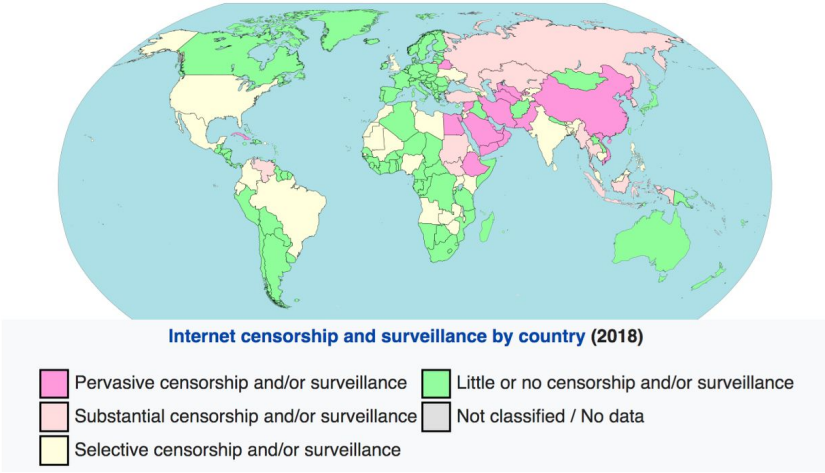
## Impacts of Internet

- Internet is a tense place




## Impacts of Internet

- Internet is a tense place






## Impacts of Internet

- Design of Internet
  - Creates or exacerbates these tensions?

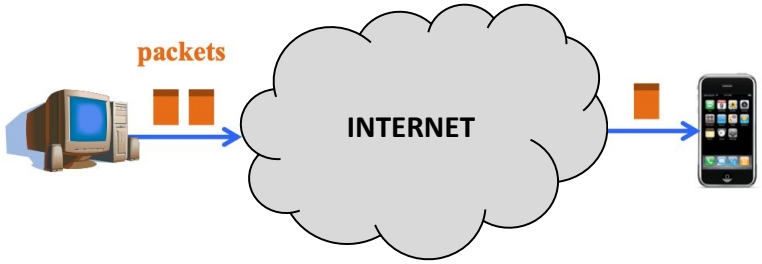


## What is Internet?

- Internet is a
  - **Publicly accessible** network of **interconnected** computer networks
    - Transmit data by **packet switching** using standard Internet Protocol
  - **Network of networks**
    - Consists of many smaller domestic/academic/business/govt networks
      - Carry various **information** and **services**

## What is Internet?

- Best-effort packet delivery service





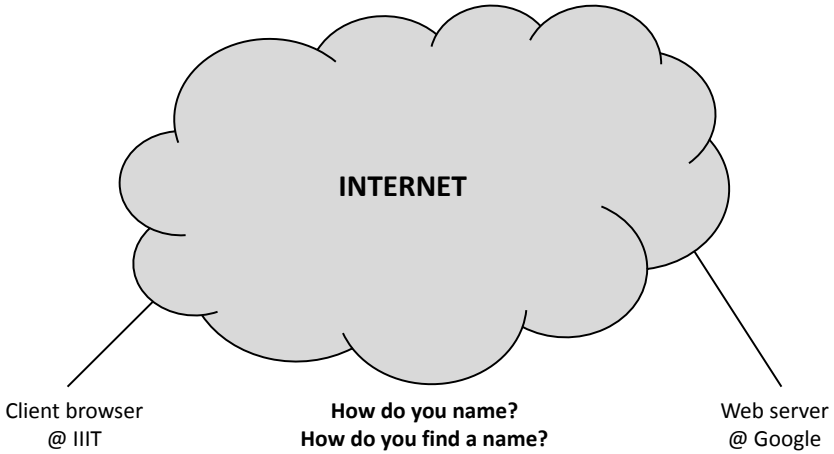

## What is Internet?

- Power at edge
  - End-to-end principle
    - Communication/protocol operations should occur at endpoints
      - Whenever possible
  - Programmability
    - New network services can be added at any time, by anyone
      - With programmable end hosts
        - Eventually, end hosts became powerful and ubiquitous



## What is Internet?

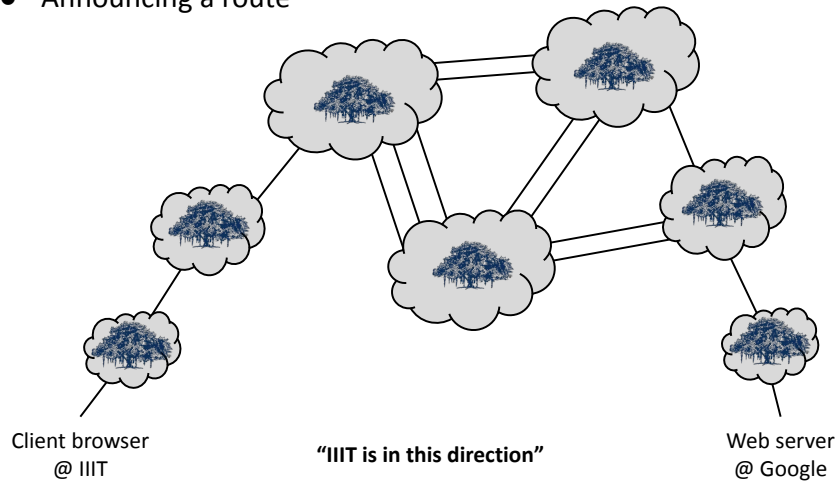
- Internet is a network of networks





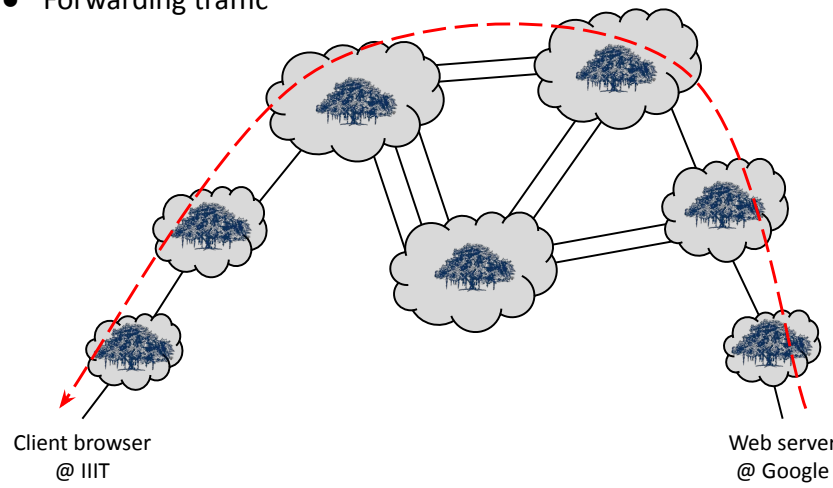

## What is Internet?

- Announcing a route



## What is Internet?

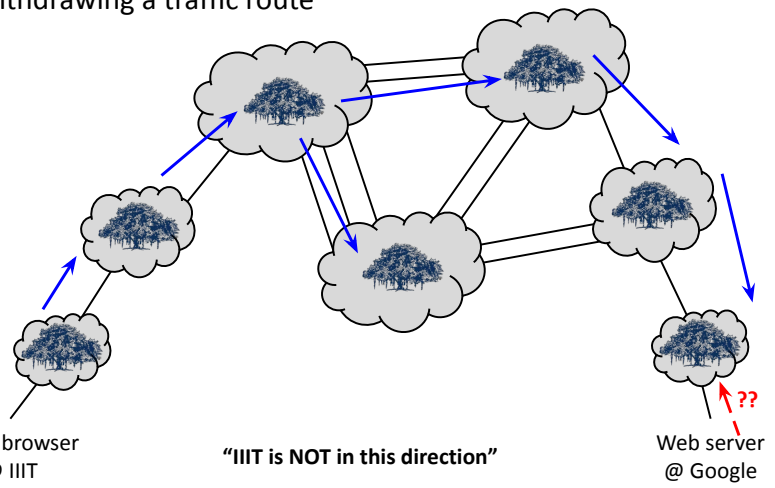
- Forwarding traffic






## What is Internet?

- Withdrawing a traffic route






## What is Internet?

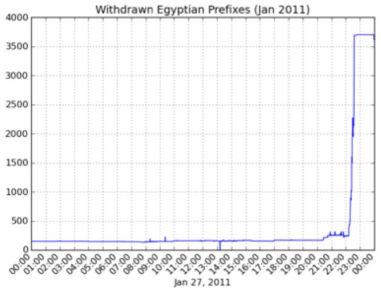
- Withdrawing a traffic route

renesys | blog

**Egypt Leaves the Internet**

By James Cowie on January 27, 2011 7:56 PM

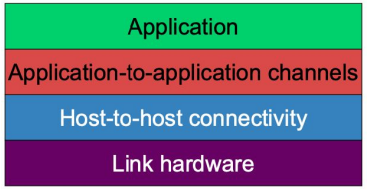
At 22:34 UTC (00:34am local time), Renesys observed the virtually simultaneous withdrawal of all routes to Egyptian networks in the Internet's global routing table. Approximately 3,500 individual BGP routes were withdrawn, leaving no valid paths by which the rest of the world could continue to exchange Internet traffic with Egypt's service providers. Virtually all of Egypt's Internet addresses are now unreachable, worldwide.





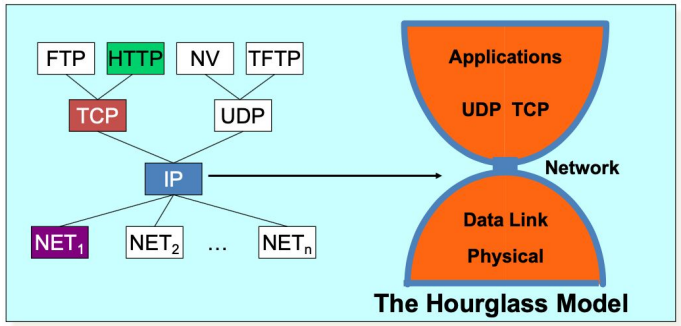

## Key concepts in networking

- Abstraction through protocol layering
  - Layers partition system
    - Each layer **solely** relies on services from layer below
    - Each layer **solely** exports services to layer above
  - Interface between layers defines interaction
    - Hides implementation details
    - Layers can change without disturbing other layers



## Key concepts in networking

- Internet Protocol (IP) suite
  - Thin Network layer facilitates interoperability






## Key concepts in networking

- Application: HyperText Transfer Protocol

```
GET /path/to/resource/ HTTP/1.1
Host: www.cs.xyz.edu
User-Agent: Mozilla/5.0
CRLF
```

Request

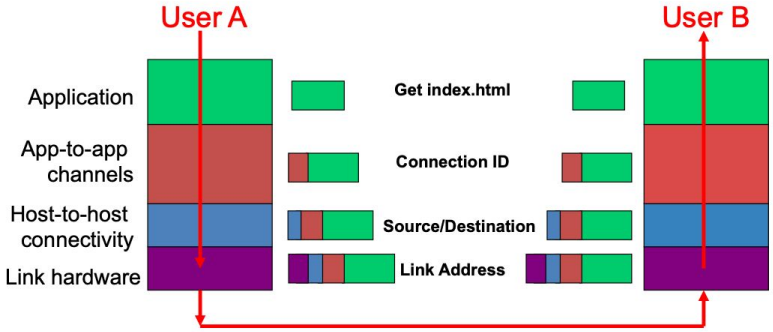
Response

```
HTTP/1.1 200 OK
Date: Wed, 11 Aug 2021 09:28:28 GMT
Server: Apache/2.4.41
Last-Modified: Fri, 06 Aug 2021 04:46:59 GMT
Content-Length: 23
CRLF
Site under construction
```



## Key concepts in networking

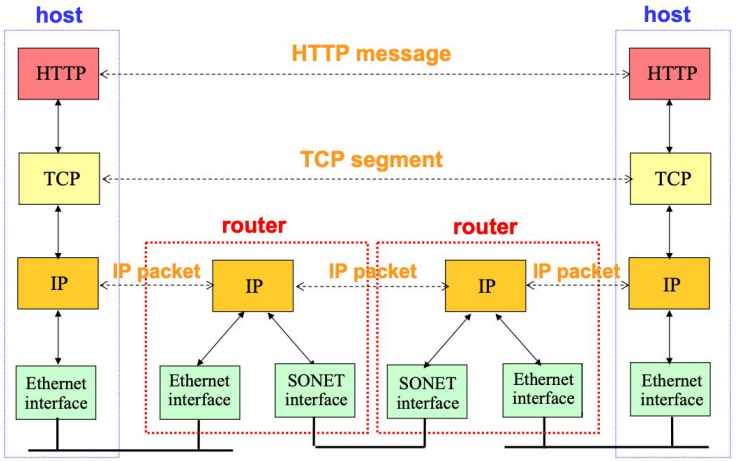
- Layer encapsulation in HTTP





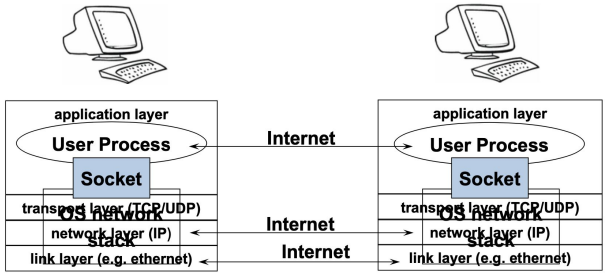

## Key concepts in networking

- End hosts vs router



# Key concepts in networking

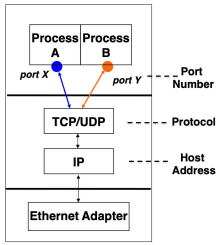
- Socket and process communication
  - Interface that OS provides to its networking subsystem






# Key concepts in networking

- Socket and process communication
  - Receiving host
    - Destination **address** that uniquely identifies host
      - IP address: 32-bit (“1.2.3.4”)
  - Receiving socket
    - Host may be running many different processes
      - Destination **port** that uniquely identifies socket
        - Port number: 16-bit (“80”)





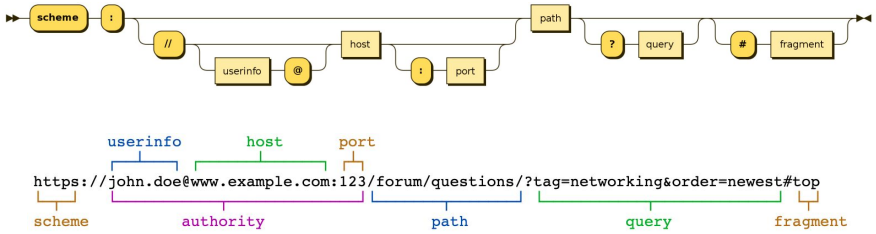

# Key concepts in networking

- Central concepts
  - Naming
    - What to call computers, services, protocols, etc.
  - Layering
    - Abstraction is key to managing complexity
  - Protocols
    - Speaking same language
      - Syntax and semantics
  - Resource allocation
    - Dividing scare resources among competing parties
      - Memory, link bandwidth, wireless spectrum, paths



# Key concepts in networking

- Uniform Resource Identifier (URI)
  - Unique sequence of characters
    - Identifies a logical or physical resource used by web
      - Real-world objects (e.g., people, places)
      - Information resources (e.g., webpages, books)
  - Syntax
    - `URI = scheme:[//authority]path[?query][#fragment]`
      - `authority = [userinfo@]host[:port]`



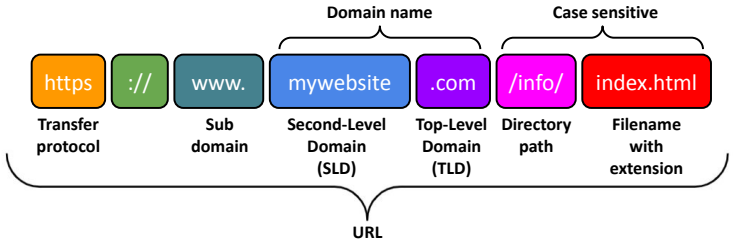
## Key concepts in networking

- Uniform Resource Name (URN)
  - Type of URI
  - Provide only a unique name
    - Without means of locating/retrieving resource/information
  - URN identifies an item, e.g., ISBN of a book



## Key concepts in networking

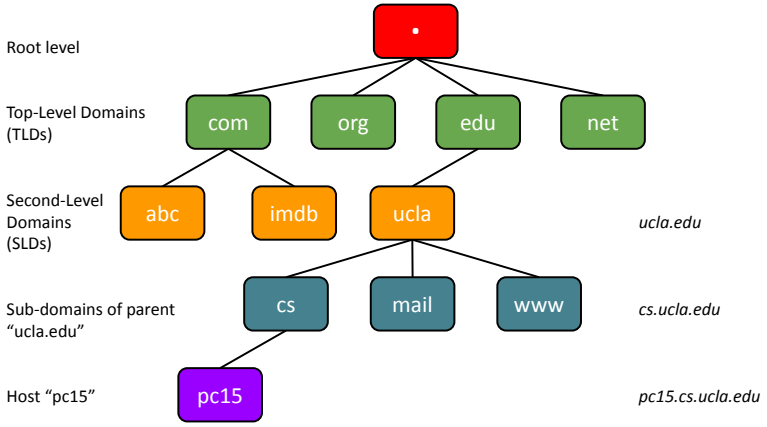
- Uniform Resource Locator (URL)
  - Type of URI
  - Provide means of locating/retrieving resources/information
  - URL provides a method for finding resources/information, e.g., web






## Key concepts in networking

- Domain Name System (DNS)
  - Serves as phone book for Internet
    - Translate human-friendly computer hostnames into IP addresses



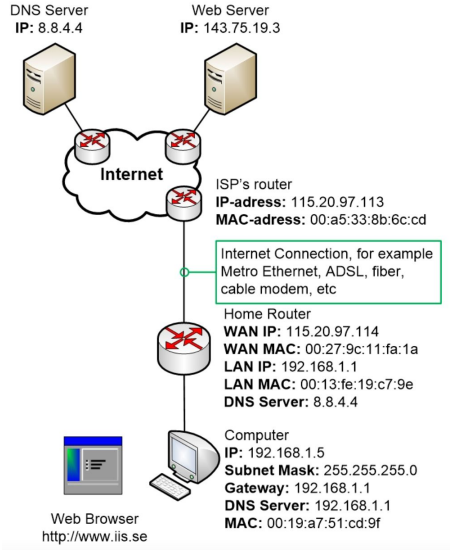



## Key concepts in networking

- Dynamic Host Configuration Protocol (DHCP)
  - A server service that dynamically assigns, or leases, IP addresses and related IP information to network clients
  - Each client gets
    - Unique IP address
    - Subnet mask
    - Default gateways
    - Domain Name System (DNS) server addresses

# How does Internet work?

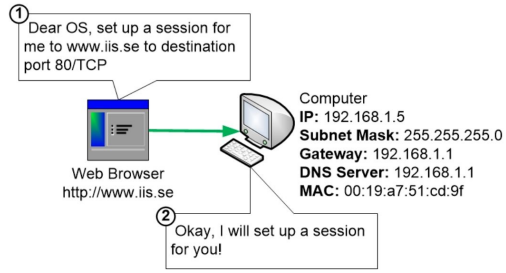
0. Sample setup






# How does Internet work?

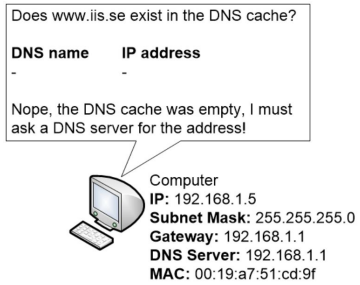
1. Computer wants to send traffic






# How does Internet work?

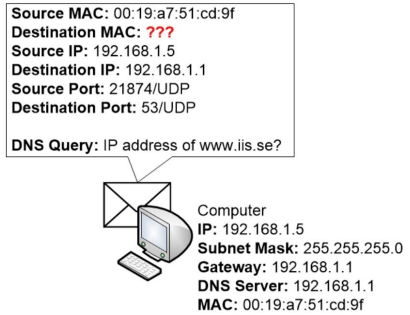
2. DNS
   a. DNS cache





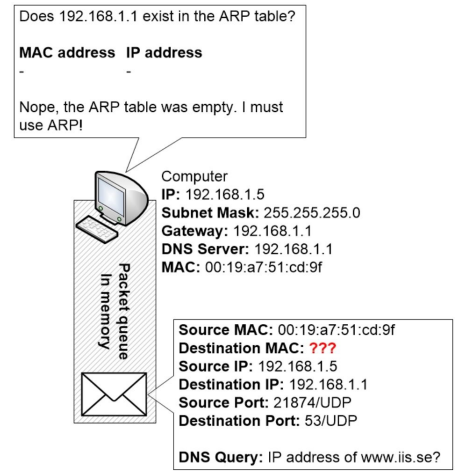

# How does Internet work?

2. DNS
   b. Putting a DNS query together



# How does Internet work?

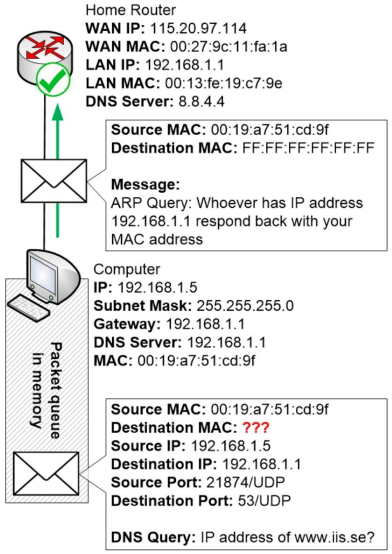
2. DNS
   c. Check ARP table for a valid MAC address






# How does Internet work?

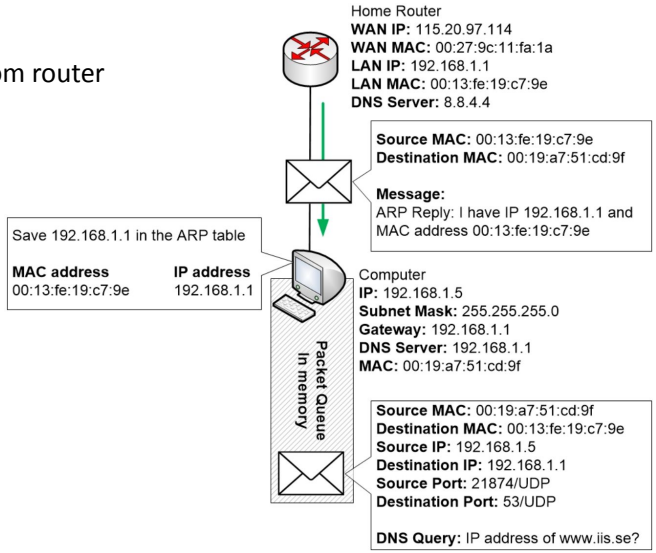
2. DNS
   d. ARP request to network






# How does Internet work?

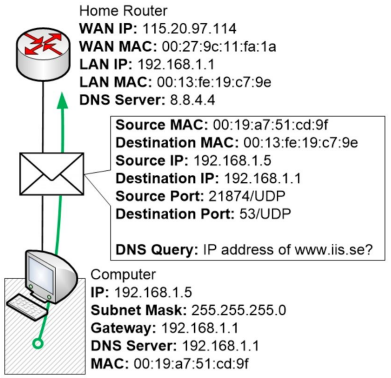
2. DNS
   e. ARP reply from router





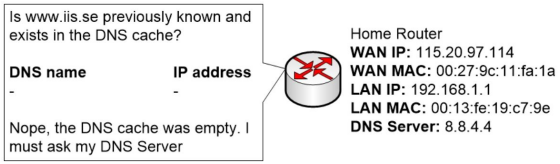

# How does Internet work?

2. DNS
   f. Send off DNS query



# How does Internet work?

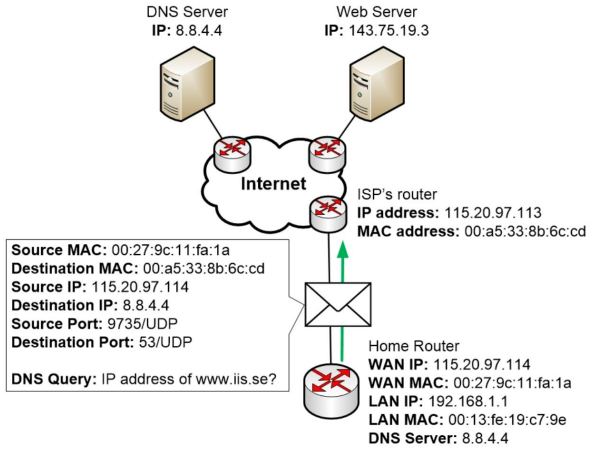
2. DNS
   g. Home router checks its DNS cache






# How does Internet work?

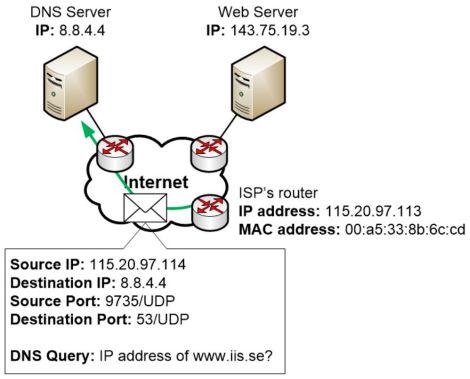
2. DNS
   h. Home router prepares and sends away its DNS query






# How does Internet work?

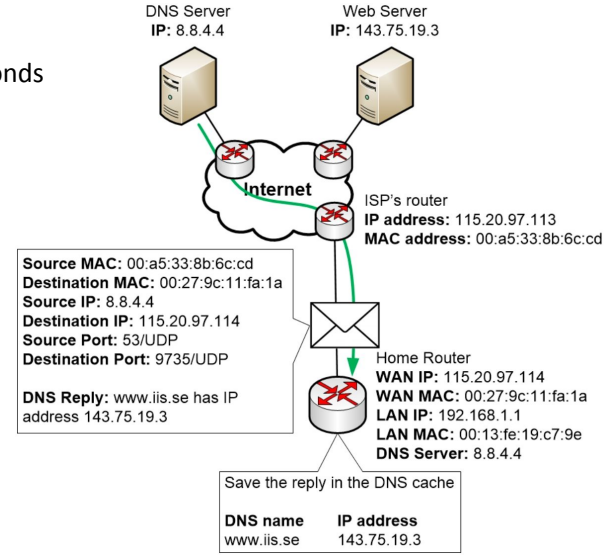
2. DNS
   i. DNS query is routed over Internet





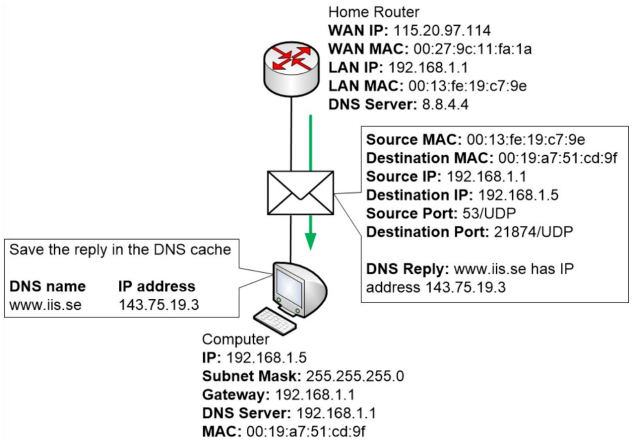

# How does Internet work?

2. DNS
   j. DNS server responds



# How does Internet work?

2. DNS
   k. Home router can send a DNS reply to computer





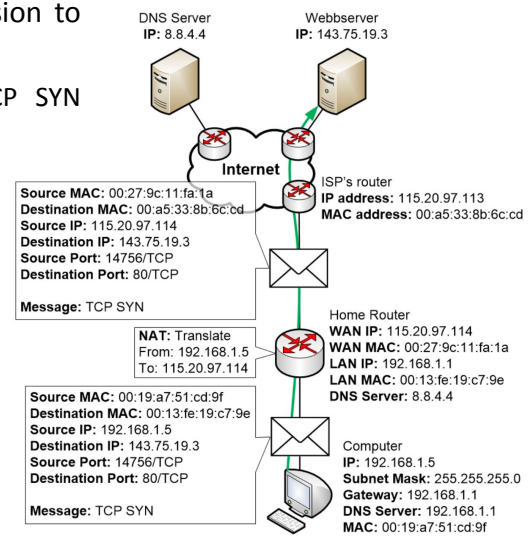

# How does Internet work?

3. Computer sets up a session to www.iis.se
   - Initialize TCP 3-way Handshake



# How does Internet work?

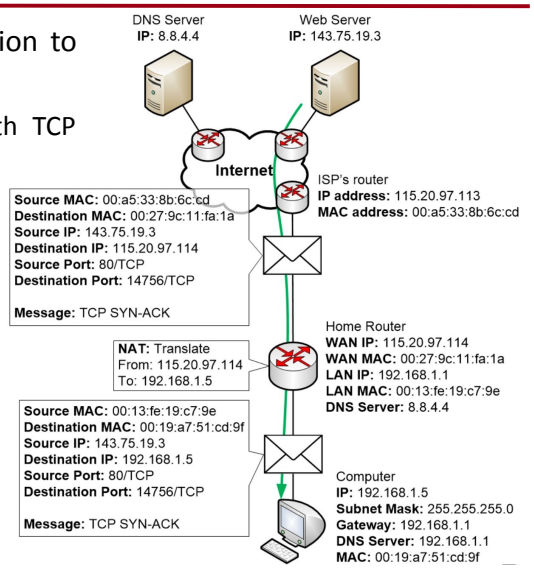
3. Computer sets up a session to www.iis.se
   a. Computer sends a TCP SYN message





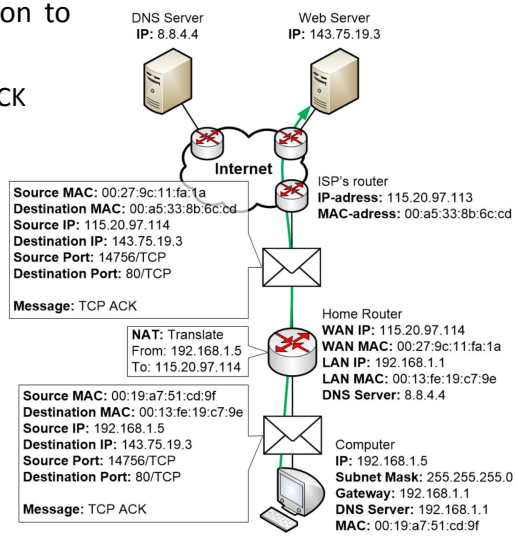

# How does Internet work?

3. Computer sets up a session to www.iis.se
   b. Web server replies with TCP SYN-ACK



## How does Internet work?

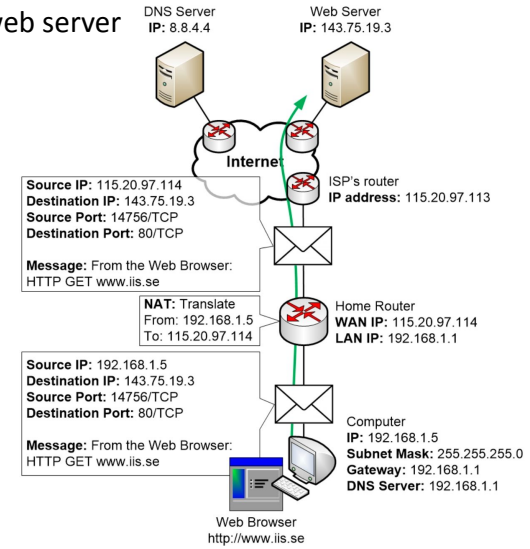
3. Computer sets up a session to www.iis.se
   c. Computer sends a TCP ACK






## How does Internet work?

4. Web browser talks with web server






## Summary and conclusions

- Growth/innovation vs create/exacerbate tensions
  - Does Internet design prevent misuse?
    - Which of following is true
      - (A) When connecting to network, individual endpoints can only use addresses given to them
      - (B) Individual endpoints can “spoof” any IP address



## Summary and conclusions

- Growth/innovation vs create/exacerbate tensions
  - Central authority IANA assigns unique IP address blocks to networks
    - Which of following is true
      - (A) Networks can only announce assigned addresses
      - (B) Networks can spoof any address

## Summary and conclusions

- Growth/innovation vs create/exacerbate tensions
  - Does Internet provide reliable packet delivery?
    - Which of following is true
      - (A) Yes, it's necessary for protocols like HTTP that require in-order streams
      - (B) No, packets can be arbitrarily dropped or reordered



Thank you!

# Software Defined Networks

Prof. Ankit Gangwal

Centre for Security, Theory, and Algorithmic Research (CSTAR),
IIIT Hyderabad, India.

# Agenda

## Part I : Traditional Networks

- Introduction
- Network Devices
- Shortcomings & Challenges

## Part II : Software Defined Networks

- Concept
- Decoupling of Control & Data Plane
- Advantages
- Architecture
  1. Infrastructure Layer
  2. Control Layer
  3. Application Layer
  4. Communication Interfaces
- Working

# Agenda

## Part III : OpenFlow Protocol

- Introduction
- Components of OpenFlow Network
  1. Controller
  2. Secure Channel
  3. Flow Table
- OpenFlow Protocol Messages
- Instruction & Action Set
- Packet Matching
- Pipeline Processing
- Table Miss
- Flow Removal
- A Flow Table Entry

# Agenda

## Part IV : SDN Resources, Issues & Use Cases

- Software Switch Implementation Compliant With OpenFlow
- Controller Implementation Compliant With OpenFlow
- SDN Programming Languages
- Issues in SDN
  1. Performance Issues
  2. Management Issues
  3. Security Issues
  4. Reliability Issues
- Use Cases of SDN
  - FlowVisor

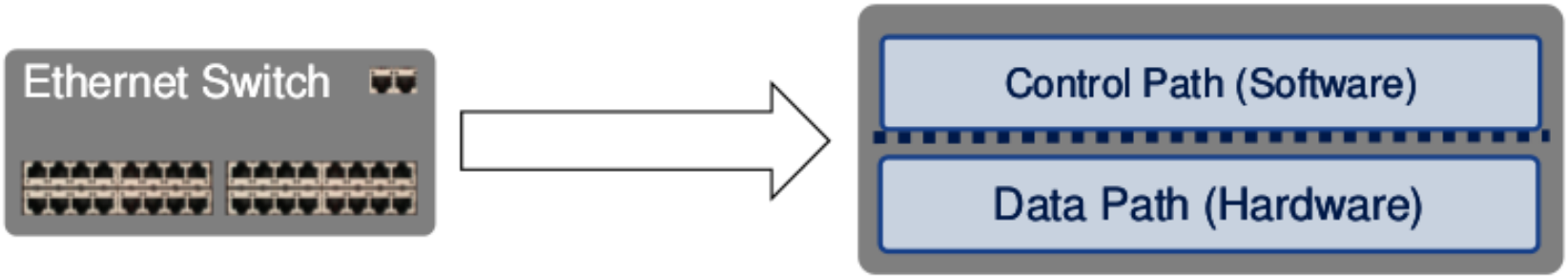
## Part V : References

# Traditional Networks

# Traditional Networks

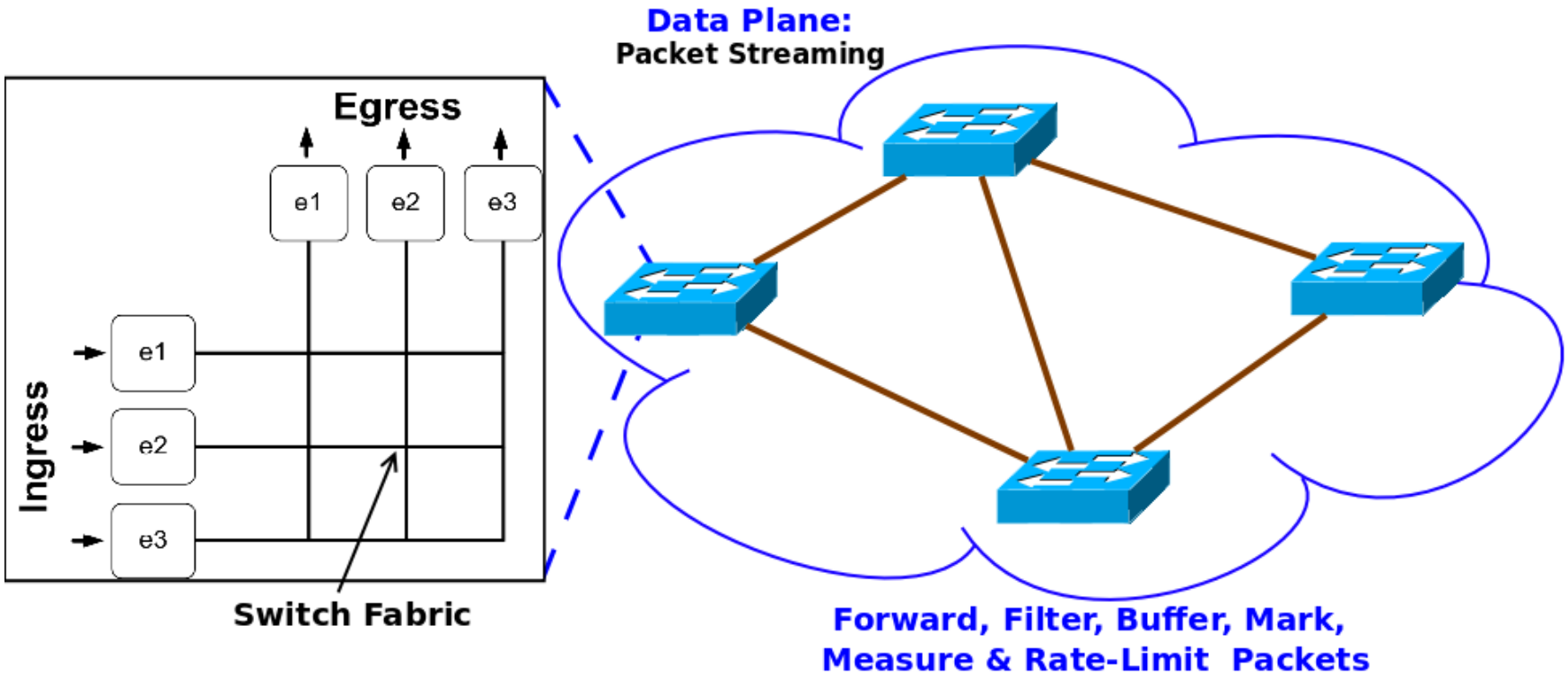
Network consists of :

- Data Plane,
- Control Plane,
- Management Plane.



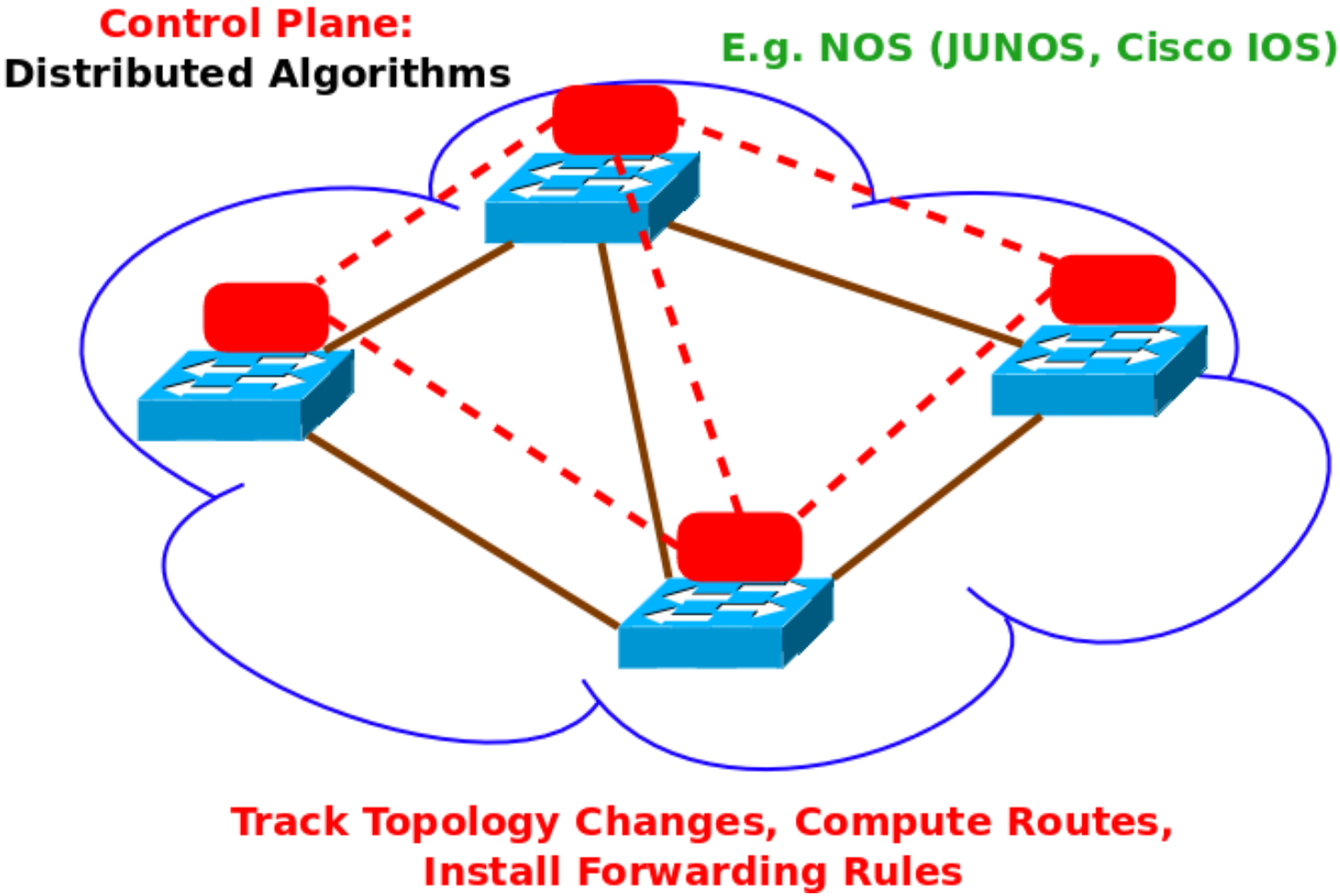
# Traditional Networks

## Data Plane



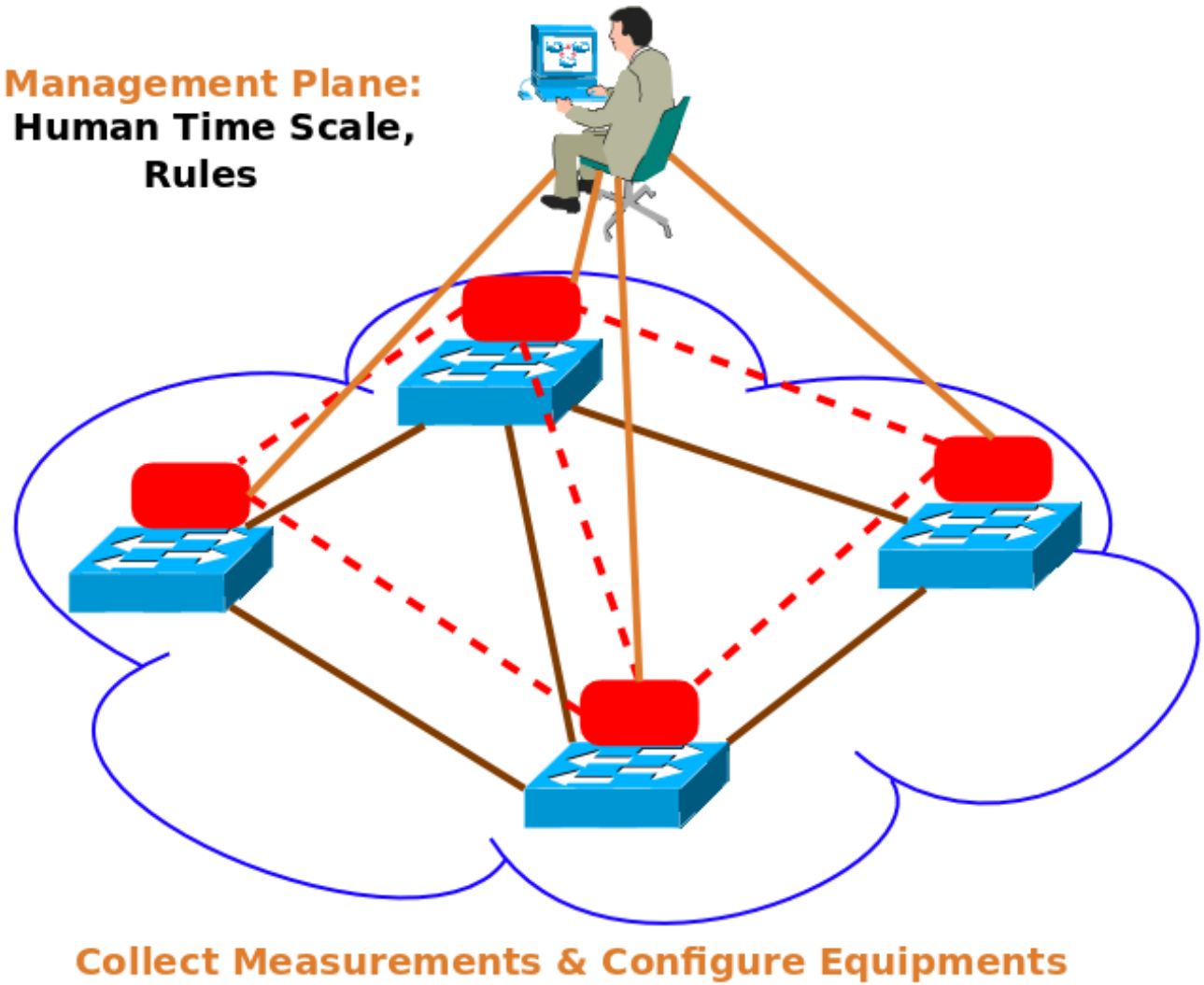
# Traditional Networks

## Control Plane



# Traditional Networks

## Management Plane



# Traditional Networks

**Network Devices:**

- ✓ Hardware + Operating Systems + Applications.
- ✓ Built into an **OPAQUE** box.
- ✓ Mix & match not allowed.

# Traditional Networks

**Results:**

- x Very high **effective** cost of box.
  - Deprivations due to replacements, to support newer functionality.
- x Networks have remained the same.
  - **Un-Programmable** by owner.
- x Complete vendor dependence.
  - **Innovations** are limited to vendors or their partners.

# Traditional Networks

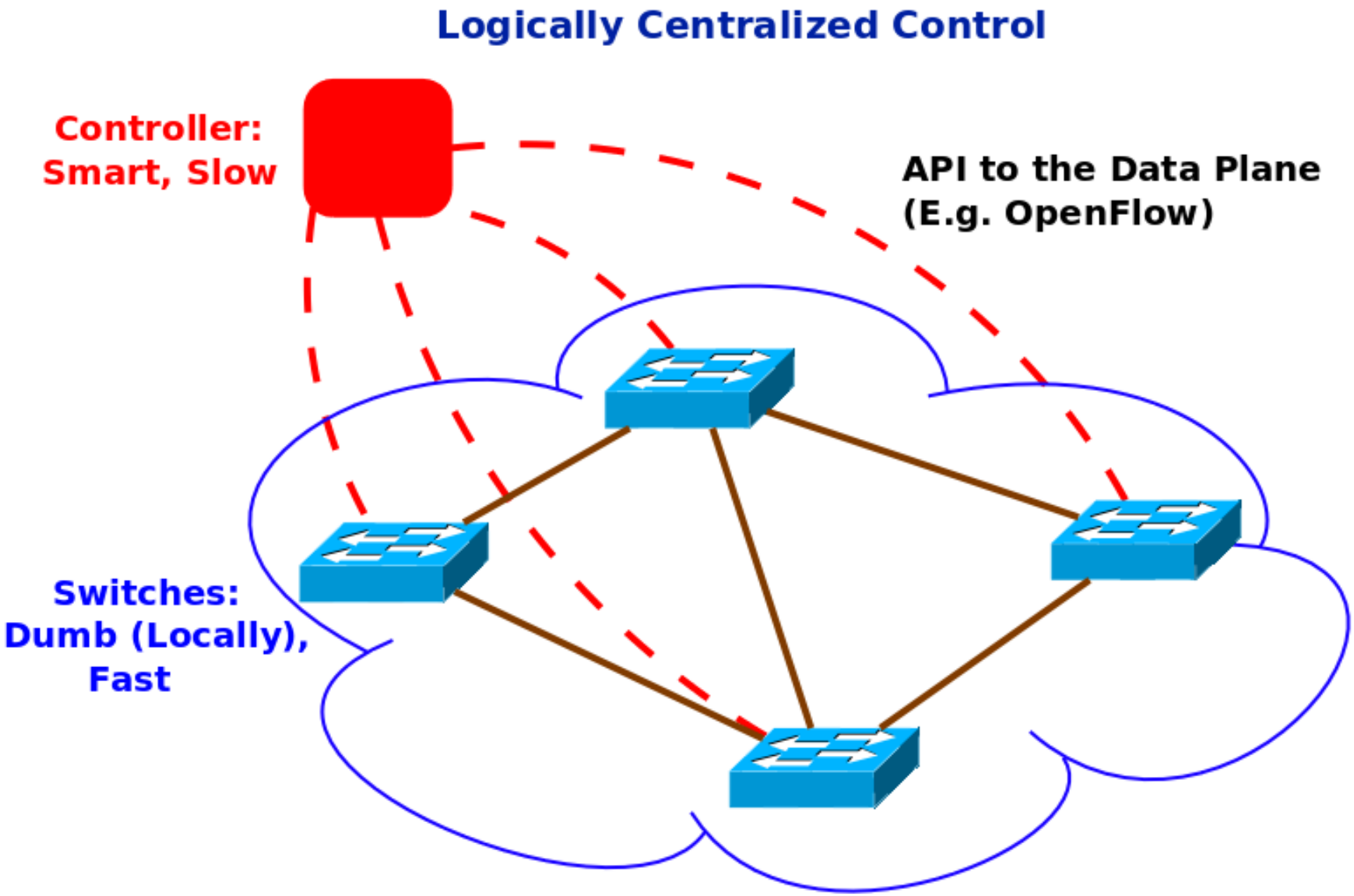
**Challenges:**

- Long hardware fabrication cycles.
- Network management is still complex.
- Testing new protocols in real network.
  - Touching the box is not allowed by IT !

# Software Defined Networks

# Software Defined Networks

## SDN = Dumb Switches + Smart Controller



# Software Defined Networks

## SDN decouples **CONTROL** and **DATA** Plane.

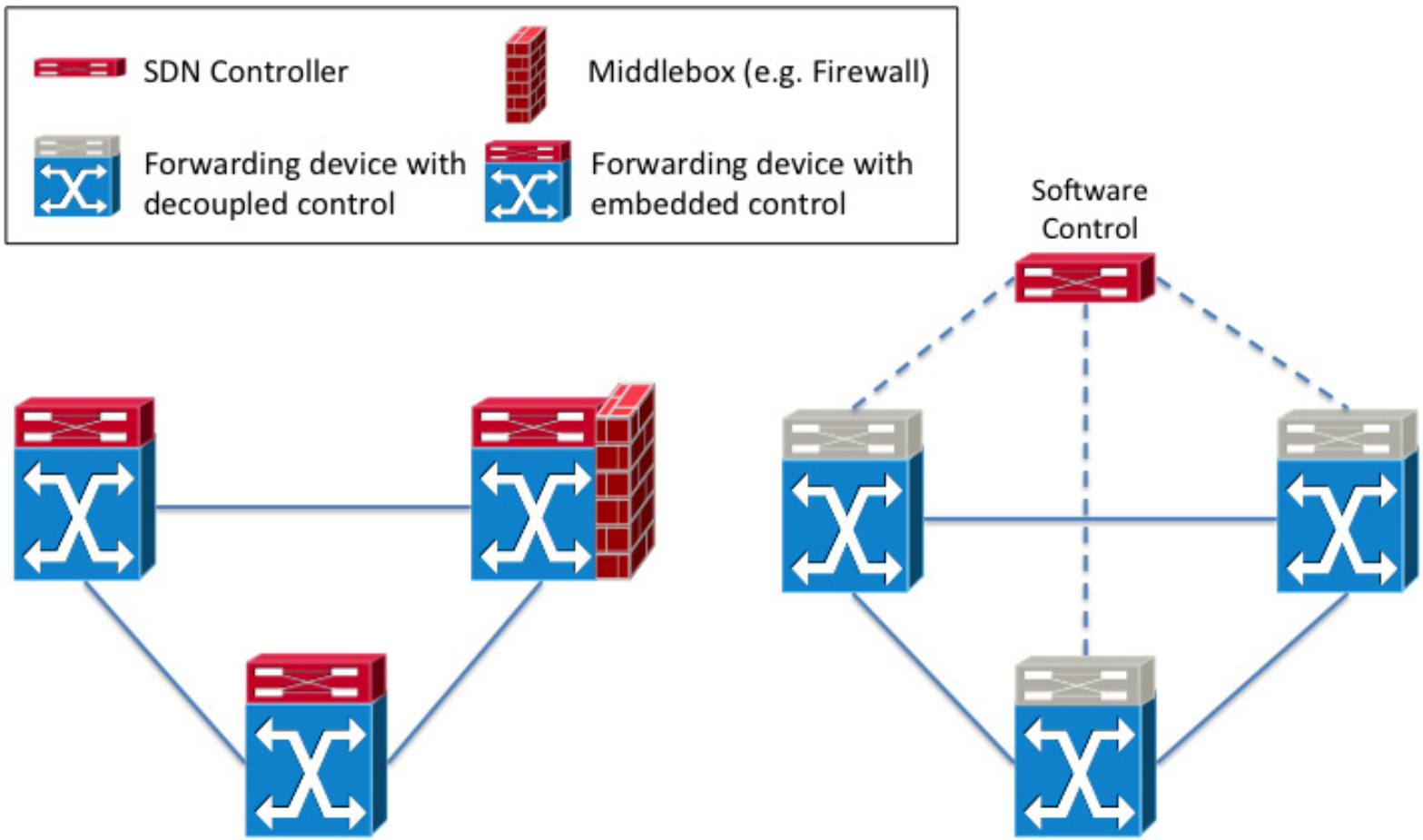
### Data Plane

- Hardware functionality.
- Forwards packet.

### Control Plane

- Creates routing (forwarding) table.
- Can sit **out** of the box.
- Protocol used: **OpenFlow** [1].

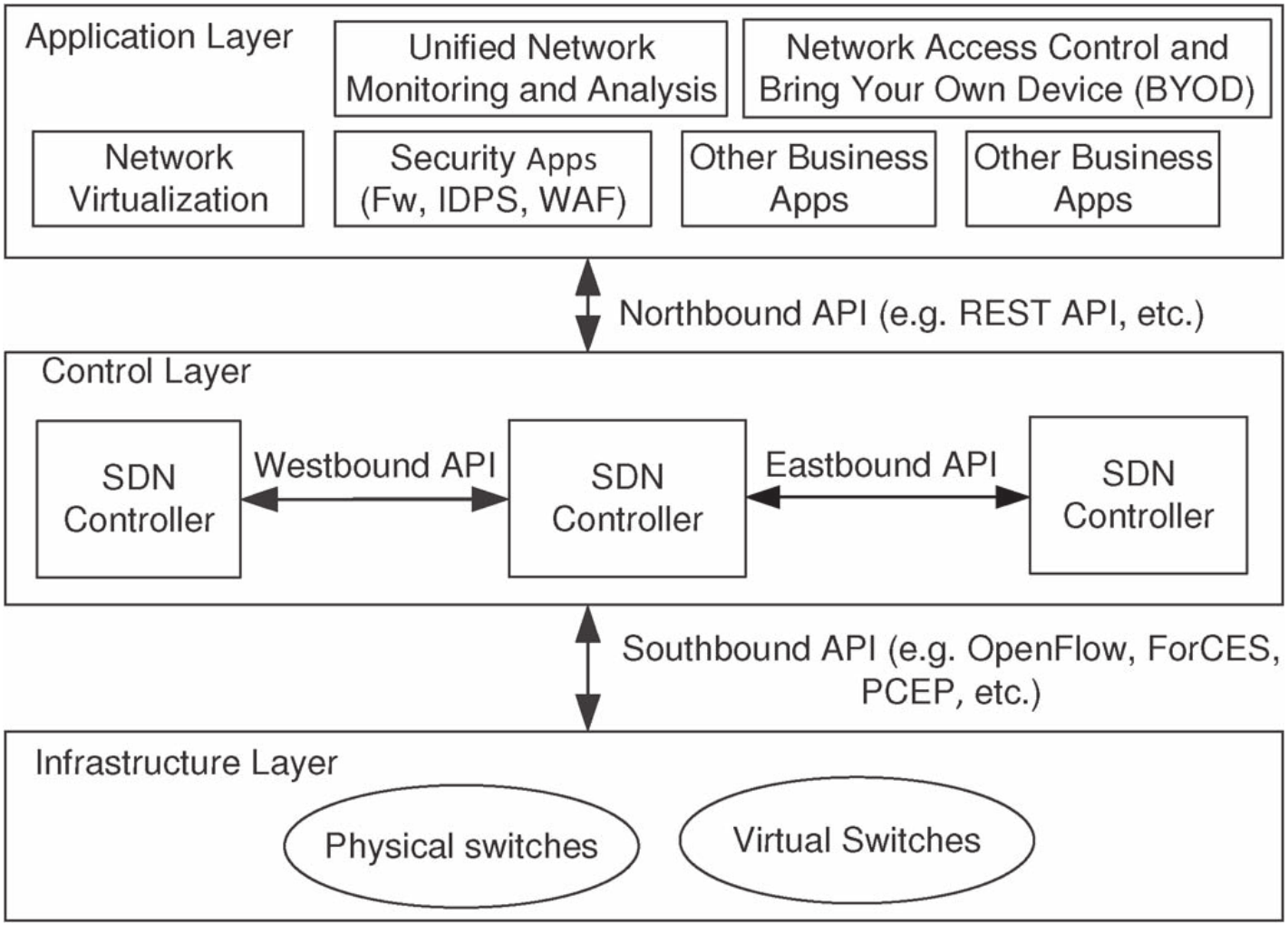
# Decoupling of CONTROL and DATA Plane



# SDN Benefits

- ~~Long hardware fabrication cycles.~~
  - ✓ Speed-to-market: No hardware fabrication cycles.
  - ✓ Fast upgrades.
- ~~Network management is still complex.~~
  - ✓ More flexibility with programmability.
  - ✓ Ease of customization and integration with other software applications.
  - ✓ Program a network V/s Configure a network.
- ~~Testing new protocols in real network.~~
  - ✓ Facilitate innovation in Network.
  - ✓ Independent innovations at each layer.

# SDN Architecture



- Also known as the **Data Plane**.
- It consists mainly of Forwarding Elements (FEs) including physical and virtual Switches.
- FEs are accessible via an open interface.
- Allows packet switching and forwarding.
- **Every FE must support Southbound APIs (The reason why traditional FE can't be used as per SDN standards).**

- Also known as the **Control Plane**.
- It consists of a set of software-based SDN Controllers.
- Provides a consolidated control functionality.
- Supervises the network forwarding behaviour through an open interface.
- Three communication interfaces allow the Controllers to interact: **Southbound, Northbound, East/Westbound**.

- **Southbound Interface**
  - Allows the Controller to interact with the forwarding elements in the infrastructure layer.
  - E.g. OpenFlow, a protocol maintained by ONF.

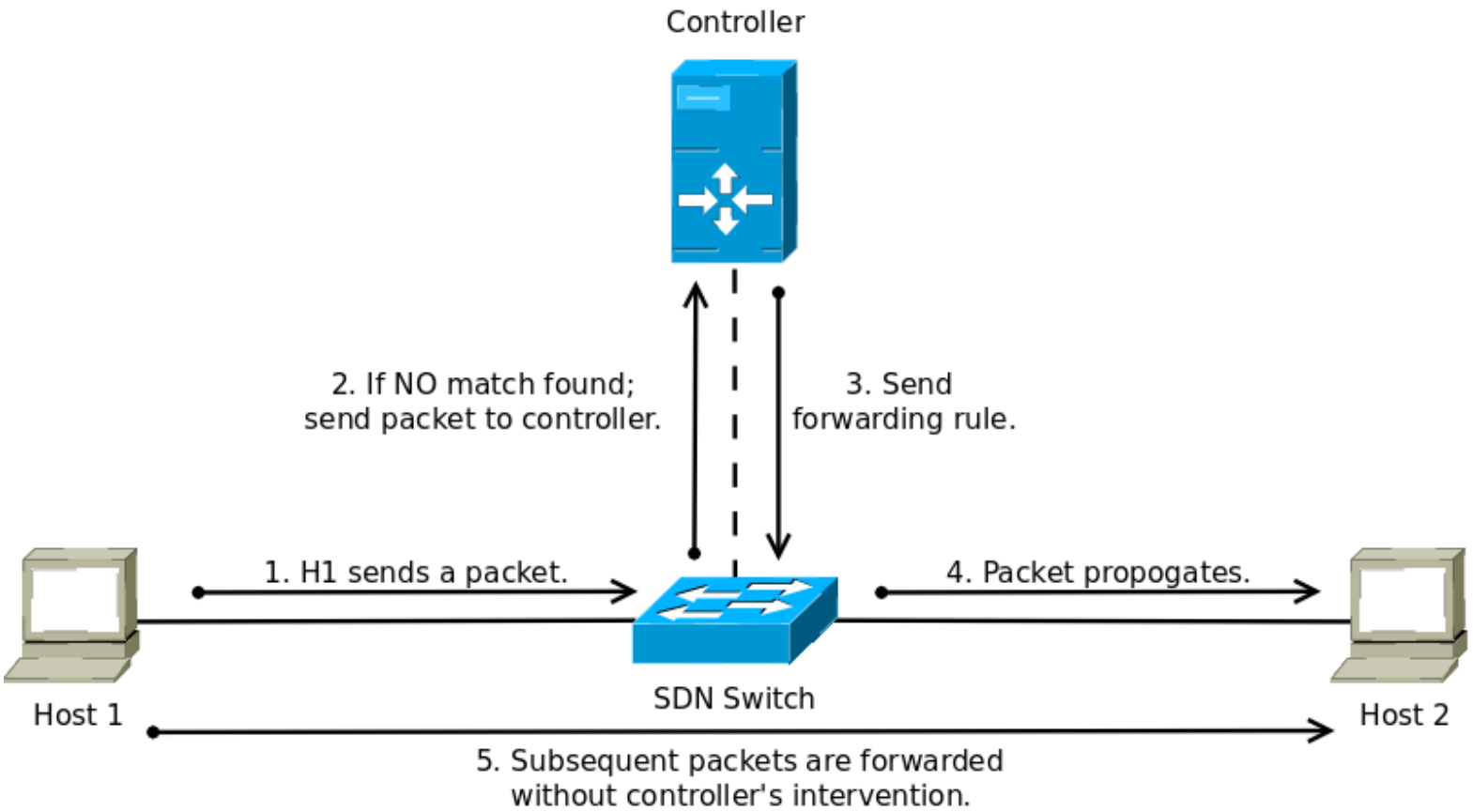
- **Southbound Interface**
  - Allows the Controller to interact with the forwarding elements in the infrastructure layer.
  - E.g. OpenFlow, a protocol maintained by ONF.
- **Northbound Interface**
  - Enables the programmability of the Controllers.
  - E.g. REpresentational State Transfer **(REST)**-based APIs.

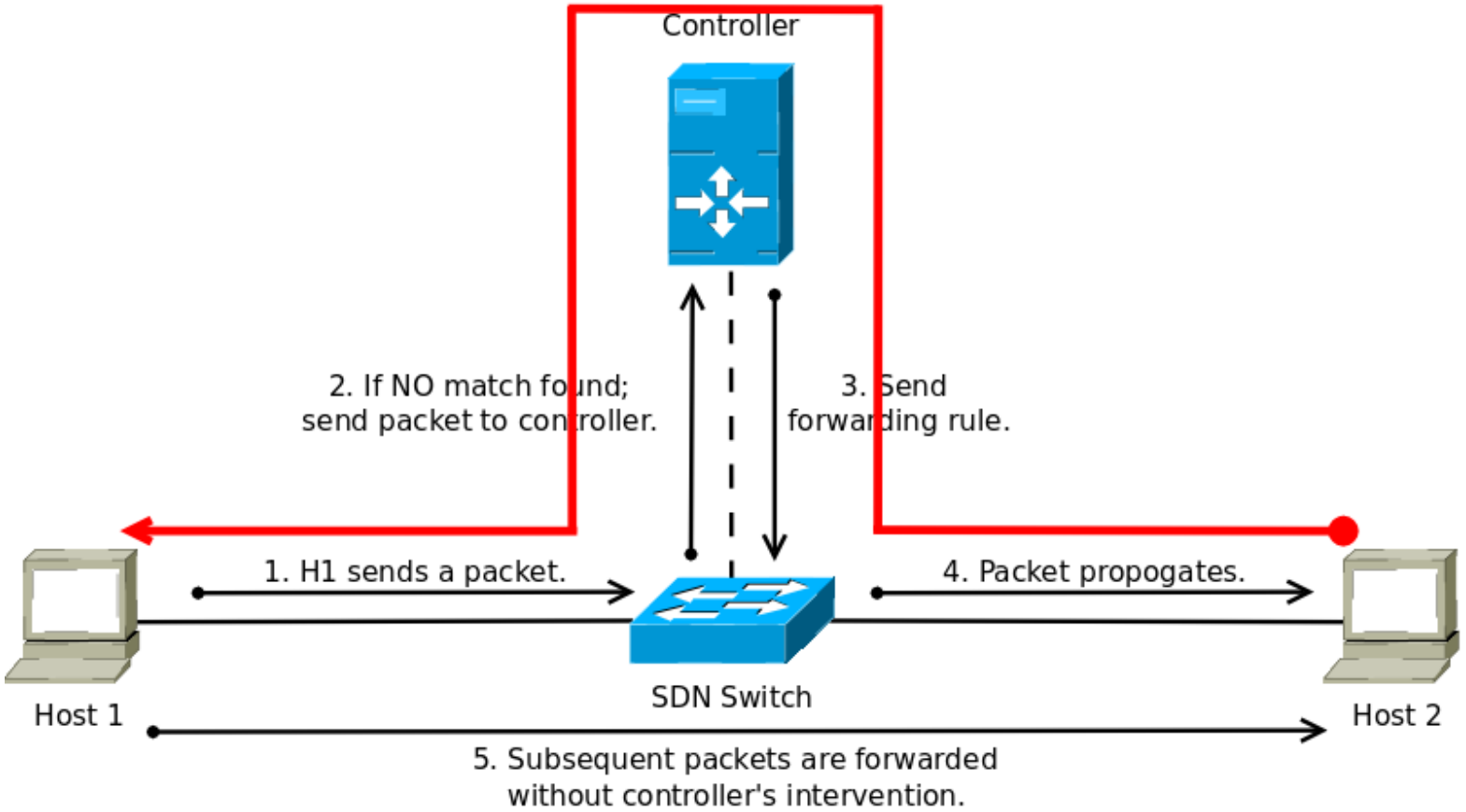
- **Southbound Interface**
  - Allows the Controller to interact with the forwarding elements in the infrastructure layer.
  - E.g. OpenFlow, a protocol maintained by ONF.
- **Northbound Interface**
  - Enables the programmability of the Controllers.
  - E.g. REpresentational State Transfer **(REST)**-based APIs.
- **East/Westbound Interface**
  - It is an envisioned communication interface.
  - Which is not currently supported by an accepted standard.
  - It is mainly meant for enabling communication between federations of Controllers to synchronize state for high availability.

# SDN Architecture

## Application Layer

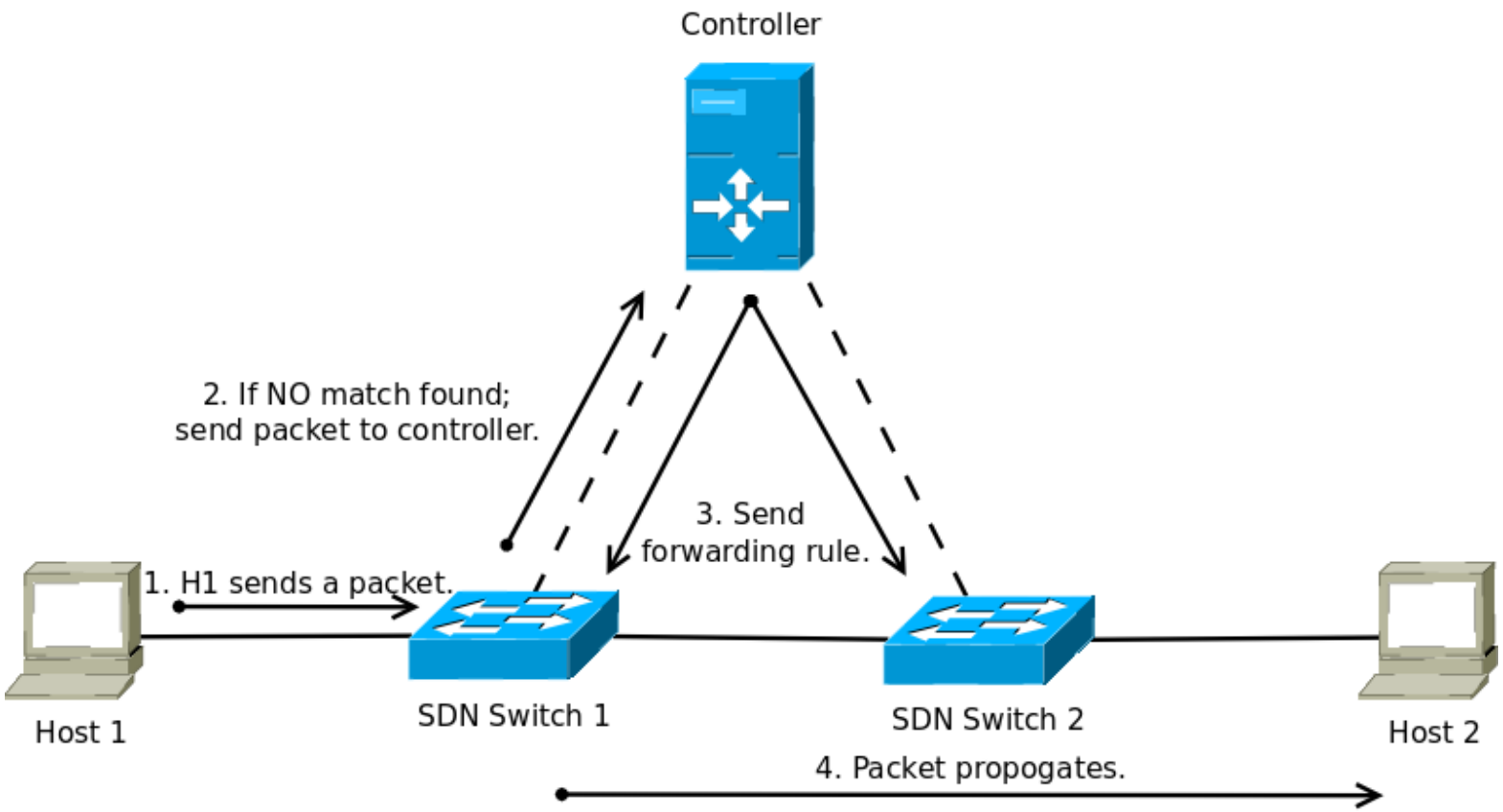
- It mainly consists of the end-user business applications.
- It consume the SDN communications and network services.
- E.g. network visualization and security business applications.


Controller
2. If NO match found;
send packet to controller.
3. Send
forwarding rule.
1. H1 sends a packet.
4. Packet propogates.
Host 1
SDN Switch
Host 2
5. Subsequent packets are forwarded
without controller's intervention.


Controller
2. If NO match found;
send packet to controller.
3. Send
forwarding rule.
1. H1 sends a packet.
4. Packet propogates.
Host 1
SDN Switch
Host 2
5. Subsequent packets are forwarded
without controller's intervention.

# SDN

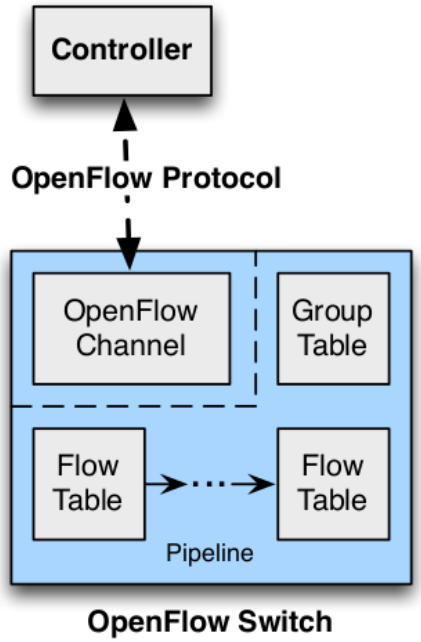
## Working - Use Global View



# OpenFlow Protocol

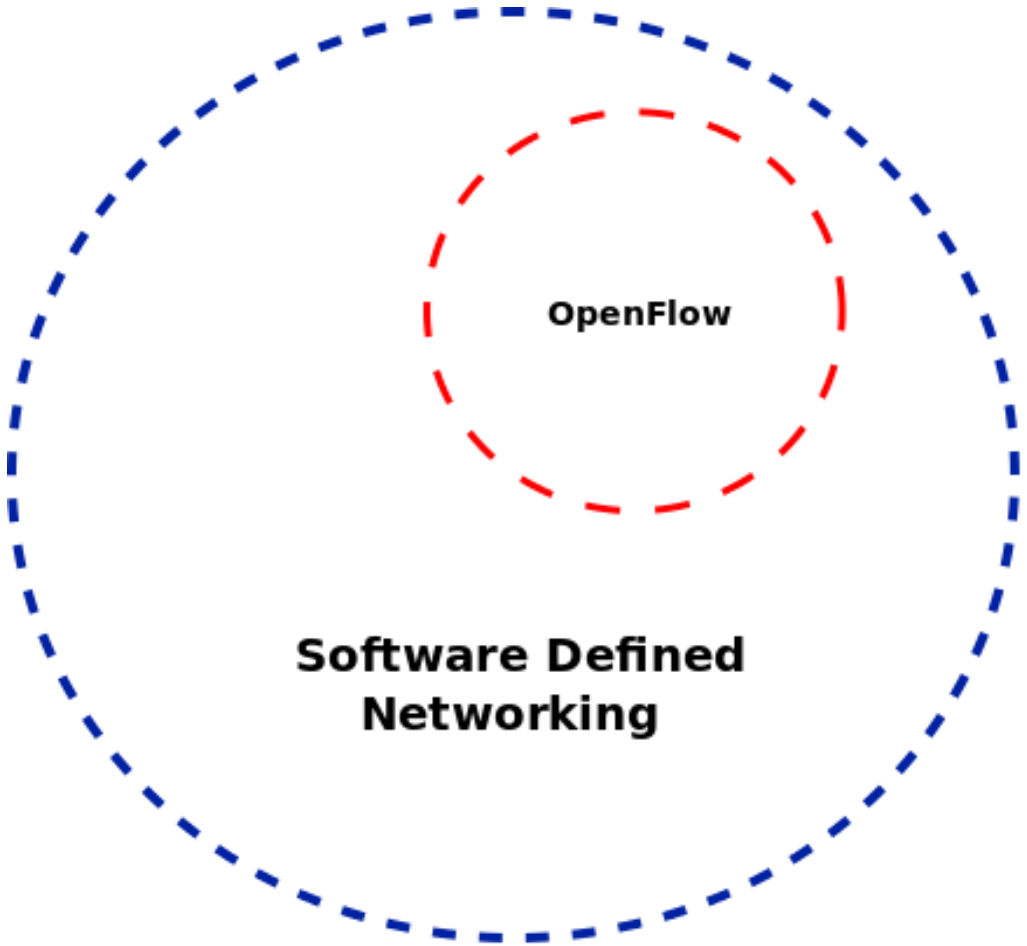
# OpenFlow Protocol

- OpenFlow is a **Layer 2** communication protocol.
- OpenFlow is the first standard communications interface defined between the **Control** and **Forwarding** layers of SDN architecture.
- OpenFlow allows **direct access** & **manipulation** of the Forwarding Plane of network devices such as Switches and Routers.

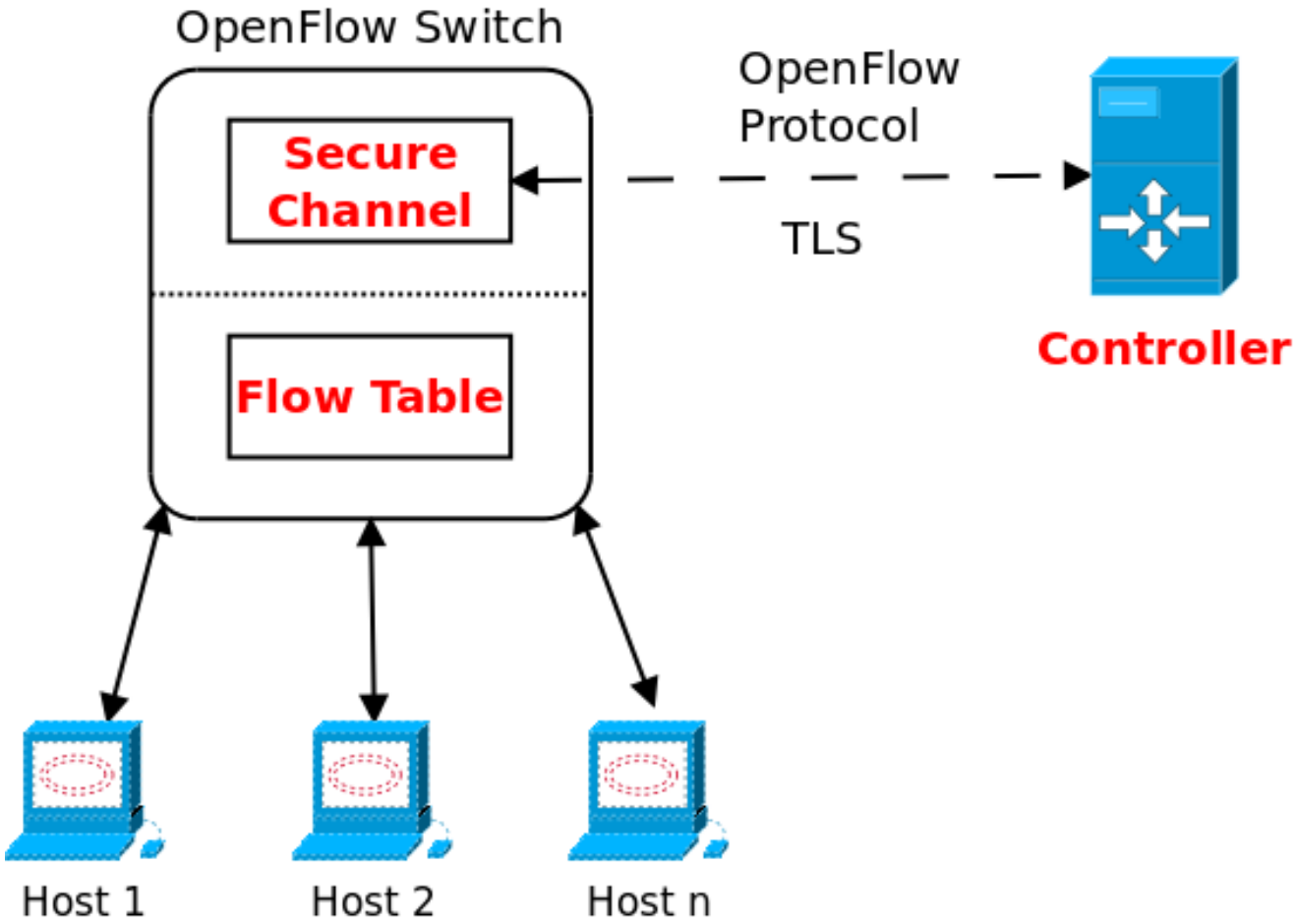


# OpenFlow **does not** equal SDN

- General Myth : OpenFlow is SDN
- Reality : OpenFlow is one flavour, or a subset, of SDN



# Components of OpenFlow Network



# Components of OpenFlow Network

1. **Controller**
   - Defines the forwarding policies (actions to take for each flow) based on information acquired from various network devices.
     - SDN relies on Controller. Any SDN must have at-least one Controller.

# Components of OpenFlow Network

1. **Controller**
   - Defines the forwarding policies (actions to take for each flow) based on information acquired from various network devices.
     - SDN relies on Controller. Any SDN must have at-least one Controller.
2. **Secure Channel**
   - Connects a Switch to the Controller, allowing commands and packets to be sent between a Controller & the Switch through OpenFlow protocol.

# Components of OpenFlow Network

1. **Controller**
   - Defines the forwarding policies (actions to take for each flow) based on information acquired from various network devices.
     - SDN relies on Controller. Any SDN must have at-least one Controller.
2. **Secure Channel**
   - Connects a Switch to the Controller, allowing commands and packets to be sent between a Controller & the Switch through OpenFlow protocol.
3. **Flow Table**
   - Consists of a **Flow Entry** & an **Action** associated with each flow entry to tell the Switch how to process the flow.

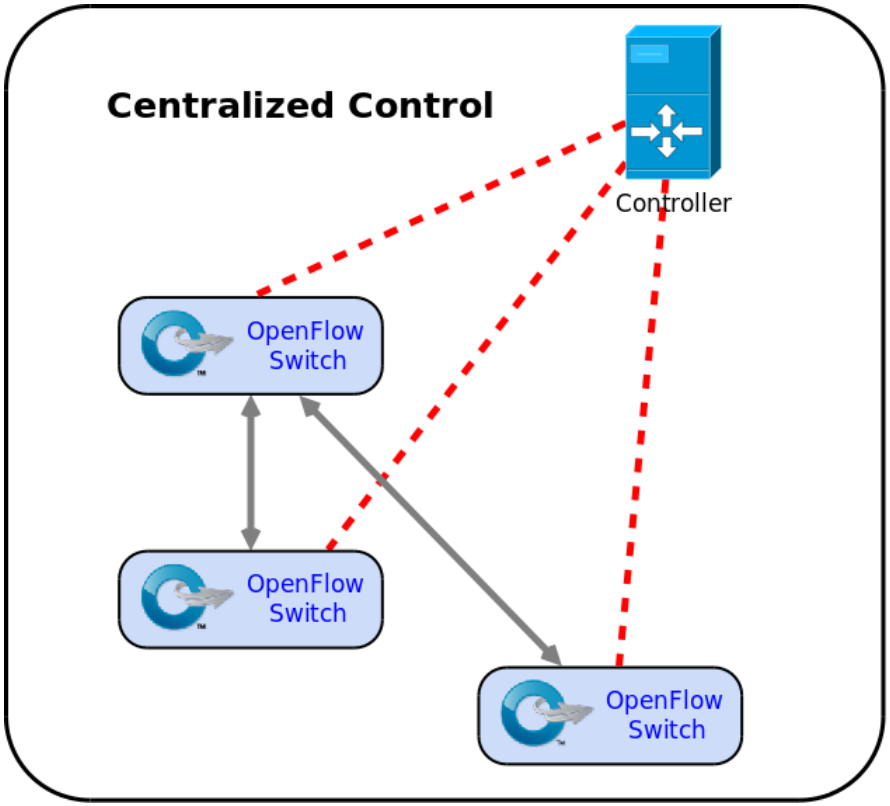
# Components of OpenFlow Network

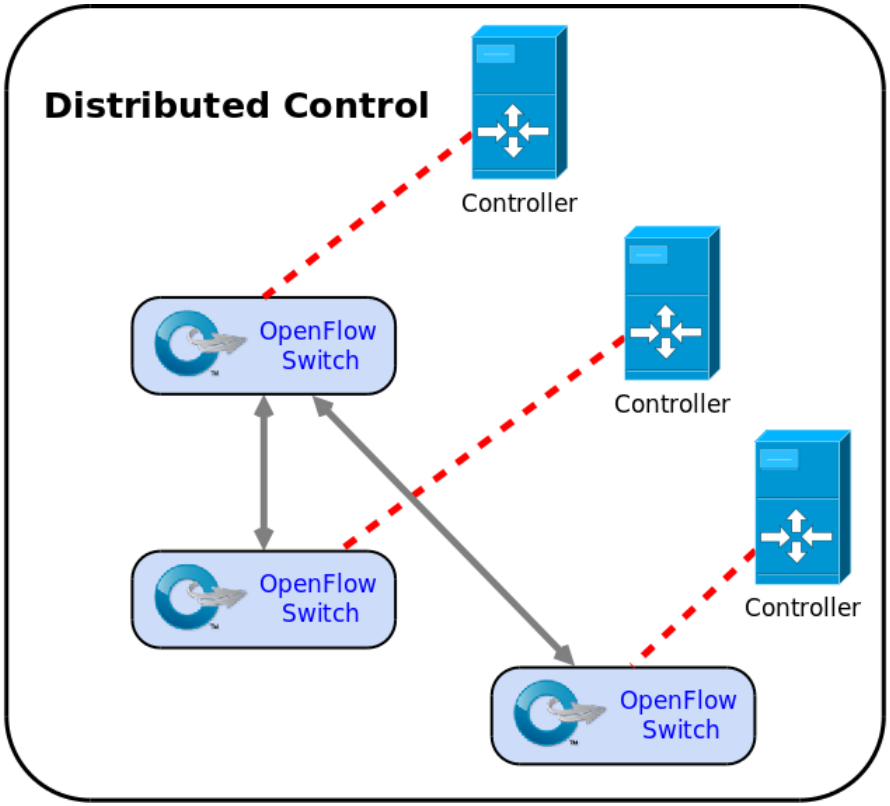
1. **Controller**
   - Defines the forwarding policies (actions to take for each flow) based on information acquired from various network devices.
     - SDN relies on Controller. Any SDN must have at-least one Controller.
2. **Secure Channel**
   - Connects a Switch to the Controller, allowing commands and packets to be sent between a Controller & the Switch through OpenFlow protocol.
3. **Flow Table**
   - Consists of a **Flow Entry** & an **Action** associated with each flow entry to tell the Switch how to process the flow.

* **Network Operating System (NOS)** is the set of Controllers that forms an execution environment.

# Controller

## Centralized V/s Distributed



## Measuring Controller Performance

- **Throughput,** measuring the maximum flow set-up rate that a Controller can maintain.
- **Latency,** measuring the Controller's request processing time under low-load conditions.

## Handling Controller Failure

- Use a backup Controller.
- Use hybrid Switches. They start working as legacy Switch upon failure of the Controller.

# Secure Channel

- SC is the **Interface** that connects each OpenFlow Switch to Controller.
- A Controller configures and manages the Switch, receives events from the Switch & send packets out the Switch via this interface.
- SC establishes and terminates the connection between OpenFlow Switch & the Controller using Connection Setup & Connection Interruption procedures.
- The SC connection is a TLS (or TCP) connection. Switch and Controller mutually authenticate by exchanging certificates signed by a site-specific private key.

1. **Controller-to-Switch** - Initiated by the Controller & used to directly manage or inspect the state of the Switch.
   - *Features, Config, Modify State, Read-State, Packet-Out, Barrier.*
2. **Asynchronous** - Asynchronous messages are sent without the Controller soliciting them, from a Switch.
   - *Packet-in, Flow Removed / Expiration, Port-status, Error.*
3. **Symmetric** - Symmetric messages are sent without solicitation, in either direction.
   - *Hello, Echo.*

# Flow Table

## Instructions & Action Set

- Each flow entry contains a set of instructions that are executed when a packet matches the entry.
- **Instructions** contain either a set of actions to be added to the action set or modify pipeline processing.
- **Action Set** contains a list of actions to be applied immediately to the packet.
- An Action set is always associated with every entry. By default it is empty.
- A flow entry modifies action set using **Write-Action** or **Clear-Action** instruction.

# Flow Table

## Instructions

List of Instructions to modify action set:

1. **Apply Actions**
   - Apply the specified actions immediately.
2. **Clear Actions**
   - Clear all the actions in the set immediately.
3. **Write Actions**
   - Merge the specified actions to the current set.
4. **Write Metadata**
   - Write the meta data field with the specified value.
5. **Goto -Table**
   - Indicated the next table in the processing pipeline.

Actions are of two type:

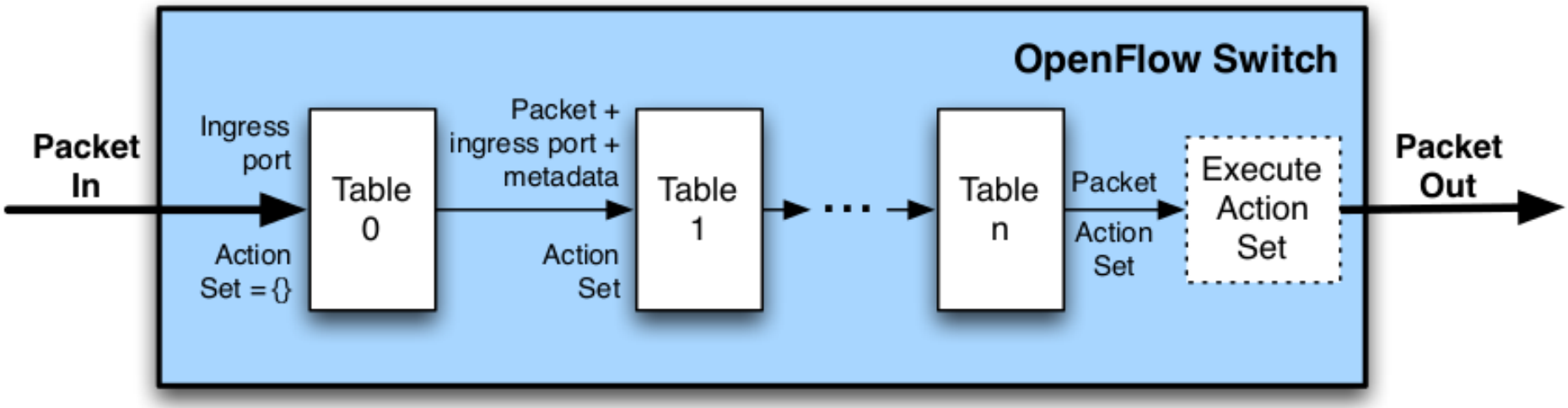
1. **Required Actions**
   - Output - Forward a packet to the specified port.
   - Drop
2. **Optional Actions**
   - Set-Queue
   - Push/Pop Tag
   - Set-Field

# Flow Table

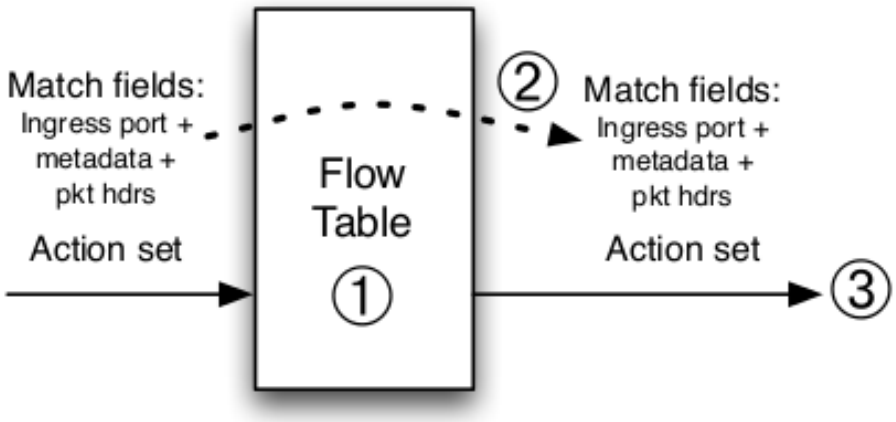
## Packet Matching

- OpenFlow pipeline contains multiple flow tables starting with **Table 0**.
- Each flow table contains one or more flow entries. Matching starts with the first flow entry.
- If a Match is found :
  - Instructions associated with flow entry are executed.
  - Instruction may direct the packet to the next flow table in pipeline.
  - When processing stops, the associated action set is applied.
- Instructions describe packet forwarding, packet modification and pipeline processing.

**Packets are matched against multiple tables in the pipeline:**



**Per-table packet processing:**



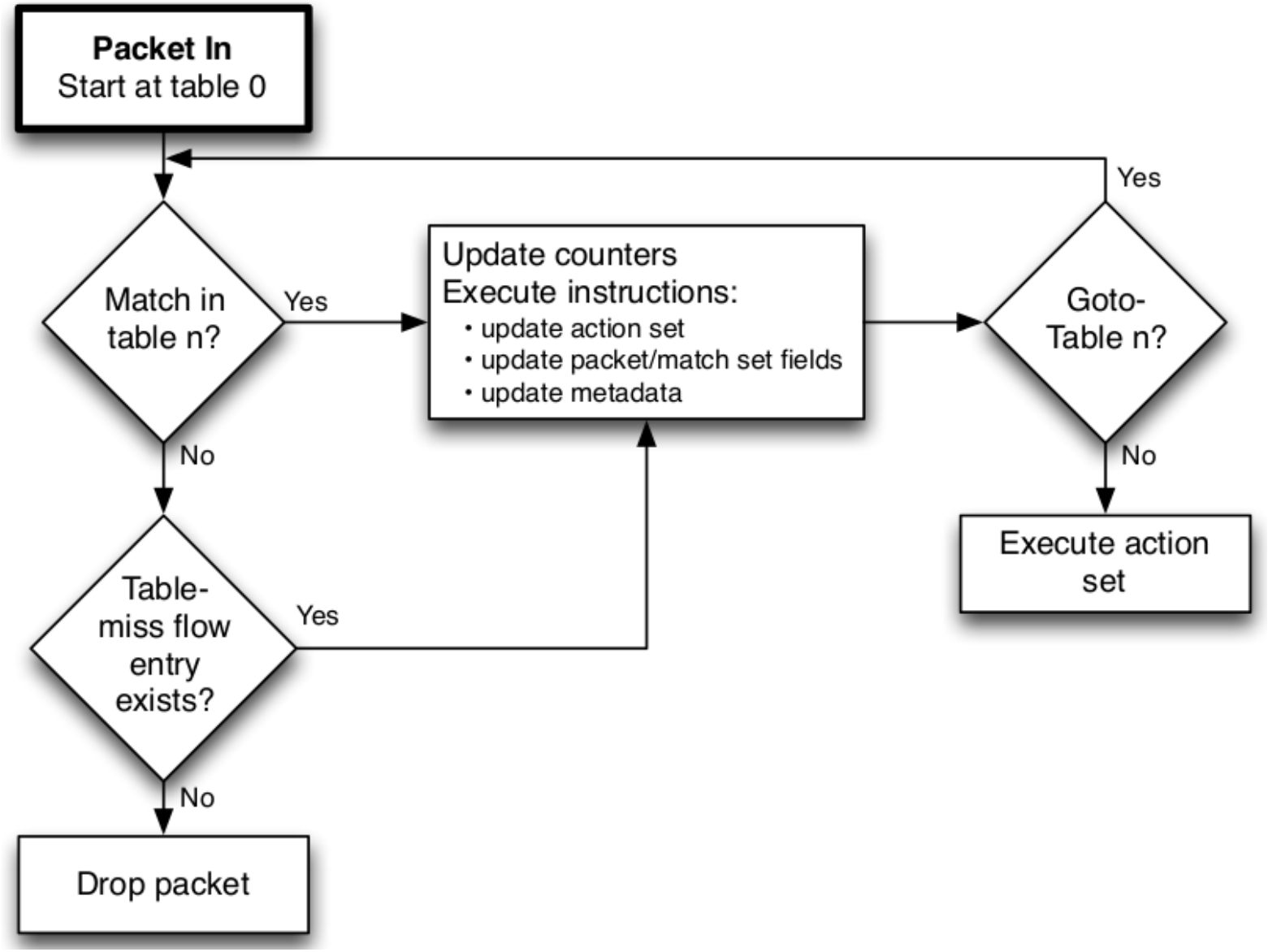

① Find highest-priority matching flow entry

② Apply instructions:
- i. Modify packet & update match fields (apply actions instruction)
- ii. Update action set (clear actions and/or write actions instructions)
- iii. Update metadata

③ Send match data and action set to next table

# Flow Table

## Flowchart detailing packet flow through an OpenFlow Switch



# Flow Table

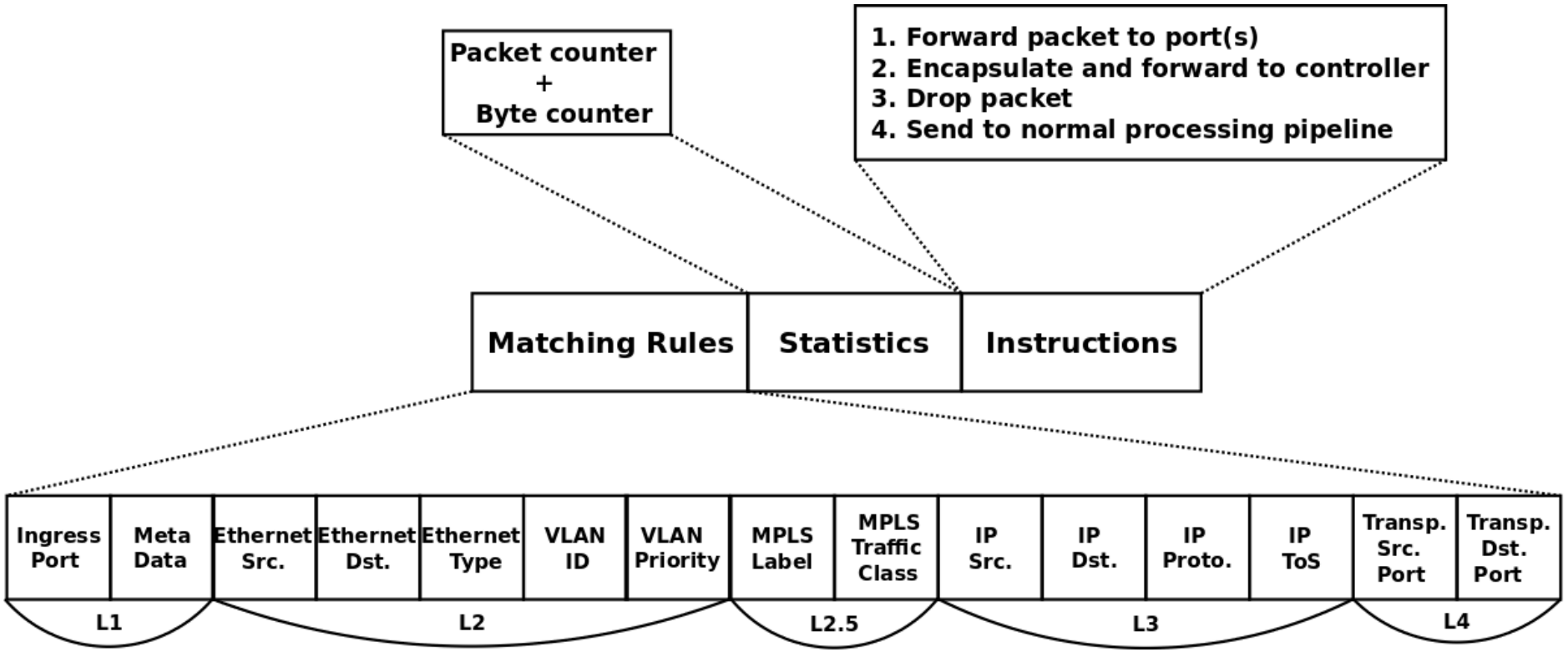
## Table Miss

- 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.
  - E.g. send packets to the Controller, direct packets to a subsequent table or drop packets.
- If a table is missed & a table-miss entry exists then it causes **penalty** in form of **delay**.

Flow entries are removed either

1. At the request of the Controller,
2. Or via the Switch flow expiry mechanism.
   - **idle_timeout** causes the flow entry to be removed when it has matched no packets in the given number of seconds.
   - **hard_timeout** causes the flow entry to be removed after the given number of seconds, regardless of how many packets it has matched.

# A Flow Table Entry



# SDN Resources, Issues & Use Cases

# Software Switch Implementation Compliant With OpenFlow

| Software Switch | Implementation | Overview | Version |
|---|---|---|---|
| Open vSwitch [2] | 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 [3] | C | Turns a commercial wireless router or Access Point into an OpenFlow-enabled Switch. | v1.0 |
| ofsoftswitch13 [4] | C/C++ | OpenFlow 1.3 compatible user-space software Switch implementation. | v1.3 |
| Indigo [5] | C | Open source OpenFlow implementation that runs on physical Switches and uses the hardware features of Ethernet Switch ASICs to run OpenFlow | v1.0 |

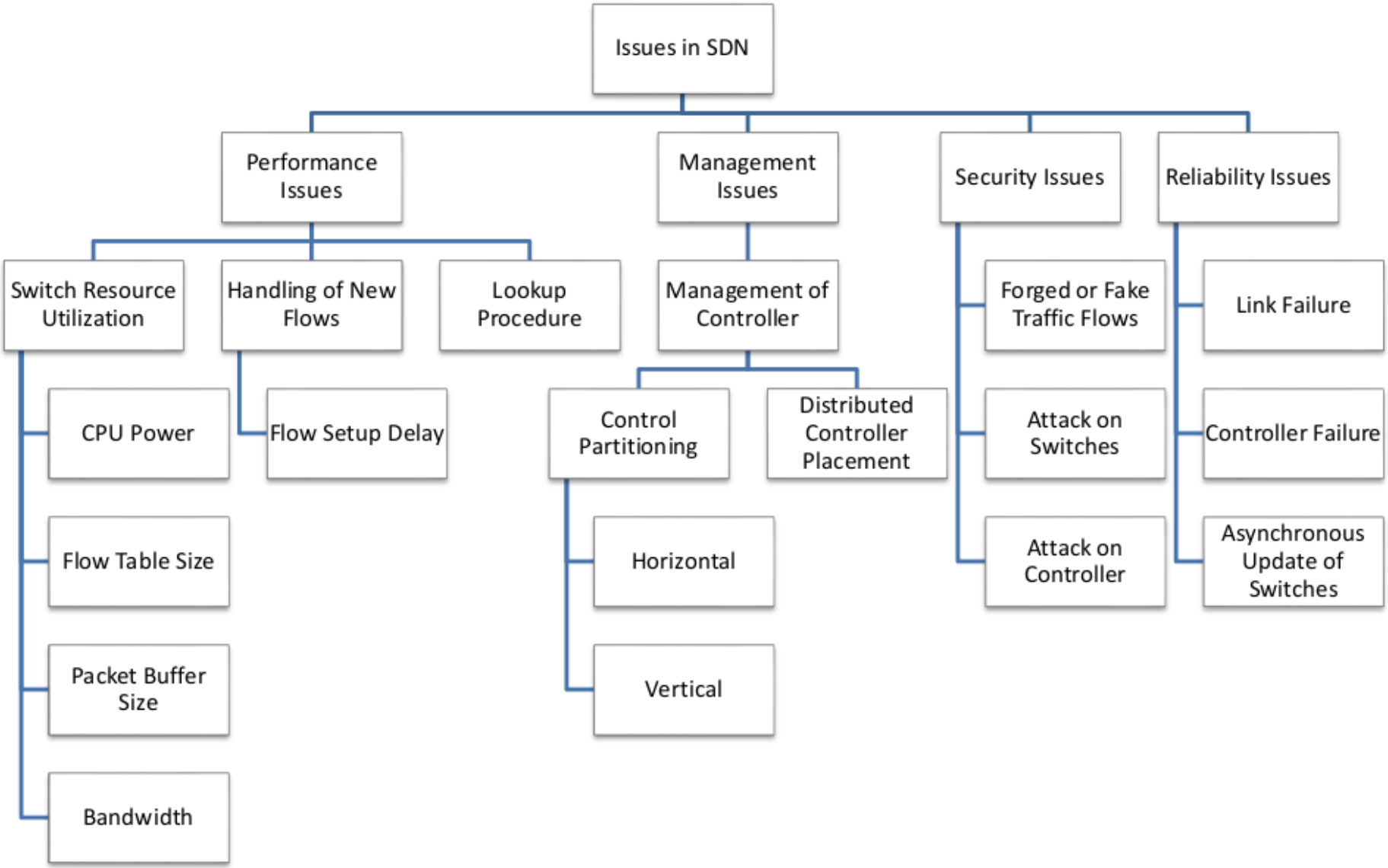
# Controller Implementation Compliant With OpenFlow

| Controller | Implementation | Open Source | Developer |
|---|---|---|---|
| POX [6] | Python | Yes | Nicira |
| NOX [7] | Python/C++ | Yes | Nicira |
| Floodlight [8] | Java | Yes | BigSwitch |
| Ryu [9] | Python | Yes | NTT, OSRG group |
| Beacon [10] | Java | Yes | Stanford |
| Trema [11] | Ruby/C | Yes | NEC |
| Maestro [12] | Java | Yes | Rice University |
| MUL [13] | C | Yes | Kulcloud |
| Jaxon [14] | Java | Yes | Independent Developers |
| Helios [15] | C | No | NEC |
| SNAC [16] | C++ | No | Nicira |
| NodeFlow [17] | JavaScript | Yes | Independent Developers |
| ovs-controller [2] | C | Yes | Independent Developers |
| Flowvisor [18] | C | Yes | Stanford/Nicira |
| RouteFlow [19] | C++ | Yes | CPqD |

# SDN Programming Languages

| **Framework** | **Level of Abstraction** | **Query Language** | **Implementation Language** | **Policies Type** |
|---|---|---|---|---|
| Frenetic [20],[21] | High | Yes | Python | Active |
| NetCore [22] | High | Yes | Python | Active |
| Nettle [23] | Low | No | Haskell | Active |
| FML [24] | High | No | Python/C++ | Passive |
| Procera [25] | High | No | Haskell | Active |

# Issues in SDN



# Performance Issues

## Switch Resource Utilization

1. CPU Power
   - Every flow is handled by the system CPU.
   - CPU is needed to encapsulate the packet to be transmitted to the Controller for flow setup through the Secure Channel.
   - Limited power of a Switch CPU can restrict the bandwidth between the Switch and the Controller.
2. Flow Tables Size
   OpenFlow Switches maintain complete visibility in a large OpenFlow network which results in increase in flow table size.

# Performance Issues

## Switch Resource Utilization

3. Packet Buffer Size
   Limited packet buffer size may lead to drop in packets and decrease in throughput.

4. Bandwidth Between Switch and Controller
   Limited bandwidth between Switch and Controller may lead to decrease in performance.

# Performance Issues

## Handling of New Flows

- Performance of Control Plane is determined by the number of new flows per second that the Controller can handle and the delay of a flow setup.
- Goals of OpenFlow was to keep the Data Plane simple and to delegate the control task to a logically centralized Controller.
- As a result, Switches consult the Controller frequently for instructions on how to handle incoming packets of new flows.
- This tends to **congest** Switch-Controller connections, which in turn adds latency to the processing of the first packets of a flow in the Switch buffer.

# Performance Issues

## Lookup Procedure

Two types of flow tables exist:

1. Hash table
   - Stored in Static RAM (SRAM) on the Switch.
   - This type of memory is **off-chip**, leading to increased lookup latencies.
2. Linear table
   - Stored on TCAM (Ternary Content Addressable Memory).
   - Located on Switch-chip, leading to decreased lookup latencies.

- In ordinary Switches, lookup mechanism is the main operation that is performed.
- In OpenFlow-enabled Switches, other operations especially the **"insert"** operation is considered that can lead to a higher power dissipation & a longer access latency.

# Management Issues

1. Distributed Controllers Placement
   Determining the number of the needed Controllers and their placement within the controlled domain is an issue.

2. Control Partitioning
   Refers to partitioning of the network into multiple controlled domains. It can be done in two ways:
   - **Horizontal**
     Multiple Controllers are organized in a flat Control Plane where each one governs a subset of the network Switches.
   - **Vertical**
     Controllers functionalities are organized vertically. Control tasks are distributed to different Controllers depending on criteria such as network view and locality requirements.

# Security Issues

1. Forged or faked traffic flows
   - To attack Switch/Controllers.
   - Can be triggered by faulty device or malicious user.
   - Aims to launch DoS/D-DoS attack against network devices.
2. Attack on Switches
   - To slow down.
   - Or to drop packets.
3. Attack on Controller
   - Inject traffic or forged requests to overload the Controller.
   - Once the Controller is attacked, all lower level Switches are misled and can't correctly deliver packets.

# Reliability Issues

1. Link Failure
   - Failure of link between Controller and Switch.
   - Failure of Switch-to-Switch path link.
2. Controller Failure
3. Asynchronous Update of Switches
   - Controller may not be able to synchronously update all the Switches.

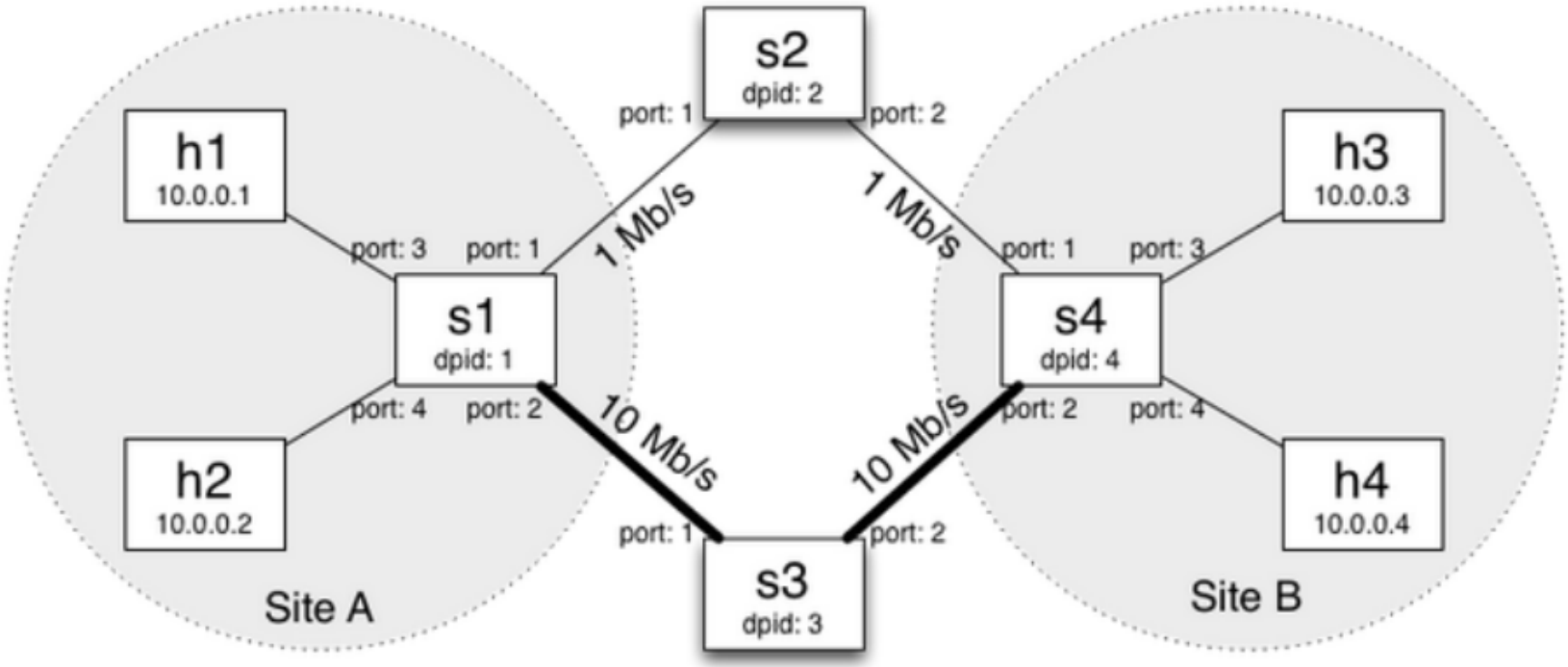
# Use Cases of SDN

## FlowVisor

- **FlowVisor** is an application Controller that sits between the physical devices and the Controllers, slices the flows (called flowspace).
- **Slicing** is to separate the flowspace in distinct subspaces.
- FlowVisor partitions the flow-table in each Switch by keeping track of which flow-entries belong to which Controller.
- Given a packet header, it can decide which flowspace contains it, and hence which slice (or slices) it belongs to.

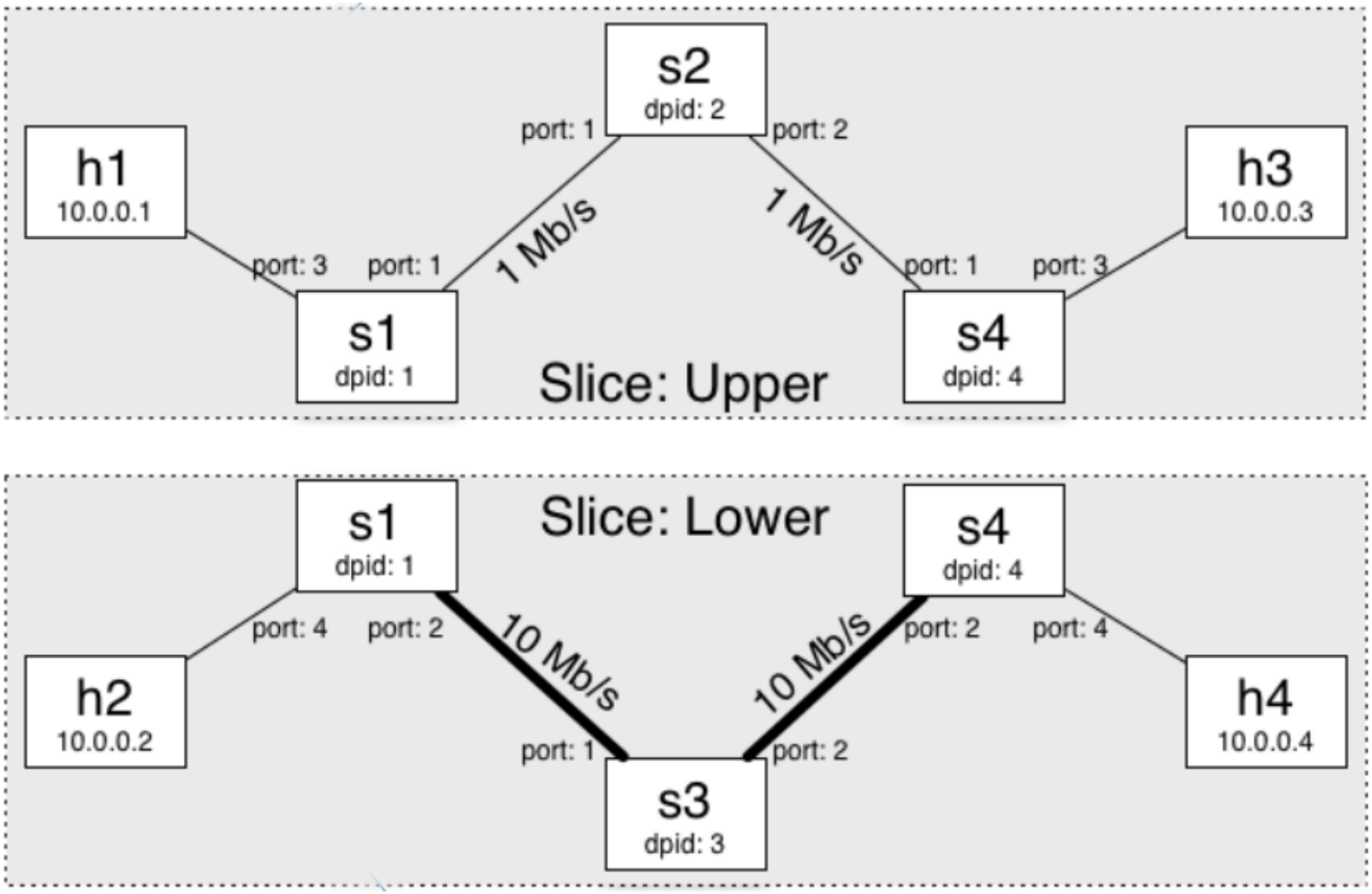
# FlowVisor

## Sample Topology



# FlowVisor

## Execution



- **Imagine a multi tenant datacenter which has multiple customers each having their applications deployed in the data center servers. Say the customers wants to run their own proprietary switching logic (Control Plane Protocols) for their respective traffic.**

- **Imagine a multi tenant datacenter which has multiple customers each having their applications deployed in the data center servers. Say the customers wants to run their own proprietary switching logic (Control Plane Protocols) for their respective traffic.**
  - With the existing network architecture there is no way to address this requirement.

- **Imagine a multi tenant datacenter which has multiple customers each having their applications deployed in the data center servers. Say the customers wants to run their own proprietary switching logic (Control Plane Protocols) for their respective traffic.**
  - With the existing network architecture there is no way to address this requirement.
  - FlowVisor solves this problem by slicing the networks based on some of the attributes either in the packet or based on the interface configurations in the OpenFlow Switches.

# References

# References


1. 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 Communications Review, Apr. 2008.

2. Open vSwitch & OVS-Controller.
   http://openvswitch.org/

3. Pantou: Openflow 1.0 for OpenWRT.
   http://www.openflow.org/wk/index.php/OpenFlow_1.0_for_OpenWRT

4. ofsoftswitch13 - cpqd.
   https://github.com/CPqD/ofsoftswitch13

5. Indigo: Open source openflow switches.
   http://www.projectfloodlight.org/indigotext

# References


6. POX.
   http://www.noxrepo.org/pox/about-pox/

7. 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 Commun. Review, 38(3), pp. 105-110, 2008.

8. Floodlight, an open sdn controller.
   http://floodlight.openflowhub.org/

9. Ryu.
   http://osrg.github.com/ryu/

10. Beacon.
    https://openflow.stanford.edu/display/Beacon/Home

# References


11. Trema openflow controller framework.
https://github.com/trema/trema

12. Z. Cai, AL Cox, and TSE Ng. **Maestro: A system for scalable openflow control**, Technical Report TR10-08, Rice University, December 2010.

13. Mul.
http://sourceforge.net/p/mul/wiki/Home/

14. Jaxon:java-based openflow controller.
http://jaxon.onuos.org/

15. Helios by nec.
http://www.nec.com/

# References


16. Simple Network Access Control (SNAC). http://www.openflow.org/wp/snac/

17. The nodeflow openflow controller. http://garyberger.net/?p=537

18. 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 Commun. Review, 40(1), pp. 129-130, 2010.

# References


19. Marcelo R. Nascimento, Christian E. Rothenberg, Marcos R. Salvador, Carlos N. A. Corr^ea, Sidney C. de Lucena, and Maur ıcio F. Magalh aes. **Virtual routers as a service: the routeflow approach leveraging software defined networks**, In Proc. 6th Int. Conf. on Future Internet Technology, CFI '11, pp. 34-37, New York, NY, USA, 2011. ACM.

20. N. Foster et al. **Frenetic: A network programming language**, SIGPLAN Not., vol. 46, no. 9, pp. 279-291, Sep. 2011.

21. N. Foster et al. **Languages for software-defined networks**, IEEE Commun. Mag., vol. 51, no. 2, pp. 128-134, Feb. 2013.

# References


22. C. Monsanto, N. Foster, R. Harrison, and D. Walker. **A compiler and run-time system for network programming languages**, in Proc. 39th Annu. ACM SIGPLAN-SIGACT Symp. POPL, pp. 217-230, 2012.

23. A. Voellmy and P. Hudak. **Nettle: Taking the sting out of programming network routers**, in Proc. 13th Intl. Conf. PADL, pp. 235-249, 2011.

24. T. L. Hinrichs, N. S. Gude, M. Casado, J. C. Mitchell, and S. Shenker. **Practical declarative network management**, in Proc. 1st ACM WREN, pp. 1-10, 2009.

25. A. Voellmy, H. Kim, and N. Feamster. **Procera: A language for high level reactive network control**, in Proc. 1st Workshop HotSDN, pp. 43-48, 2012.

# References


26. Fei Hu, Qi Hao, and Ke Bao. **A Survey on Software-Defined Network and OpenFlow: From Concept to Implementation**, IEEE Communication Surveys & Tutorials, Vol. 16, No. 4, pp. 2181-2206, 2014.

27. Yosr Jarraya, Taous Madi, and Mourad Debbabi. **A Survey and a Layered Taxonomy of Software-Defined Networking**, IEEE Communication Surveys & Tutorials, Vol. 16, No. 4, pp. 1955-1980, 2014.

28. Bruno Astuto A. Nunes, Marc Mendonca, Xuan-Nam Nguyen, Katia Obraczka, and Thierry Turletti. **Survey of Software-Defined Networking: Past, Present, and Future of Programmable Networks**, IEEE Communications Surveys & Tutorials, Vol. 16 , No. 3, pp. 1617-1634, 2014.

# References


29. Nick Feamster, Jennifer Rexford, Ellen Zegura. **The Road to SDN: An Intellectual History of Programmable Networks**, ACM SIGCOMM Computer Communication Review, Vol. 44, No. 2, pp. 87-98, 2014.

30. B. Lantz, B. Heller, and N. McKeown. **A Network in a Laptop: Rapid Prototyping for Software-Defined Networks**, Proc. of ACM Hotnets'10, 2010.

31. Diego Kreutz,Fernando M. V. Ramos, Paulo Verissimo. **Towards Secure and Dependable Software-Defined Networks**, HotSDN'13, ACM, New York, USA, pp. 55-60, 2013.

32. https://www.opennetworking.org/

33. http://www.opennetsummit.org/

# Thank You!!! Enjoy Networking!!!

# Mininet

Prof. Ankit Gangwal

Centre for Security, Theory, and Algorithmic Research (CSTAR),
IIIT Hyderabad, India.

# Agenda

- Comparison of Various Network Testing Platform
- Introduction to Mininet
- Mininet - Comparison
- Installing Mininet
  1. Mininet VM Installation
  2. Native Installation from Source
  3. Installation from Package
- Preparing Mininet
- Running Mininet
- Mininet Commands
- Mininet Other Utilities
- References
- Warm-up Exercises

# Platforms for Network Testing

| Platforms | Advantages | Disadvantages |
|---|---|---|
| **Hardware Testbed** | Fast.<br>Accurate. | Expensive.<br>Hard to reconfigure.<br>Hard to change. |
| **Simulators** | Inexpensive.<br>Flexible.<br>Easy to download. | May not be "Believable".<br>May be slow. |
| **Emulators** | Inexpensive.<br>Flexible.<br>Easy to download.<br>Reasonably accurate. | Slower than hardware.<br>Experiments may not fit.<br>Possibility of inaccuracy from multiplexing. |

# Introduction to Mininet

- Mininet is an **Open Source Project**.
- Mininet depends on the **Linux Kernel**.
- Mininet is a **Network Emulator**.
- Mininet is used to develop, share & experiment with **OpenFlow & SDN**.
- Mininet experiments are **Python Scripts**.

# Introduction to Mininet

- It makes a **single system** look like a **complete network** that includes end-hosts, switches, routers & links on a single Linux kernel.
- Mininet host behaves as a real machine, where you can **SSH** into it & run arbitrary programs.
- Mininet uses **process-based virtualisation** to run many (upto **4096**) hosts and switches on a single OS kernel.
- You can create **custom** topologies.

# Introduction to Mininet

- Programs can send packets through virtual Ethernet interface, with given link speed & delay.
- Packets are processed by virtual Ethernet switch, router, or middle-box, with given amount of queueing.
- You can **customize** packet forwarding as Mininet's switches are programmable using the OpenFlow protocol.
- For custom routing or switching behaviour separate OpenFlow controller must be developed for required features.
- Mininet doesn't do NAT out of the box. This means that your virtual hosts will be **isolated** from your LAN by default.

# Comparing Mininet

Mininet combines many of the best features of emulators, hardware testbeds, and simulators.

## Compared to full system virtualization based approaches, Mininet

- **Boots faster:** seconds instead of minutes.
- **Scales larger:** hundreds of hosts and switches.
- **Provides more bandwidth:** typically 2Gbps total bandwidth on modest hardware.
- **Installs easily:** Apart from pre-built VM image, Mininet Ubuntu Package is also available.

# Comparing Mininet

## Compared to hardware testbeds, Mininet is

- **inexpensive**.
- **quickly reconfigurable and restart-able**.

## Compared to simulators, Mininet

- easily **connects to real networks**.
- offers **interactive performance** - you can type at it.

## Limitation

- Mininet cannot (currently) run **non-Linux-compatible** OpenFlow switches or applications; this has not been a major issue in practice.

# Installing Mininet

- Go to http://mininet.org/download/
- Option 1: Mininet VM Installation (easy, recommended)
- Option 2: Native Installation from Source
- Option 3: Installation from Packages

# Option 1: Mininet VM Installation

1. Download the **Mininet VM image** @ https://github.com/mininet/mininet/wiki/Mininet-VM-Images
   - Go under **Mininet 2.2.0 on Ubuntu 14.04** & select: Ubuntu 14.04 - 32 bit (recommended for Windows users using VirtualBox or Hyper-V)

2. Download and install **VirtualBox** @ https://www.virtualbox.org/wiki/Downloads
   - Go under **VirtualBox platform packages** & select appropriate version for your host OS.

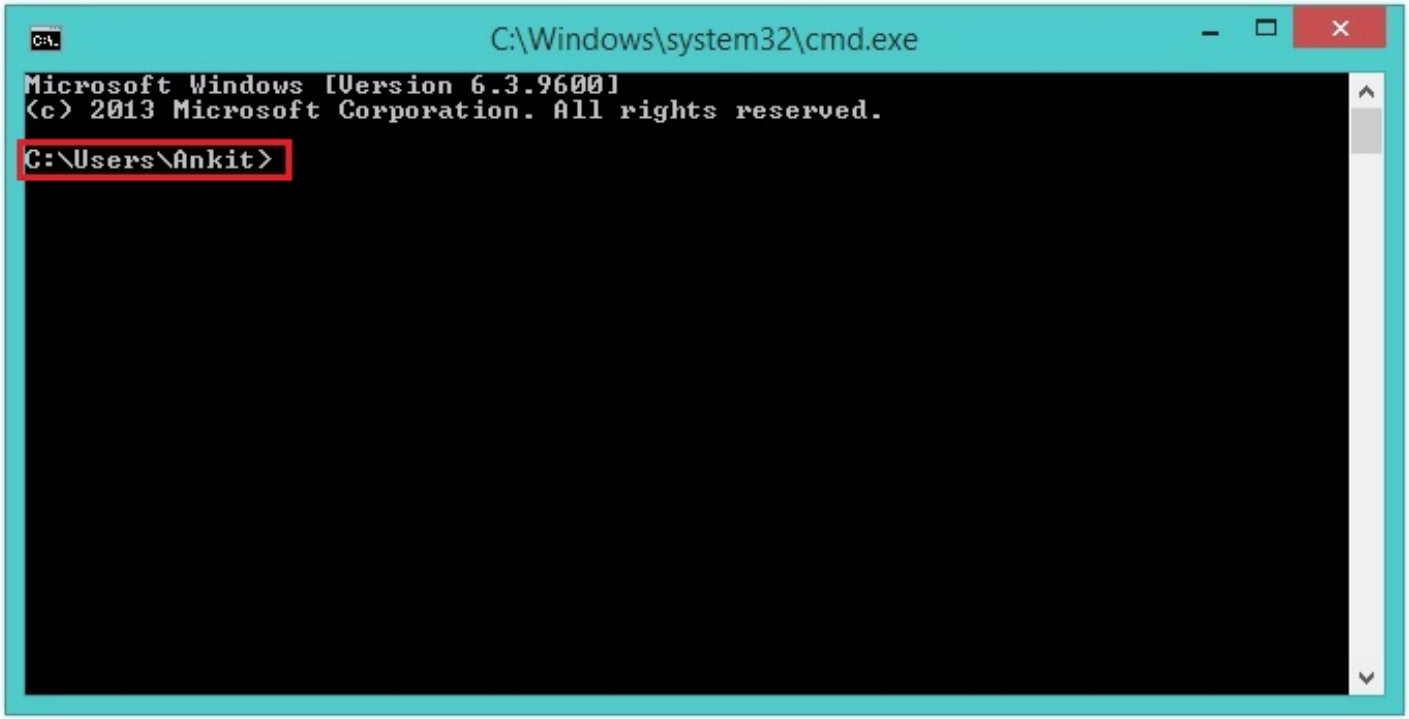
# Option 1: Mininet VM Installation

3. Download X server & Terminal for your host OS @ https://github.com/mininet/openflow-tutorial/wiki/Installing-Required-Software

| OS Type | OS Version | Virtualization Software | X Server | Terminal |
|---|---|---|---|---|
| Windows | 7+ | VirtualBox | Xming | PuTTY |
| Windows | XP | VirtualBox | Xming | PuTTY |
| Mac | OS X 10.7 - 10.9 Lion/Mountain Lion/Mavericks | VirtualBox | download & install XQuartz | Terminal.app |
| Mac | OS X 10.5-10.6 Leopard/Snow Leapord | VirtualBox | X11 (install from OS X main system DVD, preferred), or download XQuartz | Terminal.app |
| Linux | Ubuntu 10.04+ | VirtualBox | X server already installed | gnome terminal + SSH |

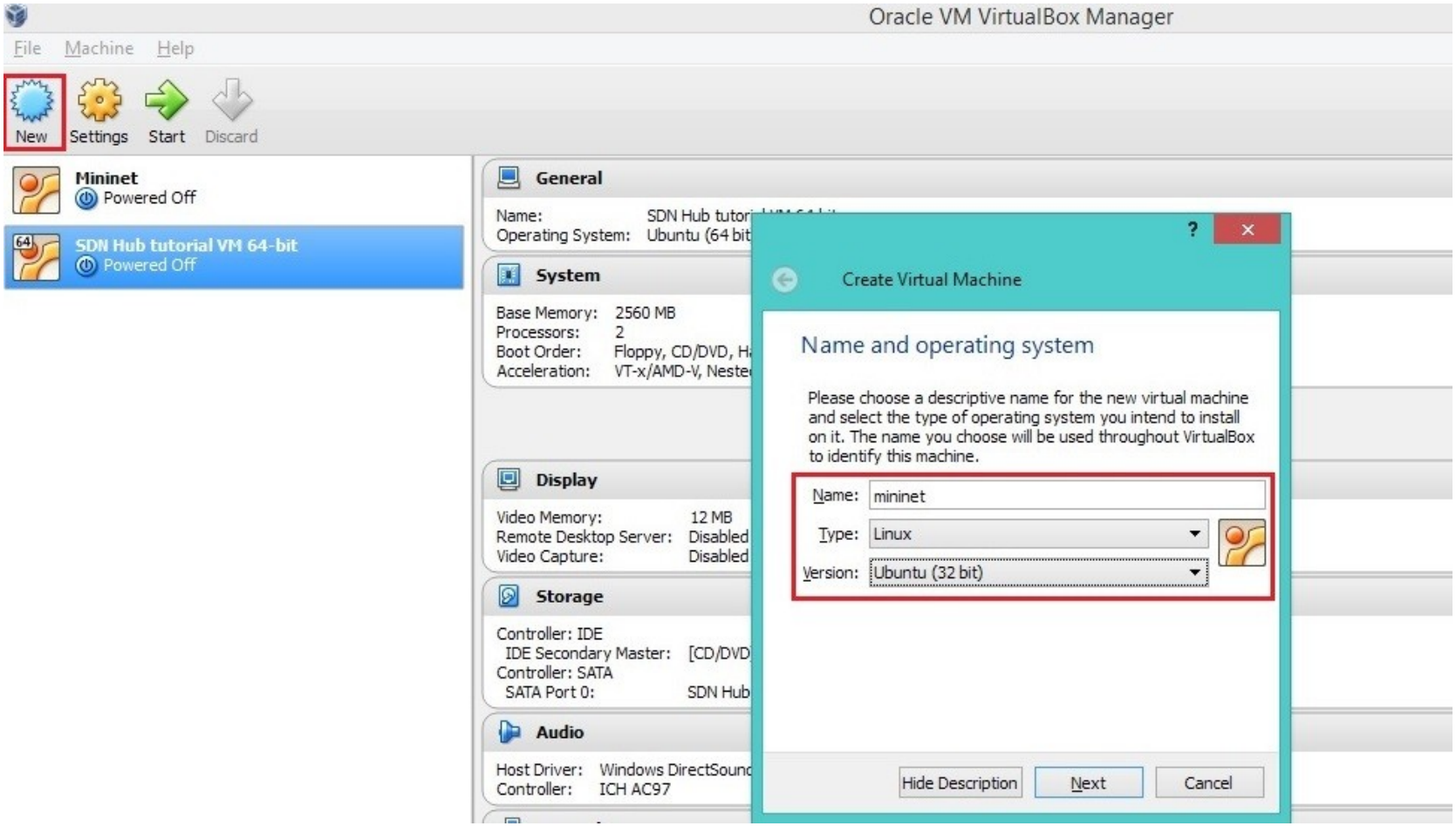
# Option 1: Mininet VM Installation

4. Extract the Mininet image you downloaded in step1.
5. Install the X server & launch it.
6. In case of windows copy the putty.exe to the folder where the command prompt starts up. In this case "c:\users\ankit".



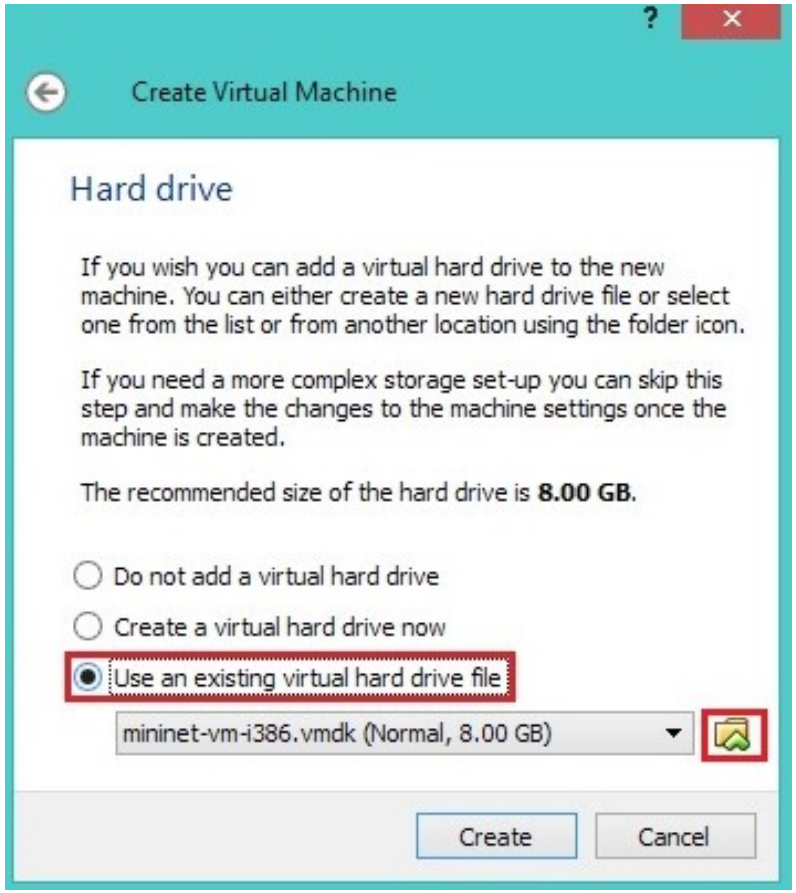
# Option 1: Mininet VM Installation

7. Launch VirtualBox; select New & Name your VM; Type ->Linux; Version ->Ubuntu (32 bit).



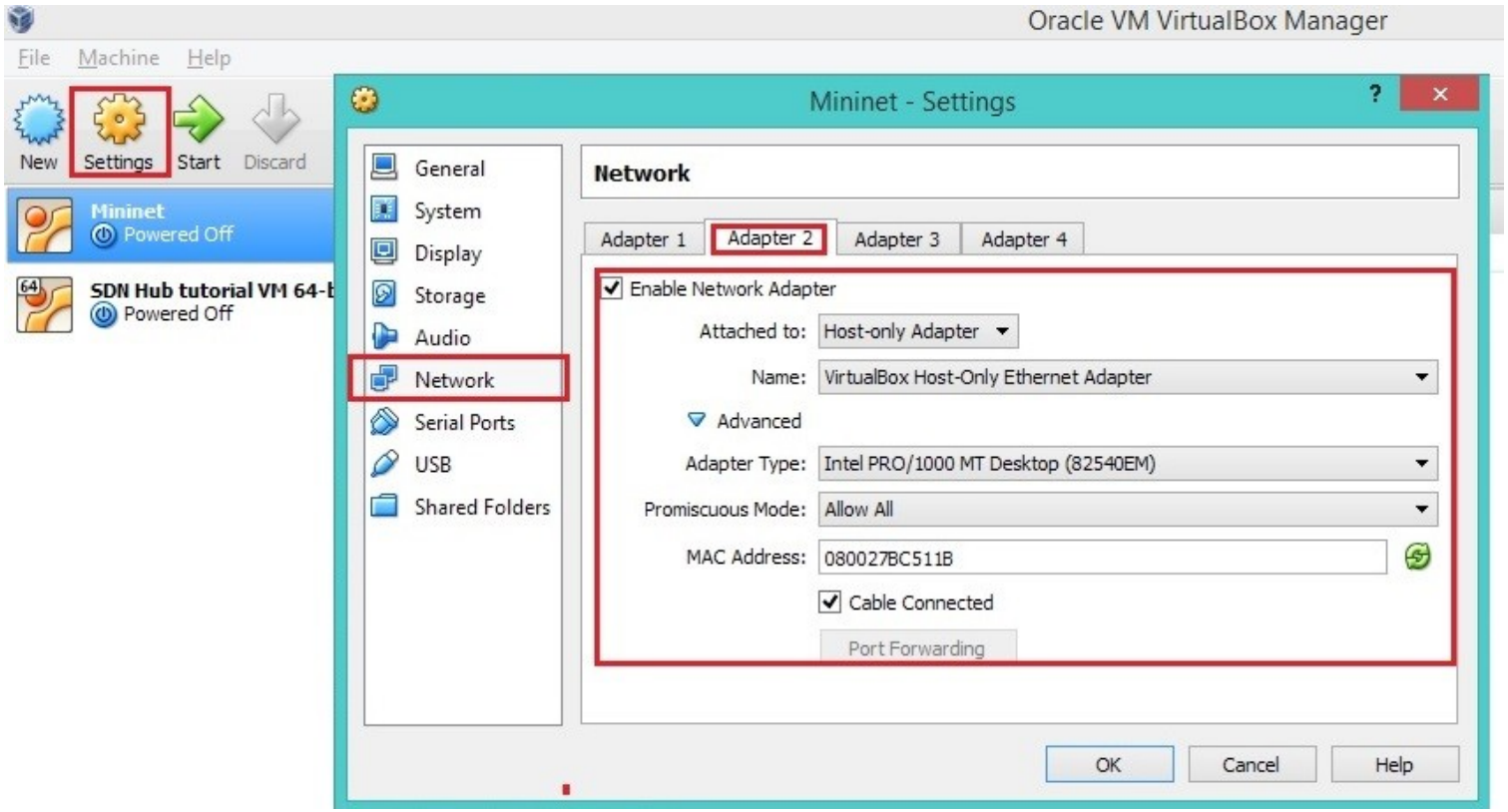
# Option 1: Mininet VM Installation

8. Allocate RAM 512 MB is Good.
9. Next "Select Use an existing virtual hard drive file" & locate .vmdk file. Then click "Create".



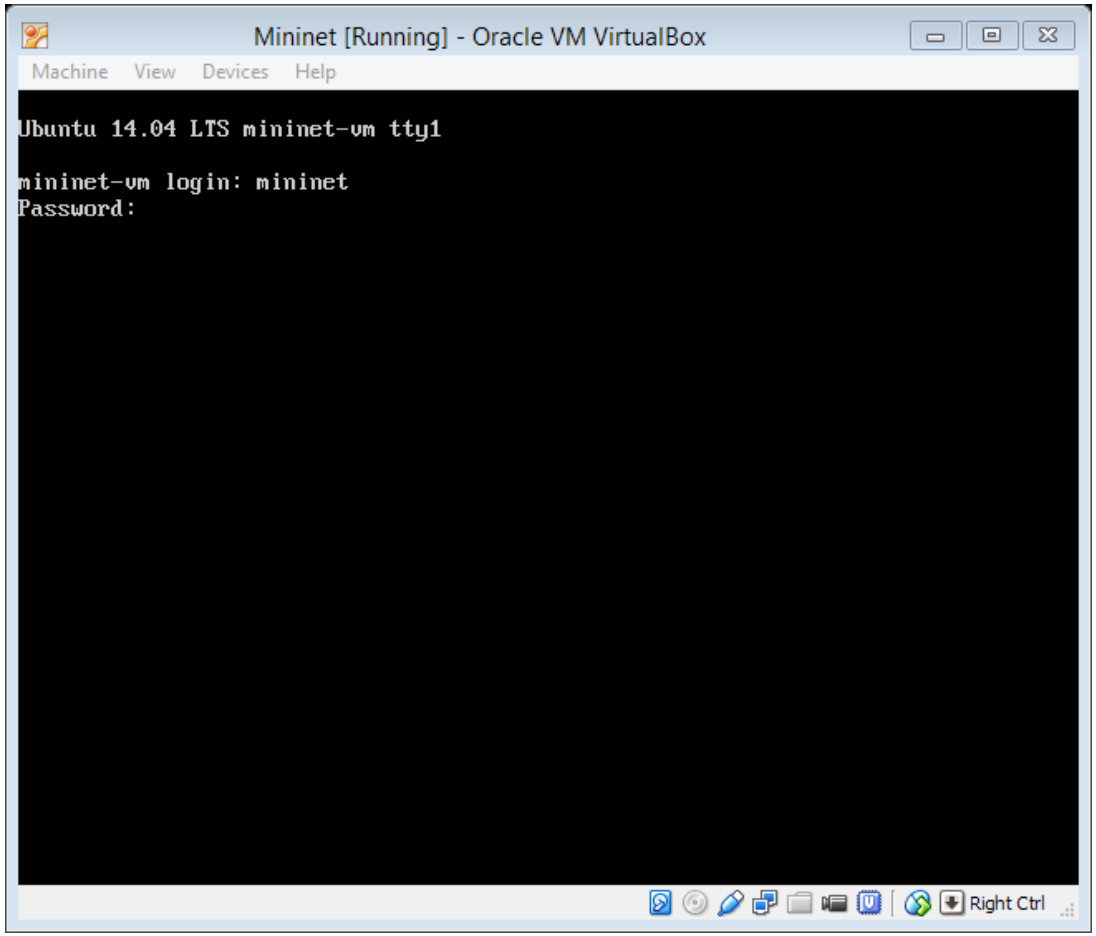
# Option 1: Mininet VM Installation

10. Click on Settings ->Network ->Adapter 2 ->check "Enable Network Adapter" ->Attached to "Host-only Adapter" ->Advanced ->Promiscuou Mode -> "Allow All".



# Option 1: Mininet VM Installation

11. Double click on VM name or click on Start to boot.
12. Username & Password is "mininet".



# Option 2: Native Installation from Source

1. To install natively from source, first get the source code:

**$ git clone git://github.com/mininet/mininet**

**Note that the above command will check out the latest Mininet. If you want to run any other version, you may checkout that version explicitly:

- **$ cd mininet**
- **$ git tag** /*lists available versions*/
- **$ git checkout -b 2.2.1 2.2.1** /*or other version required*/
- **$ cd ..**

2. Once you have the source tree, install Mininet with:

**$ mininet/util/install.sh [options]**

# Option 2: Native Installation from Source

## **Typical *install.sh* options include:

- To install everything (using your home directory):
  **install.sh -a**
- To install everything (using another directory):
  **install.sh -s mydir -a**
- To install Mininet + user switch + OVS (using your home dir):
  **install.sh -nfv**
- To install Mininet + user switch + OVS (using other dir):
  **install.sh -s mydir -nfv**
- For other options:
  **install.sh -h**

## 3. After the installation, test the basic Mininet functionality:

**$ sudo mn --test pingall**

# Option 3: Installation from Packages

**1. Remove any traces of earlier versions of Mininet and Open vSwitch from /usr/local/:**

```
sudo rm -rf /usr/local/bin/mn /usr/local/bin/mnexec \
sudo rm -rf /usr/local/lib/python*/*/*mininet* \
sudo rm -rf /usr/local/bin/ovs-* /usr/local/sbin/ovs-*
```

**2. Confirm which OS version you are running, run the command:**

```
lsb_release -a
```

# Option 3: Installation from Packages

## 3. Install the base Mininet package:

Mininet 2.1.0 on Ubuntu 13.10:

**sudo apt-get install mininet**

Mininet 2.0.0 on Ubuntu 13.04:

**sudo apt-get install mininet**

Mininet 2.0.0 on Ubuntu 12.10:

**sudo apt-get install mininet/quantal-backports**

Mininet 2.0.0 on Ubuntu 12.04:

**sudo apt-get install mininet/precise-backports**

## 4. After this deactivate openvswitch-controller if it is running:

**sudo service openvswitch-controller stop**

**sudo update-rc.d openvswitch-controller disable**

# Option 3: Installation from Packages

5. Test Mininet :

**sudo mn --test pingall**

6. If Mininet complains that Open vSwitch isn't working:

**sudo dpkg-reconfigure openvswitch-datapath-dkms**
**sudo service openflow-switch restart**

7. To install additional software:

**git clone git://github.com/mininet/mininet**
**mininet/util/install.sh -fw**

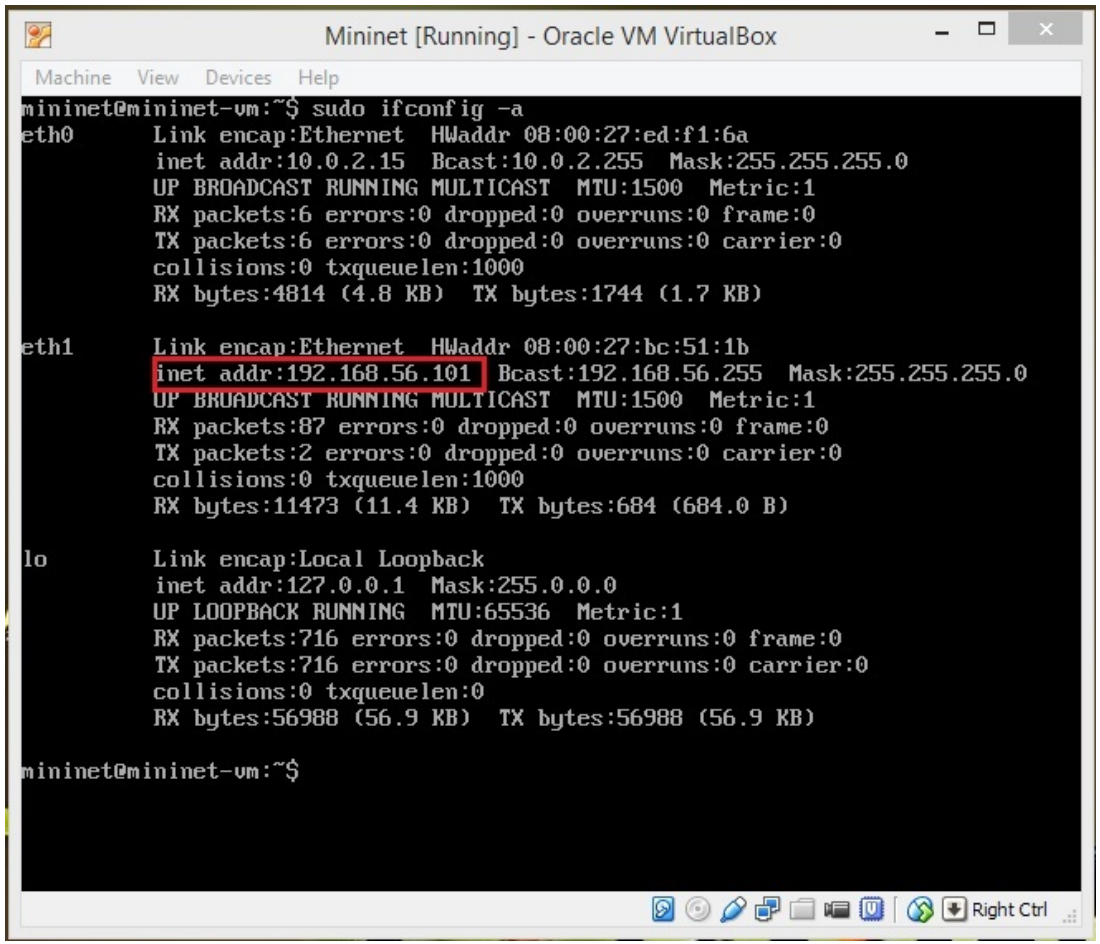
# Preparing Mininet

- **Only VM based users need to follow preparation steps !!!**
- **With VM booted & logged in:**

1. List all the network interfaces installed on the system:
   **sudo ifconfig -a**
2. Assign an address (e.g. for **eth1** interface) run:
   **sudo dhclient eth1**

# Preparing Mininet

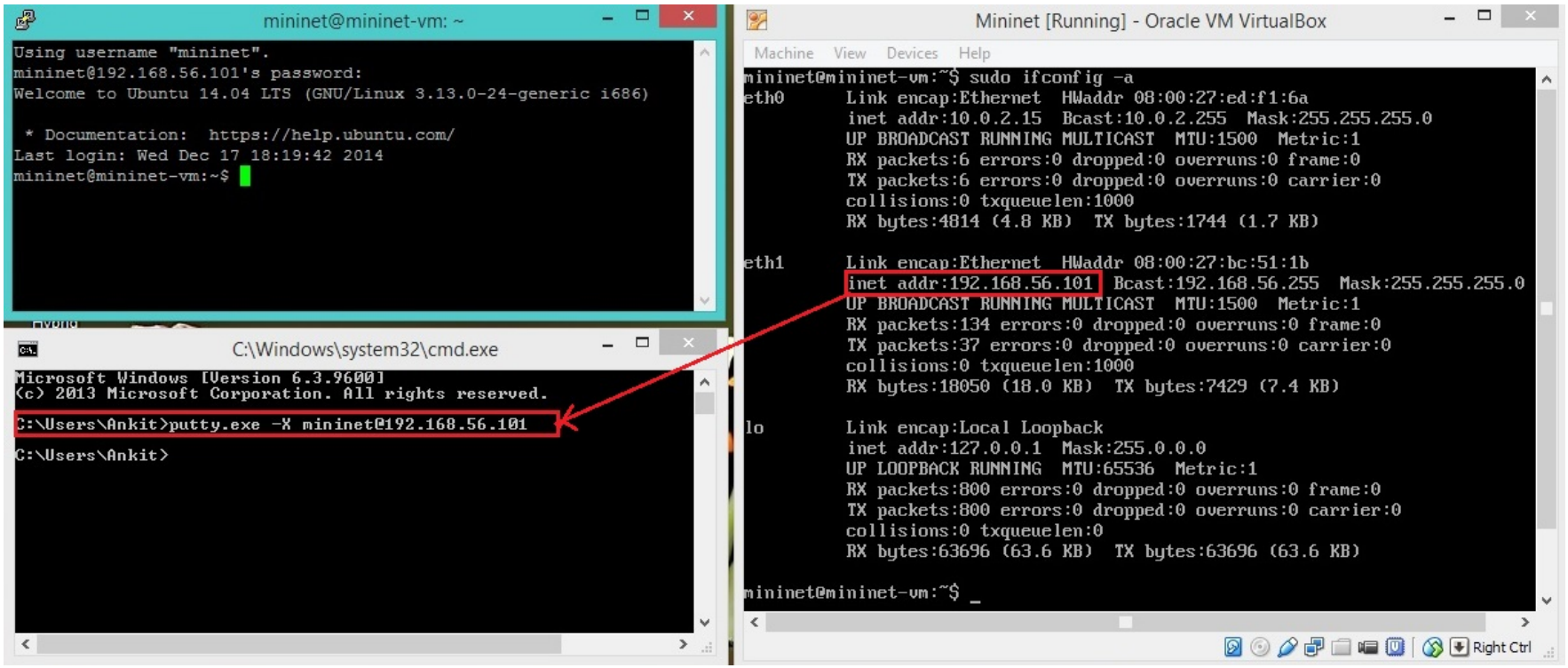
3. Repeat step 1 & note "inet addr" for eth1.



# Preparing Mininet

4. Windows users : From Cmd run putty.exe with eth1's inet addr. Password : "mininet".

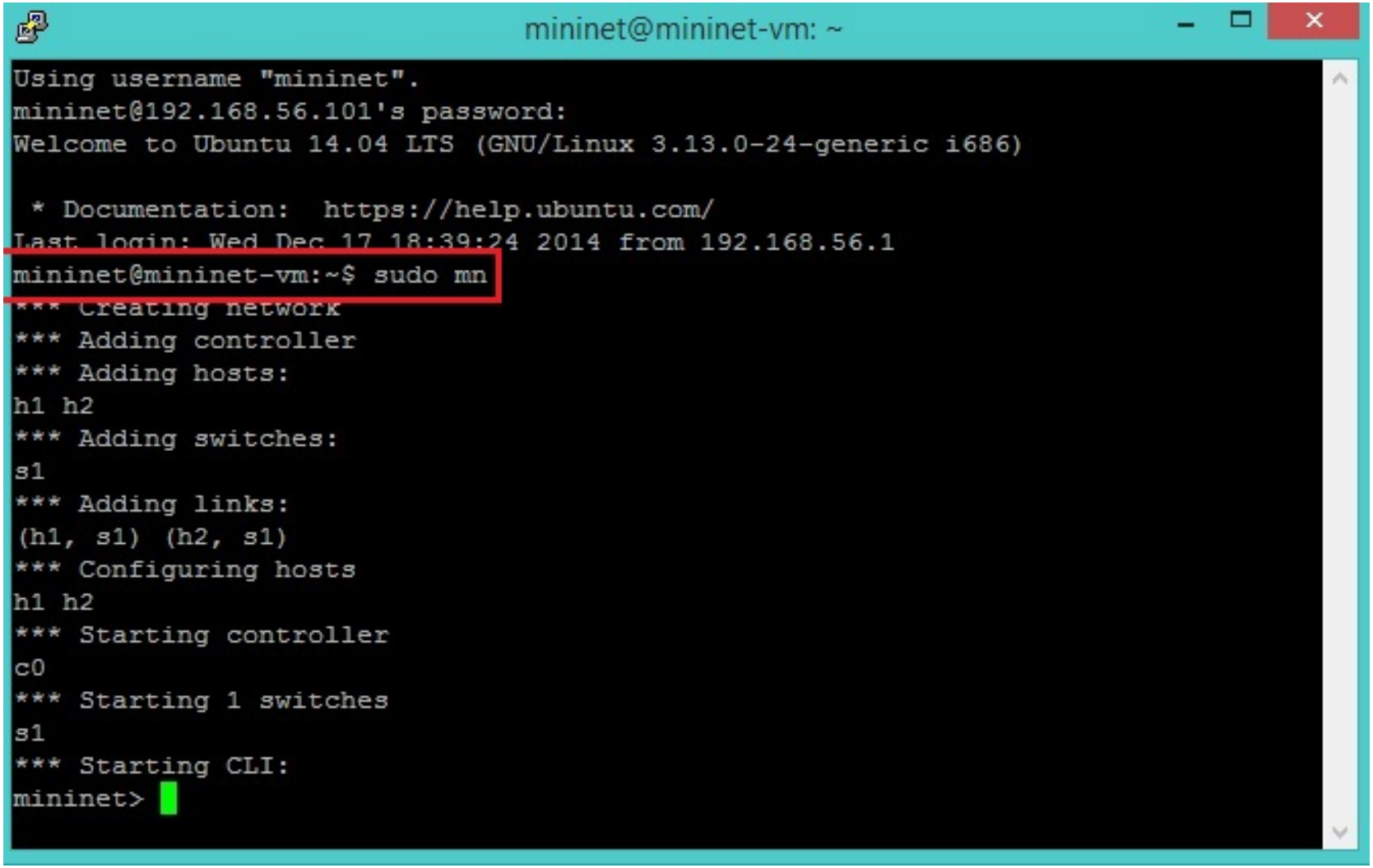
**putty.exe -X mininet@192.168.56.101**




** For other OS replace putty.exe with "Compatible Terminal".

# Running Mininet

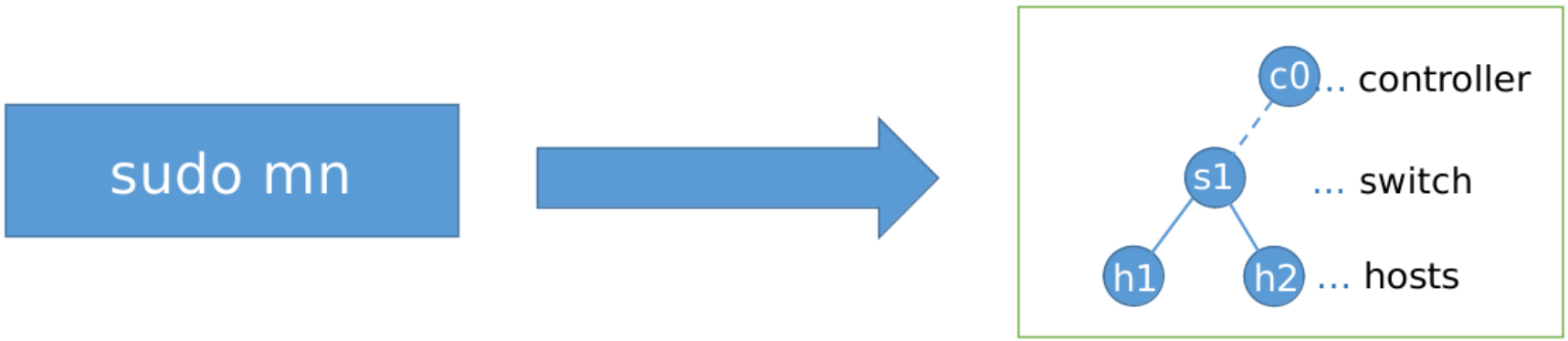
**sudo mn**



# Running Mininet

**sudo mn** runs a network with the default minimal topology:

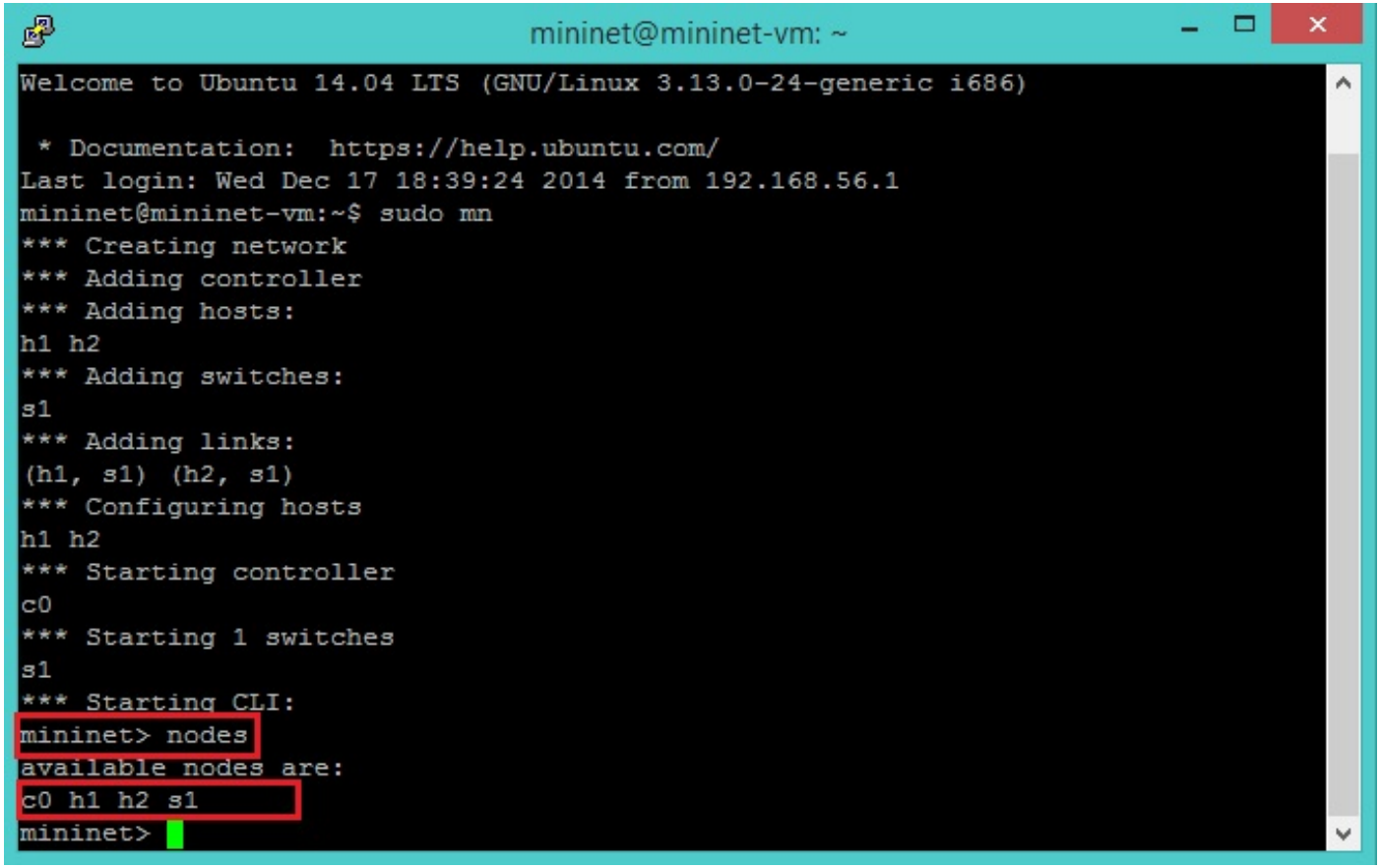
- 1 Controller
- 1 Switch
- 2 Hosts
- 2 links (h1,s1 & h2,s1)



# Mininet Command - **nodes**

## Lists available **nodes**.

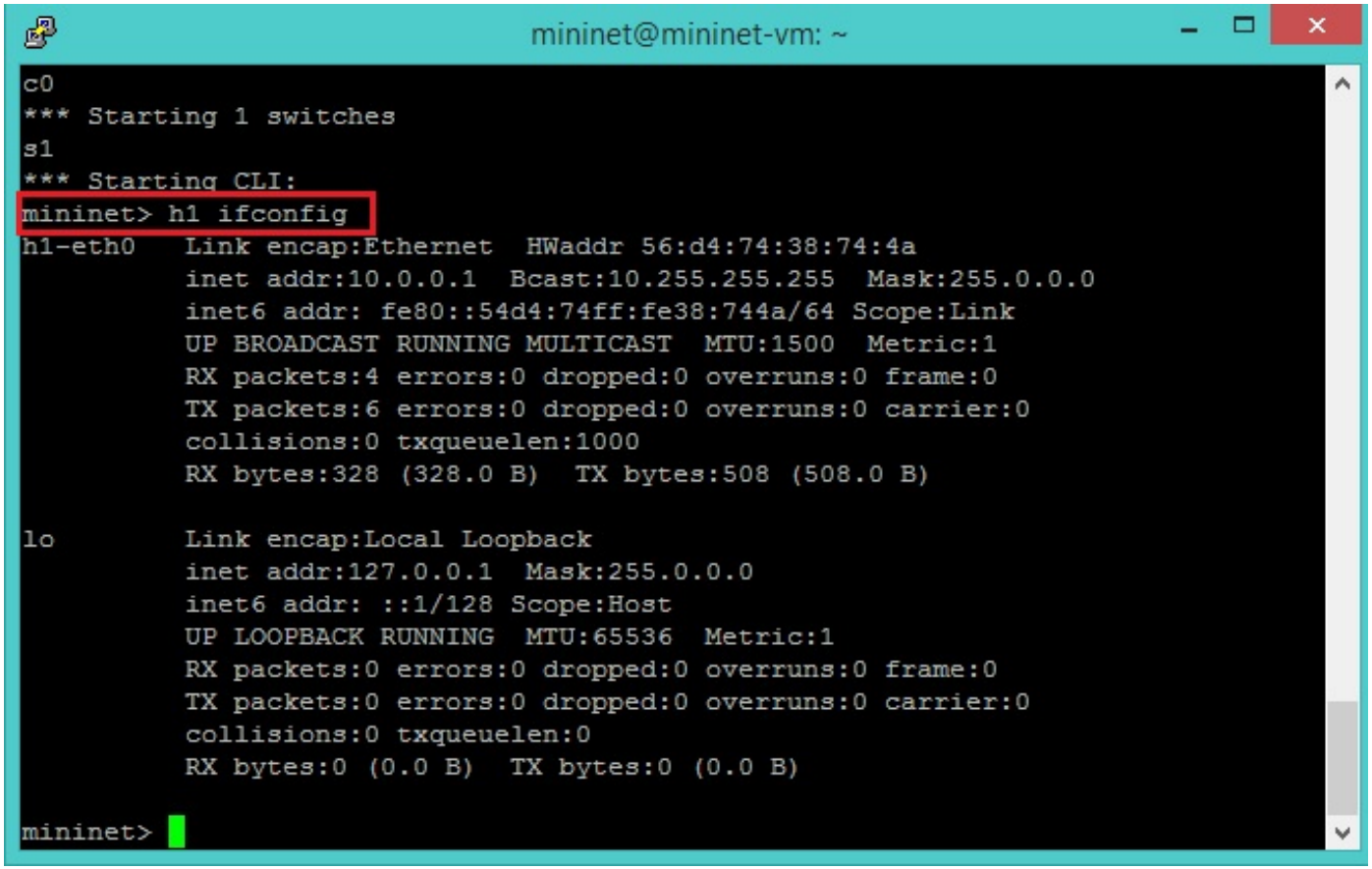
**mininet>nodes**



# Mininet Command - **ifconfig**

**Running a command on a node:** prepend the command with the name of the node. E.g. to check the NI config. of a virtual host :

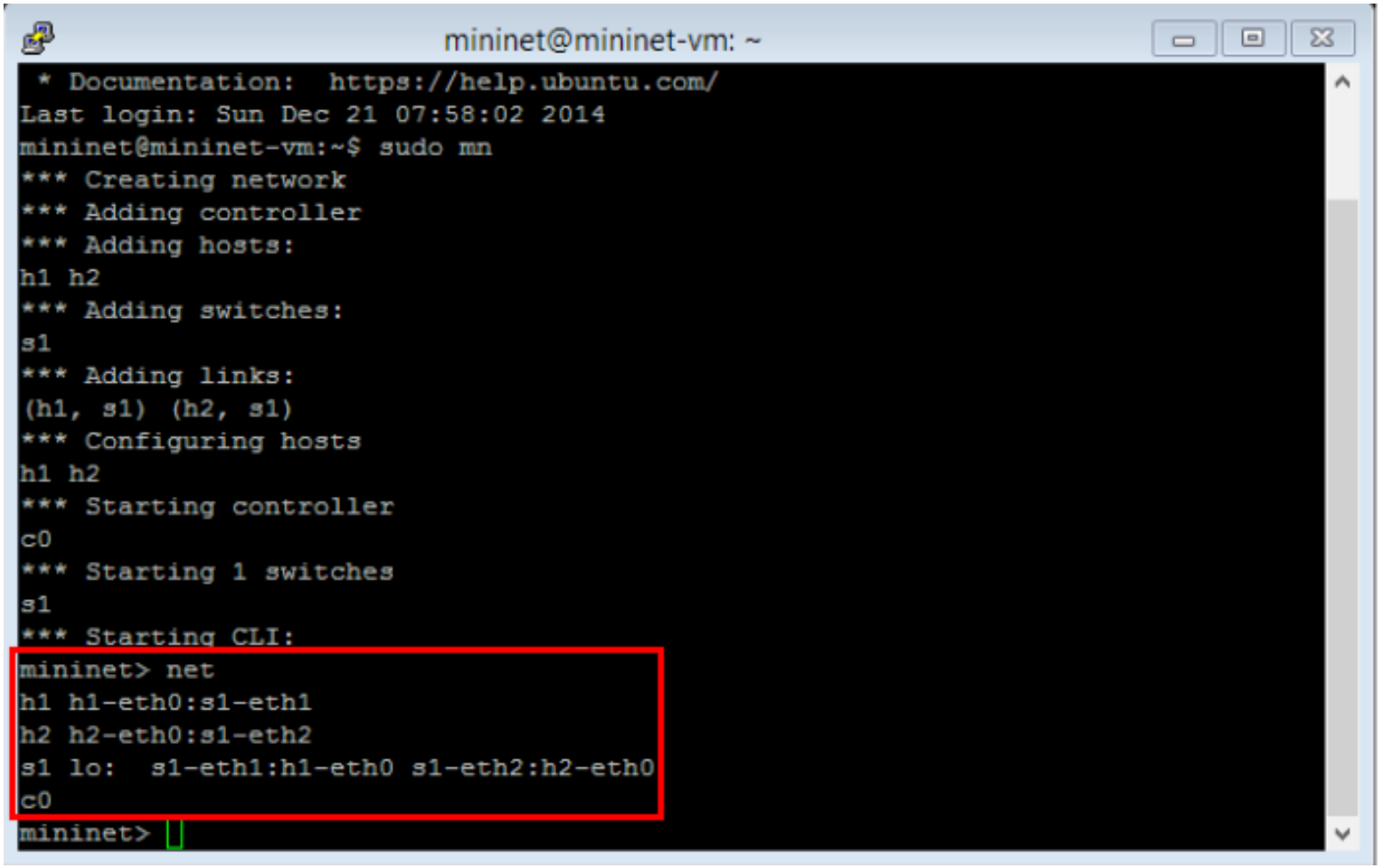
**mininet>h1 ifconfig**



# Mininet Command - **net**

Lists **NI-to-NI** connection information.

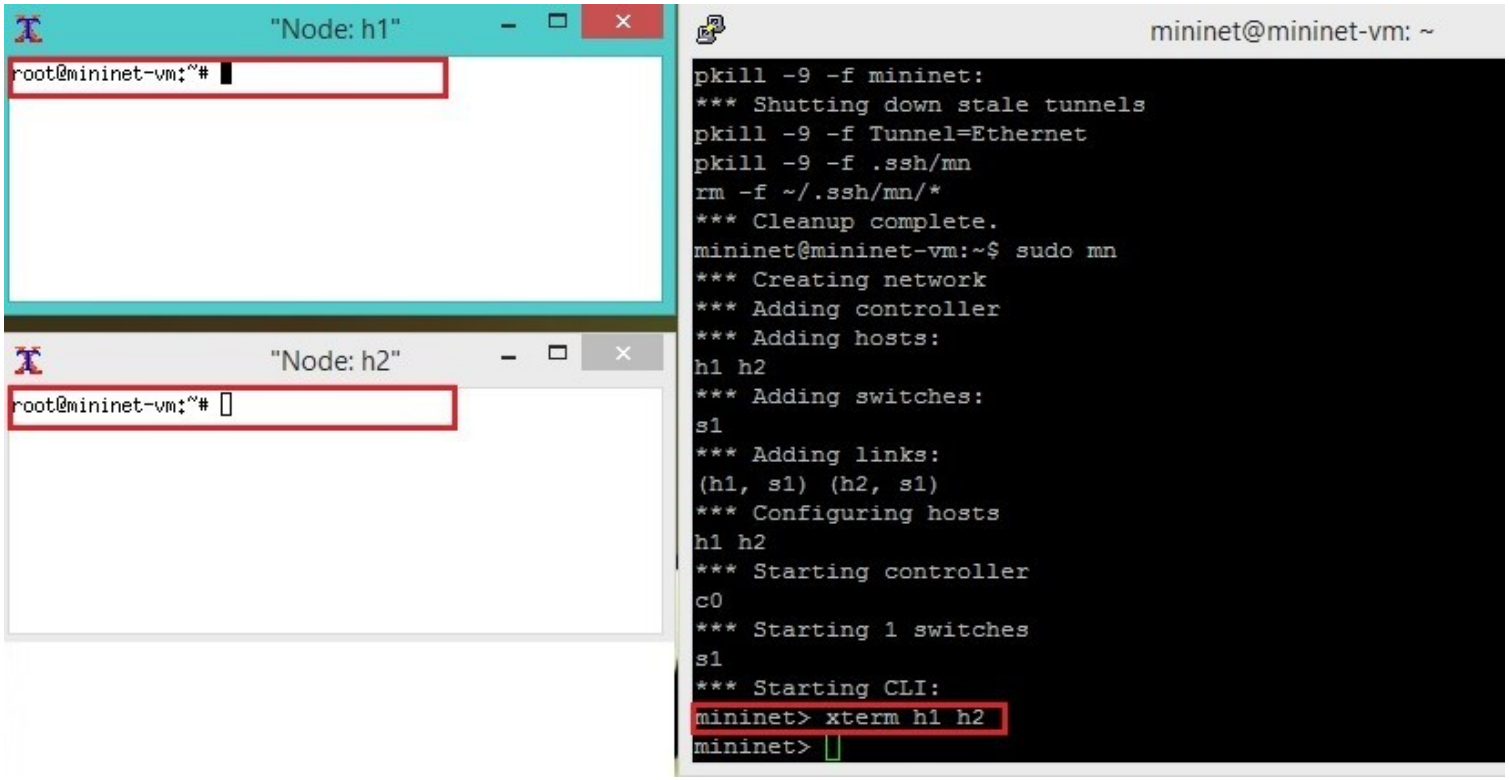
**mininet>net**



# Mininet Command - **xterm**

## Spawning an **xterm** for one or more virtual hosts.

**mininet>xterm h1 h2**



# Mininet Commands - **help, exit, -c**

**Lists** available commands

**mininet>help**

**Exiting** Mininet

**mininet>exit**

**Clearing** any residual state or processes

**$sudo mn -c**

# Mininet Commands - **ping & iperf**

**Ping** from one host to another

**mininet>h1 ping h2**

**Ping reachability** from every host to every other host

**mininet>pingall**

Testing **bandwidth**

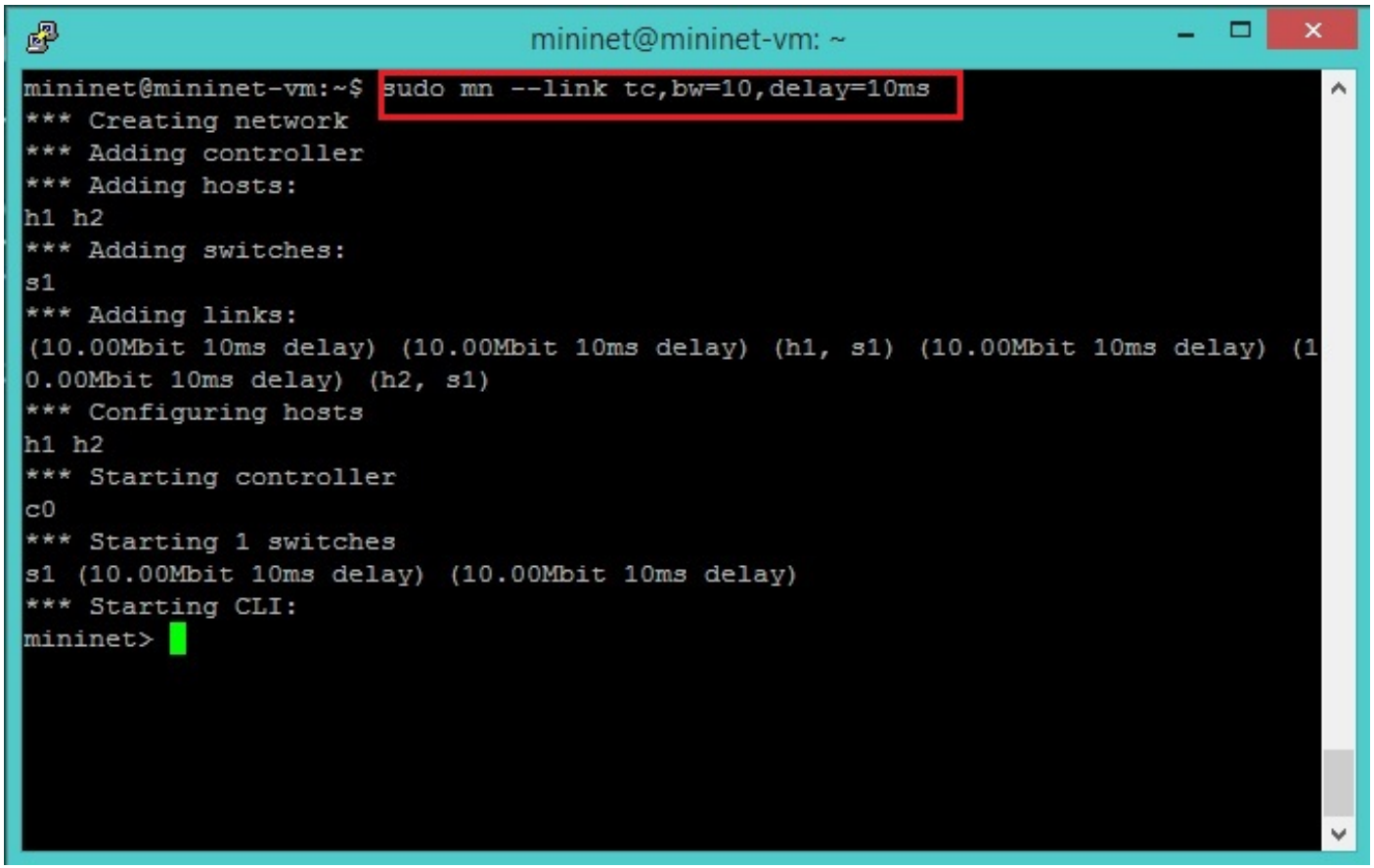
TCP : **mininet>iperf**

UDP : **mininet>iperfudp**

# Mininet Command - **link**

## Making links with custom **Bandwidth** & **Delay**

**$ sudo mn --link=tc,bw=10,delay=10**



# Remote Controllers

## To Run Remote Controllers:

- Open separate terminal & navigate to controller:

  **$ cd pox**

- To run native POX:

  **$ ./pox.py**

- To run your module of POX (paste in /pox/ext):

  **$ ./pox.py mycontroller**

- To run POX with DEBUG Level Info:

  **$ ./pox.py mycontroller log.level --DEBUG**

- To run POX with CRITICAL Level Info:

  **$ ./pox.py mycontroller log.level --CRITICAL**

# Mininet Commands - **topo, mac, switch, controller**

**$ sudo mn --topo single,3 --mac --switch ovsk --controller remote**

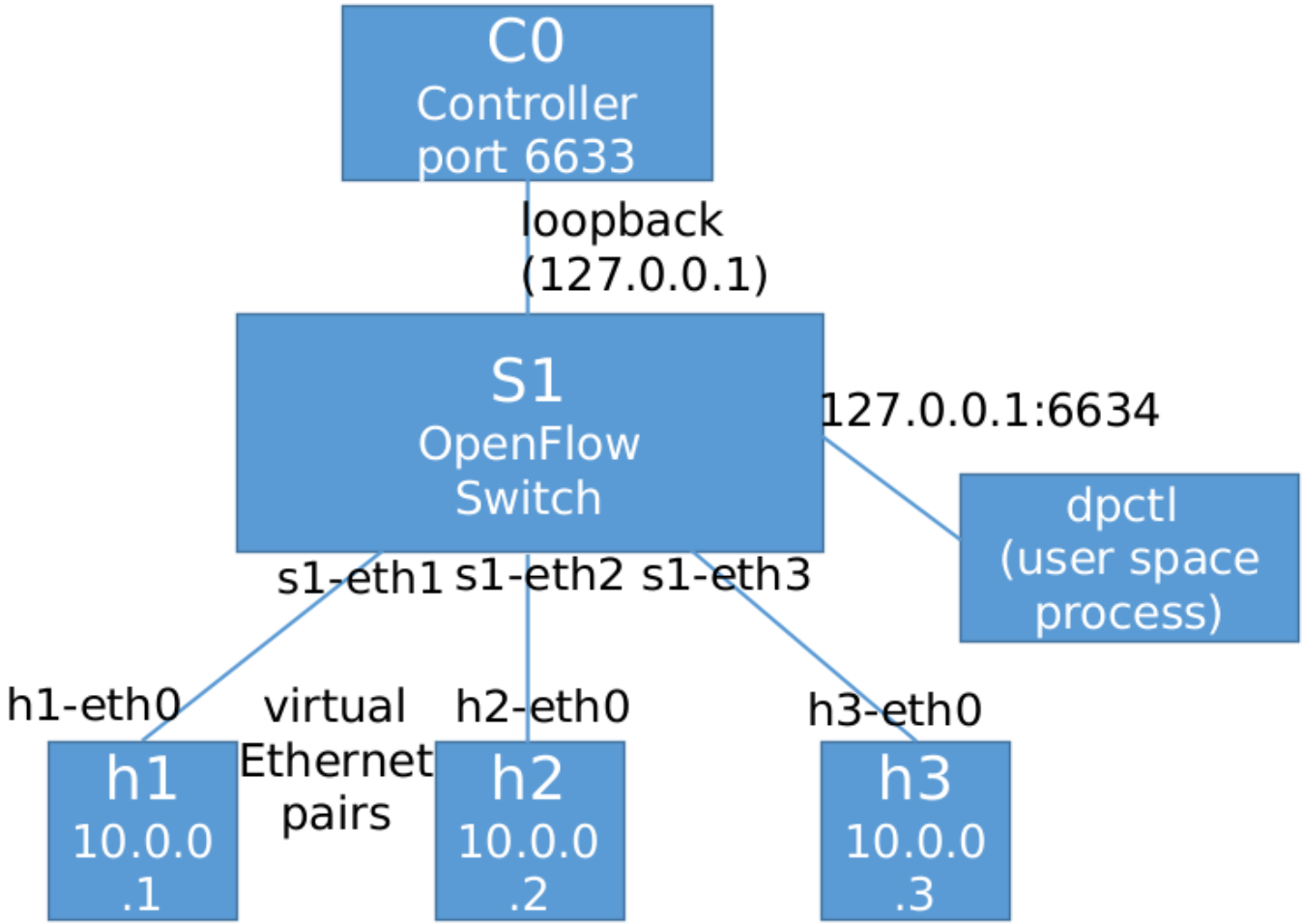
Mininet has

- Created 3 virtual hosts, each with a separate IP address.
- Created a "single" OpenFlow software (openvSwitch-based) switch in kernel with 3 ports.
- Connected each virtual host to the switch with a virtual Ethernet cable.
- Set the MAC address of each host equal to its IP.
- Configure the OpenFlow switch to connect to a remote controller (default is localhost).

# Mininet Commands - **topo, mac, switch, controller**

## Creating custom networks

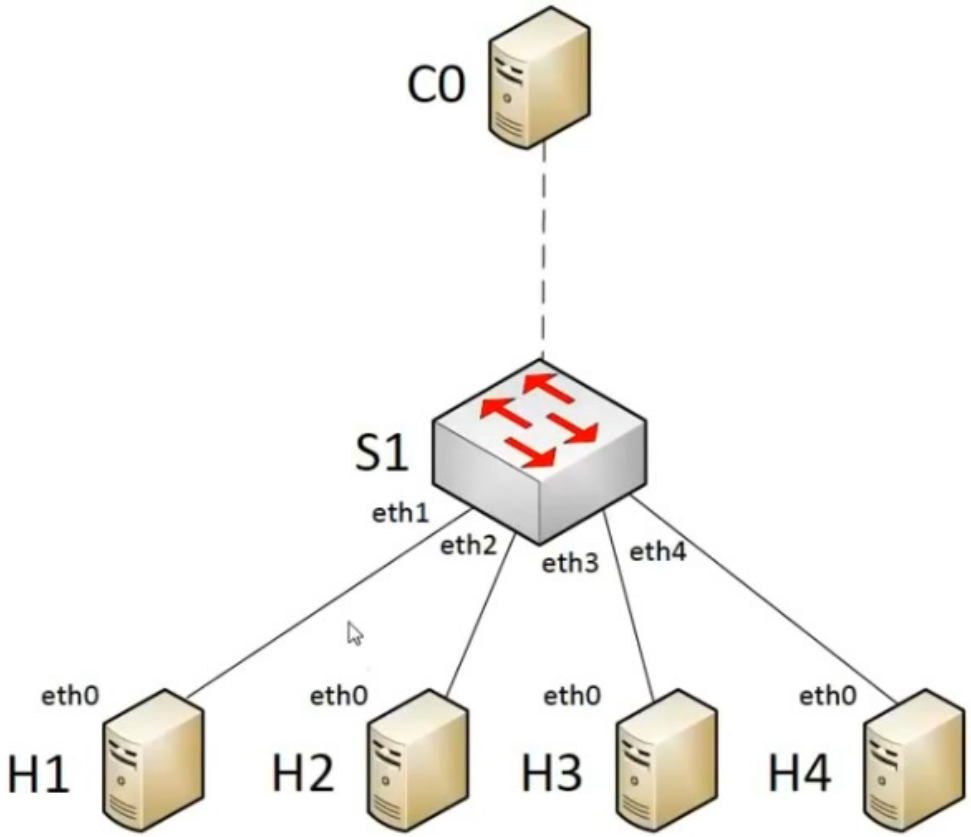
**$ sudo mn --topo single,3 --mac --switch ovsk --controller remote**



# Mininet Command - **topo**

## 4 Nodes in **Single** Topology

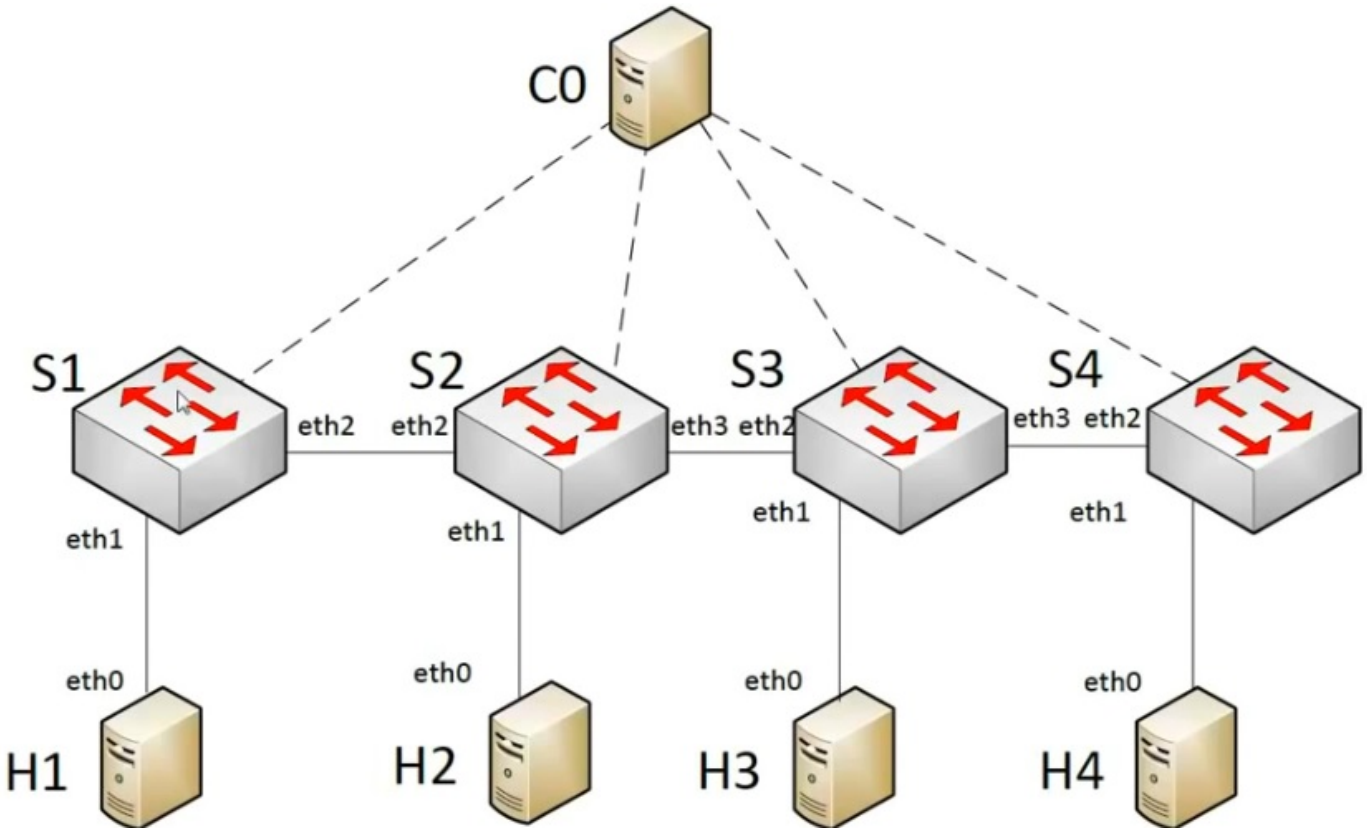
**$ sudo mn --topo=single,4**



# Mininet Command - **topo**

## 4 Nodes, 4 Switches in **Linear** Topology

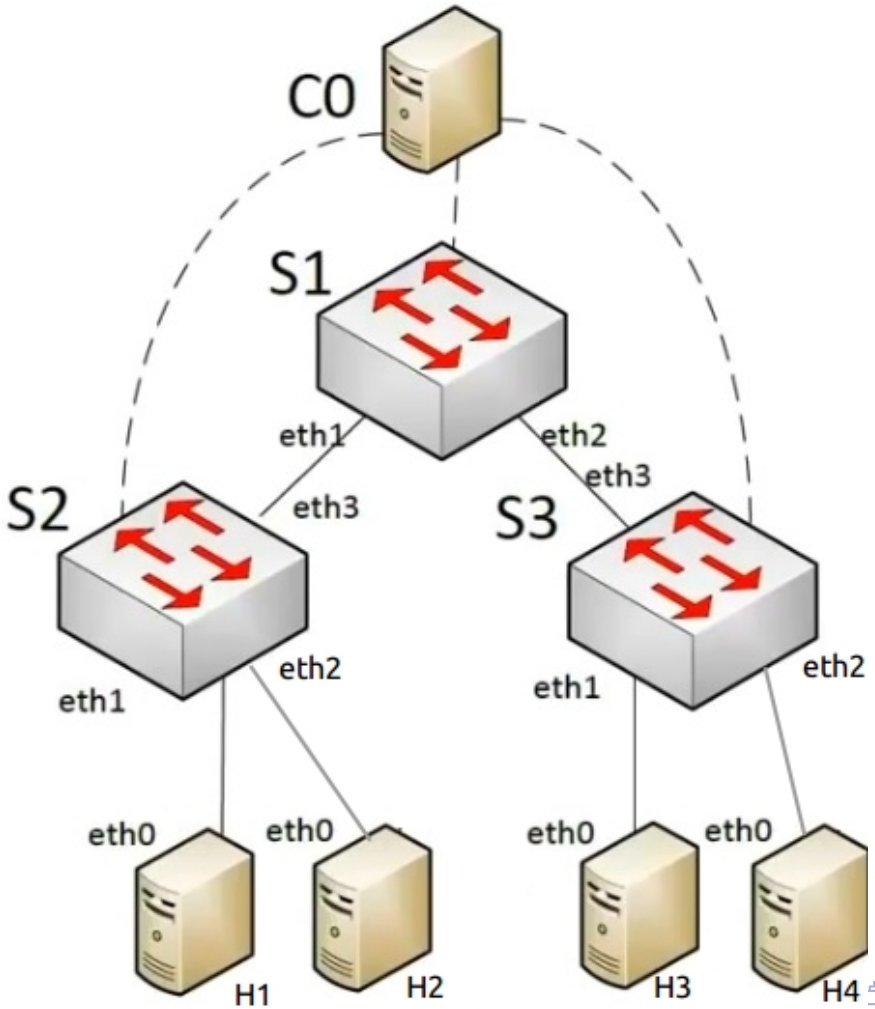
**$ sudo mn --topo=linear,4**



# Mininet Command - **topo**

**Tree** Topology: 2 levels deep & fan-out of 2

**$ sudo mn --topo=tree,2,2**



# Mininet Other Utilities - **dpctl**

- **dpctl** : enables visibility and control over a single switch's flow table. It is useful for debugging.

**dpctl show** : dumps out switch's port state and capabilities.

```
$ dpctl show tcp:127.0.0.1:6634
```

**dpctl dump-flows** : shows flows currently installed in SDN switch.

```
$ dpctl dump-flows tcp:127.0.0.1:6634
```

**dpctl add-flow** : installs the necessary flows. In your SSH terminal:

```
$ dpctl add-flow tcp:127.0.0.1:6634 in_port=1,actions=output:2
```

# Mininet Other Utilities - **Wireshark**

**Wireshark** : general (non-OF-specific) graphical utility for viewing packets.

- **$ sudo wireshark /dev/null**
- Click on Capture ->Interfaces.
- Check 'lo', the loopback interface.
- Click on the Start button.

# References



http://mininet.org/


B. Lantz, B. Heller, and N. McKeown. **A Network in a Laptop: Rapid Prototyping for Software-Defined Networks**, Proc. of ACM Hotnets'10, 2010.

# Thank You!!!

## Check Warm-Up Exercise.

# Exercise 1

Creating a network manually & running a simple performance test.

ex1.py

```
def perfTest():
    "Create network and run simple performance test"
    topo = SingleSwitchTopo(n=4)
    net = Mininet(topo=topo,host=CPULimitedHost, link=TCLink)
    net.start()
    print "Dumping host connections"
    # To Do 1: Dump hosts connections here
    print "Testing network connectivity"
    # To Do 2: Ping all hosts via single command
    print "Testing bandwidth between h1 and h3"
    # To Do 3: Test bandwidth between host 1 & host 3
    # HINT: First get the names of hosts from network then iperf them.
    net.stop()
```

## Commands to use:

1. dumpNodeConnections(net.hosts)
2. net.pingAll()
3. host_1, host_3 = net.get('host_1', 'host_3')
4. net.iperf((host_1, host_3))

# Exercise 2

Installing rules manually using *ovs-ofctl*.

*ex2.py*

```
for intf in switch1.intfs.values():
    print intf
    print switch1.cmd( 'ovs-vsctl add-port dp1 %s' % intf )

# To Do 1:
# add-flow for switch 1 & 2 for both to & fro
# Use Syntax :
# switch_name.cmd('ovs-ofctl add-flow datapath_id \"in_port=port_X actions=output:port_Y\"' )
```

## Info:

*DatapathId(dpid) = Unique identifier for switches.*

# Exercise 3

Simple controller.

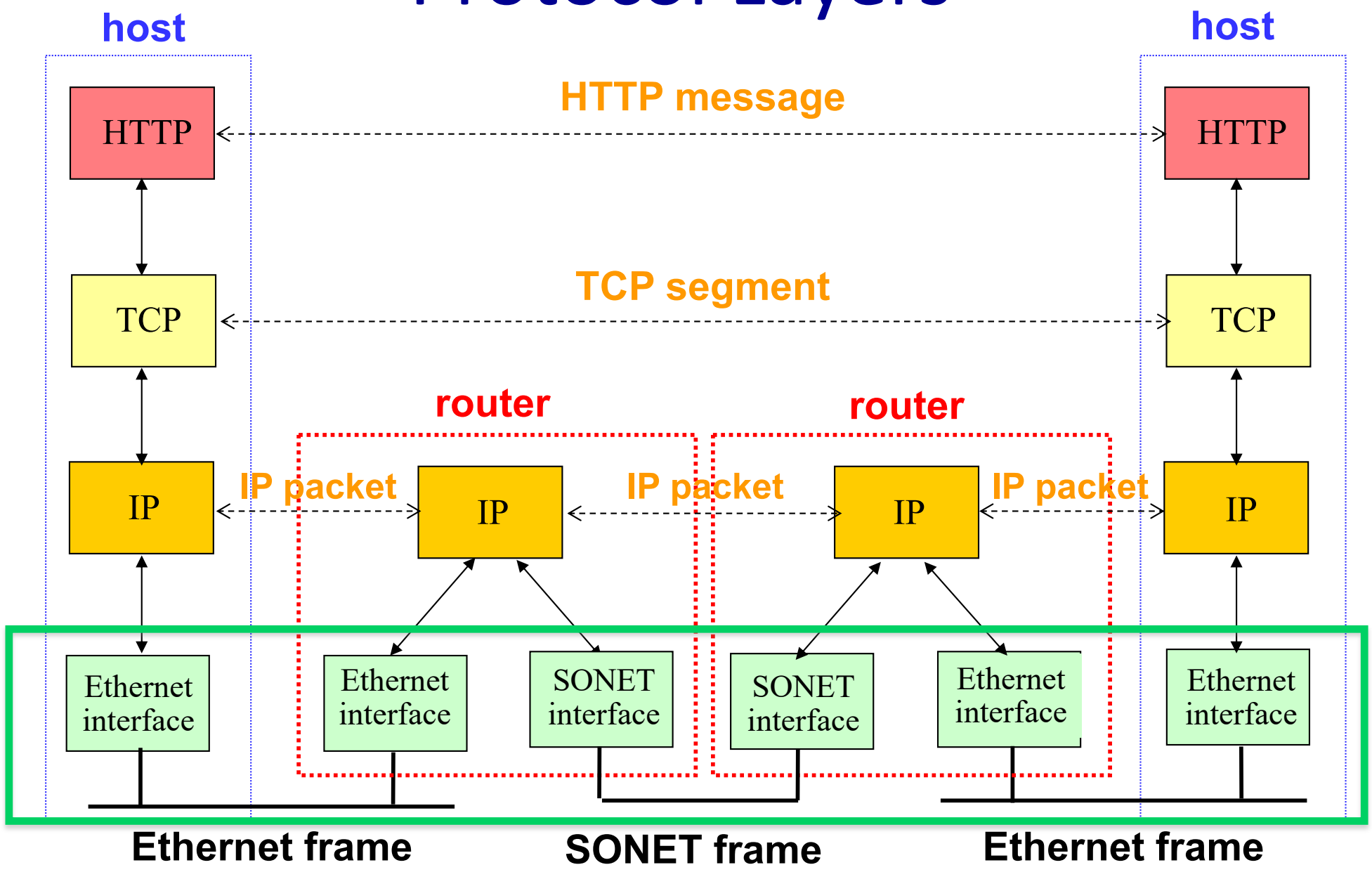
```
   print "s0_dpid=", s0_dpid
  #To Do 1: Do it for the rest of the switches as well!!! SEE TOPOLOGY FILE FOR DETAILS OF THE SWITCHES.


def _handle_PacketIn (event):
 global s0_dpid, s1_dpid, s2_dpid, s3_dpid
 print "PacketIn: ", dpidToStr(event.connection.dpid)

 if event.connection.dpid==s0_dpid:
   msg= of.ofp_flow_mod()
   msg.idle_timeout = 0
   msg.hard_timeout = 0
   # To Do 2: FLOOD THE PACKETS. USE SYNTAX:
   # msg.actions.append(of.ofp_action_output(port = FLOOD_TO_ALL_CMD))
   event.connection.send(msg)

 #To Do 3: Repeat for the rest of the switches as well!!!
```

*ex3_ctrl.py*

## How to:

1. PASTE "ex3_ctrl.py" UNDER /pox/ext/
2. Use Command: *of.OFPP_ALL* & run as remote controller.
3. Open separate terminal & run "ex3_tp.py".
4. If everything is OK; hosts should ping each other.

# Protocol Layers



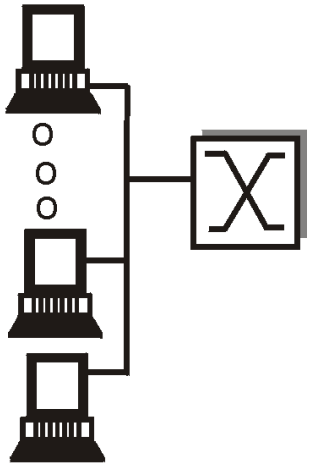
# Link = Medium + Adapters

# What is a Link?

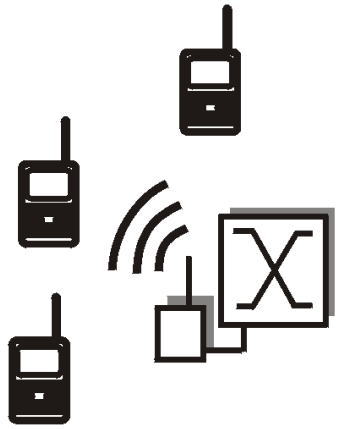
## Communication Medium

## Network Adapter

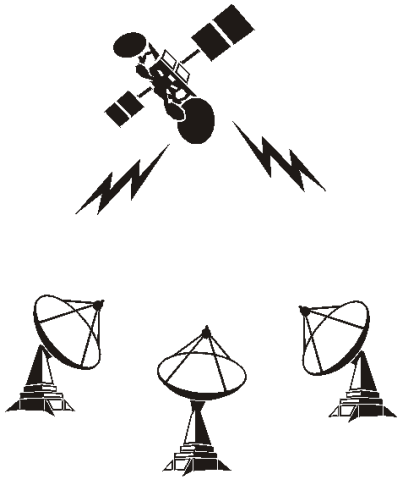

# Broadcast Links: Shared Media



shared wire
(e.g. Ethernet)



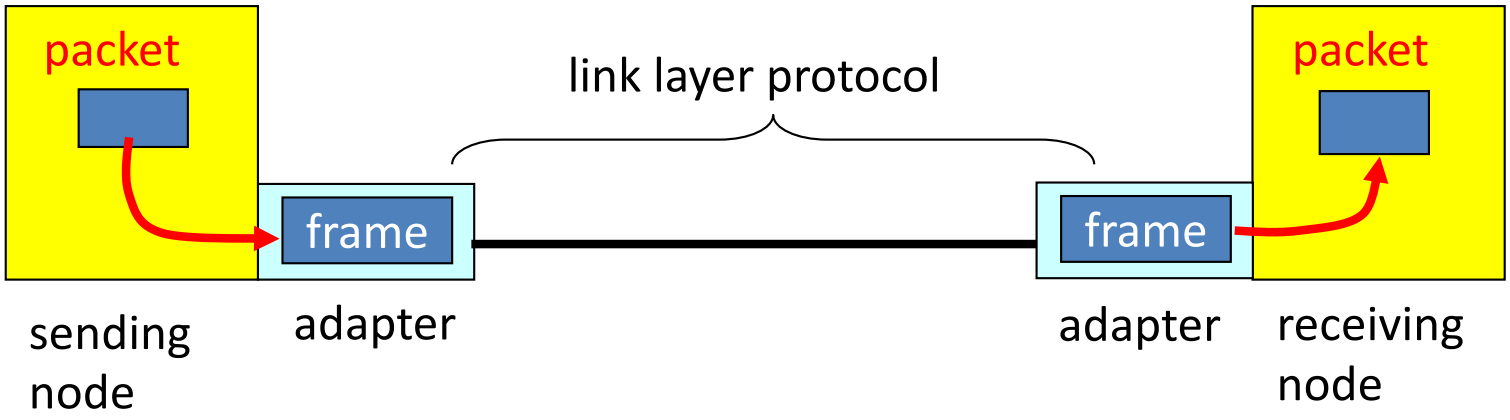
shared wireless
(e.g. Wavelan)



satellite


cocktail party

# Adaptors Communicating



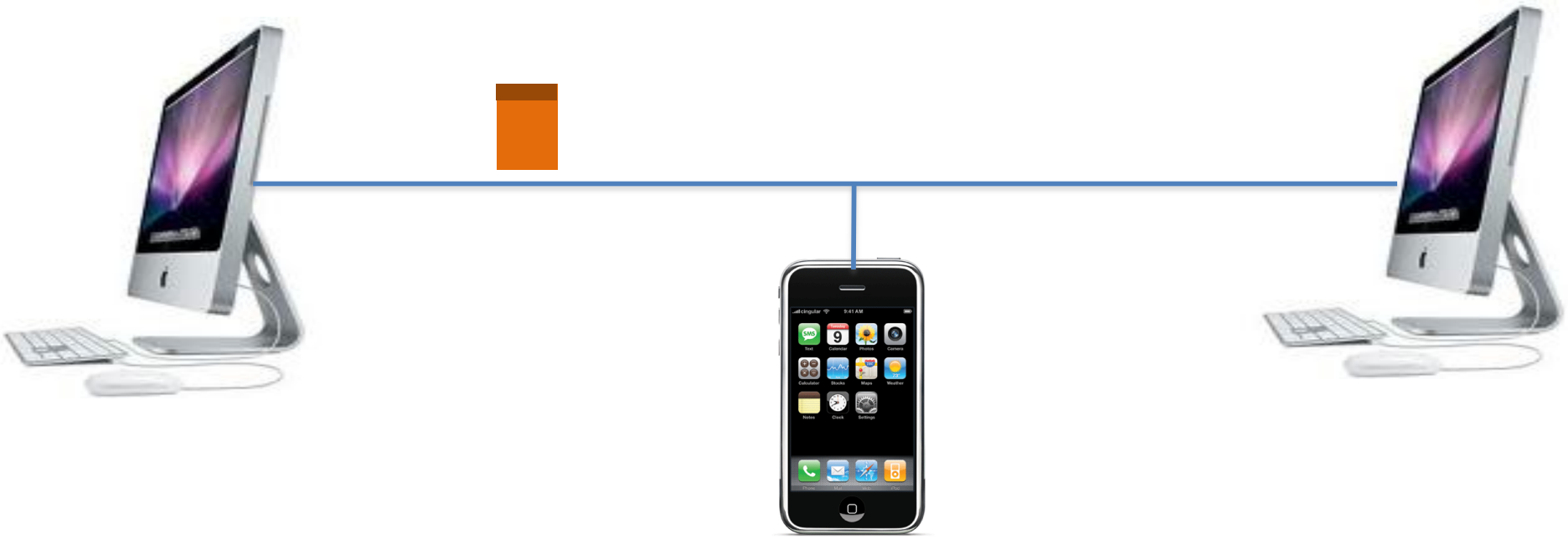

- Sending side
  - Encapsulates packet in a frame
  - Adds error checking bits, flow control, etc.
- Receiving side
  - Looks for errors, flow control, etc.
  - Extracts datagram and passes to receiving node

# Link-Layer Services

- Encoding
  - Represent the 0s and 1s
- Framing
  - Encapsulate packet into frame, adding header/trailer
- Error detection
  - Receiver detecting errors with checksums
- Error correction
  - Receiver optionally correcting errors
- Flow control
  - Pacing between sending and receiving nodes

# Addresses

# Medium Access Control Address



- Identify the sending and receiving adapter
  - Unique identifier for each network adapter
  - Identifies the intended receiver(s) of the frame
  - … and the sender who sent the frame

# Medium Access Control Address

- MAC address (e.g., 00-15-C5-49-04-A9)
  - Numerical address used within a link
  - Unique, hard-coded in the adapter when it is built
  - Flat name space of 48 bits
- Hierarchical allocation: Global uniqueness!
  - Blocks: assigned to vendors (e.g., Dell) by the IEEE
  - Adapters: assigned by the vendor from its block
- Broadcast address (i.e., FF-FF-FF-FF-FF-FF)
  - Send the frame to *all* adapters

# As an Aside: Promiscuous Mode

- Normal adapter: receives frames sent to
  - The local MAC address
  - Broadcast address FF-FF-FF-FF-FF-FF
- Promiscuous mode
  - Receive *everything*, independent of destination MAC
- Useful for packet sniffing
  - Network monitoring
  - E.g., wireshark, tcpdump

# Why Not Just Use IP Addresses?

- Links can support *any* network protocol
  - Not just for IP (e.g., IPX, Appletalk, X.25, …)
  - Different addresses on different kinds of links
- An adapter may move to a new location
  - So, cannot simply assign a static IP address
  - Instead, must reconfigure the adapter's IP address
- Must identify the adapter during bootstrap
  - Need to talk to the adapter to assign it an IP address

# Who Am I: Acquiring an IP Address


71-65-F7-2B-08-53

????

1A-2F-BB-76-09-AD

DHCP server

- Dynamic Host Configuration Protocol (DHCP)
  - Broadcast “I need an IP address, please!”
  - Response “You can have IP address 1.2.3.4.”

# Who Are You: Discovering the Receiver


71-65-F7-2B-08-53
1.2.3.4

1A-2F-BB-76-09-AD
1.2.3.5

0C-C4-11-6F-E3-98
1.2.3.6

- Address Resolution Protocol (ARP)
  - Broadcast “who has IP address 1.2.3.6?”
  - Response “0C-C4-11-6F-E3-98 has 1.2.3.6!”

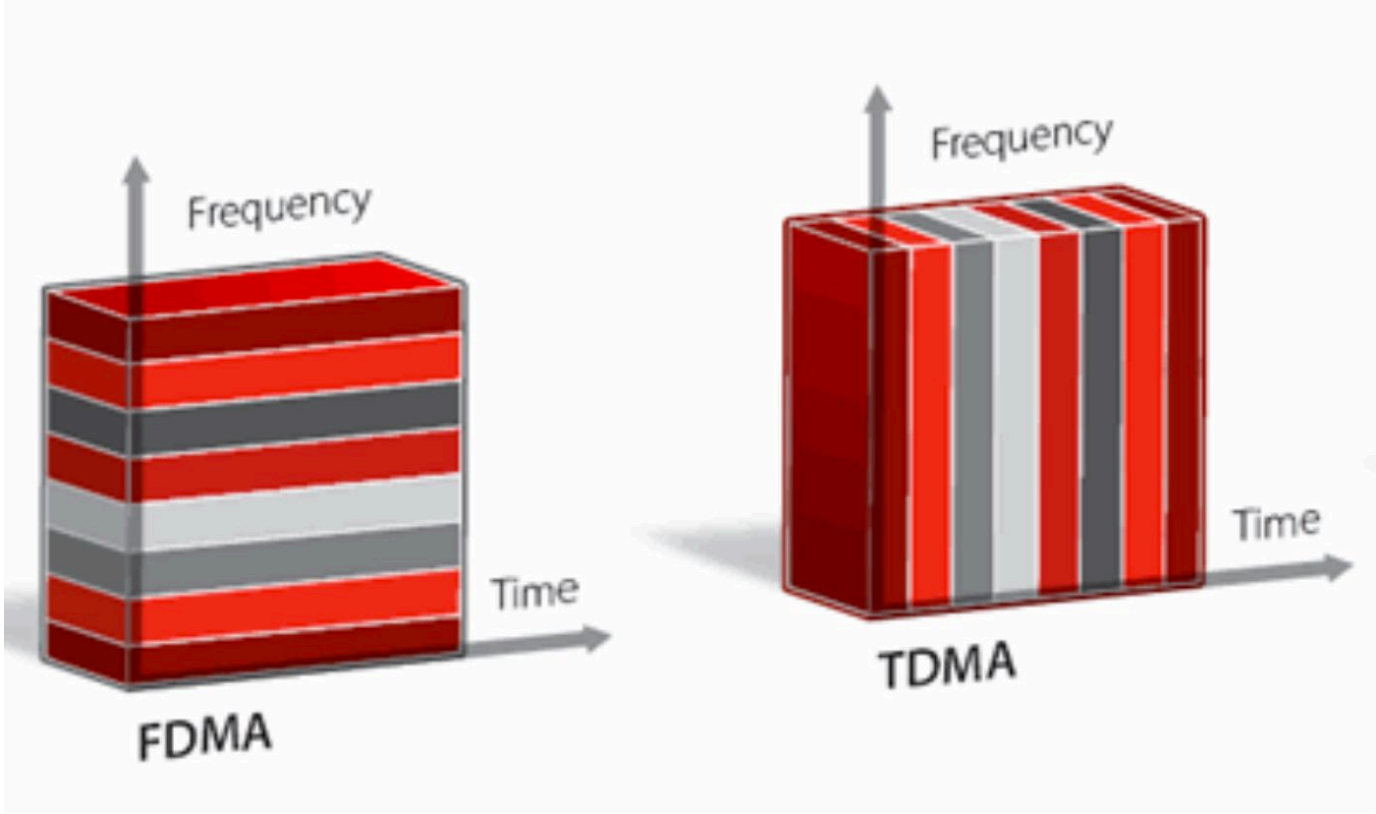
# Sharing the Medium

# Collisions


- Single shared broadcast channel
  - Avoid having multiple nodes speaking at once
  - Otherwise, collisions lead to garbled data

# Multi-Access Protocol

- Divide the channel into pieces
  - in frequency vs. in time

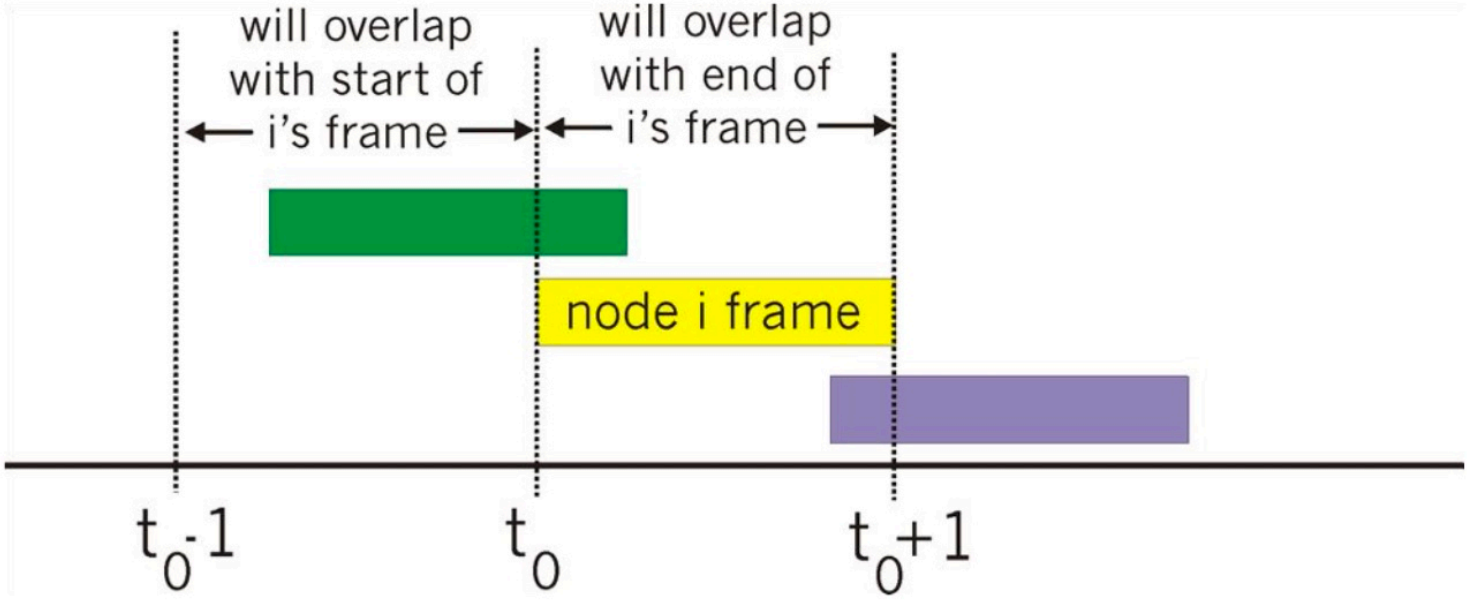


- Random access protocols
  - When node has packet to send
    - Transmit at full channel data rate R
    - No a priori coordination among nodes
    - Two or more transmitting nodes $\Longrightarrow$ Collision
  - Random access MAC protocol specifies
    - How to detect collisions
    - How to recover from collisions (e.g., via delayed retransmissions)
  - Examples of random access MAC protocols
    - Slotted and pure ALOHA
    - CSMA and CSMA/CD

- Random access protocols
  - Slotted ALOHA
    - Time is divided into equal size slots
      - Slot = Packet transmission time
    - Node with new packet
      - Transmit at beginning of next slot
    - If collision happen
      - Retransmit in future slots with probability p, until successful

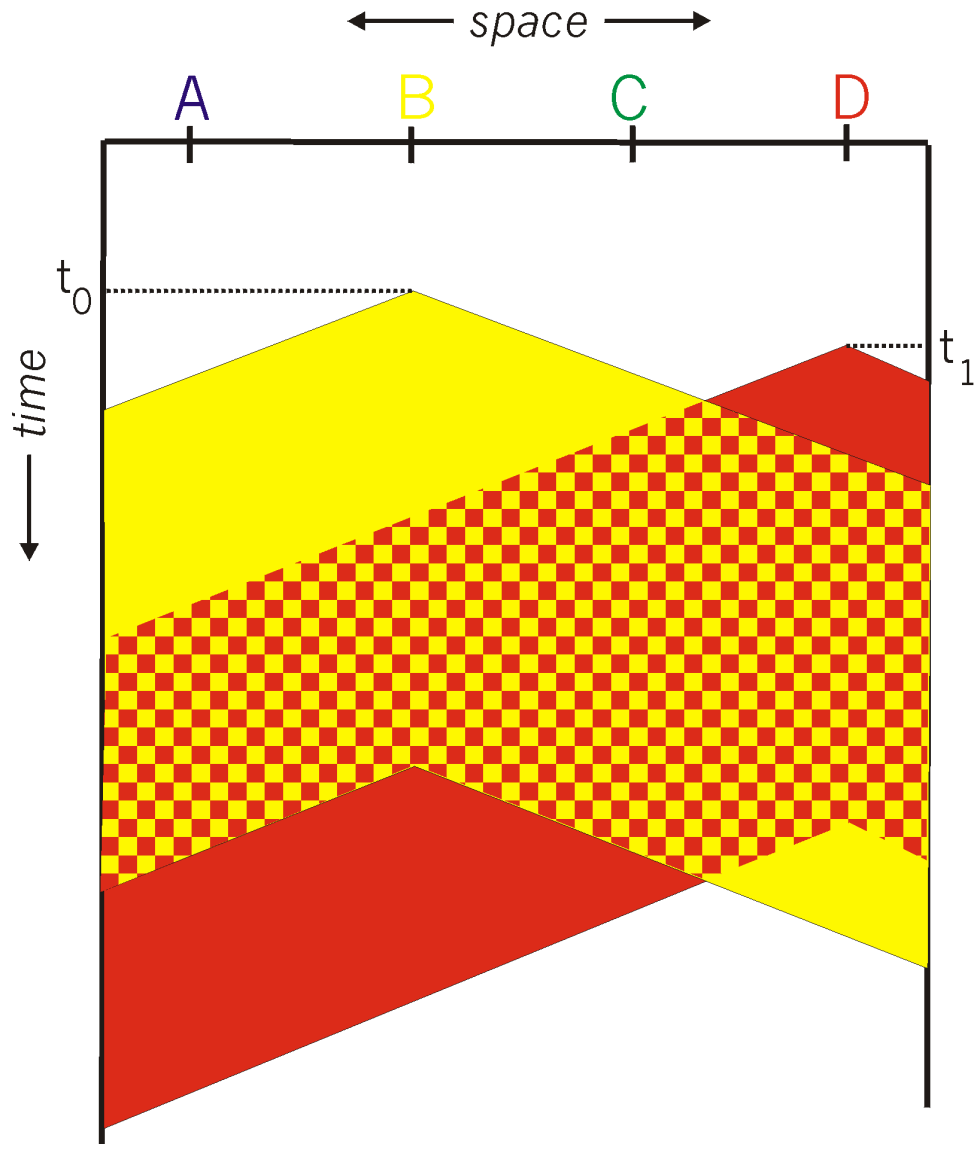

Success (S), Collision (C), Empty (E) slots

- Random access protocols
  - Pure (unslotted) ALOHA
    - Simpler, no synchronization
    - Node has new packet to transmit
      - Send without awaiting for beginning of slot
    - Collision probability increases
    - Packet sent at $t_0$ collide with other packets sent in $[t_0-1, t_0+1]$



# Carrier Sense Multiple Access

- Listen for other senders
  - Then transmit your data
- Collisions can still occur
  - Propagation delay
  - Wasted transmission



# Like Human Conversation…

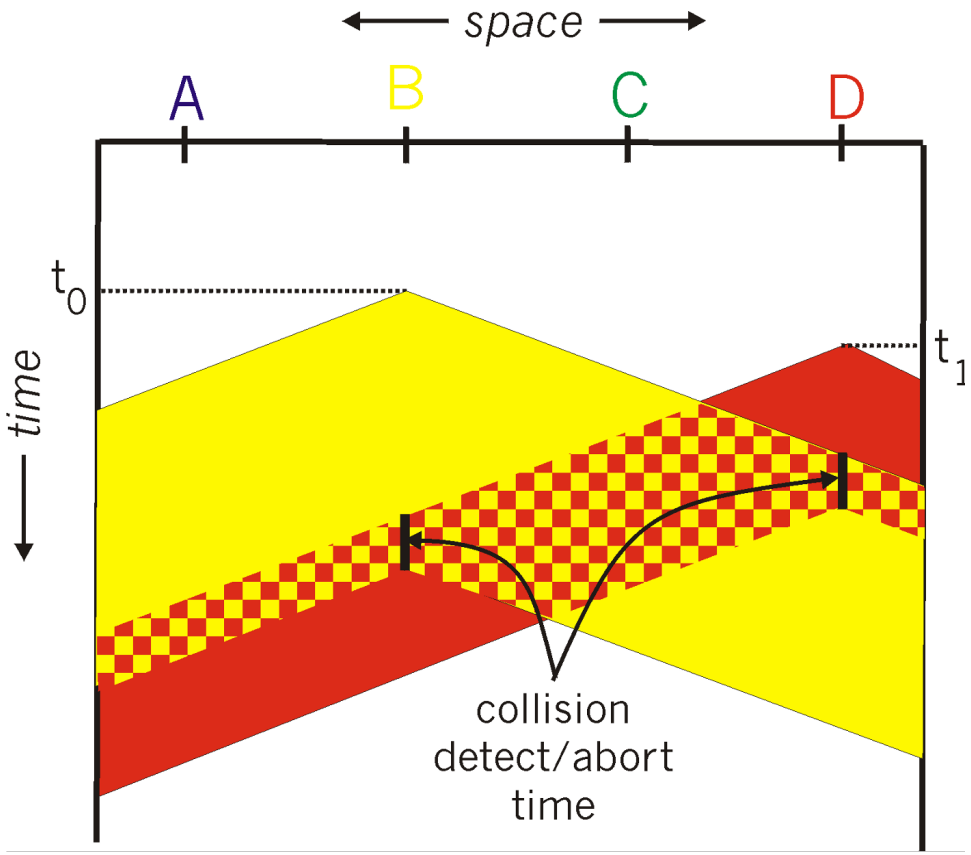
- Carrier sense
  - Listen before speaking
  - …and don't interrupt!


- Collision detection
  - Detect simultaneous talking
  - … and shut up!


- Random access
  - Wait for a random period of time
  - … before trying to talk again!

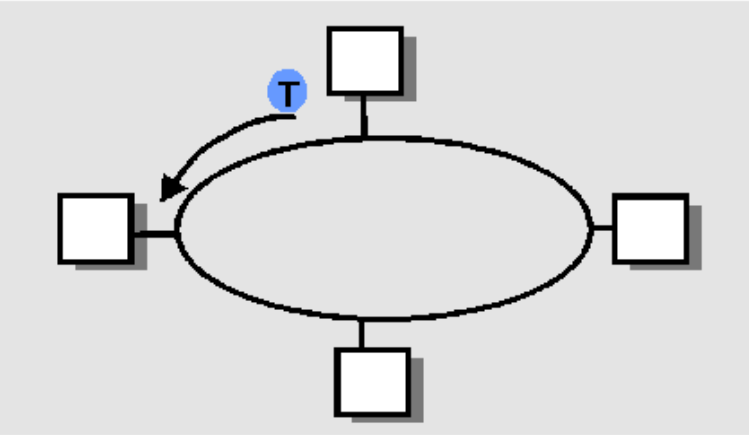
# CSMA/CD Collision Detection

- Detect collision
  - Abort transmission
  - Jam the link
- Wait random time
  - Transmit again
- Hard in wireless
  - Must receive data while transmitting



# Multi-Access Protocol

- Take turns
  - Do not transmit w/o token
  - With token, transmit for up to some max time/length
  - Pass token to right



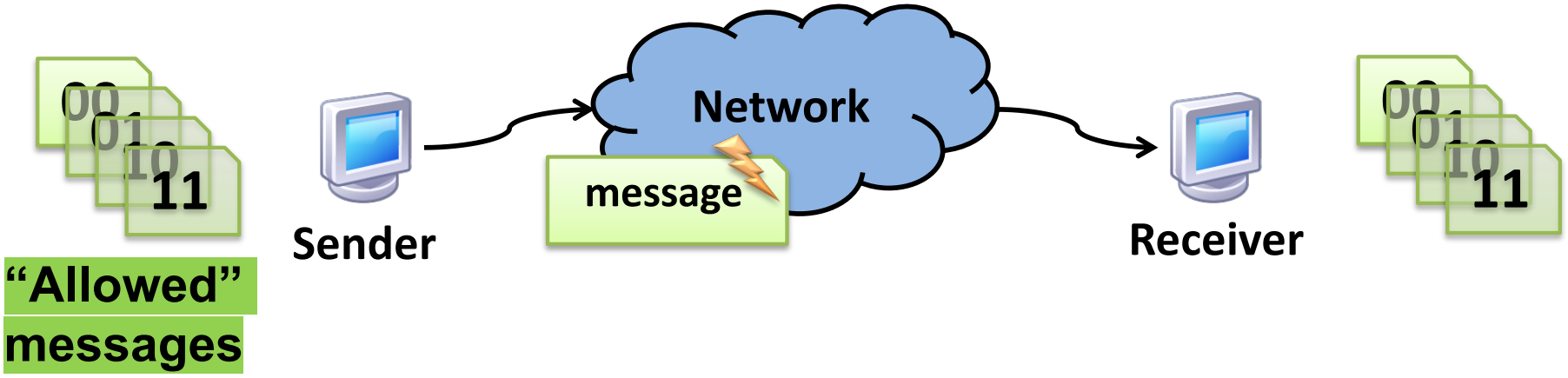

- Punt
  - Let collisions happen
  - … and detect and recover from them

# Comparing the Three Approaches

- Channel partitioning is
  - (a) Efficient/fair at high load, inefficient at low load
  - (b) Inefficient at high load, efficient/fair at low load
- "Taking turns"
  - (a) Inefficient at high load
  - (b) Efficient at all loads
  - (c) Robust to failures
- Random access
  - (a) Inefficient at low load
  - (b) Efficient at all load
  - (c) Robust to failures

# Error Control

# Error control: Motivation

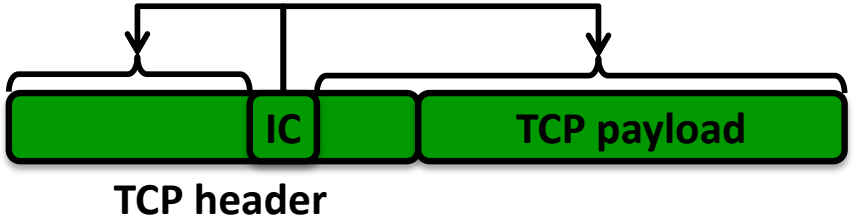



- *A priori,* any string of bits is an "allowed" **message**
  - Hence any **changes to the bits** (***bit errors***) the sender transmits produce "**allowed**" messages

- **Therefore without error control, receiver wouldn't know errors happened!**

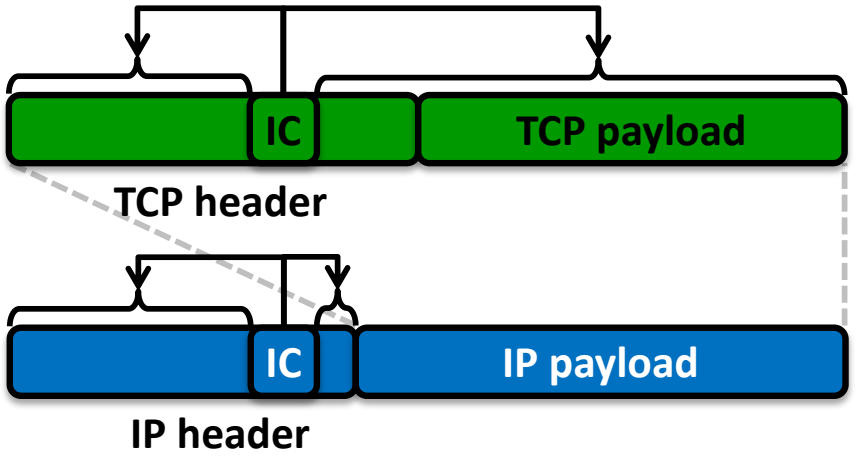
# Error control in the Internet stack

- **Transport layer**
  - **Internet Checksum (IC)** over TCP/UDP header, data



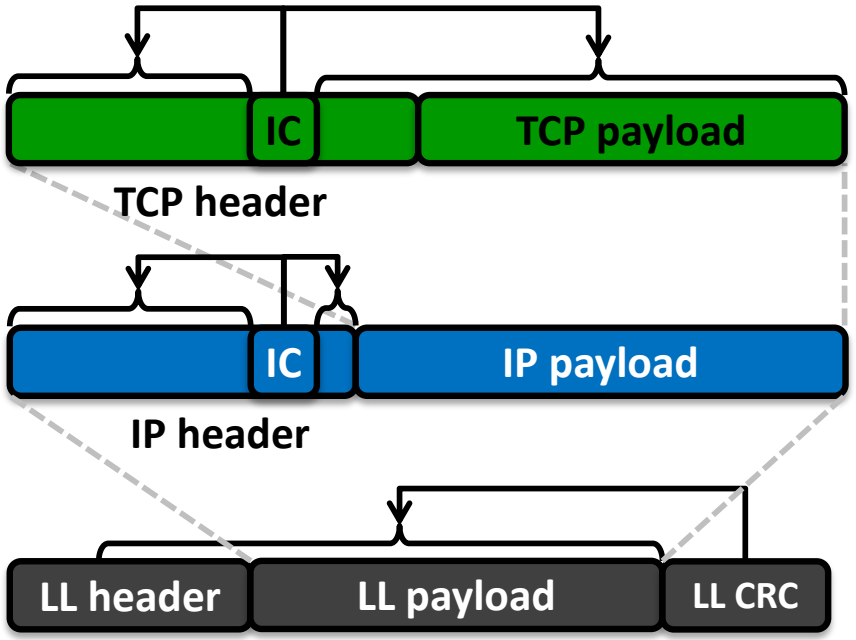
# Error control in the Internet stack

- **Transport layer**
  - **Internet Checksum (IC)** over TCP/UDP header, data
- **Network layer (L3)**
  - **IC** over IP header only



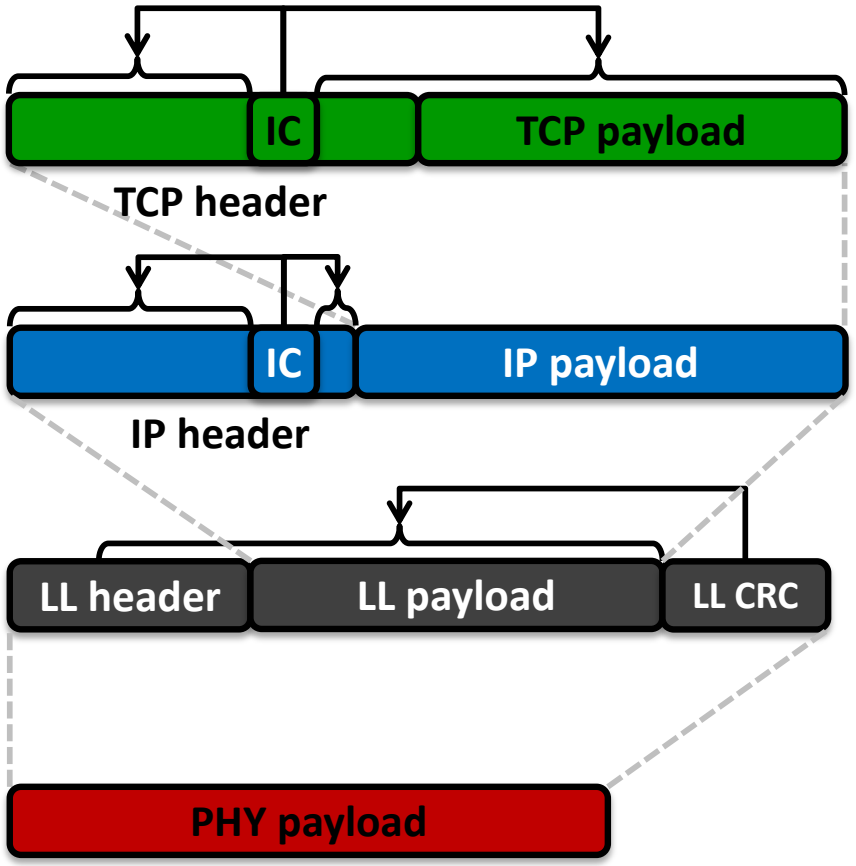
# Error control in the Internet stack

- **Transport layer**
  - **Internet Checksum (IC)** over TCP/UDP header, data
- **Network layer (L3)**
  - **IC** over IP header only
- **Link layer (L2)**
  - **Cyclic Redundancy Check (CRC)**



# Error control in the Internet stack

- **Transport layer**
  - **Internet Checksum (IC)** over TCP/UDP header, data
- **Network layer (L3)**
  - **IC** over IP header only
- **Link layer (L2)**
  - **Cyclic Redundancy Check (CRC)**
- **Physical layer (PHY)**
  - **Error Control Coding (ECC), or**
  - **Forward Error Correction (FEC)**



# Error control: Key Ideas

- **Reduce the set of "allowed" messages**
  - **Not every string of bits** is an "allowed" message
  - Receipt of a **disallowed** string of bits means that the **message was garbled** in transit over the network

- We call an allowable message (of *n* bits) a ***codeword***
  - **Not all** *n*-bit strings are codewords!
  - The remaining *n*-bit strings are **"space" between codewords**

- **Plan:** Receiver will **use that space** to both **detect** and **correct** errors in transmitted messages

# Encoding and decoding

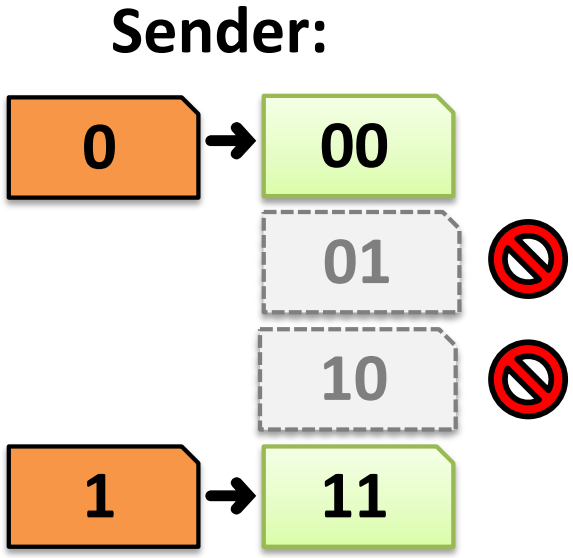
- **Problem: Not every string of bits is "allowed"**
  - But we want to be **able to send any** message!
  - *How can we send a "disallowed" message?*

- **Answer: Codes,** as a sender-receiver **protocol**
  - The sender must ***encode*** its messages ➜ codewords
  - The receiver then ***decodes*** received bits ➜ messages

- The **relationship between messages and codewords** isn't always obvious!

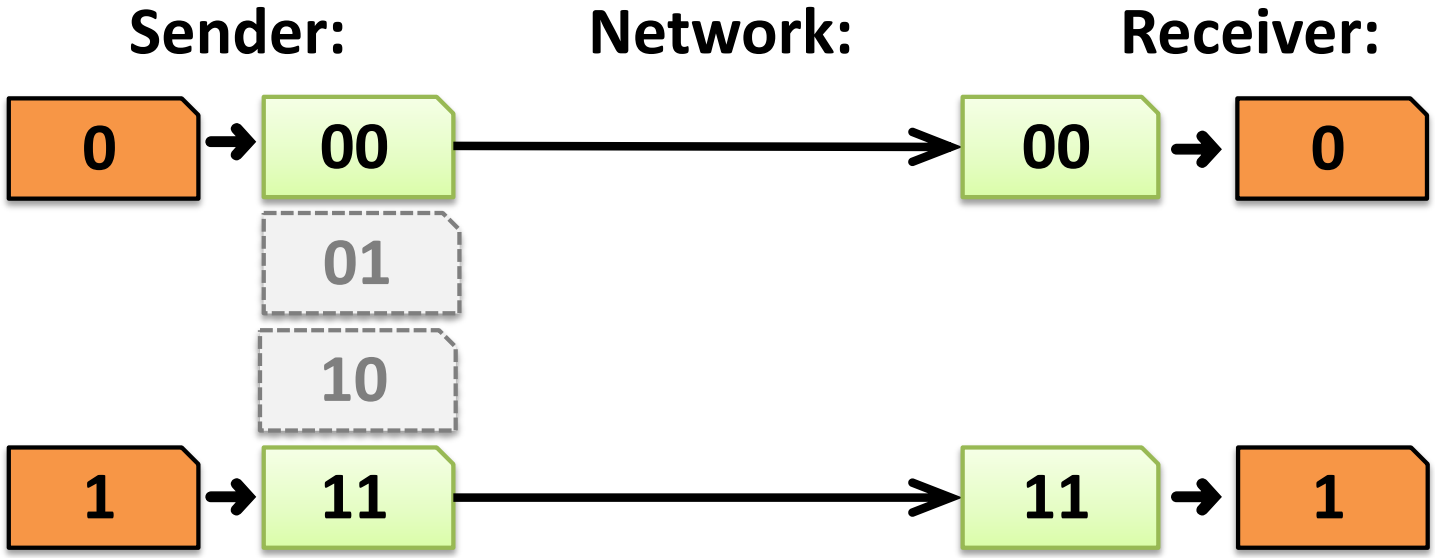
# A simple error-detecting code

- Let's start simple: suppose messages are one bit long
- Take the message bit, and **repeat** it once
  - This is called a ***two-repetition code***



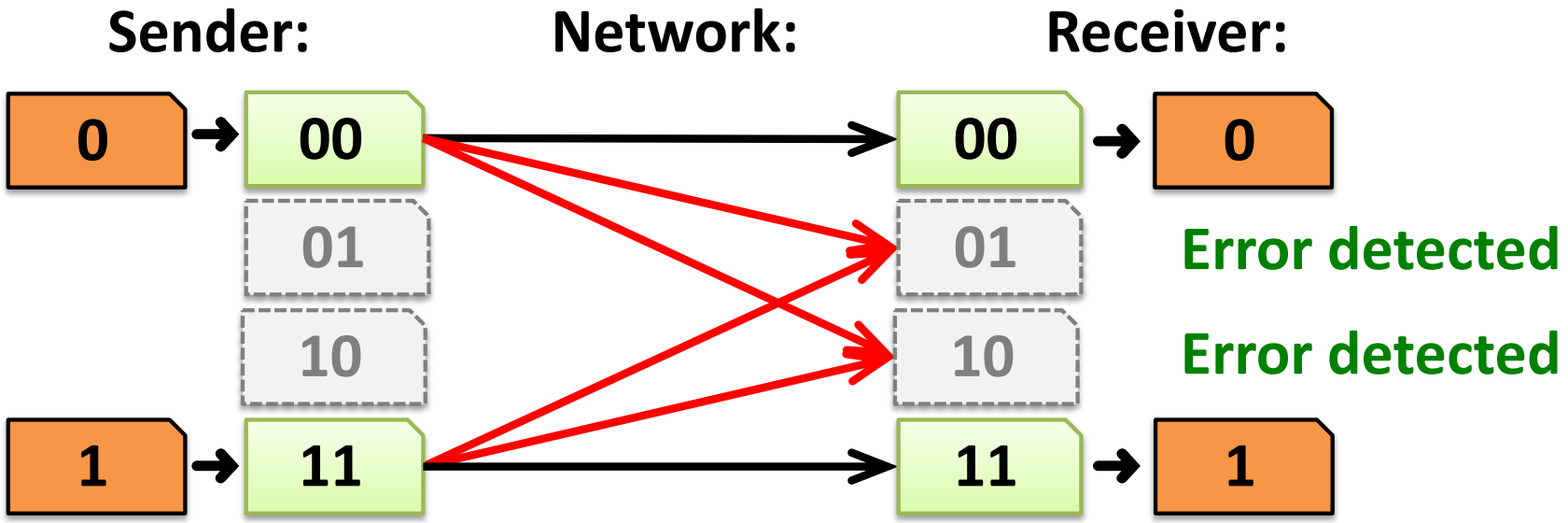
# Receiving the two-repetition code

- Suppose the network causes **no bit error**
- Receiver **removes repetition** to **correctly *decode*** the message bits



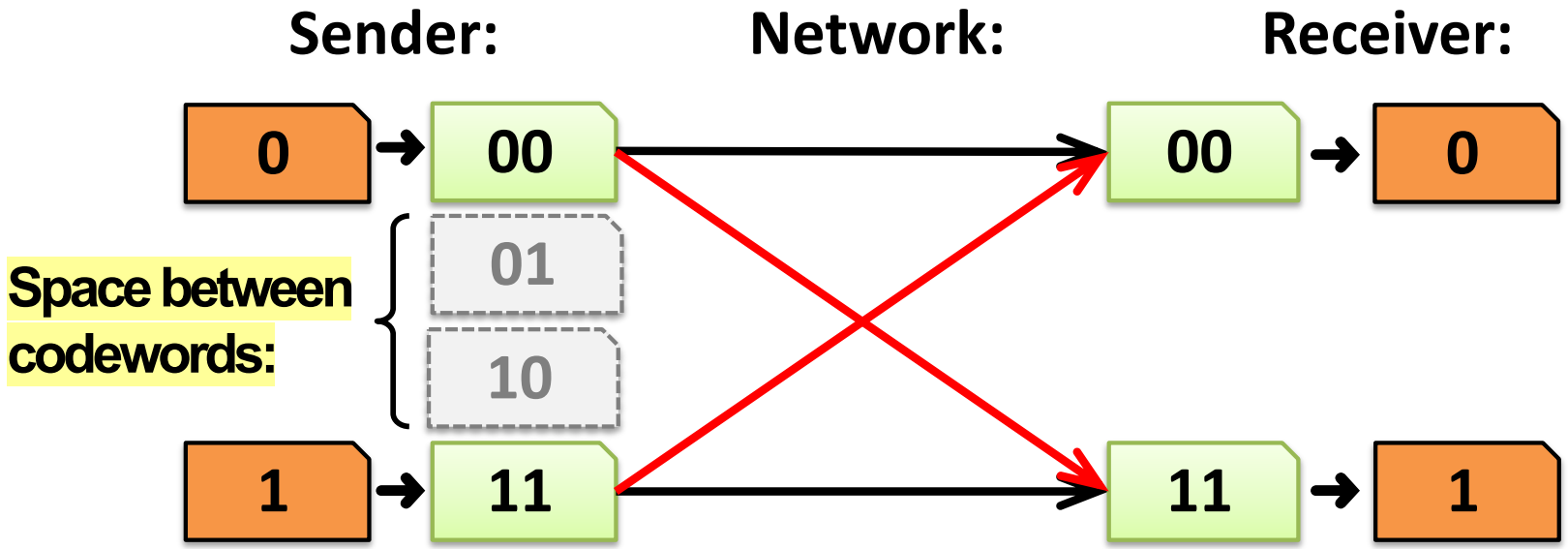
# Detecting one bit error

- Suppose the network causes up to **one bit error**
- The receiver **can detect** the error:
  - It received a **non-codeword**
- Can the receiver **correct** the error?
  - **No!** The **other** codeword could have been sent as well



# Reception with two bit errors

- Can receiver **detect** presence of **two bit errors?**
  - **No:** It has no way of telling which codeword was sent!
    - Enough bit errors that the sent codeword **"jumped over" the space between** codewords



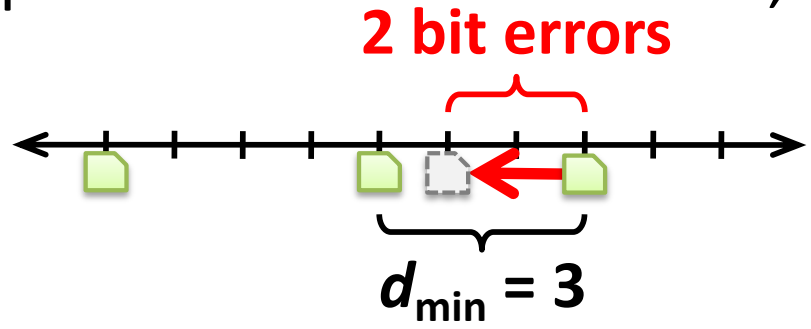
# Hamming distance

- Measures the **number of bit flips** to change **one codeword into another**
- ***Hamming distance*** between two messages $m_1$, $m_2$: The number of bit flips needed to change $m_1$ into $m_2$
- **Example:** Two bit flips needed to change codeword 00 to codeword 11, so they are Hamming distance of **two** apart:

# How many bit errors can we detect?

- Suppose the **minimum Hamming distance** between **any pair** of codewords is $d_{min}$

- Then, we **can detect at most $d_{min} - 1$ bit errors**
  - Will land in space between codewords, as we just saw




  - Receiver will flag message as **"Error detected"**

# Decoding error **detecting** codes

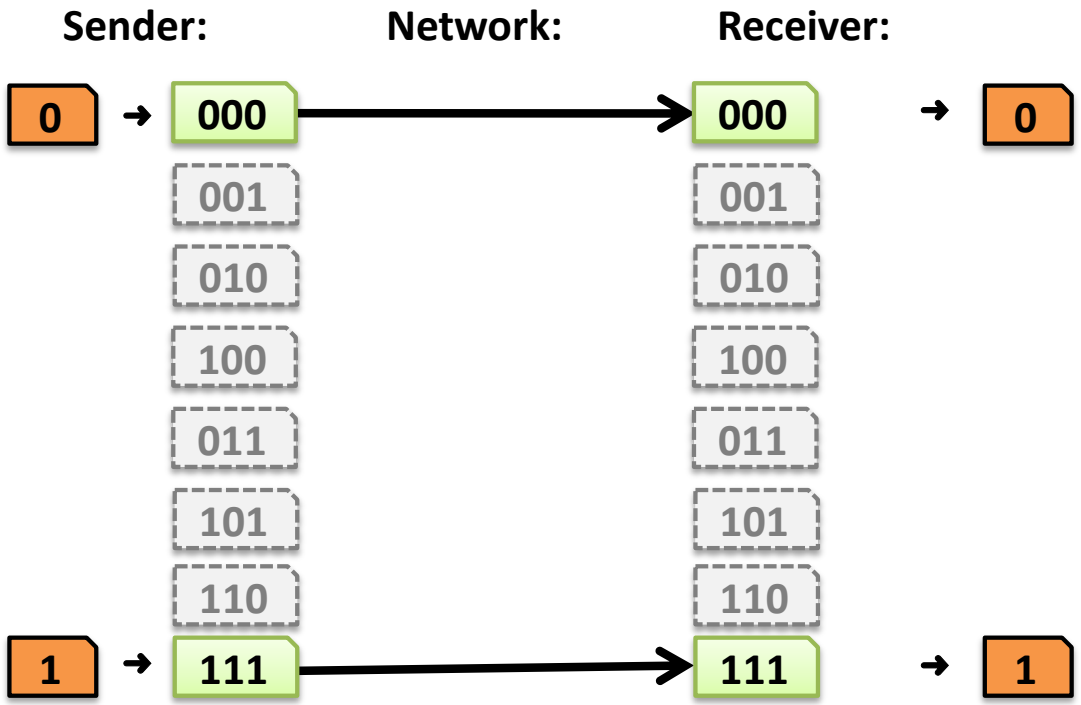
- **The receiver decodes** in a **two-step** process:

1. Map **received bits → codeword**
   - **Decoding rule:** Consider all codewords
     - **Choose** the one that **exactly matches** the received bits
     - **Return "error detected"** if none match

2. Map **codeword → source bits** and **"error detected"**
   - Use the **reverse map** of the sender

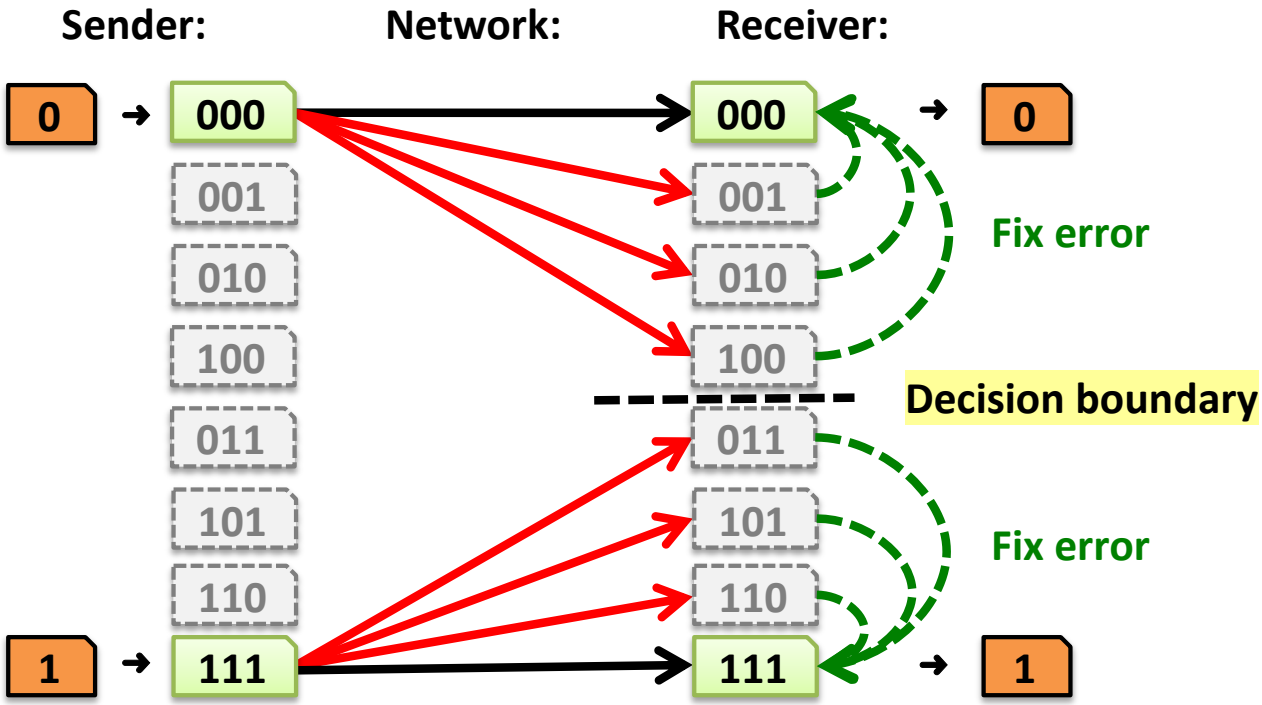
# A simple error-correcting code

- Let's look at **a three-repetition code**
- If **no errors,** it works like the two-repetition code:



# Correcting one bit error

- Receiver **chooses the closest codeword** (measured by Hamming distance) **to the received bits**
  - A ***decision boundary*** **exists** halfway between codewords



# Decoding error correcting codes

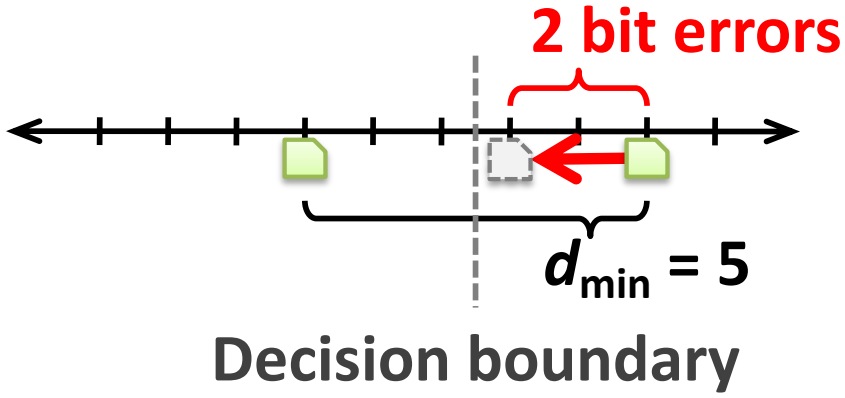
- **The receiver decodes** in a two-step process:

1. Map **received bits → codeword**
   - **Decoding rule:** Consider all codewords
     - **Choose one** with the **minimum Hamming distance** to the received bits

2. Map **codeword → source bits**
   - Use the **reverse map** of the sender

# How many bit errors can we correct?

- There is **≥ $d_{min}$ Hamming distance** between any two codewords
- So we can **correct ≤ $\lfloor (d_{min} - 1)/2 \rfloor$ bit flips:**
  - This many bit flips can't move received bits closer to another codeword, **across the decision boundary:**



# Code rate

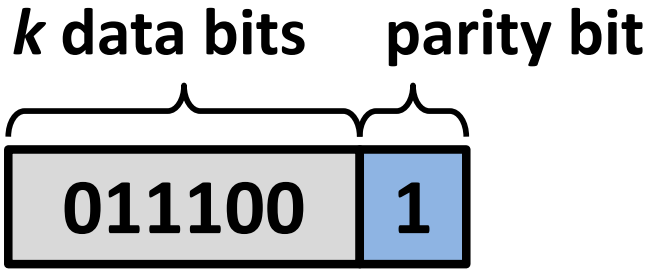
- Suppose **codewords** of length ***n*, messages** length ***k*** ($k < n$)
- The **code rate *R = k/n*** is a fraction between 0 and 1
- So, we have a **tradeoff:**
  - **High-rate codes** (*R* approaching one) generally **correct fewer errors,** but **add less overhead**
  - **Low-rate codes** (*R* close to zero) generally **correct more errors,** but **add more overhead**

# Parity bit

- Given a message of $k$ data bits $D_1, D_2, \ldots, D_k$, append a ***parity bit P*** to make a codeword of length $n = k + 1$

  – P is the exclusive-or of the data bits:
    - $P = D_1 \oplus D_2 \oplus \cdots \oplus D_k$

  – Pick the parity bit so that **total number of 1's is even**



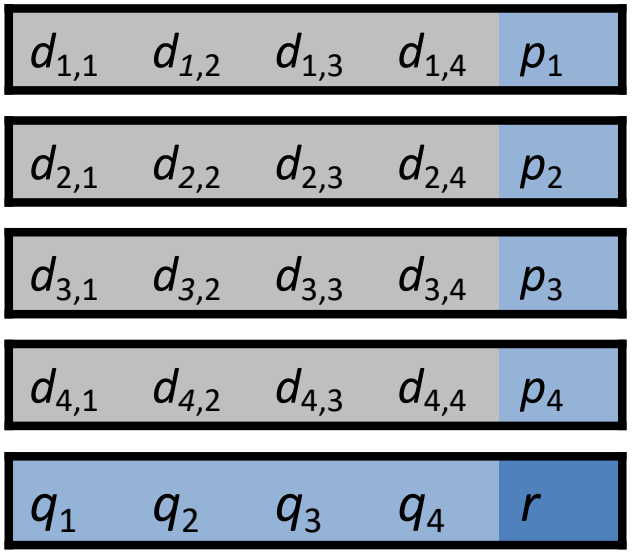
# Checking the parity bit

- **Receiver: counts number of 1s** in received message
  - **Even:** received message is a codeword
  - **Odd:** isn't a codeword, and **error detected**
    - But receiver doesn't know where, so **can't correct**
- What about $d_{min}$?
  - Change one data bit → change parity bit, so $d_{min} = 2$
    - So parity bit **detects 1 bit error, corrects 0**
- Can we **detect and correct more errors,** in general?

# Two-dimensional parity

- Break up data into multiple rows
  - Parity bit **across** each row ($p_i$)
  - Parity bit **down each column** ($q_i$)
  - Add a parity bit $r$ covering row parities

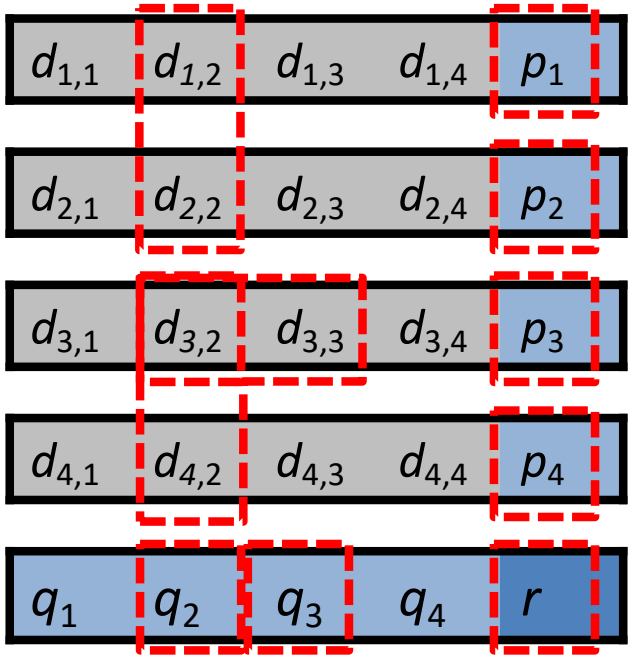
$$p_j = d_{j,1} \oplus d_{j,2} \oplus d_{j,3} \oplus d_{j,4}$$
$$q_j = d_{1,j} \oplus d_{2,j} \oplus d_{3,j} \oplus d_{4,j}$$
$$r = p_1 \oplus p_2 \oplus p_3 \oplus p_4$$




- This example has rate 16/25:

# Two-dimensional parity: Properties

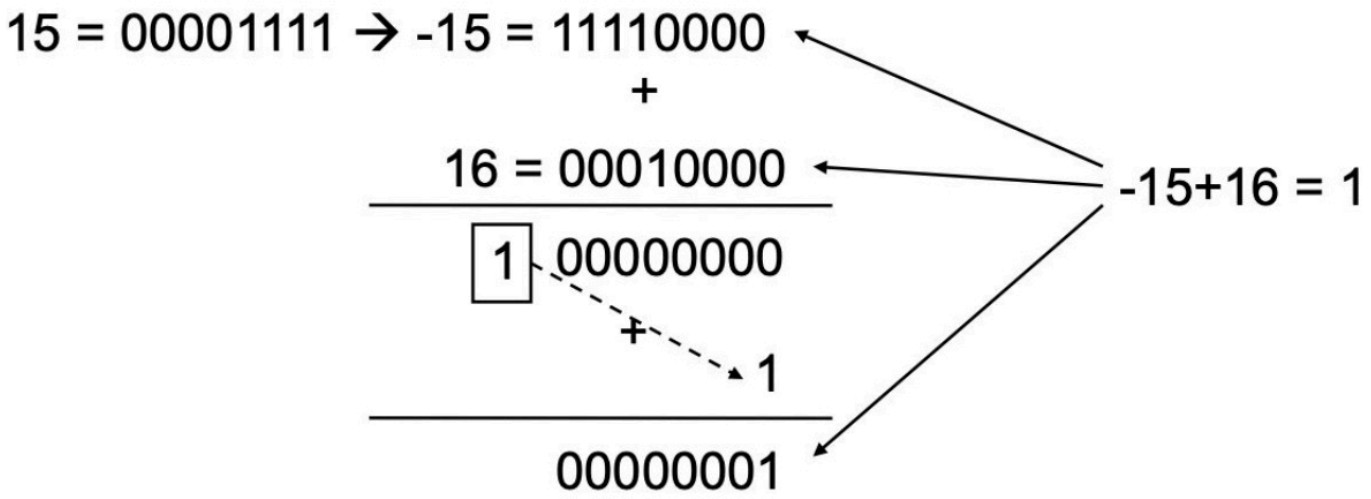
- Flip **1 data bit, 3 parity bits** flip
- Flip **2 data bits, ≥ 2 parity bits** flip
- Flip **3 data bits, ≥ 3 parity bits** flip




- Therefore, $\boldsymbol{d_{min} = 4}$, so
  - Can detect ≤ 3 bit errors
  - Can correct single-bit errors (*how?*)

- 2-D parity detects **most** four-bit errors

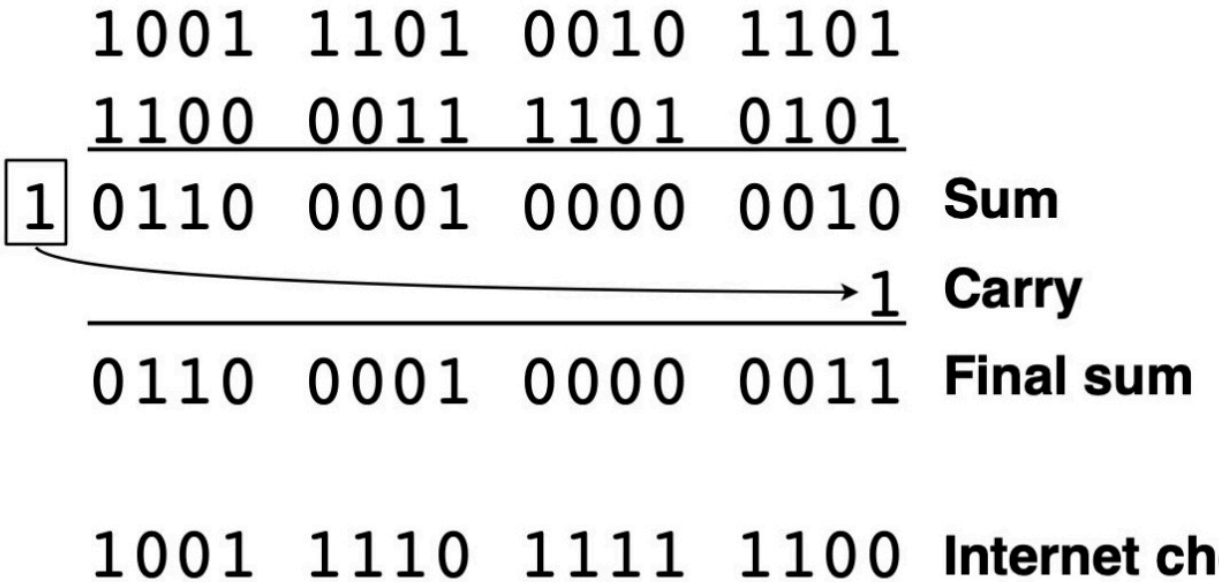
- Checksum
  - Internet checksum
    - Uses 1's complement addition
      - Negative number –x is x with all bits inverted
      - When two numbers are added, carry-on is added to result



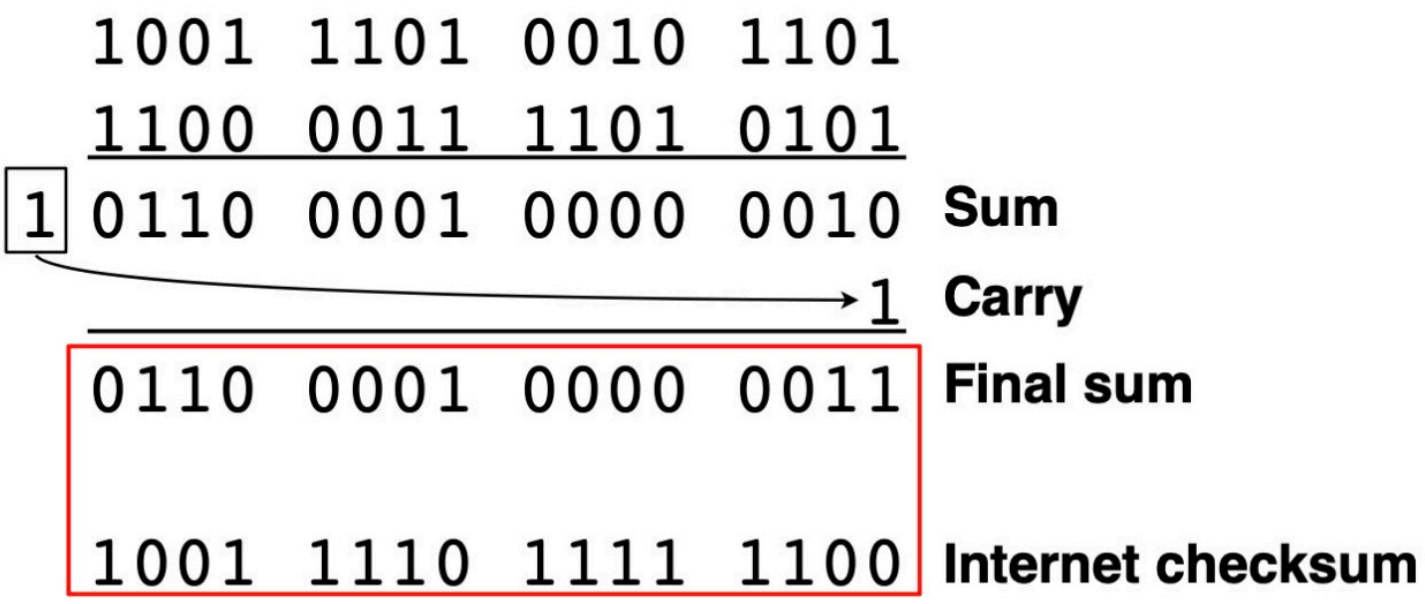
- Checksum
  - Internet checksum
    - Example
      - If data is

        1001 1101 0010 1101 1100 0011 1101 0101

      - Convert to 16-bit words, then add, carry, and invert



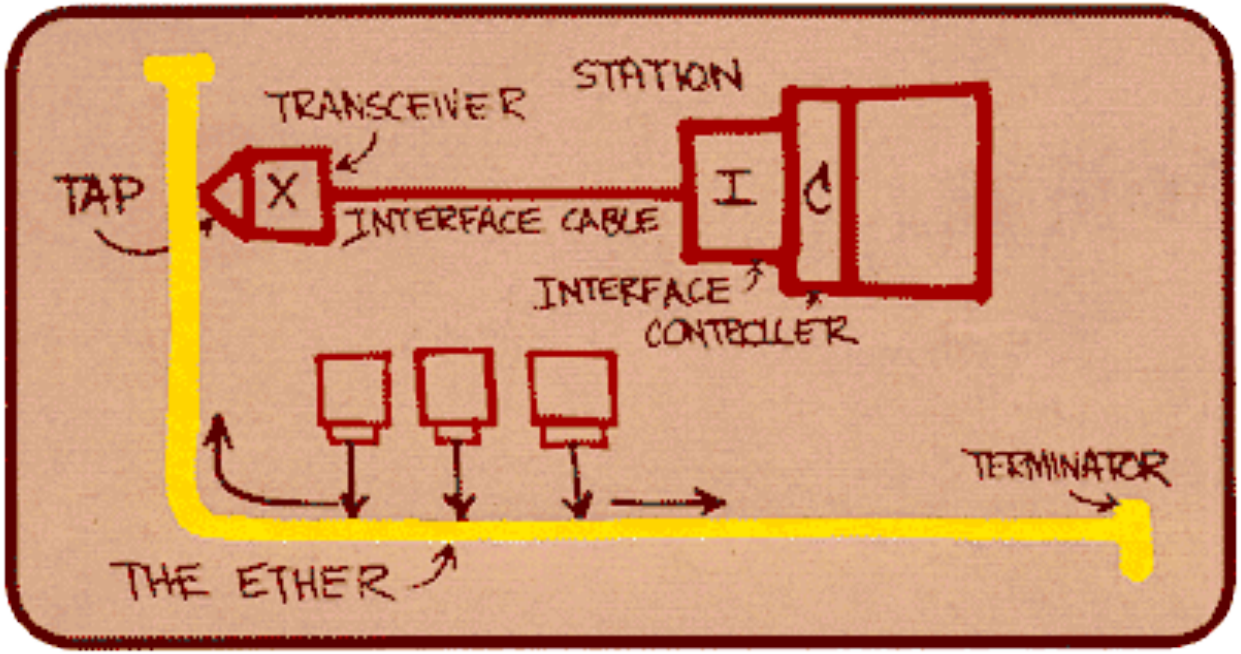
- Checksum
  - Internet checksum
    - Why 1's complement?
      - Receiver computes final sum, inverts it, then compares with received Internet checksum
        - Instead just add final sum to received Internet checksum
          - See if all 1s (1111 1111 1111 1111)



# Ethernet

# Ethernet

- Dominant wired LAN technology
- First widely used LAN technology
- Kept up with speed race: 10 Mbps – 40 Gbps

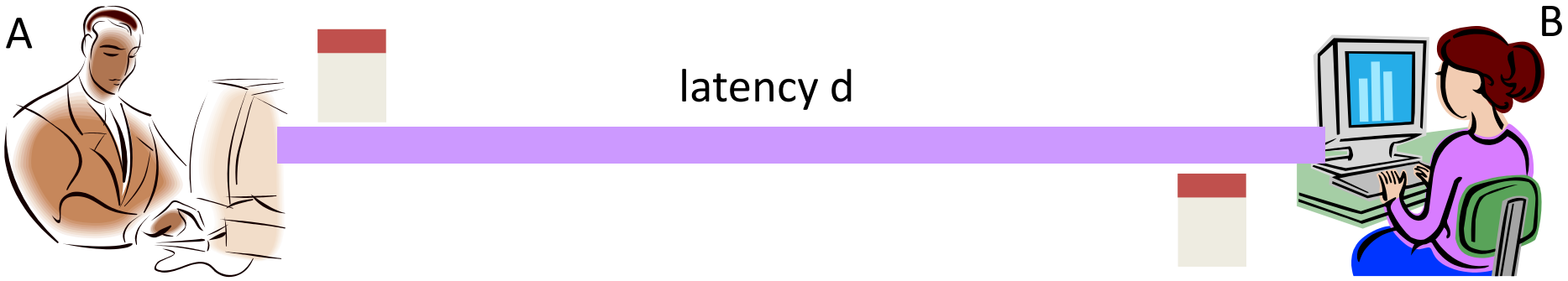



Metcalfe's Ethernet sketch

# Ethernet Uses CSMA/CD

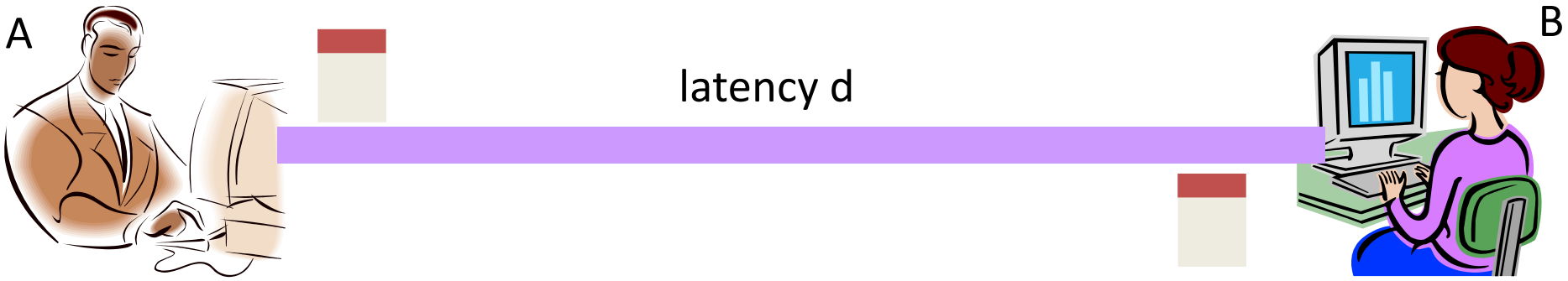
- Carrier Sense: wait for link to be idle
  - Channel idle: start transmitting
  - Channel busy: wait until idle
- Collision Detection: listen while transmitting
  - No collision: transmission is complete
  - Collision: abort transmission, and send jam signal
- Random Access: exponential back-off
  - After collision, wait random time before trying again
  - After $m^{th}$ collision, choose K randomly from {0, …, $2^m$-1}
  - … and wait for K*512 bit times before trying again

# Limitations on Ethernet Length




- Latency depends on physical length of link
  - Time to propagate a packet from one end to other
- Suppose A sends a packet at time t
  - And B sees an idle line at a time just before t+d
  - … so B happily starts transmitting a packet
- B detects a collision, and sends jamming signal
  - But A doesn't see collision till t+2d

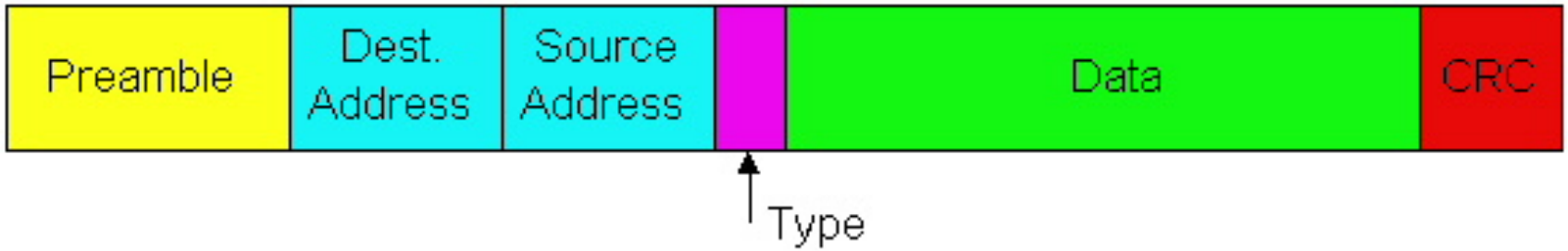
# Limitations on Ethernet Length




- A needs to wait for time 2d to detect collision
  - So, A should keep transmitting during this period
  - … and keep an eye out for a possible collision
- Imposes restrictions on Ethernet
  - Maximum length of the wire: 2500 meters
  - Minimum length of the packet: 512 bits (64 bytes)

# Ethernet Frame Structure

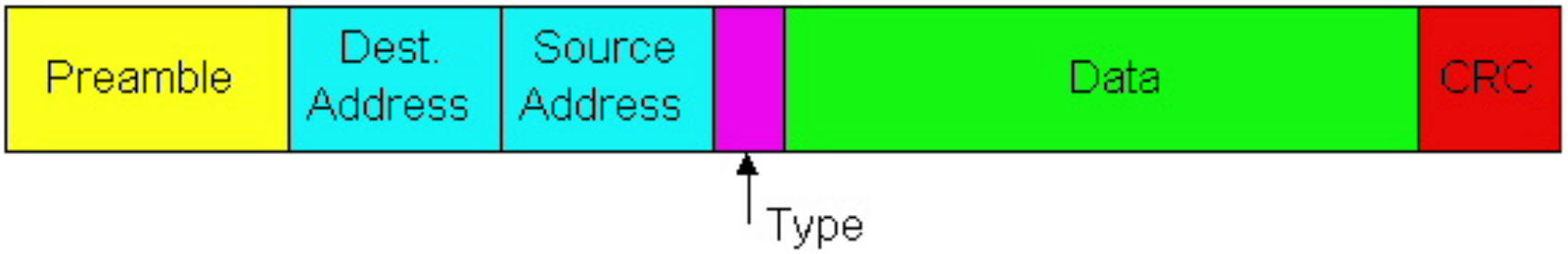
- Sending adapter encapsulates packet in frame




- Preamble: synchronization
  - Seven bytes with pattern 10101010, followed by one byte with pattern 10101011
  - Used to synchronize receiver, sender clock rates

# Ethernet Frame Structure

- Addresses: source and destination MAC addresses
  - Adaptor passes frame to network-level protocol
    - If destination is local MAC address or broadcast address
  - Otherwise, adapter discards frame
- Type: indicates the higher layer protocol
  - Usually IP
  - But also Novell IPX, AppleTalk, …
- CRC: cyclic redundancy check

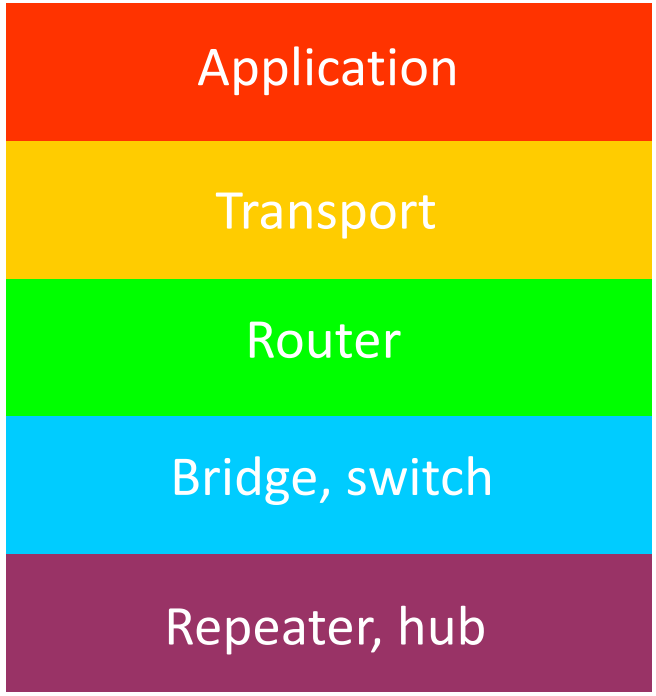


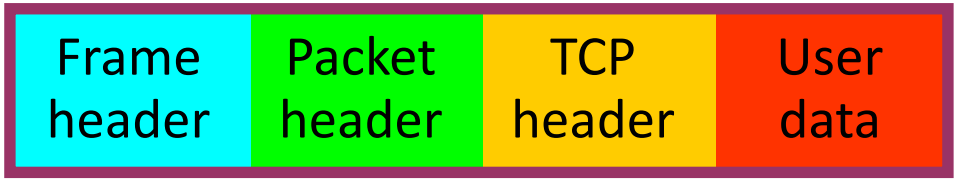
# Unreliable, Connectionless Service

- Connectionless
  - No handshaking between send and receive adapter
- Unreliable
  - Receiving adapter doesn't send ACKs or NACKs
  - Packets passed to network layer can have gaps
  - Gaps can be filled by transport protocol (e.g., TCP)
  - Otherwise, the application will see the gaps

# Summary: Multiple Layers

- Different devices switch different things
  - Network layer: packets (routers)
  - Link layer: frames (bridges and switches)
  - Physical layer: electrical signals (repeaters and hubs)



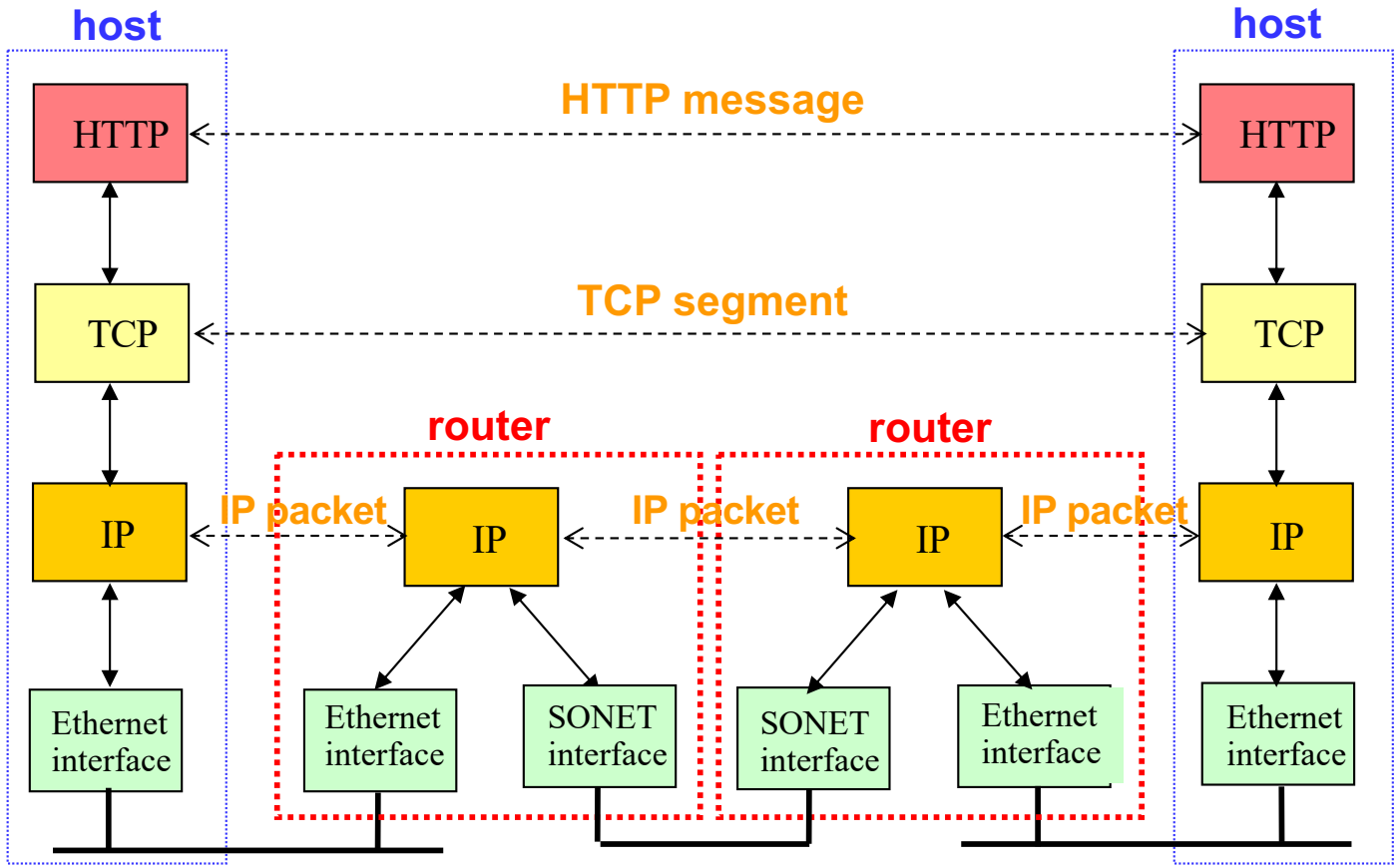



# Conclusion

- Links
  - Connect two or more network adapters
  - … each with a unique address
  - … over a shared communication medium

- Coming next
  - Network layer (IP)

# Today: Hubs, Switches, and Routers, Oh My!



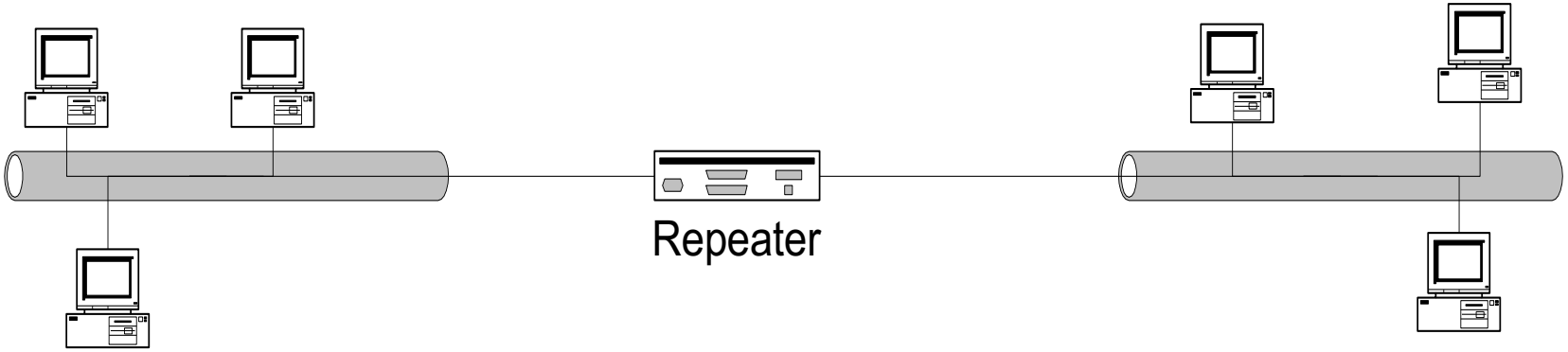
# Terminology

- Hubs and Repeaters
  - Connect machines on same “layer 2” LAN
  - Broadcast: All frames are sent out all physical ports
- Switches and Bridges
  - Connect machines on same “layer 2” LAN
  - Only send frames to selected physical port based on destination MAC address
- Routers
  - Connect between LANs at “layer 3”, e.g., wide area
  - Only send packet to selected physical port based on destination IP address

# “Layer 2”
# Hubs and Switches

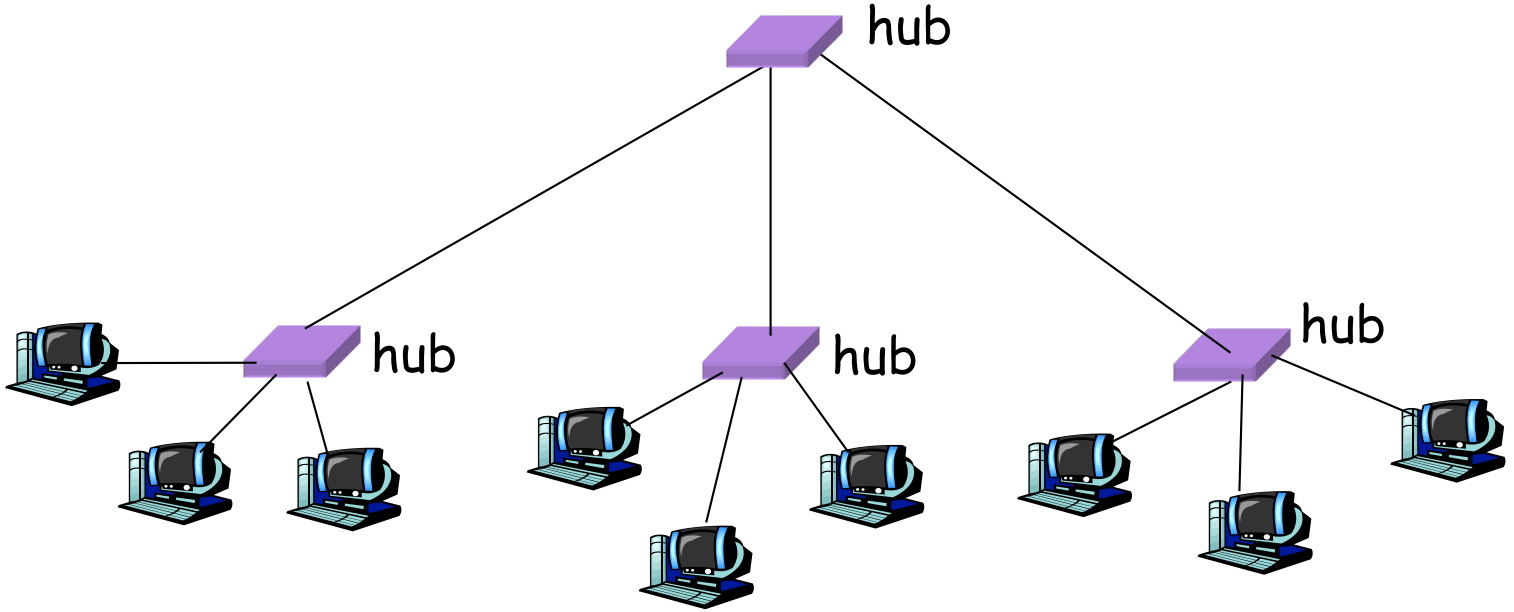
# Physical Layer: Repeaters

- Distance limitation in local-area networks
  - Electrical signal becomes weaker as it travels
  - Imposes a limit on the length of a LAN
- Repeaters join LANs together
  - Analog electronic device
  - Continuously monitors electrical signals
  - Transmits an amplified copy



# Physical Layer: Hubs

- Joins multiple input lines electrically
  - Designed to hold multiple line cards
  - Do not necessarily amplify the signal
- Very similar to repeaters
  - Also operates at the physical layer



# Hub: Overview

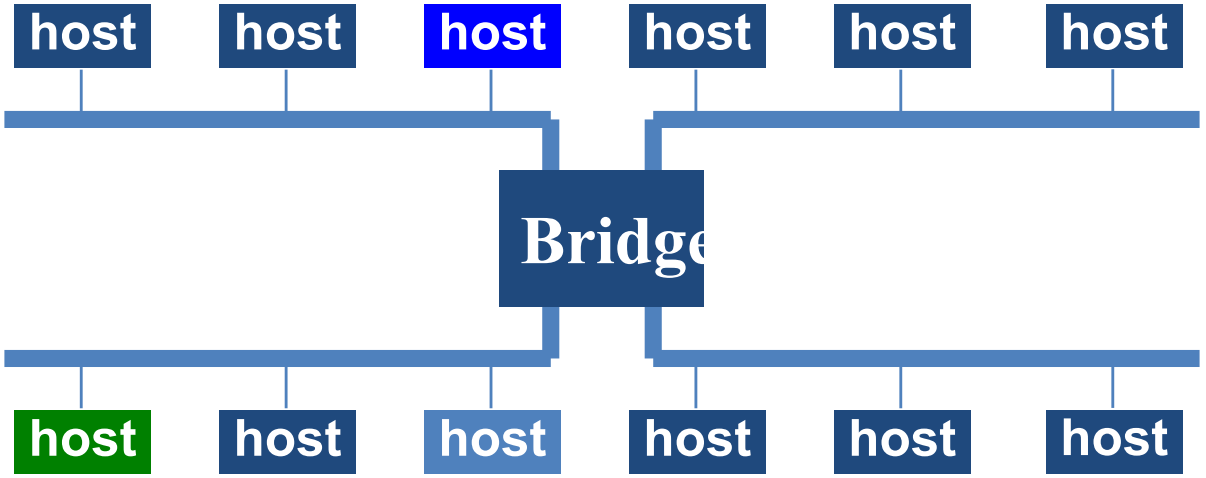
- **Layer 1** Device
- **1** Collision Domain
- **Half-**Duplex
- **Wasted** Bandwidth
- **Security** Risks
- **Replaced** by Switches

# Limitations of Repeaters and Hubs

- One large shared link
  - Each bit is sent everywhere
  - So, aggregate throughput is limited
- Cannot support multiple LAN technologies
  - Does not buffer or interpret frames
  - Can't interconnect between different rates/formats
- Limitations on maximum nodes and distances
  - Shared medium imposes length limits
  - E.g., cannot go beyond 2500 meters on Ethernet

# Link Layer: Bridges

- Connects two or more LANs at the link layer
  - Extracts destination address from the frame
  - Looks up the destination in a table
  - Forwards the frame to the appropriate segment
- Each segment can carry its own traffic

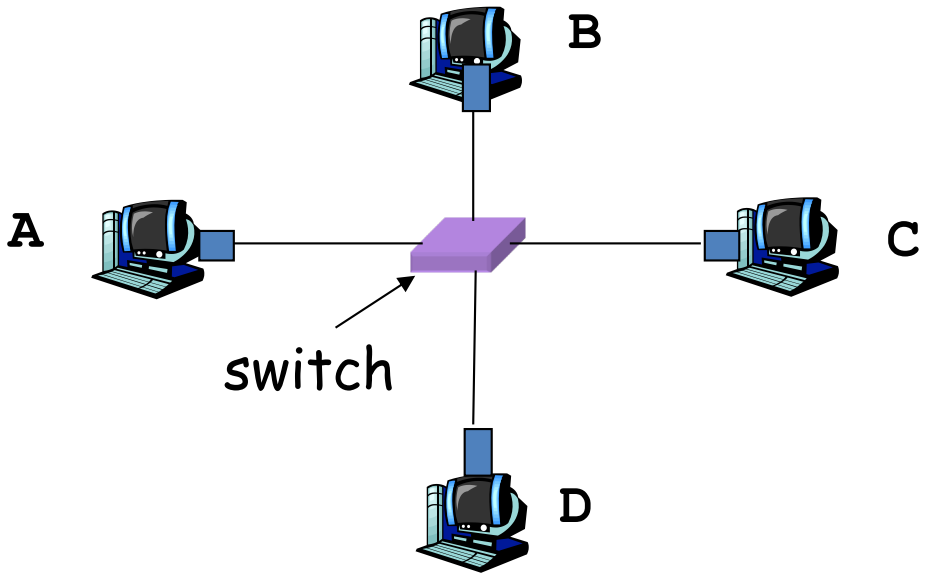


# Bridge: Overview

- **Layer 2** Device
- **Segments** Lans
- **2** Collision Domains
- **Fewer** Ports
- **Replaced** by Switches

# Link Layer: Switches

- Typically connects individual computers
  - A switch is essentially the same as a bridge
  - … though typically used to connect hosts
- Supports concurrent communication
  - Host A can talk to C, while B talks to D

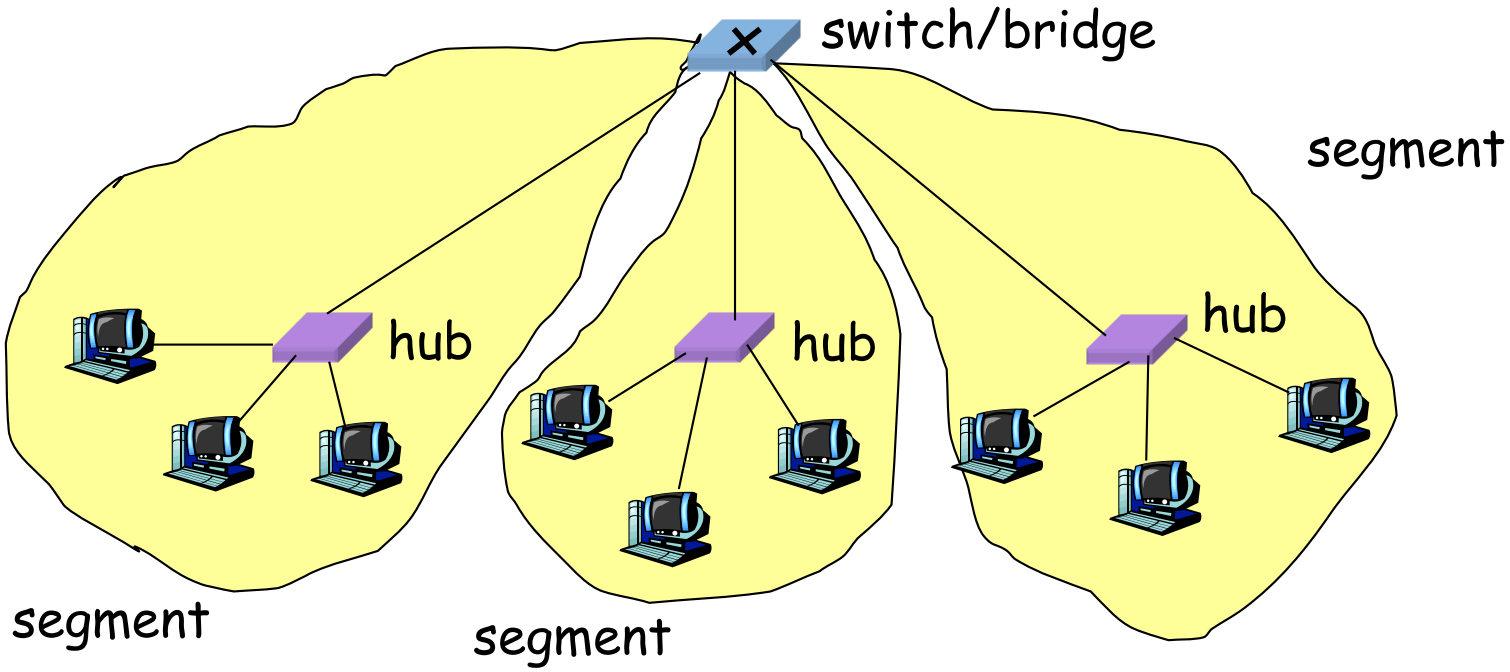


# Switch: Overview

- **Layer 2** Device
- **Full-**Duplex
- **Multiple** Collision Domains
- **Saves Bandwidth**
- **Increased Security**

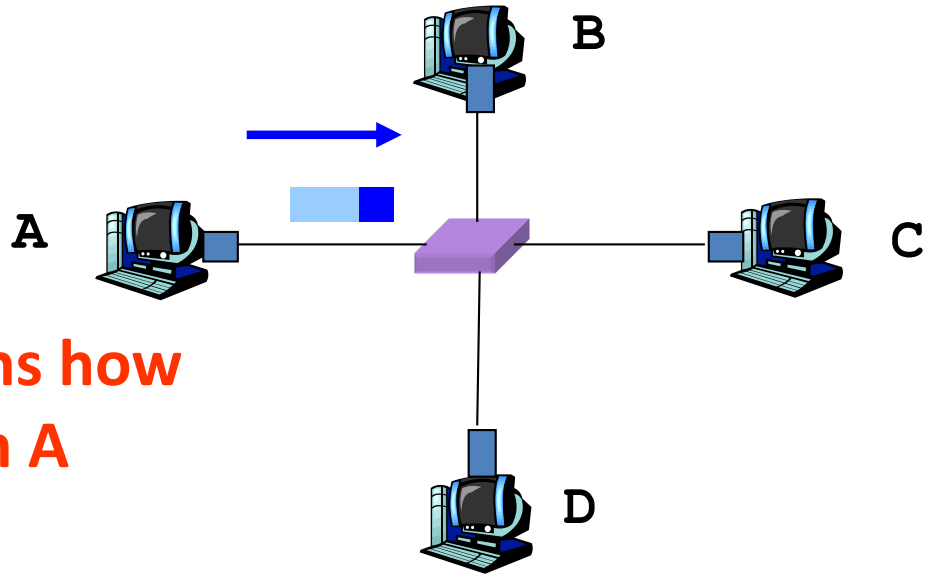
# Bridges/Switches: Traffic Isolation

- Switch filters packets
  - Frame only forwarded to the necessary segments
  - Segments can support separate transmissions



# Self Learning: Building the Table

- When a frame arrives
  - Inspect the *source* MAC address
  - Associate the address with the *incoming* interface
  - Store the mapping in the switch table
  - Use a timer to eventually forget the mapping

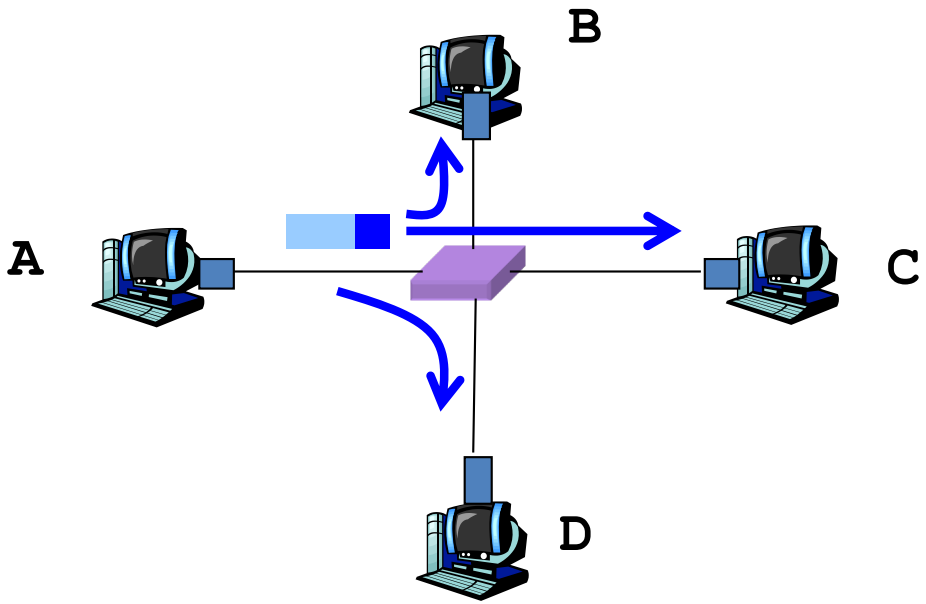



**Switch learns how to reach A**

# Self Learning: Handling Misses

- When frame arrives with unfamiliar destination
  - Forward the frame out all of the interfaces
  - … except for the one where the frame arrived
  - Hopefully, this case won't happen very often!

**When in doubt, shout!**



# Switches vs. Hubs

- Compared to hubs, Ethernet switches support
  - (Y) Larger geographic span
  - (M) Similar span
  - (C) Smaller span
- Compared to hubs, switches provide
  - (Y) Higher load on links
  - (M) Less privacy
  - (C) Traffic isolation

# Routers: Looking closer…

# Router: Overview

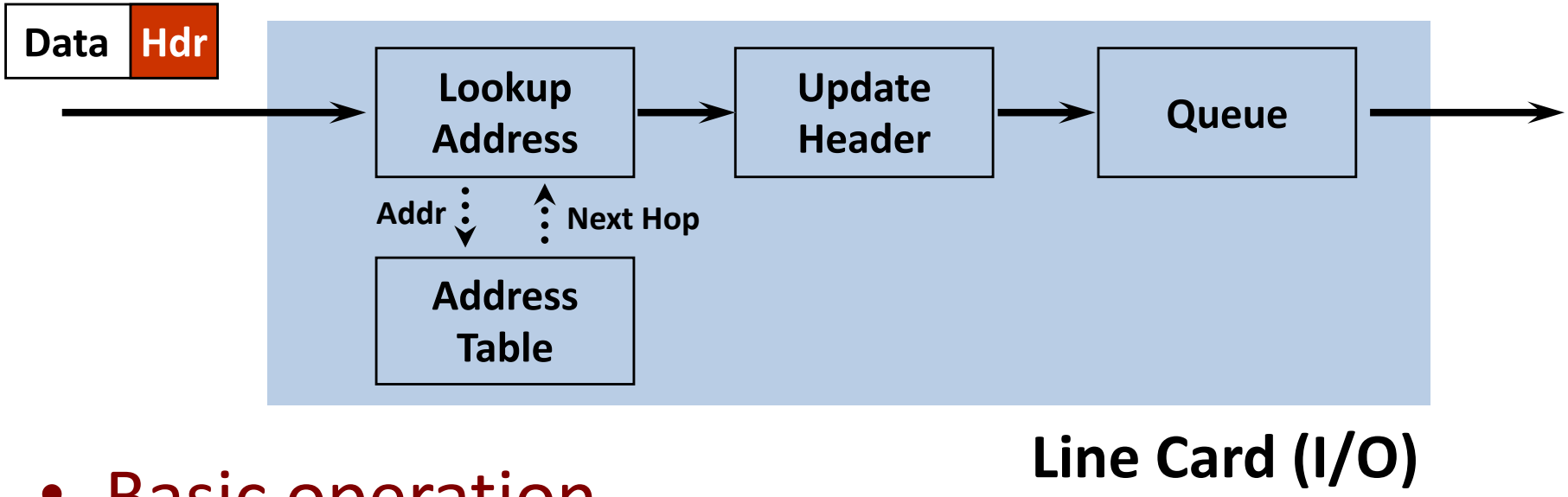
- **Routes Traffic** Between Networks
- **Layer 3** Device
- **Fewer Ports**

# Basic Router Architecture

- Each switch/router has a forwarding table
  - Maps destination address to outgoing interface

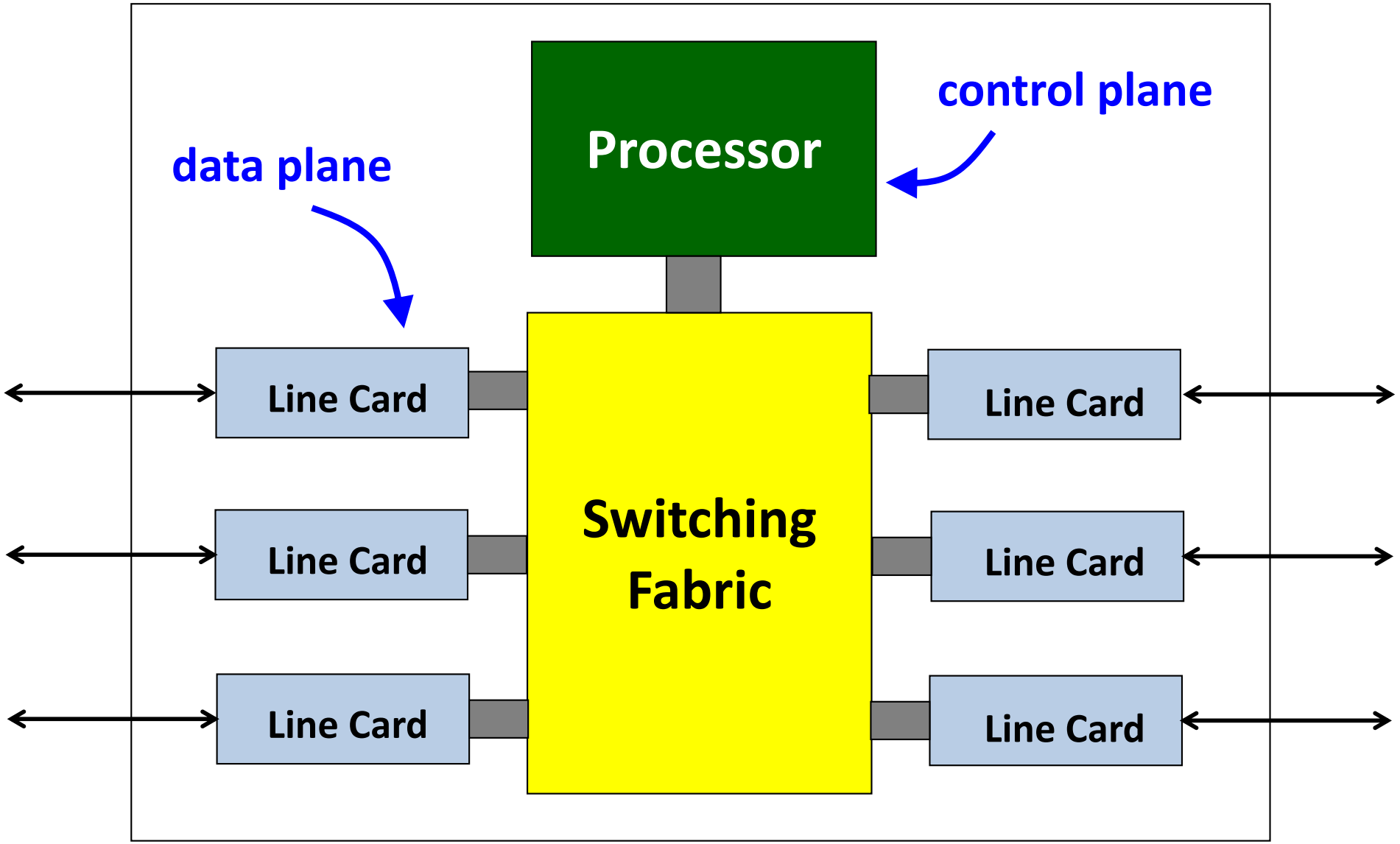
- Basic operation
  1. Receive packet
  2. Look at header to determine destination address
  3. Look in forwarding table to determine output interface
  4. Modify packet header (e.g., decr TTL, update chksum)
  5. Send packet to output interface

# Basic Router Architecture

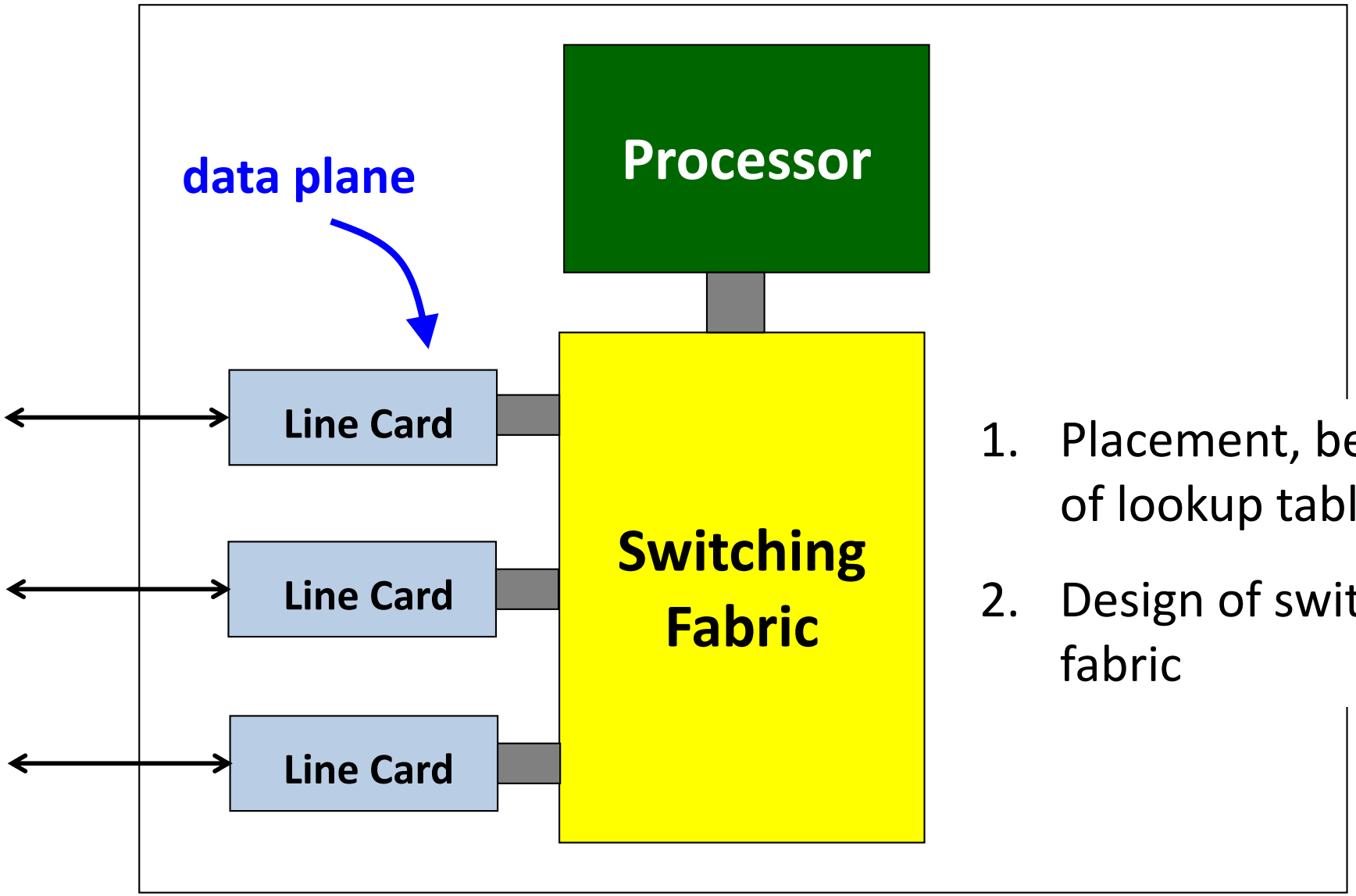



- Basic operation
  1. Receive packet
  2. Look at header to determine destination address
  3. Look in forwarding table to determine output interface
  4. Modify packet header (e.g., decr TTL, update chksum)
  5. Send packet to output interface

# Router


Processor
control plane
data plane
Line Card
Line Card
Line Card
Switching Fabric
Line Card
Line Card
Line Card

# Router




1. Placement, behavior of lookup tables
2. Design of switching fabric

# Lookup algorithm depends on protocol

| Protocol | Mechanism | Techniques |
|---|---|---|
| Ethernet (48 bits)<br>MPLS<br>ATM | Exact Match | • Direct lookup<br>• Associative lookup<br>• Hashing<br>• Binary tree |
| IPv4 (32 bits)<br>IPv6 (128 bits) | Longest-Prefix Match | • Radix trie<br>• Compressed trie<br>• TCAM |

# Longest Prefix Match (LPM)

- Each packet has destination IP address
- Router looks up table entry that matches address

**68.211.6.120**

| Prefix | Output |
|---|---|
| 68.208.0.0/12 | 1 |
| 68.211.0.0/17 | 1 |
| 68.211.128.0/19 | 2 |
| 68.211.160.0/19 | 2 |
| 68.211.192.0/18 | 1 |

# LPM: Motivation

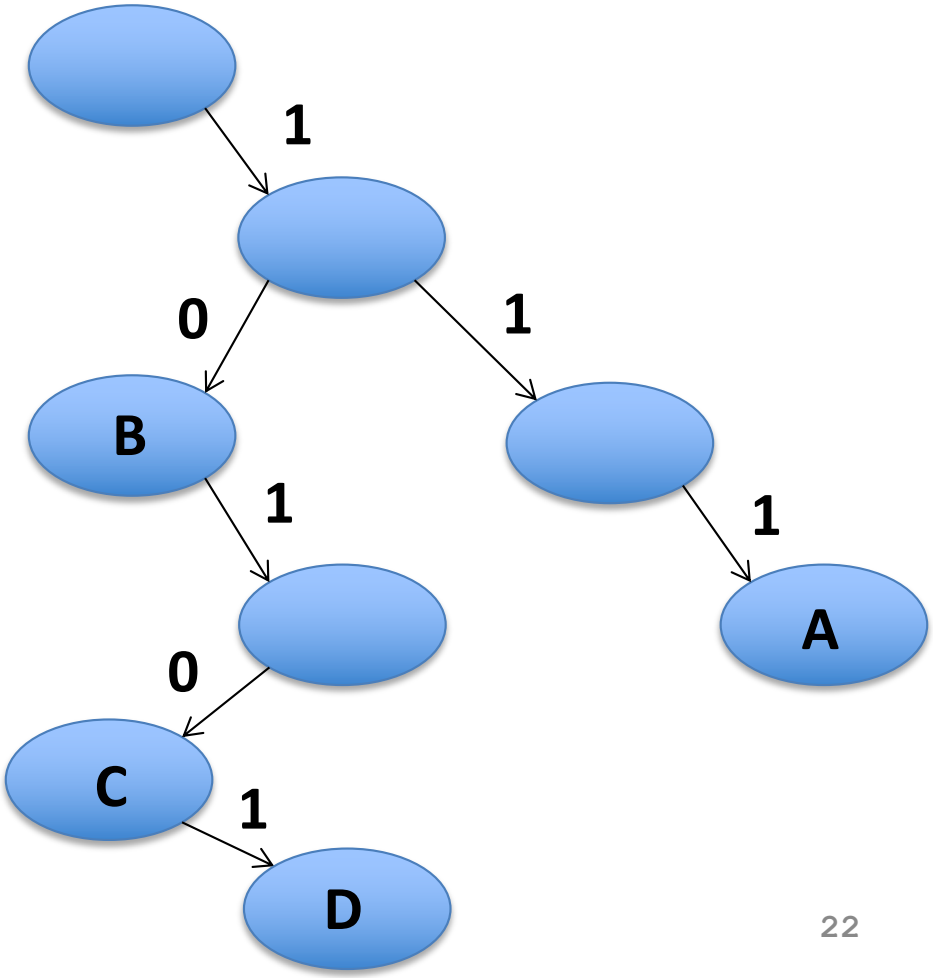
- Each packet has destination IP address
- Router looks up table entry that matches address

- Benefits of CIDR allocation and LPM
  - **Efficiency:** Prefixes can be allocated at much finer granularity
  - **Hierarchical aggregation:** Upstream ISP can aggregate 2 contiguous prefixes from downstream ISPs to shorter prefix

# Software LPM lookup using trie

- Prefixes “spelled out” by following path from root
- To find the best prefix spell out address in trie

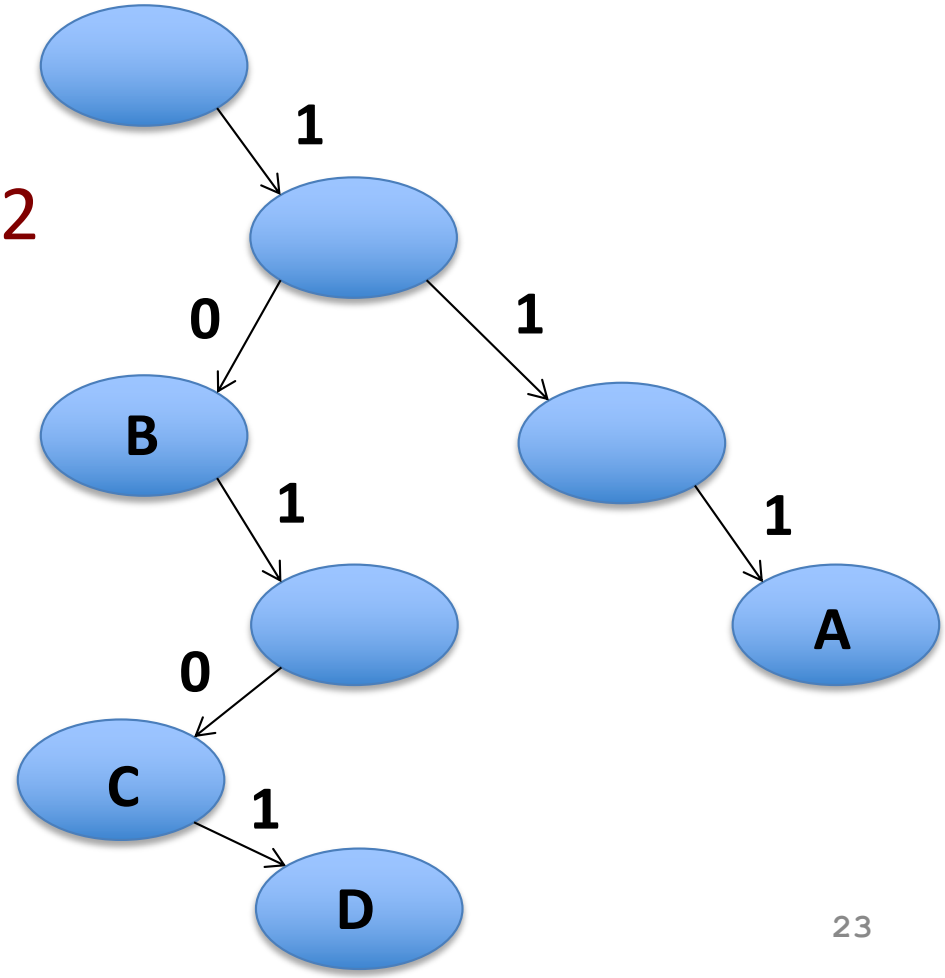
| | Prefixes |
|---|---|
| A | 111* |
| B | 10* |
| C | 1010* |
| D | 10101 |



# Software LPM lookup using trie

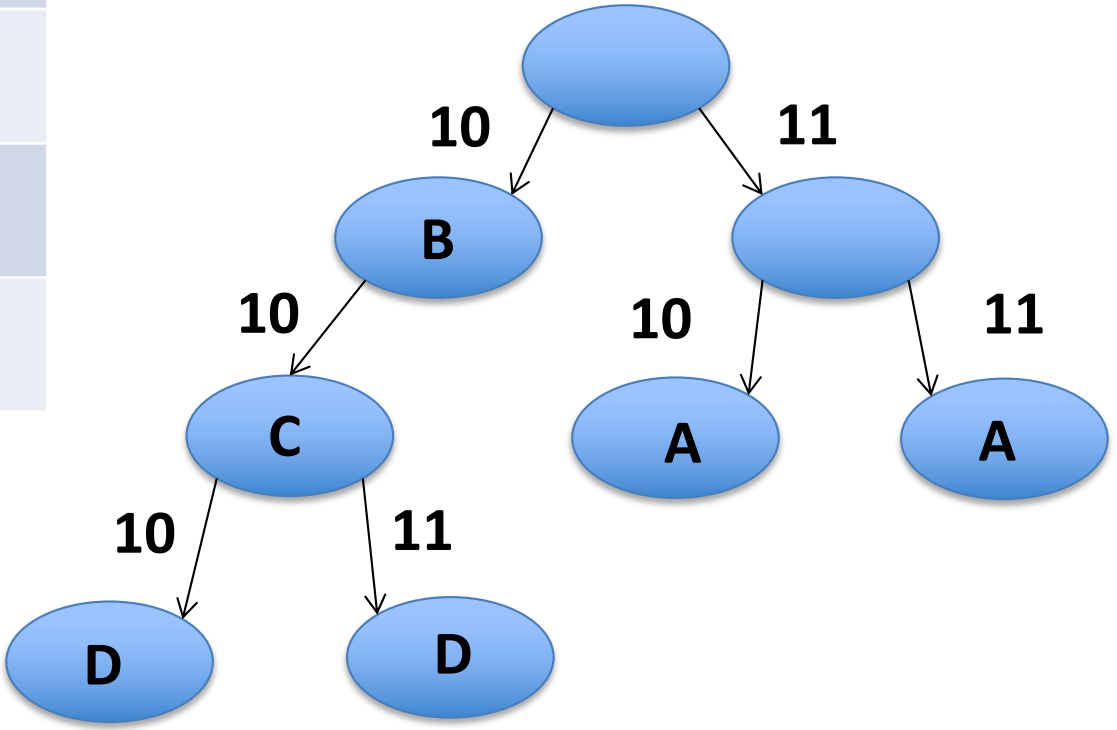
- Prefixes "spelled out" by following path from root
- To find the best prefix spell out address in trie

- 1 lookup per level → max 32 lookups/address!
- Too slow:
  - E.g., "Optical Carrier 48" line (2.5 Gbps) requires 160ns lookup … or 4 memory accesses

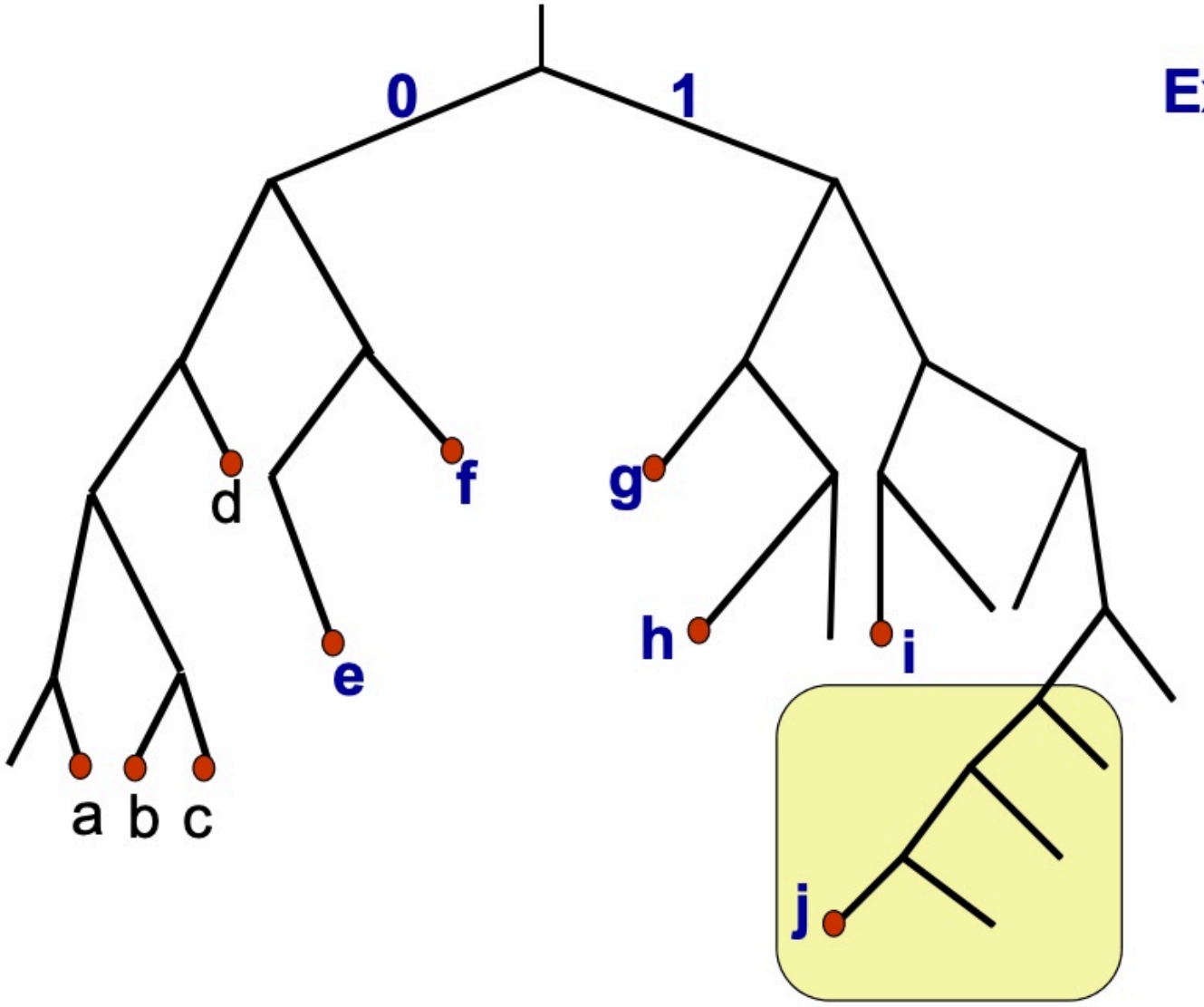


# Software LPM lookup: k-ary trie (k=2)

| | Prefixes |
|---|---|
| A | 111* |
| B | 10* |
| C | 1010* |
| D | 10101 |



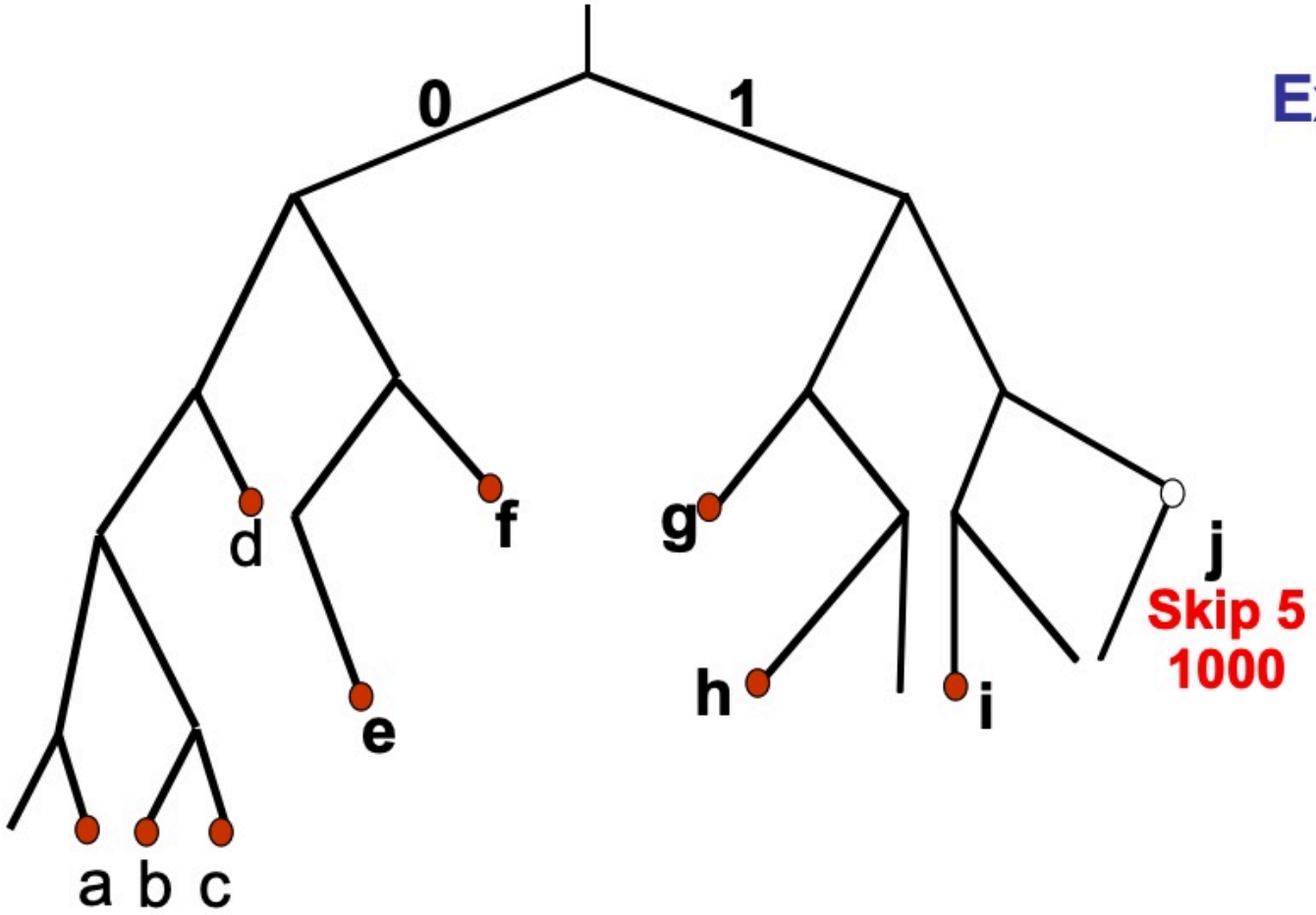
# IP Address Lookup: Binary Tries




**Example Prefixes:**

a) 00001
b) 00010
c) 00011
d) 001
e) 0101
f) 011
g) 100
h) 1010
i) 1100
j) 11110000

# IP Address Lookup: Patricia Trie



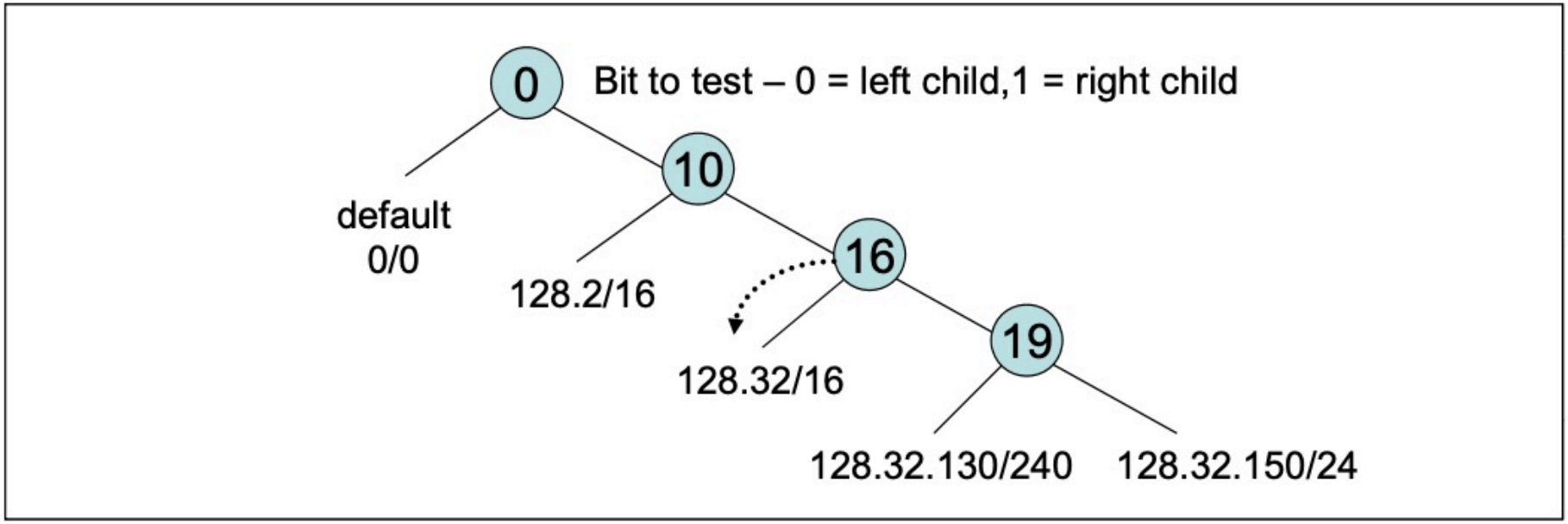

**Example Prefixes**

a) 00001
b) 00010
c) 00011
d) 001
e) 0101
f) 011
g) 100
h) 1010
i) 1100
j) 11110000

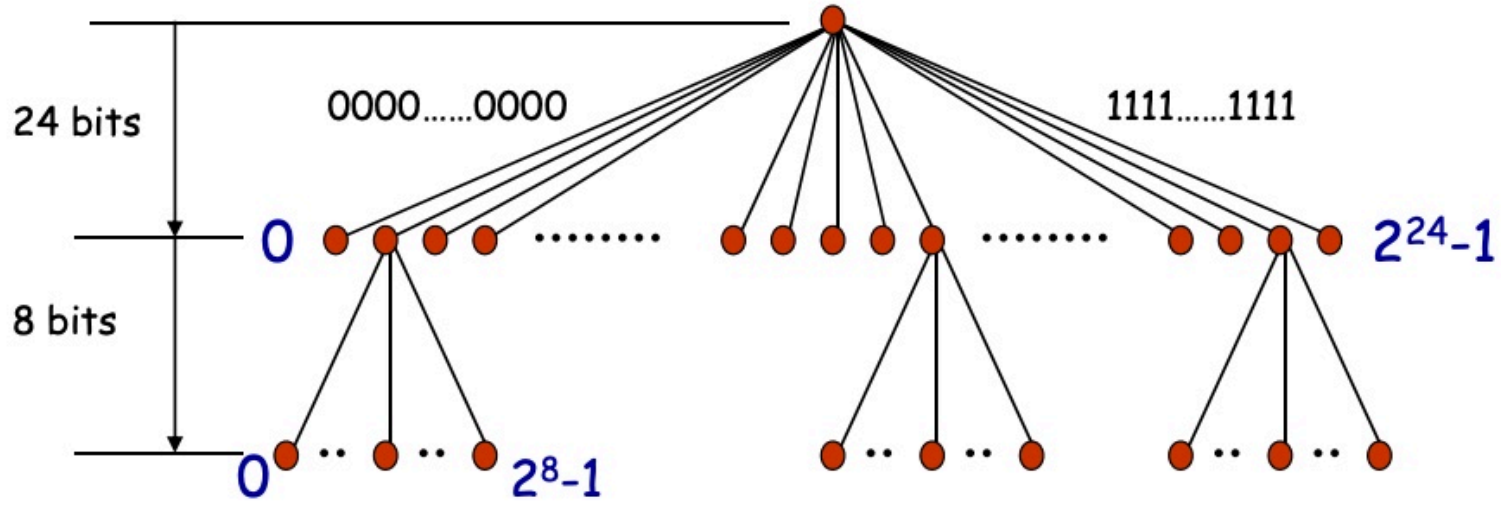
**Problem:** Lots of (slow) memory lookups

# LPM with PATRICIA Tries

- Traditional method – Patricia Tree
  - Arrange route entries into a series of bit tests
- Worst case = 32 bit tests
  - Problem: memory speed, even w/SRAM!



# Address Lookup: Direct Trie

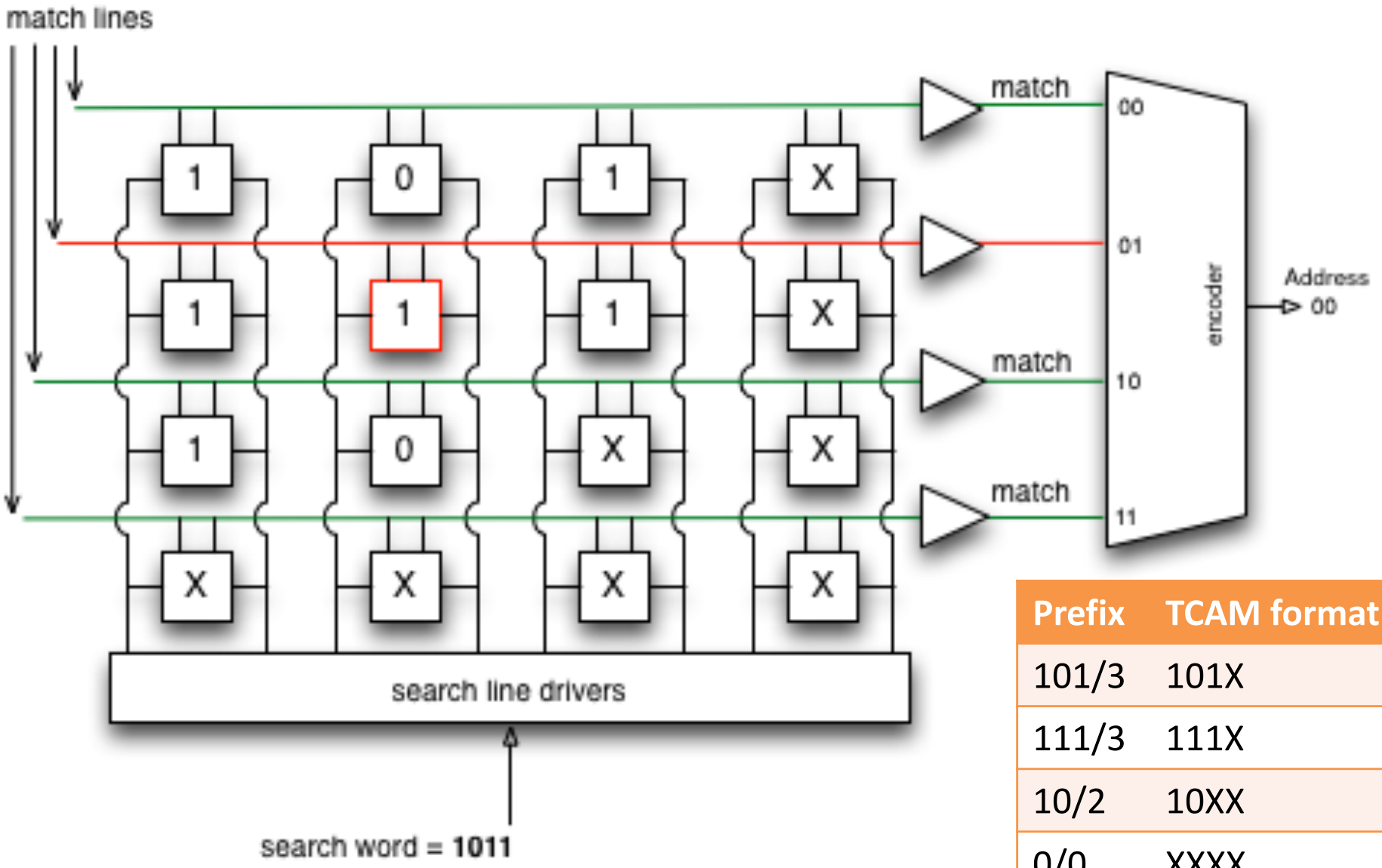



- When pipelined, one lookup per memory access
- **Inefficient use of memory**

# Hardware for LPM lookup

- Content-Address Memory (CAM)
  - Input:  tag (address)
  - Output: value (port
  - Exact match, but O(1) in hardware
- Ternary CAM
  - Can have wildcards: 0, 1, *
  - “value” memory cell and “mask” (care / don’t care) cell
- LPM via TCAM
  - In parallel, search all prefixes for all matches
  - Then choose longest match
    - Trick: choose first match, but already sorted by prefix length

# Example: LPM with a TCAM

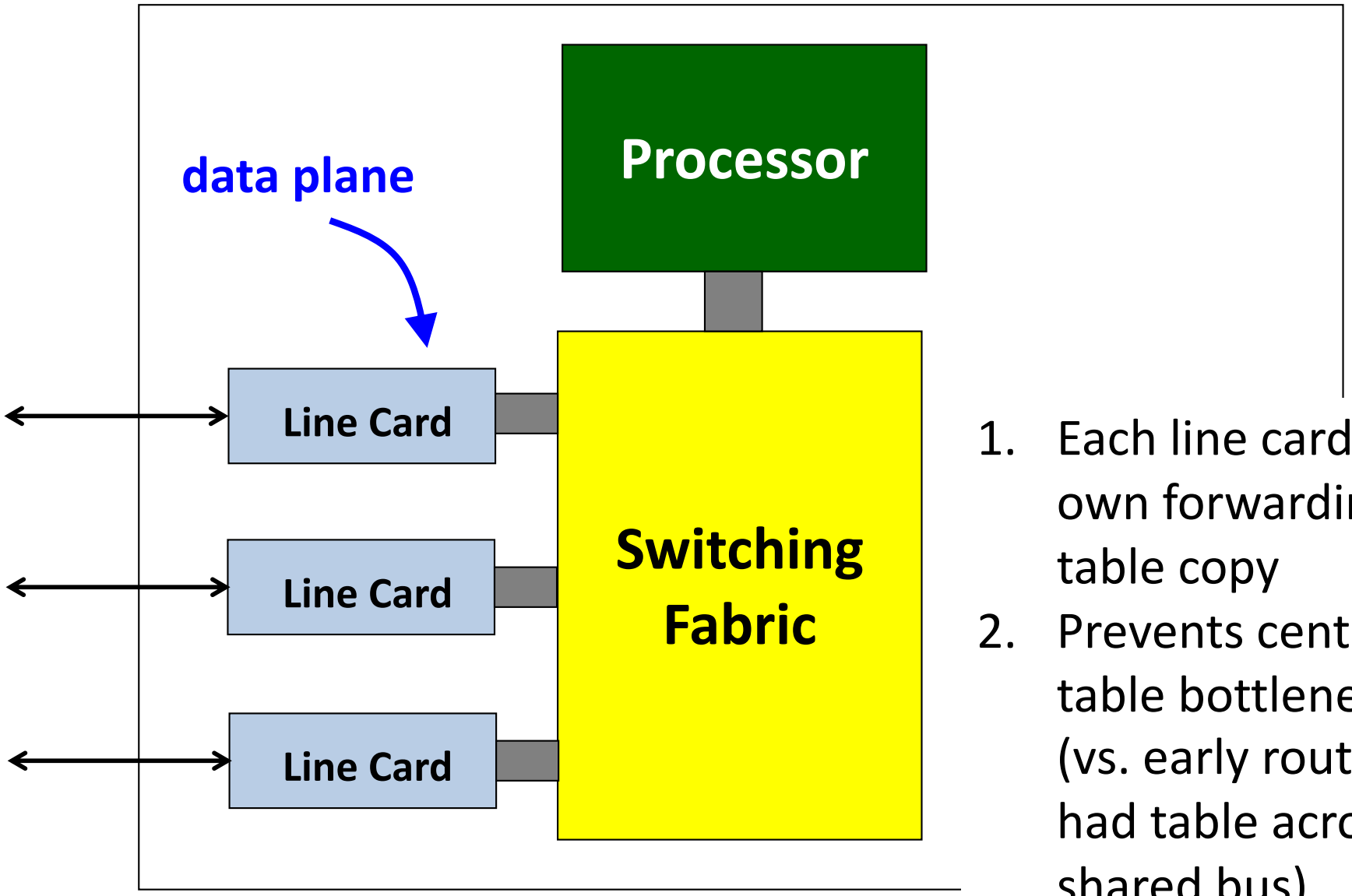



| Prefix | TCAM format |
|---|---|
| 101/3 | 101X |
| 111/3 | 111X |
| 10/2 | 10XX |
| 0/0 | XXXX |



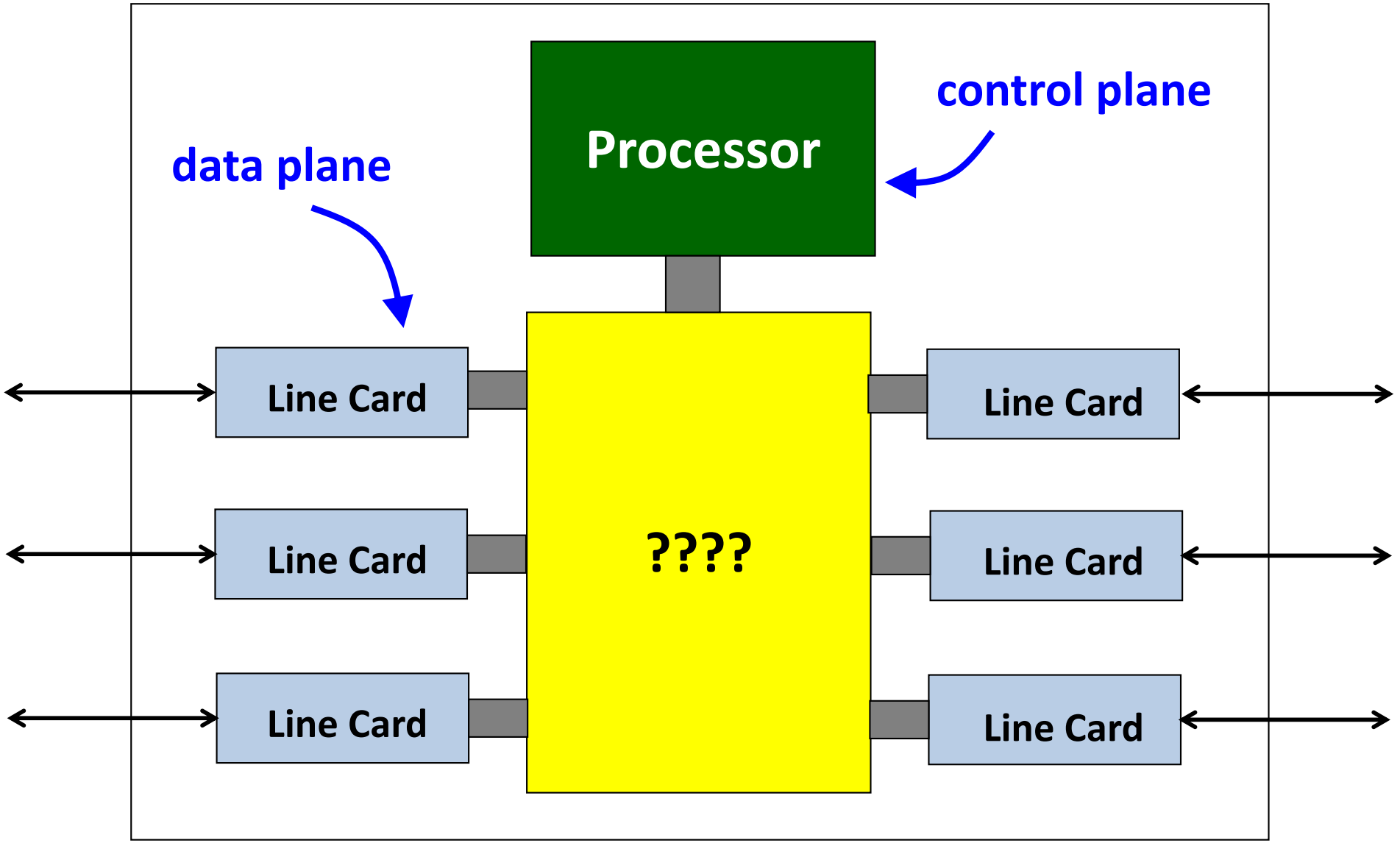
[thenetworksherpa.com/tcam-in-the-forwarding-engine]

# Decision: Forwarding table per line card




1. Each line card has own forwarding table copy
2. Prevents central table bottleneck (vs. early routers had table across shared bus)

# Decision: Crossbar switch

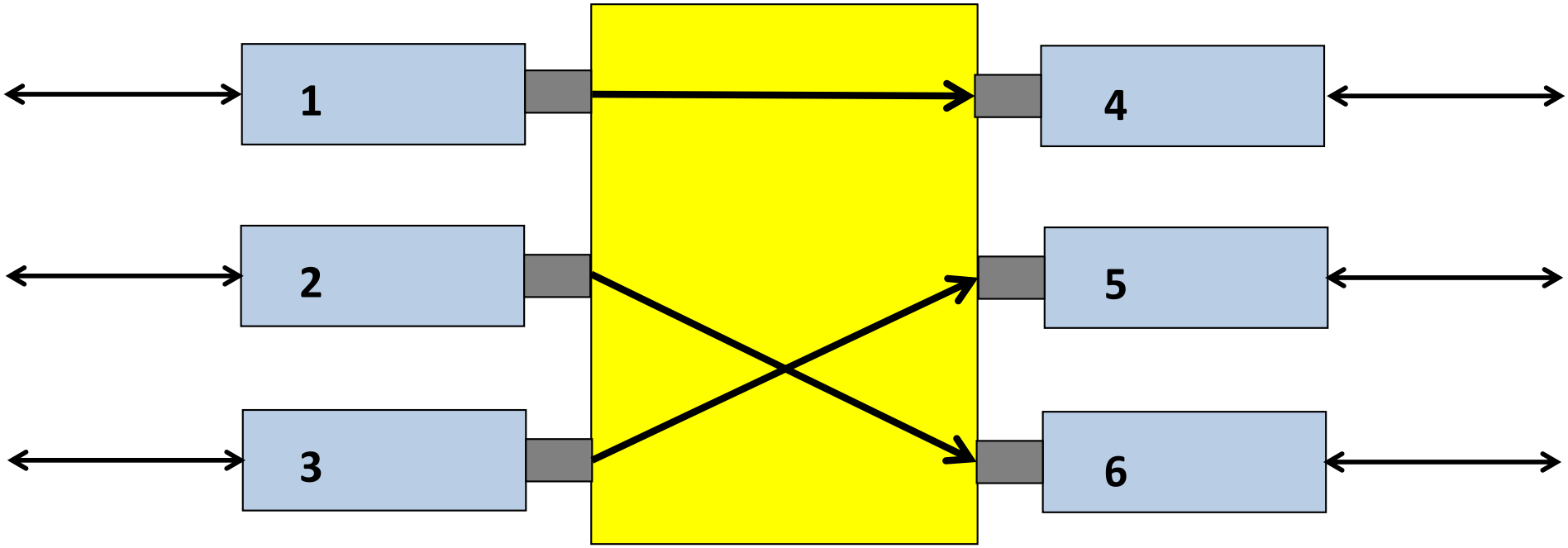


# Decision: Crossbar switch

- Shared bus
  - Only one input can speak to one output at a time
- Crossbar switch / switched backplane
  - Input / output pairs that don't compete can send in same timeslot

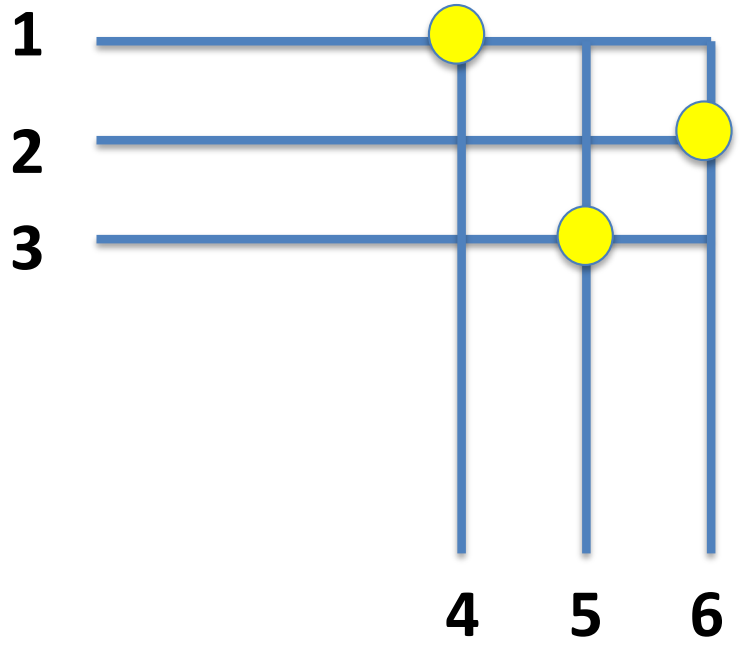
# Crossbar switching

- Every input port has connection to every output port
- In each timeslot, each input connected to zero or more outputs



# Crossbar switching

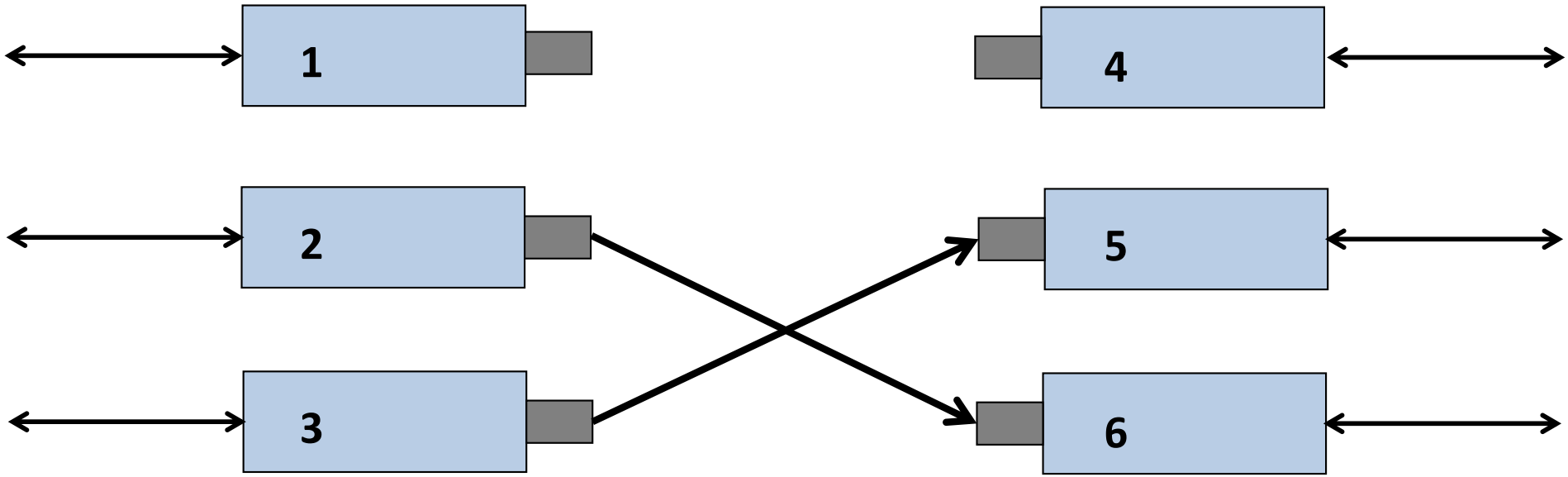
- Every input port has connection to every output port
- In each timeslot, each input connected to zero or more outputs




- Good parallelism
- Needs scheduling

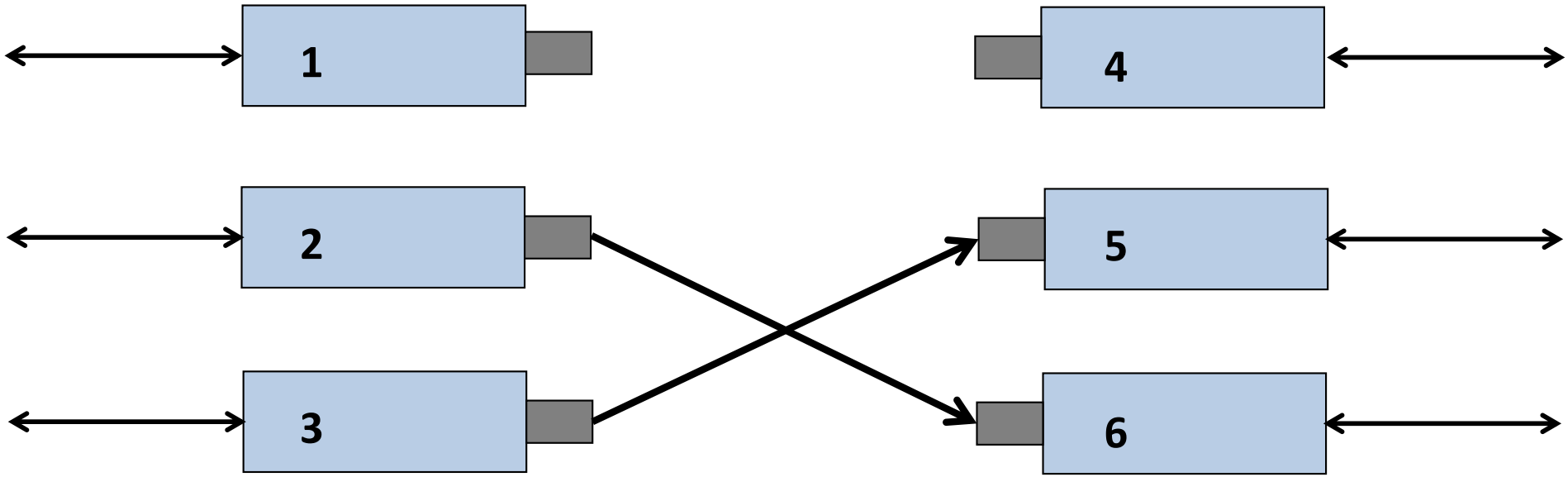
# Everything gets complicated…

- **Problem: Head-of-line blocking**
  - The packet in front of queue blocks packets behind it from being processed
  - Say first packet at input 1 wants to go to output 5; second packet at 1 that wants 4 is still blocked



# Everything gets complicated…

- **Solution: *Virtual output queues***
  - One queue at input, **per output port** (for all inputs)
  - So **avoids head-of-line blocking** during crossbar scheduling

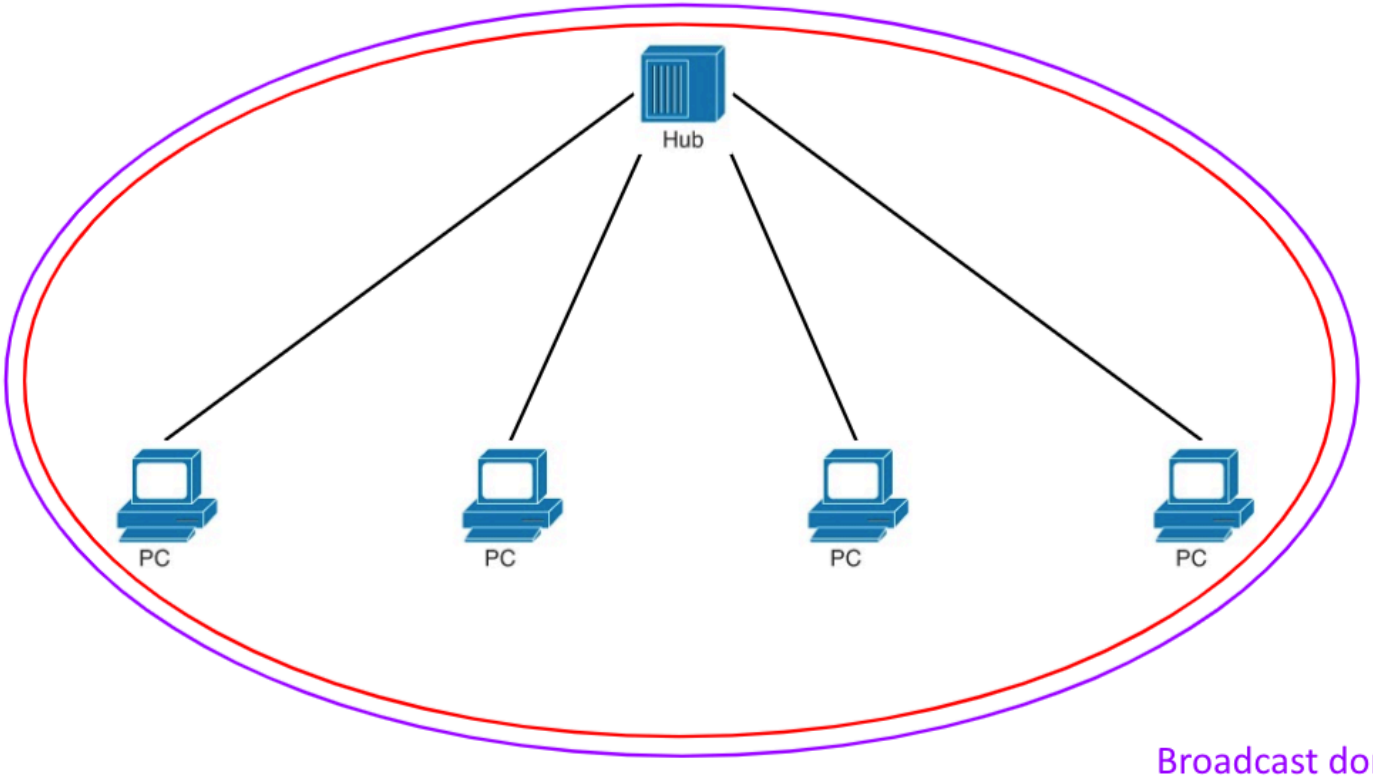


# Cisco 8000 Series Routers

- Up to 648 400 GbE
- 260 Tbps backplane

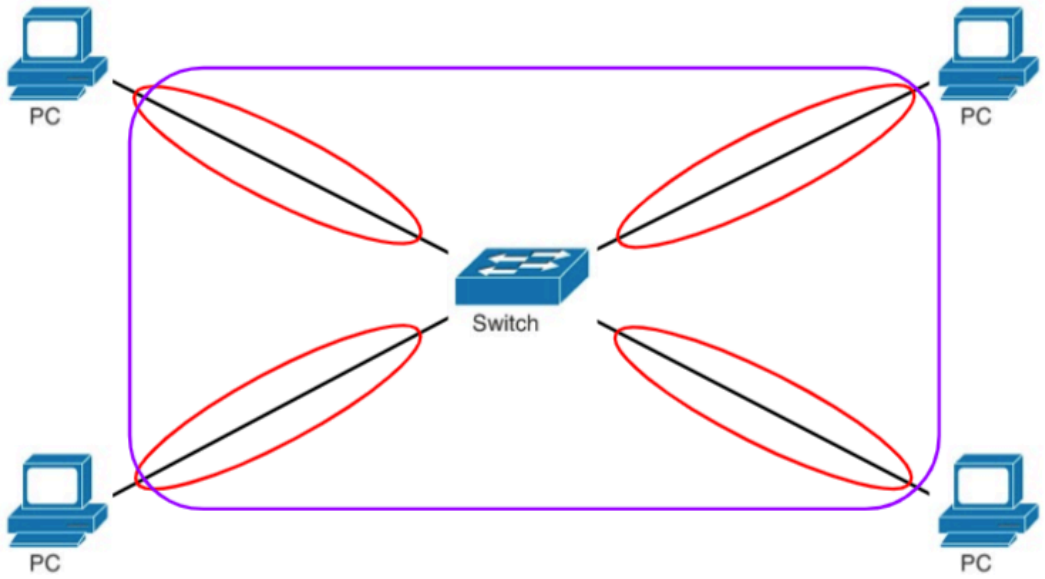
# Collision and broadcast domains

- Hub



# Collision and broadcast domains

- Switch

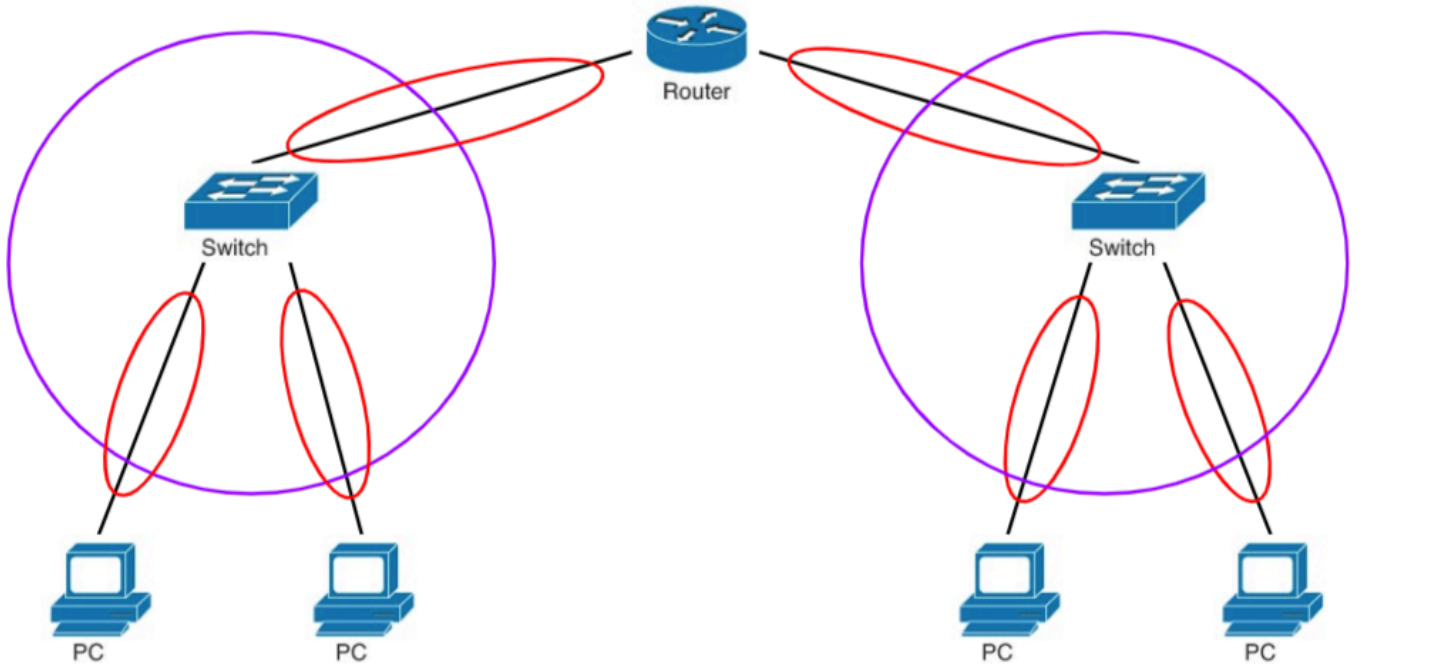



Broadcast domain

Collision domain

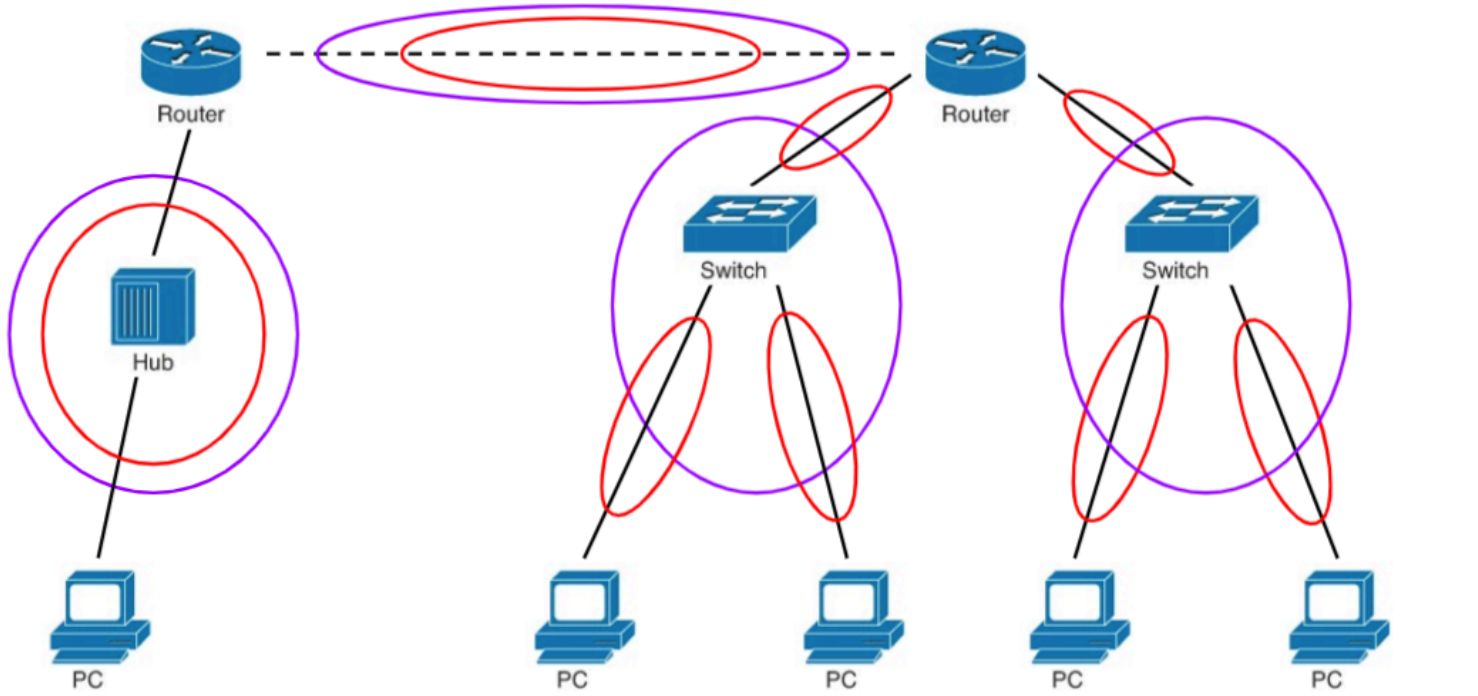
# Collision and broadcast domains

- Switch and router



# Collision and broadcast domains

- Hub, switch, and router



# Conclusions

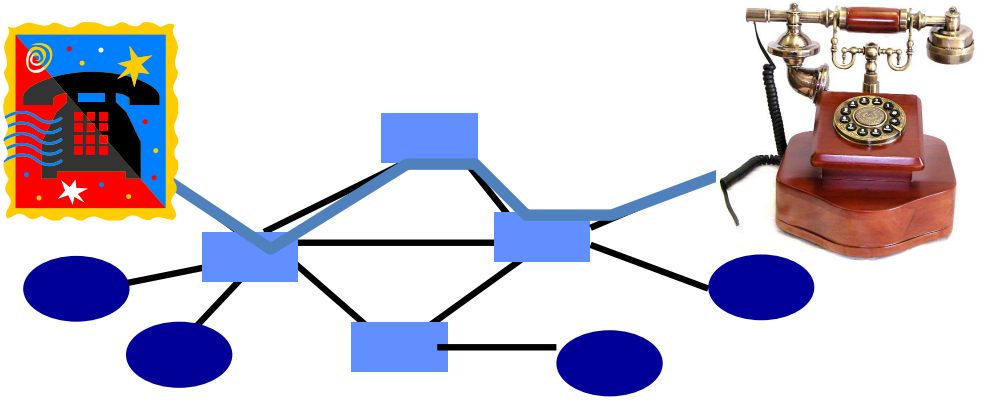
- Physical devices sharing L2 & L3 networks have many common features
  - Forward table lookups
  - Queueing and backplane switching
  - Fast vs. slow paths
    - Switches and routers separate routing decisions (control plane) from forwarding actions (data plane)
- High speed necessitates innovation
  - Specialized hardware
  - Software algorithms

# IP Protocol Stack: Key Abstractions

# Best-Effort Global Packet Delivery

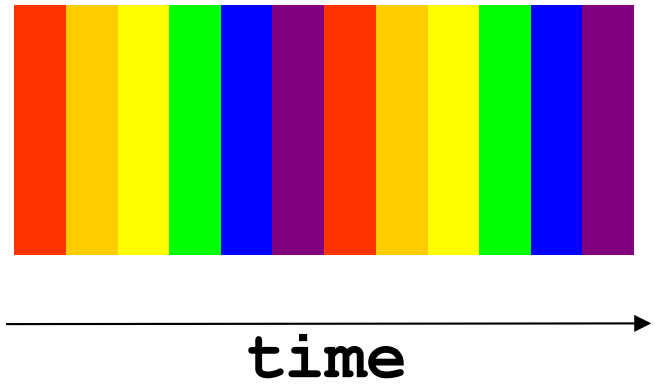
# Circuit Switching (e.g., Phone Network)

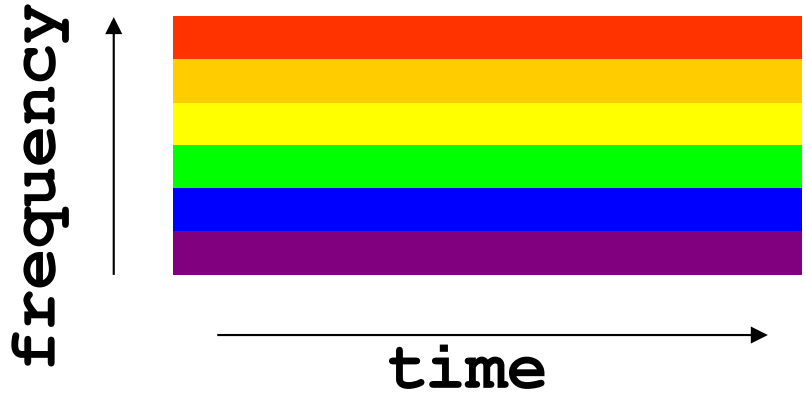
- Source establishes connection
  - Reserve resources along hops in the path
- Source sends data
  - Transmit data over the established connection
- Source tears down connection
  - Free the resources for future connections



# Circuit Switching: Static Allocation

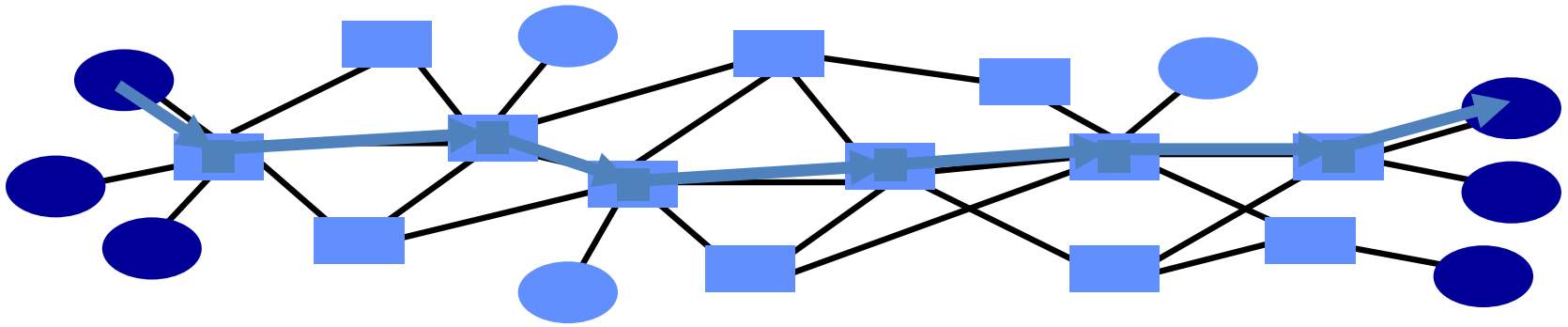
- Time-division
  - Each circuit allocated certain time slots
- Frequency-division
  - Each circuit allocated certain frequencies



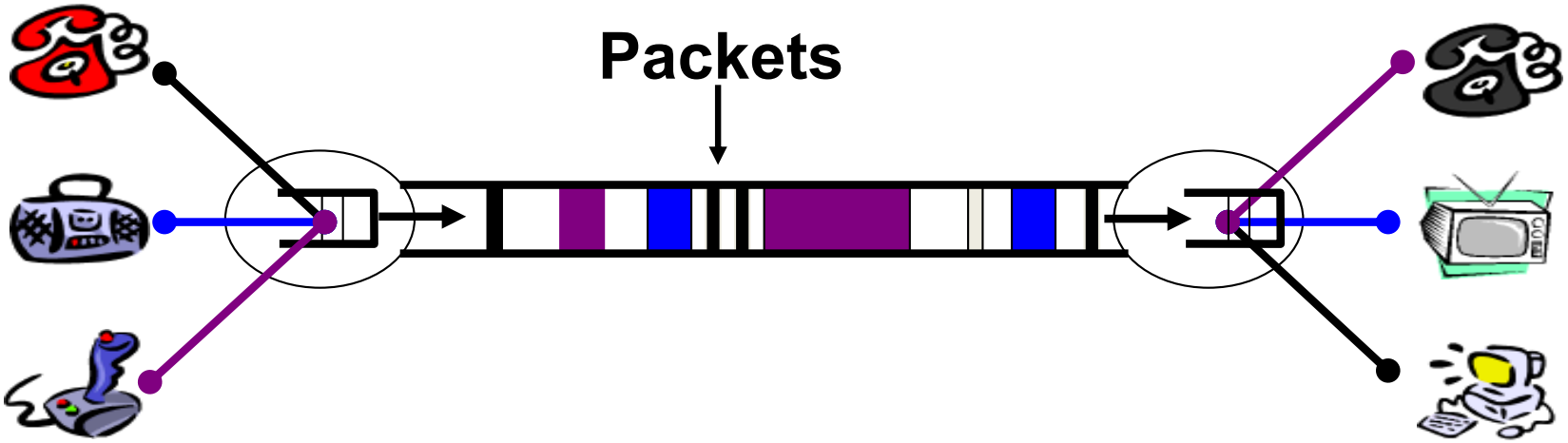



# Packet Switching

- Message divided into packets
  - Header identifies the destination address
- Packets travel separately through the network
  - Forwarding based on the destination address
  - Packets may be buffered temporarily
- Destination reconstructs the message



# Packet Switching: Statistical (Time Division) Multiplexing




- Intuition: Traffic by computer end-points is bursty!
  - Versus: Telephone traffic not bursty (e.g., constant 56 kbps)
  - One can use network while others idle
- Packet queuing in network: tradeoff space for time
  - Handle short periods when outgoing link demand > link speed

# Is Best Effort Good Enough?

- Packet loss and delay
  - Sender can resend
- Packet corruption
  - Receiver can detect, and sender can resend
- Out-of-order delivery
  - Receiver can put the data back in order
- Packets follow different paths
  - Doesn't matter
- Network failure
  - Drop the packet
- Network congestion
  - Drop the packet

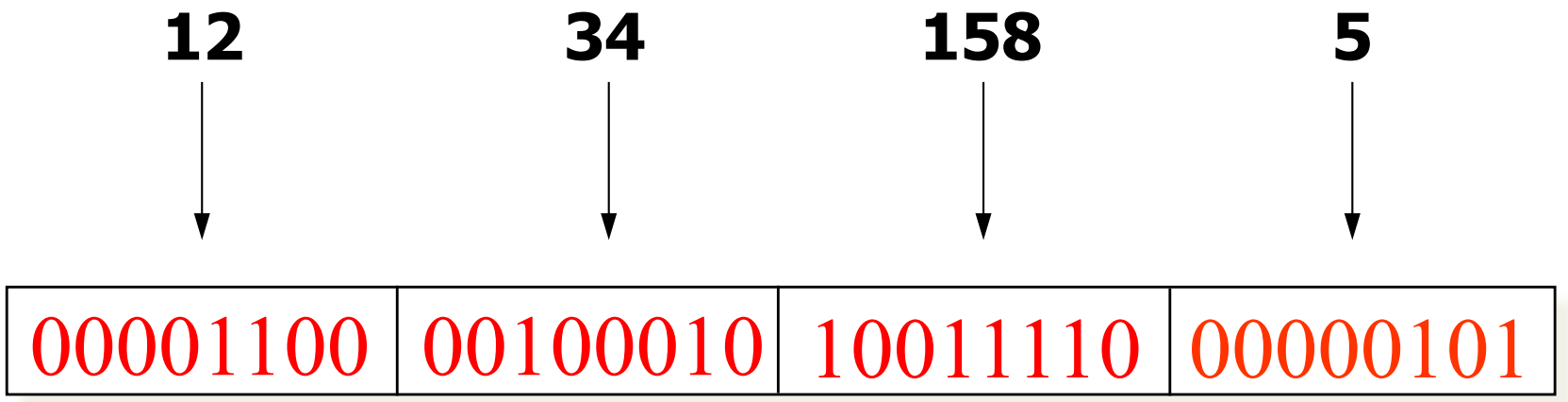
# Packet (Y) vs. Circuit Switching (A)?

| | |
|---|---|
| • Predictable performance | Circuit |
| • Network never blocks senders | Packet |
| • Reliable, in-order delivery | Circuit |
| • Low delay to send data | Packet |
| • Simple forwarding | Circuit |
| • No overhead for packet headers | Circuit |
| • High utilization under most workloads | Packet |
| • No per-connection network state | Packet |

# Network Addresses

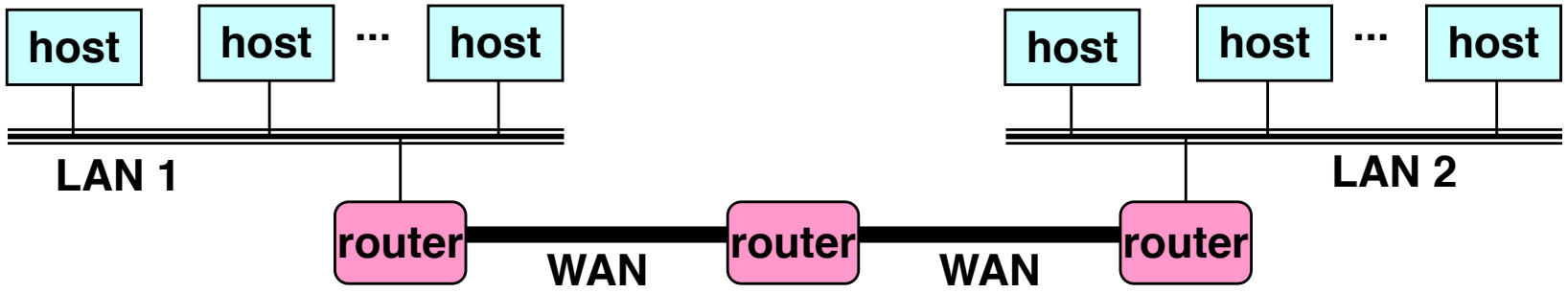
# IP Address (IPv4)

- A unique 32-bit number
- Identifies an interface (on a host, on a router, …)
- Represented in dotted-quad notation



# Grouping Related Hosts

- The Internet is an "inter-network"
  - Used to connect networks together, not hosts
  - Need to address a network (i.e., group of hosts)

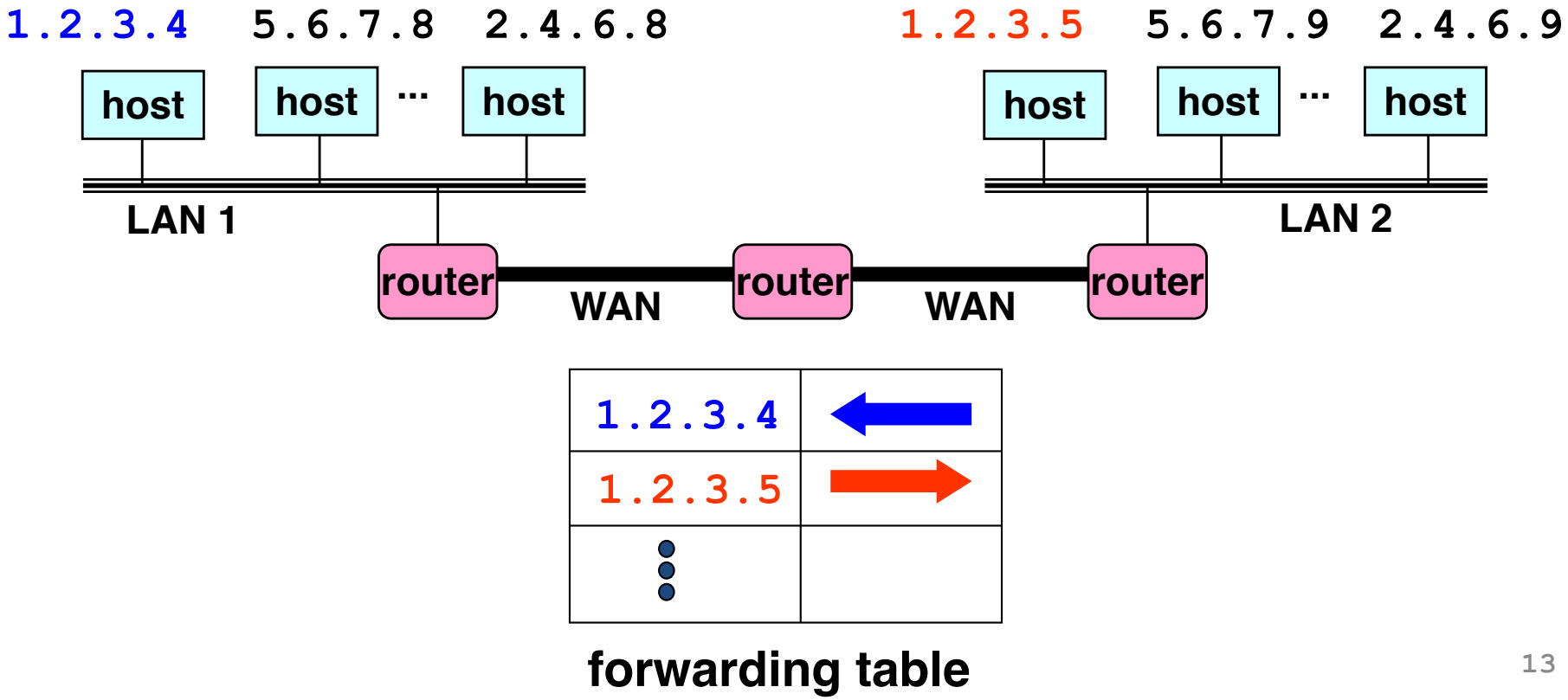



LAN = Local Area Network

WAN = Wide Area Network

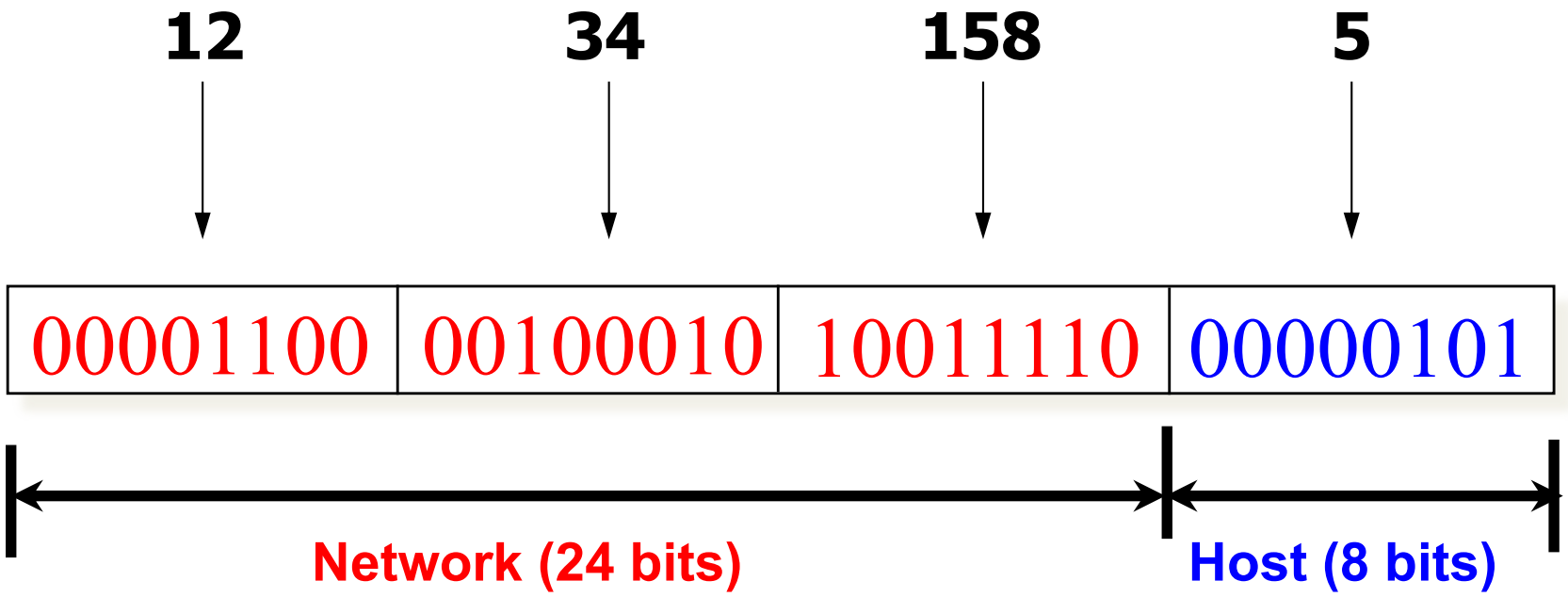
# Scalability Challenge

- Suppose hosts had arbitrary addresses
  - Then every router would need a lot of information
  - …to know how to direct packets toward every host



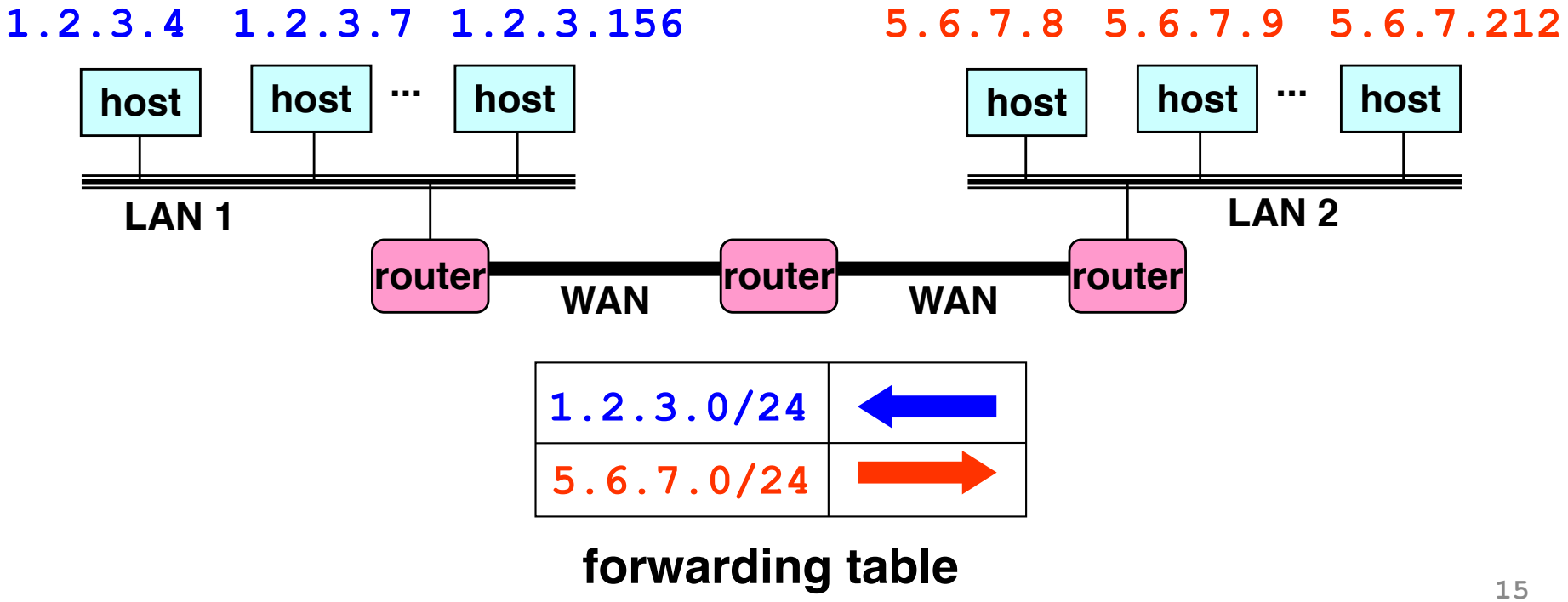
# Hierarchical Addressing: IP Prefixes

- Network and host portions (left and right)
- 12.34.158.0/24 is a 24-bit **prefix** with $2^8$ addresses



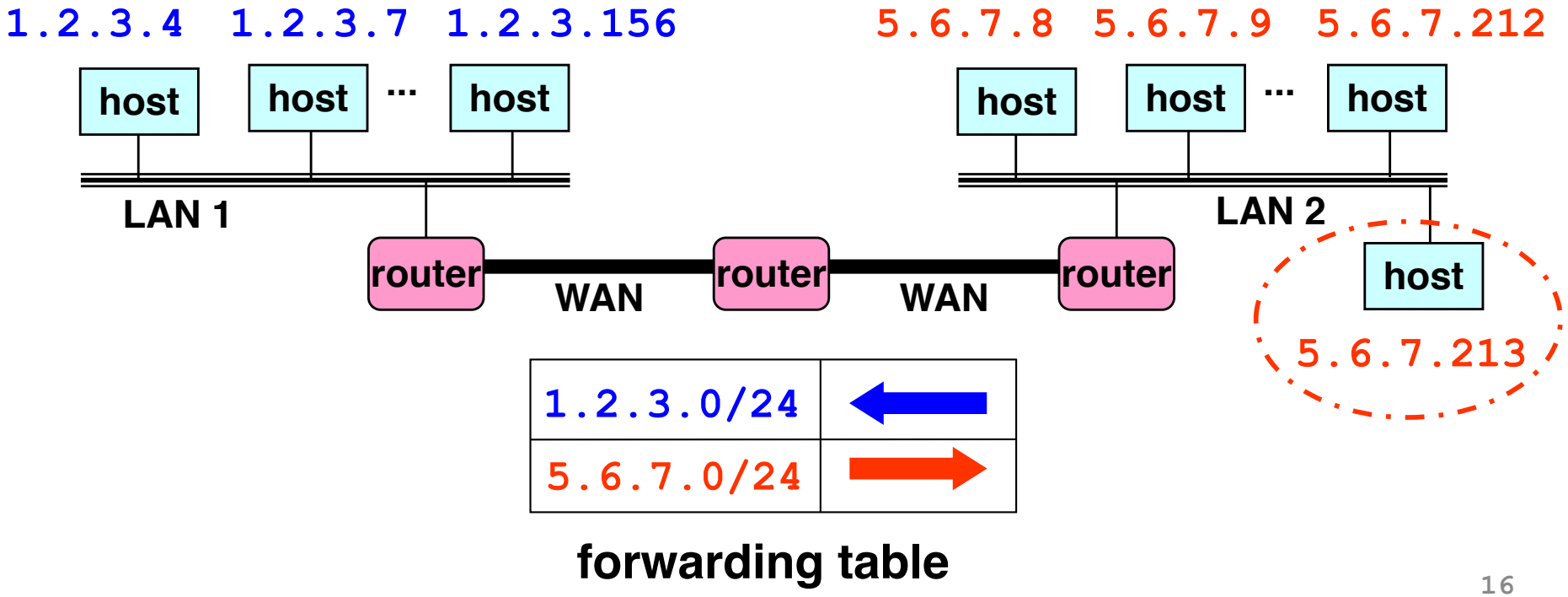
# Scalability Improved

- Number related hosts from a common subnet
  - 1.2.3.0/24 on the left LAN
  - 5.6.7.0/24 on the right LAN



# Easy to Add New Hosts

- No need to update the routers
  - E.g., adding a new host 5.6.7.213 on the right
  - Doesn't require adding a new forwarding-table entry



# History of IP Address Allocation

# Classful Addressing

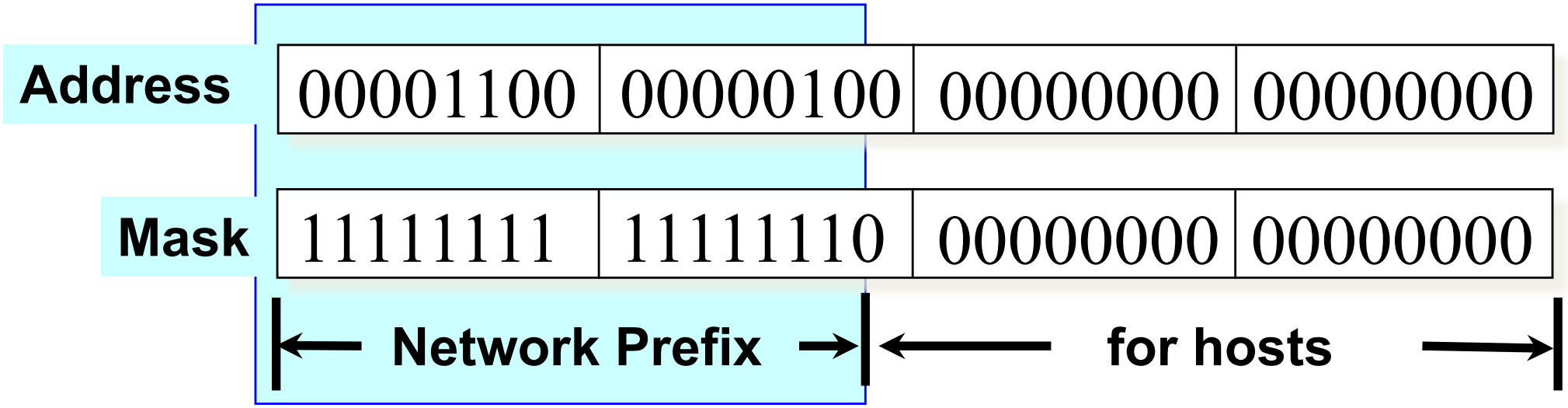
- In the olden days, only fixed allocation sizes
  - Class A: 0*
    - Very large /8 blocks (e.g., MIT has 18.0.0.0/8)
  - Class B: 10*
    - Large /16 blocks (e.g,. Princeton has 128.112.0.0/16)
  - Class C: 110*
    - Small /24 blocks (e.g., AT&T Labs has 192.20.225.0/24)
  - Class D: 1110* for multicast groups
  - Class E: 11110* reserved for future use
- This is why folks use dotted-quad notation!

# Classless Inter-Domain Routing (CIDR)

- Use two 32-bit numbers to represent network:

  Network number = IP address + Mask

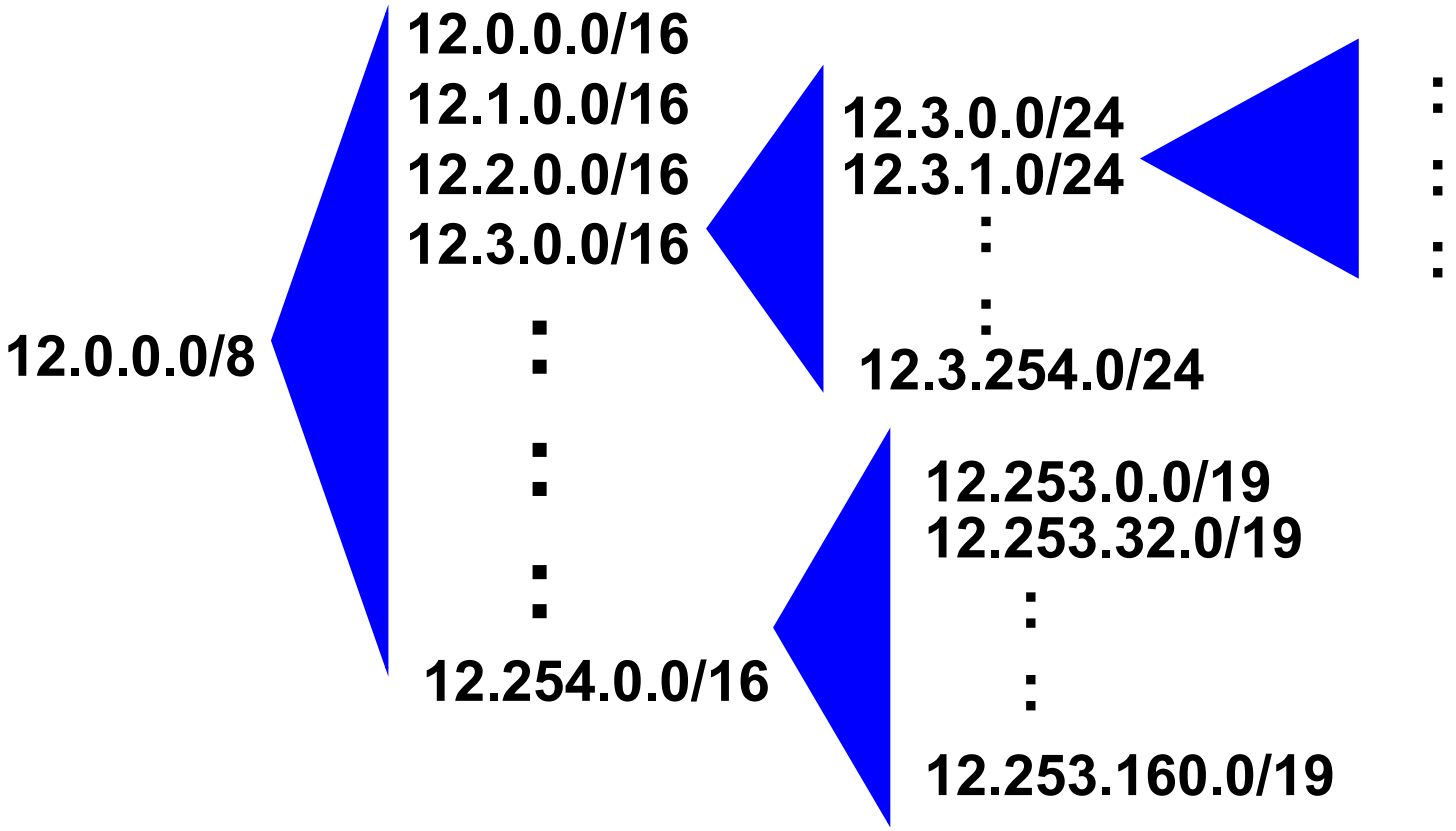
**IP Address : 12.4.0.0  IP Mask: 255.254.0.0**




**Written as 12.4.0.0/15**

# Hierarchical Address Allocation

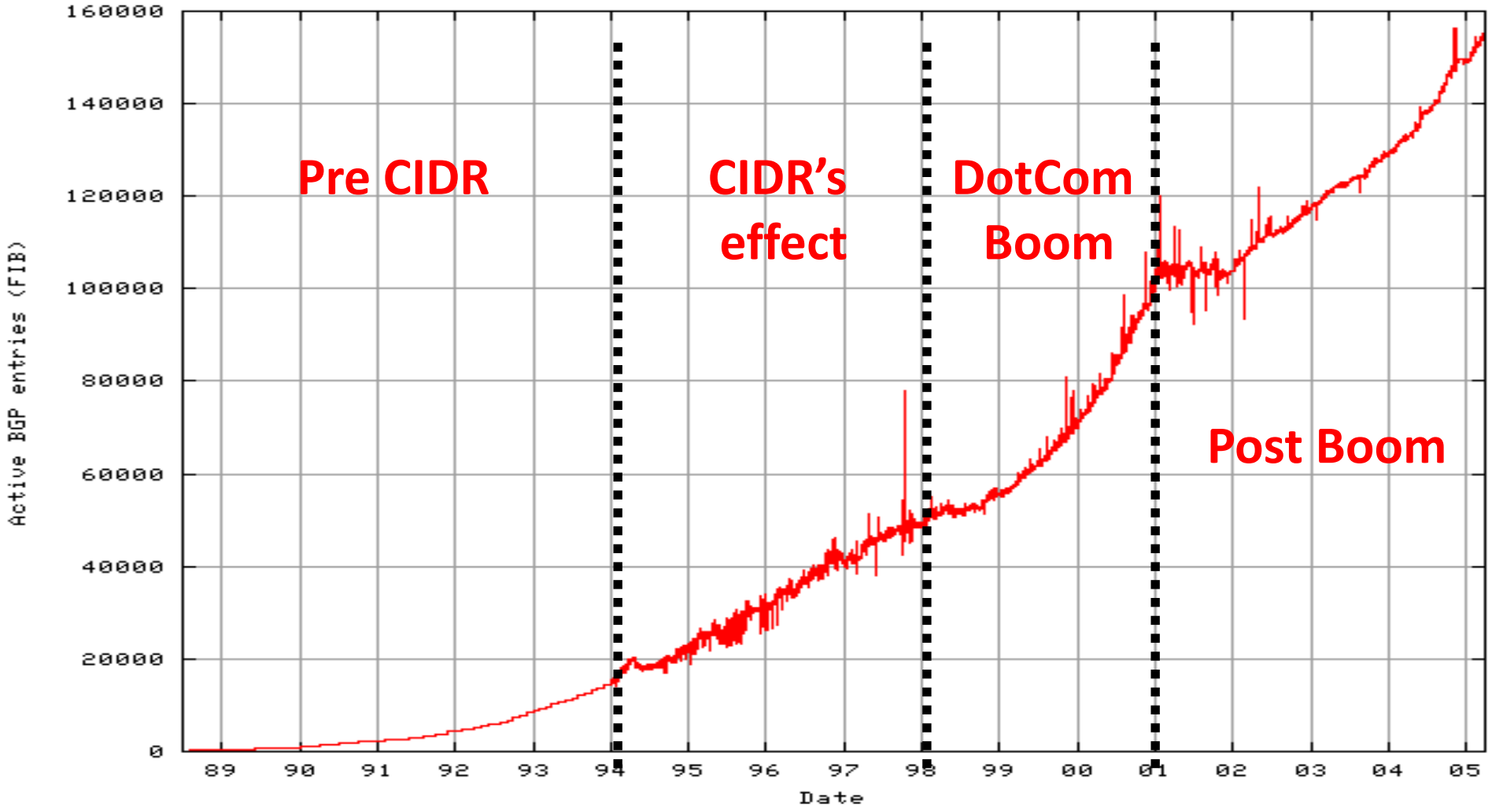
- Hierarchy is key to scalability
  - Address allocated in contiguous chunks (prefixes)
  - Today, the Internet has about 600-800,000 prefixes



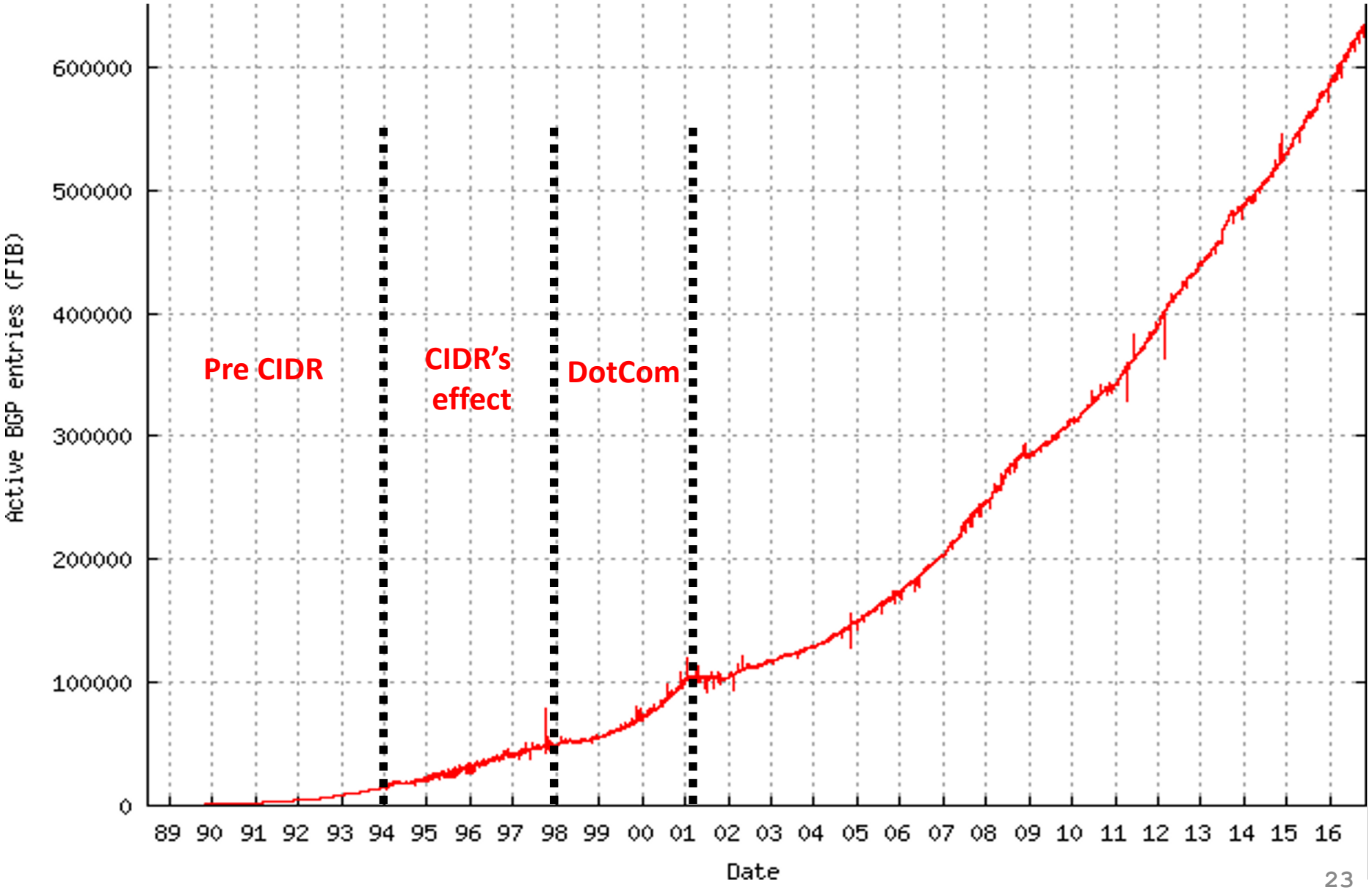
# Obtaining a Block of Addresses

- Internet Corporation for Assigned Names and Numbers (ICANN)
  - Allocates large blocks to Regional Internet Registries
- Regional Internet Registries (RIRs)
  - E.g., ARIN (American Registry for Internet Numbers)
  - Allocates to ISPs and large institutions
- Internet Service Providers (ISPs)
  - Allocate address blocks to their customers
  - Who may, in turn, allocate to their customers…

# Long Term Growth (1989-2005)



# Long Term Growth (1989-2017)



# Network addresses

- Classless addressing
  - Can we have exactly same IP addresses with different CIDR notations?
    - IP address would be 194.24.0.20 (regardless of subnet mask)
      - 194.24.0.20/15: 194.24.0.0 - 194.25.255.255
      - 194.24.0.20/16: 194.24.0.0 - 194.24.255.255 [subnet of above]
      - 194.24.0.20/20: 194.24.0.0 - 194.24.15.255 [subnet of above]
      - 194.24.0.20/21: 194.24.0.0 - 194.24.7.255 [subnet of above]
      - 194.24.0.20/24: 194.24.0.0 - 194.24.0.255 [subnet of above]
      - 194.24.0.20/29: 194.24.0.16 - 194.24.0.23 [subnet of above]

# Network addresses

- Special addresses in each block
  - First and last network address
    - E.g., 192.168.5.0/24 (subnet mask 255.255.255.0)
      - Network address: 192.168.5.0
      - Directed broadcast address: 192.168.5.255

| | Binary form | Dot-decimal |
|---|---|---|
| Network address | 11000000.10101000.00000101.00000000 | 192.168.5.0 |
| Directed broadcast address | 11000000.10101000.00000101.11111111 | 192.168.5.255 |

# Network addresses

- Special addresses in each block
  - First and last network address
    - Can address ending with 0 or 255 be used as a host address?
      - **Yes!** only exception are **FIRST** and **LAST** addresses of network
        - E.g., 192.168.0.0/16 (subnet mask 255.255.0.0)
          - Network address: 192.168.0.0
          - Directed broadcast address: 192.168.255.255
          - Addresses usable for hosts
            - 192.168.**1.**0,
            - 192.168.**2.**0,
            - …
            - 192.168.**1.**255,
            - 192.168.**2.**255,
            - …

# Network addresses

- Special addresses in each block
  - First and last network address
    - Network address/directed broadcast address always end with 0/255?
      - **No!**
        - E.g., CIDR 203.0.113.16/28
          - Network address: 203.0.113.16
          - Directed broadcast address: 203.0.113.31

| | **Binary form** | **Dot-decimal** |
|---|---|---|
| Network address | 11001011.00000000.01110001.00010000 | 203.0.113.16 |
| Directed broadcast address | 11001011.00000000.01110001.00011111 | 203.0.113.31 |

# Network addresses

- Special addresses in each block
  - First and last network address
    - A special case is /31 network (subnet mask 255.255.255.254)
      - Capacity for just two hosts
      - Typically, used for point-to-point connections
      - No network address or directed broadcast address

# Network addresses

- Special addresses in each block
  - First and last network address
    - A special case is /32 network (subnet mask 255.255.255.255)
      - Capacity for just one host
      - Cannot be used for assigning address to network links
        - Need more than one address per link
      - Is strictly reserved for use on links that can have only one address
        - E.g., loopback interface
      - No network address or directed broadcast address

# Network addresses

- Special blocks of addresses
  - 0.0.0.0/8
    - RFC1700, page 4: *“0.0.0.0/8 - Addresses in this block refer to source hosts on "this" network. Address 0.0.0.0/32 may be used as a source address for this host on this network; other addresses within 0.0.0.0/8 may be used to refer to specified hosts on this network.”*

    - Non-routable address, describes an invalid or unknown target

# Network addresses

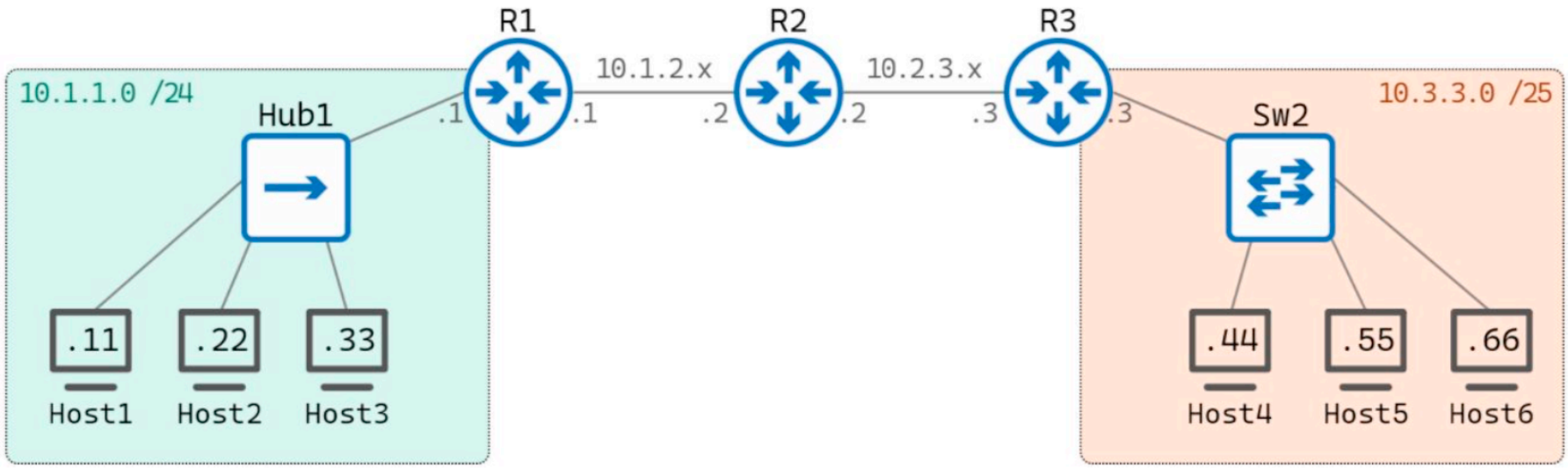
- Special blocks of addresses
  - 0.0.0.0/8
    - 0.0.0.0/0
      - "All addresses"
      - Covers every IP on Internet
      - Used in routing (specify default gateway)
      - Used in firewalls (specify default rules)
    - Is different from
    - 0.0.0.0/32 (same as 0.0.0.0)
      - Used on application-level as uninitialized IP address
      - "Unspecified address" (host without IP uses it in DHCPDISCOVER)
      - INADDR_ANY (configuring server to bind listening sockets)

# Network addresses

- Special blocks of addresses
  - Link-local addresses
    - What happens when
      - Host is unable to get IP address from DHCP server
      - And, no IP address assigned manually
        - Host assigns itself an IP address from link-local addresses
    - 169.254.0.0/16
      - 169.254.0.0 - 169.254.255.255 (65,536 addresses)
    - Normally used when
      - No external, stateful address config mechanism (e.g., DHCP) exists
      - Or, primary configuration method has failed

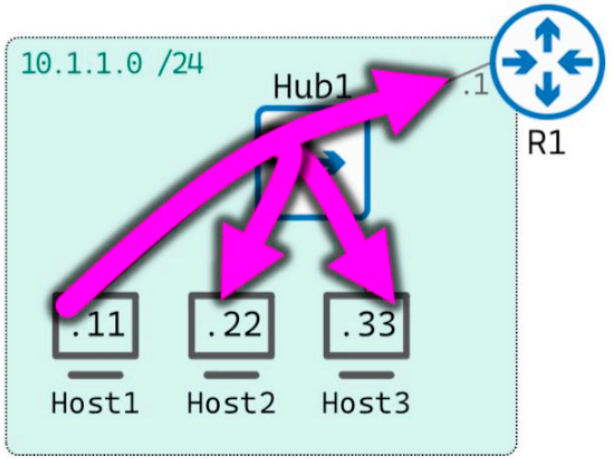
# Network addresses

- Broadcast addresses
  - E.g., consider the following topology



# Network addresses

- Broadcast addresses
  - Local broadcast (255.255.255.255)



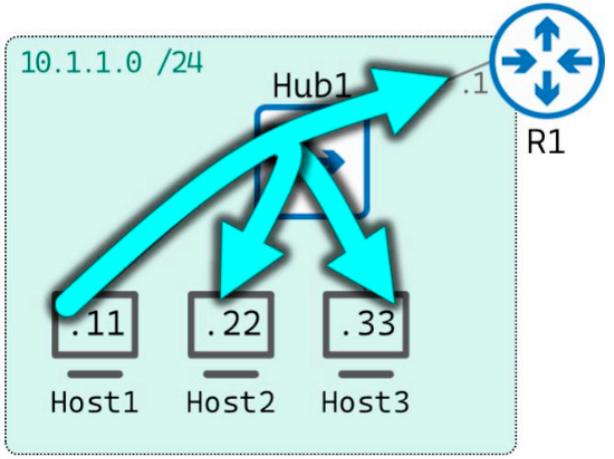

```
Host1# ping 255.255.255.255
PING 255.255.255.255 (255.255.255.255): 56 data bytes
64 bytes from 10.1.1.11: seq=0 ttl=64 time=0.044 ms
64 bytes from 10.1.1.33: seq=0 ttl=64 time=0.944 ms (DUP!)
64 bytes from 10.1.1.22: seq=0 ttl=64 time=1.108 ms (DUP!)
64 bytes from 10.1.1.1: seq=0 ttl=255 time=1.324 ms (DUP!)
^C
--- 255.255.255.255 ping statistics ---
1 packets transmitted, 1 packets received, 3 duplicates, 0% packet loss
round-trip min/avg/max = 0.044/0.855/1.324 ms
Host1#
```

| Protocol | Source | Destination | Info |
|---|---|---|---|
| ICMP | 10.1.1.11 | 255.255.255.255 | Echo (ping) request id=0x6801, seq=0/0, ttl=64 (broadcast) |
| ICMP | 10.1.1.33 | 10.1.1.11 | Echo (ping) reply id=0x6801, seq=0/0, ttl=64 |
| ICMP | 10.1.1.22 | 10.1.1.11 | Echo (ping) reply id=0x6801, seq=0/0, ttl=64 |
| ICMP | 10.1.1.1 | 10.1.1.11 | Echo (ping) reply id=0x6801, seq=0/0, ttl=255 |

```
> Frame 3: 98 bytes on wire (784 bits), 98 bytes captured (784 bits) on interface 0
> Ethernet II, Src: ee:ee:ee:11:11:11, Dst: ff:ff:ff:ff:ff:ff
> Internet Protocol Version 4, Src: 10.1.1.11, Dst: 255.255.255.255
> Internet Control Message Protocol
```

# Network addresses

- Broadcast addresses
  - Directed broadcast (10.1.1.255)



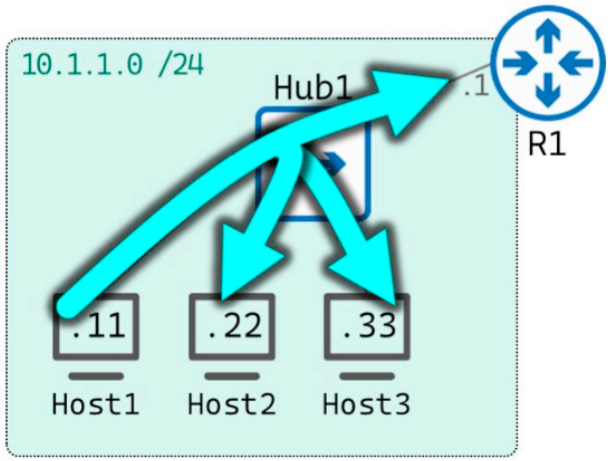

```
Host1# ping 10.1.1.255
PING 10.1.1.255 (10.1.1.255): 56 data bytes
64 bytes from 10.1.1.11: seq=0 ttl=64 time=0.046 ms
64 bytes from 10.1.1.33: seq=0 ttl=64 time=0.615 ms (DUP!)
64 bytes from 10.1.1.22: seq=0 ttl=64 time=0.835 ms (DUP!)
64 bytes from 10.1.1.1: seq=0 ttl=255 time=1.261 ms (DUP!)
^C
--- 10.1.1.255 ping statistics ---
1 packets transmitted, 1 packets received, 3 duplicates, 0% packet loss
round-trip min/avg/max = 0.046/0.689/1.261 ms
Host1#
```

| Protocol | Source | Destination | Info |
|---|---|---|---|
| ICMP | 10.1.1.11 | 10.1.1.255 | Echo (ping) request id=0x6901, seq=0/0, ttl=64 (no response … |
| ICMP | 10.1.1.33 | 10.1.1.11 | Echo (ping) reply id=0x6901, seq=0/0, ttl=64 |
| ICMP | 10.1.1.22 | 10.1.1.11 | Echo (ping) reply id=0x6901, seq=0/0, ttl=64 |
| ICMP | 10.1.1.1 | 10.1.1.11 | Echo (ping) reply id=0x6901, seq=0/0, ttl=255 |

```
> Frame 7: 98 bytes on wire (784 bits), 98 bytes captured (784 bits) on interface 0
> Ethernet II, Src: ee:ee:ee:11:11:11, Dst: ff:ff:ff:ff:ff:ff
> Internet Protocol Version 4, Src: 10.1.1.11, Dst: 10.1.1.255
> Internet Control Message Protocol
```

# Network addresses

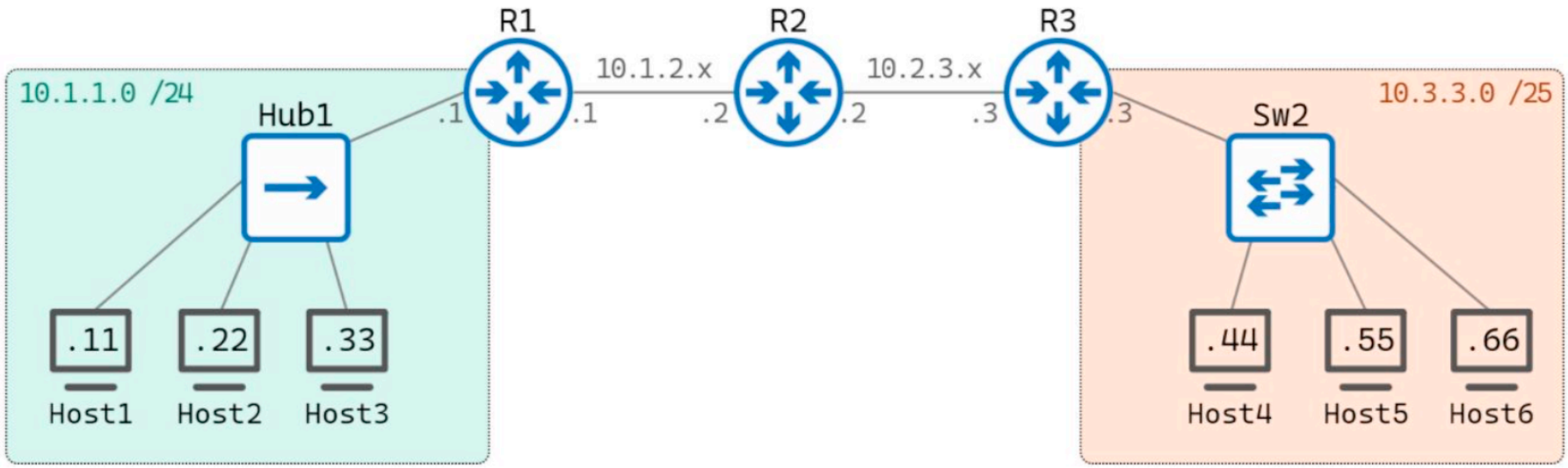
- Broadcast addresses
  - Directed broadcast (10.1.1.255)




```
> Ethernet II, Src: ee:ee:ee:11:11:11, Dst: ff:ff:ff:ff:ff:ff
> Internet Protocol Version 4, Src: 10.1.1.11, Dst: 10.1.1.255
v Internet Control Message Protocol
    Type: 8 (Echo (ping) request)
    Code: 0
    Checksum: 0xb20d [correct]
    [Checksum Status: Good]
    Identifier (BE): 26881 (0x6901)
    Identifier (LE): 361 (0x0169)
    Sequence number (BE): 0 (0x0000)
    Sequence number (LE): 0 (0x0000)
  v [No response seen]
    > [Expert Info (Warning/Sequence): No response seen to ICMP request]
  > Data (56 bytes)
```

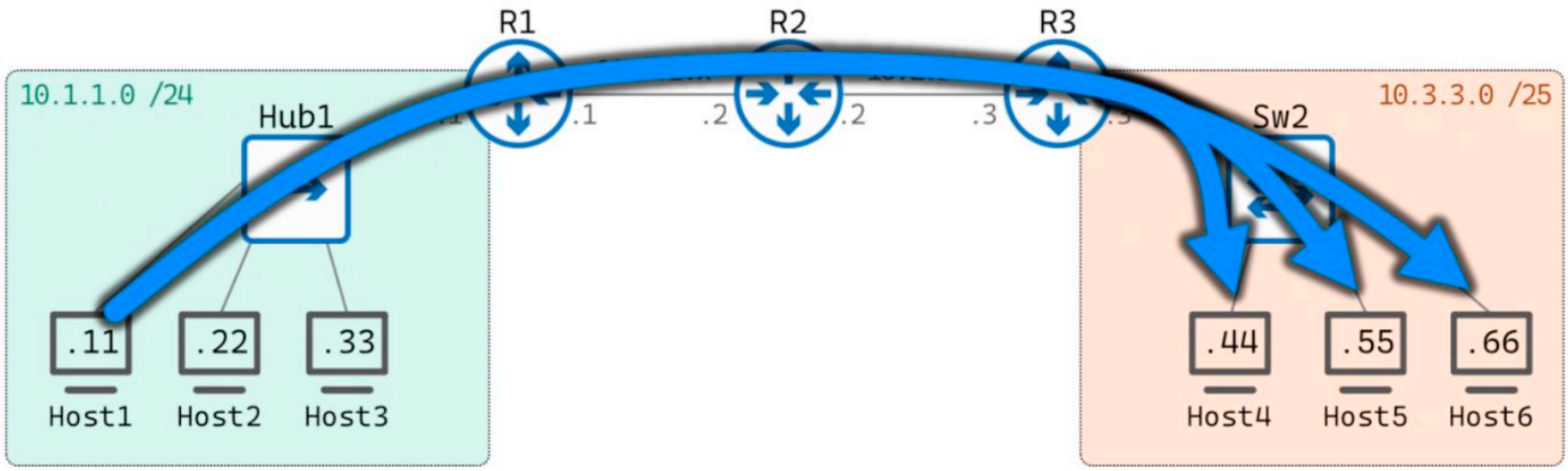
# Network addresses

- Broadcast addresses
  - Directed broadcast to a foreign network



# Network addresses

- Broadcast addresses
  - Directed broadcast to a foreign network



# Network addresses

- Broadcast addresses
  - Directed broadcast to a foreign network

```
Host1# ping 10.3.3.127
PING 10.3.3.127 (10.3.3.127): 56 data bytes
64 bytes from 10.2.3.3: seq=0 ttl=253 time=1.171 ms
64 bytes from 10.3.3.66: seq=0 ttl=61 time=3.683 ms (DUP!)
64 bytes from 10.3.3.55: seq=0 ttl=61 time=7.340 ms (DUP!)
64 bytes from 10.3.3.44: seq=0 ttl=61 time=9.838 ms (DUP!)
^C
--- 10.3.3.127 ping statistics ---
1 packets transmitted, 1 packets received, 3 duplicates, 0% packet loss
round-trip min/avg/max = 1.171/5.508/9.838 ms
```

| Protocol | Source | Destination | Info |
|---|---|---|---|
| ICMP | 10.1.1.11 | 255.255.255.255 | Echo (ping) request id=0x6b01, seq=0/0, ttl=61 (broadcast) |
| ICMP | 10.3.3.66 | 10.1.1.11 | Echo (ping) reply id=0x6b01, seq=0/0, ttl=64 |
| ICMP | 10.3.3.55 | 10.1.1.11 | Echo (ping) reply id=0x6b01, seq=0/0, ttl=64 |
| ICMP | 10.3.3.44 | 10.1.1.11 | Echo (ping) reply id=0x6b01, seq=0/0, ttl=64 |

```
> Frame 3: 98 bytes on wire (784 bits), 98 bytes captured (784 bits) on interface 0
> Ethernet II, Src: ee:ee:10:33:33:33, Dst: ff:ff:ff:ff:ff:ff
> Internet Protocol Version 4, Src: 10.1.1.11, Dst: 255.255.255.255
> Internet Control Message Protocol
```

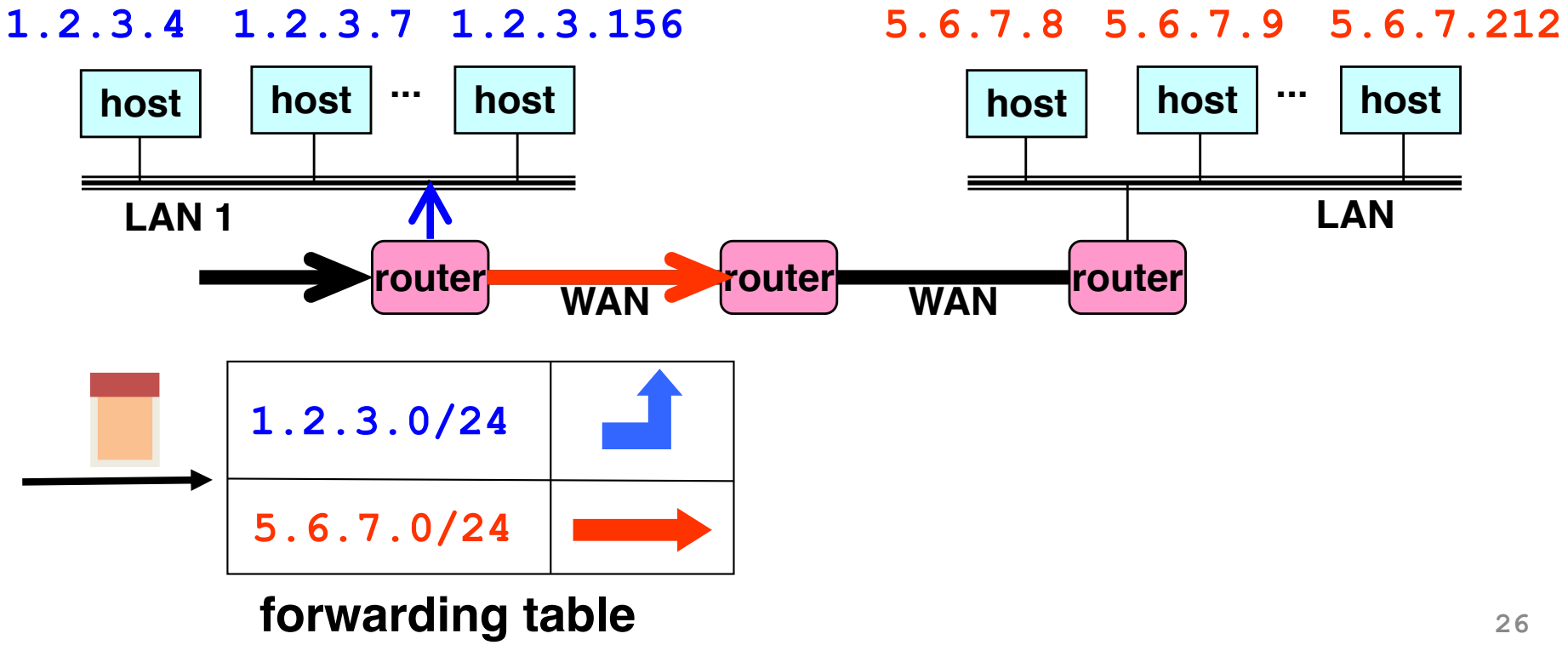
# Packet Forwarding

# Hop-by-Hop Packet Forwarding

- Each router has a forwarding table
  - Maps destination address to outgoing interface
- Upon receiving a packet
  - Inspect the destination address in the header
  - Index into the table
  - Determine the outgoing interface
  - Forward the packet out that interface
- Then, the next router in the path repeats

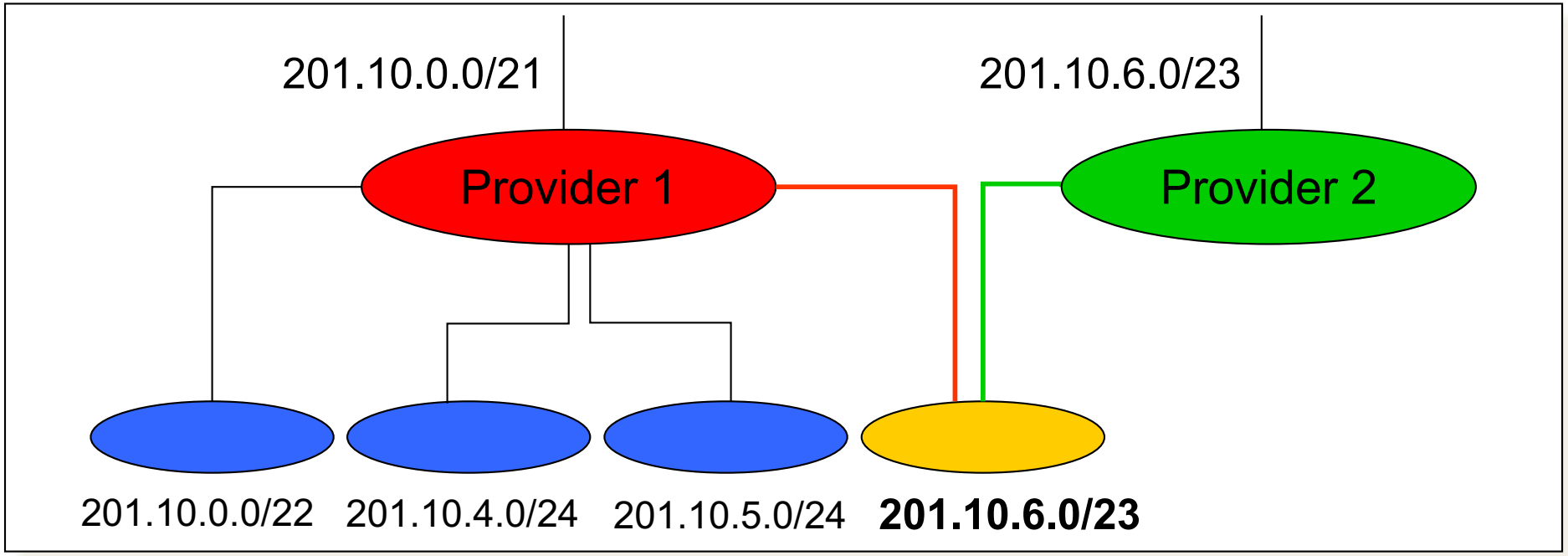
# Separate Forwarding Entry Per Prefix

- Prefix-based forwarding
  - Map the destination address to matching prefix
  - Forward to the outgoing interface



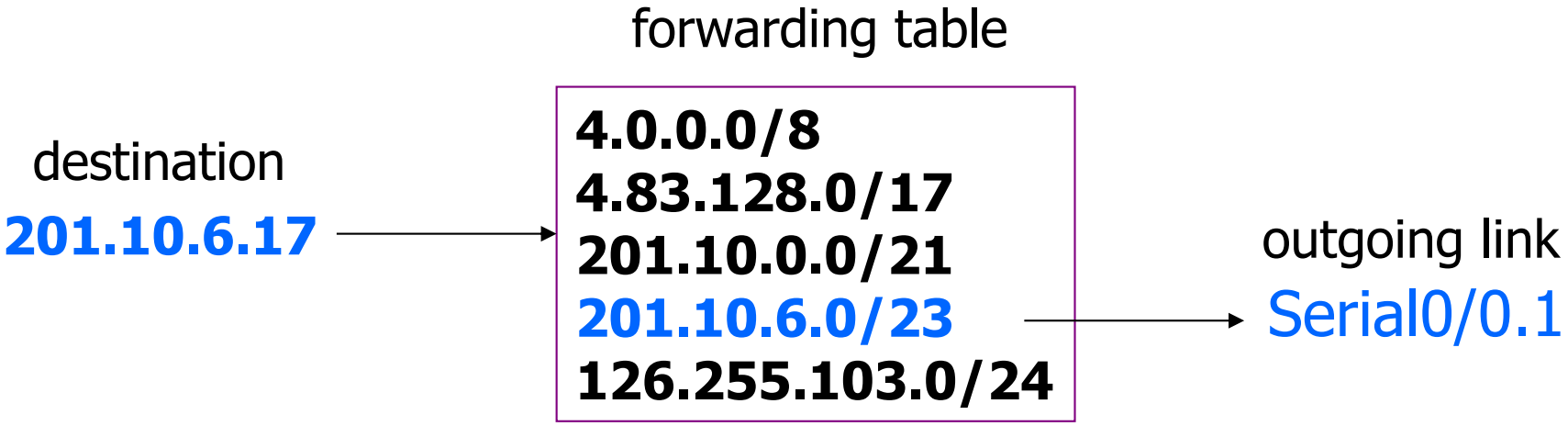
# CIDR Makes Packet Forwarding Harder

- Forwarding table may have many matches
  - E.g., entries for 201.10.0.0/21 and 201.10.6.0/23
  - The IP address 201.10.6.17 would match both!



# Longest Prefix Match Forwarding

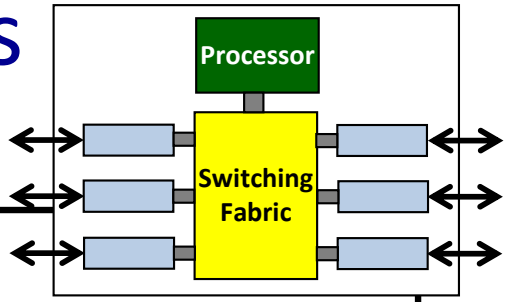
- Destination-based forwarding
  - Packet has a destination address
  - Router identifies longest-matching prefix
  - Cute algorithmic problem: very fast lookups



# Creating a Forwarding Table

- Entries can be statically configured
  - E.g., “map 12.34.158.0/24 to Serial0/0.1”
- But, this doesn’t adapt
  - To failures
  - To new equipment
  - To the need to balance load
- That is where the *control plane* comes in
  - Routing protocols

# Data, Control, & Management Planes




| | Data | Control | Management |
|---|---|---|---|
| Time-scale | Packet (ns) | Event (10 ms to sec) | Human (min to hours) |
| Tasks | Forwarding, buffering, filtering, scheduling | Routing, signaling | Analysis, configuration |
| Location | Line-card hardware | Router software | Humans or scripts |

# Q's: MAC vs. IP Addressing

- Hierarchically allocated

  Y) MAC M) IP C) Both A) Neither

- Organized topologically

  Y) MAC M) IP C) Both A) Neither

- Forwarding via exact match on address

  Y) MAC M) IP C) Both A) Neither

- Automatically calculate forwarding by observing data

  Y) Ethernet switches M) IP routers C) Both A) Neither

- Per connection state in the network

  Y) MAC M) IP C) Both A) Neither

- Per host state in the network

  Y) MAC M) IP C) Both A) Neither

# Q's: MAC vs. IP Addressing

- Hierarchically allocated

  Y) MAC  M) IP  **C) Both**  A) Neither

- Organized topologically

  Y) MAC  **M) IP**  C) Both  A) Neither

- Forwarding via exact match on address

  **Y) MAC**  M) IP  C) Both  A) Neither

- Automatically calculate forwarding by observing data

  **Y) Ethernet switches**  M) IP routers  C) Both  A) Neither

- Per connection state in the network

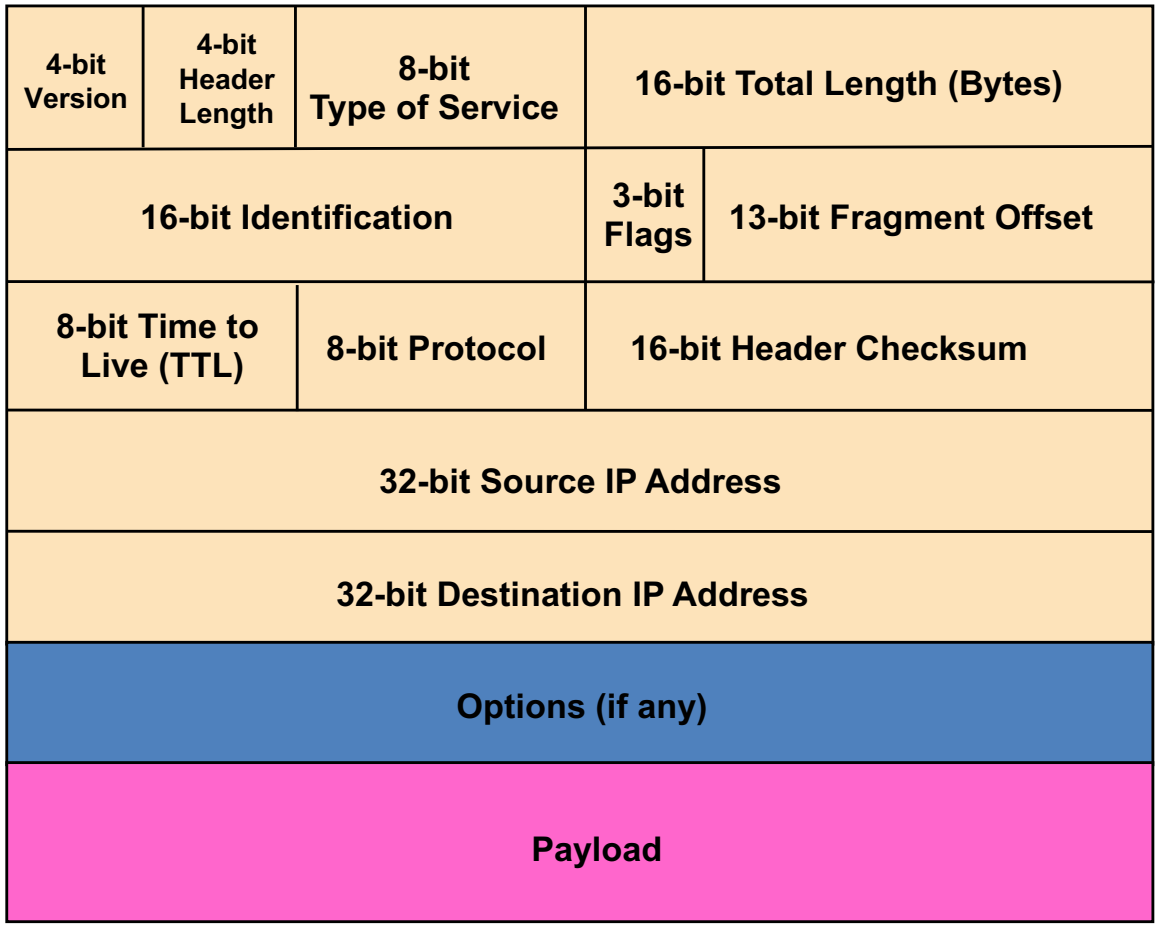
  Y) MAC  M) IP  C) Both  **A) Neither**

- Per host state in the network

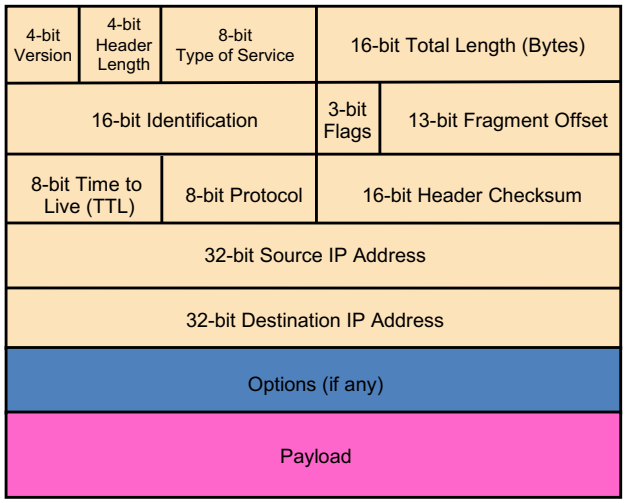
  **Y) MAC**  M) IP  C) Both  A) Neither

# IP Packet Format

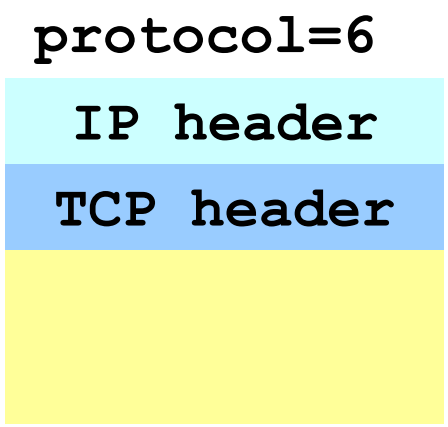
# IP Packet Structure

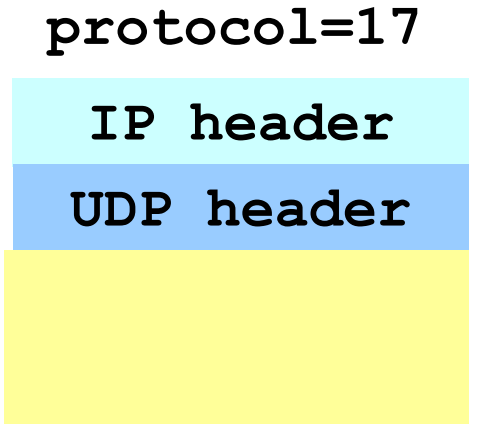


# IP Header: Transport Protocol

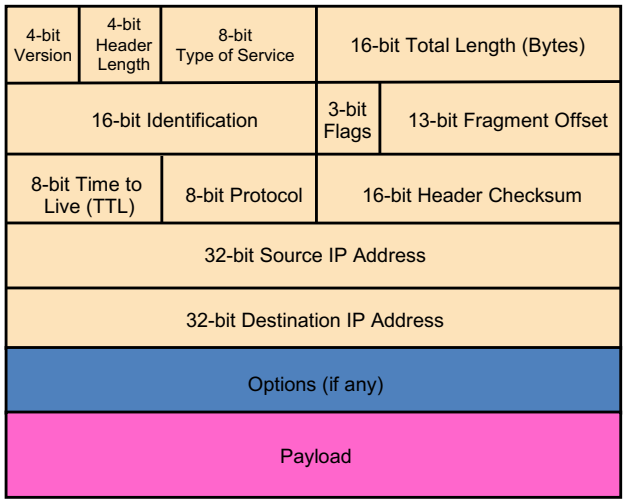



- Protocol (8 bits)
  - Identifies the higher-level protocol
    - E.g., "6" for the Transmission Control Protocol (TCP)
    - E.g., "17" for the User Datagram Protocol (UDP)
  - Important for demultiplexing at receiving host
    - Indicates what kind of header to expect next



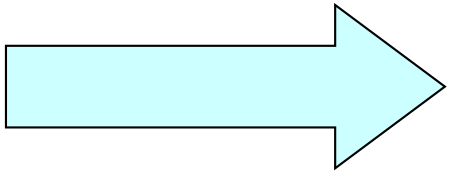



# IP Header: Header Checksum




- Checksum (16 bits)
  - Sum of all 16-bit words in the header
  - If header bits are corrupted, checksum won't match
  - Receiving  discards corrupted packets

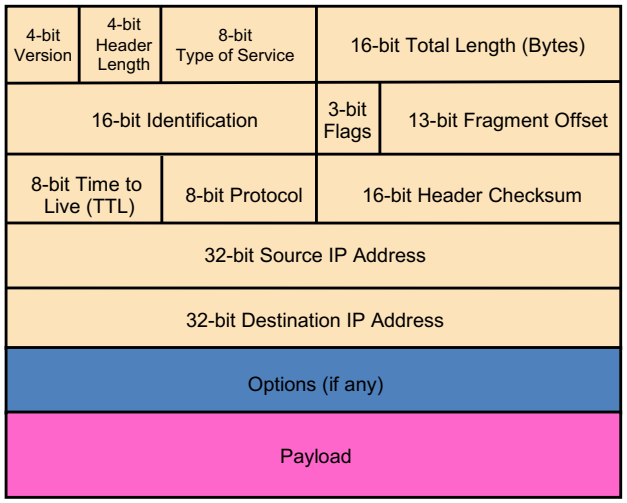
```
  134
+ 212
-----
= 346
```



Mismatch!

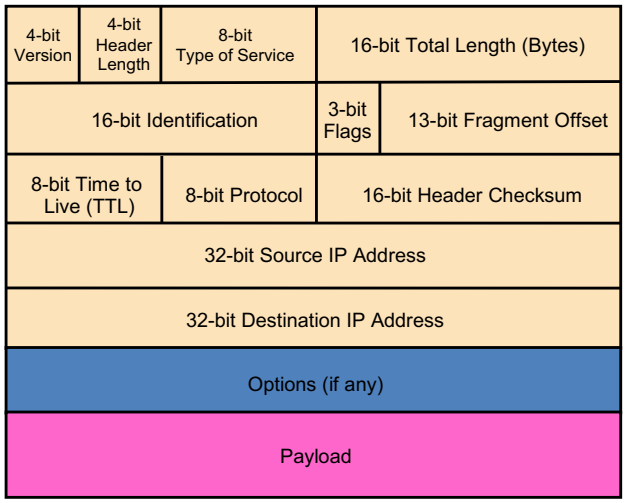
```
  134
+ 216
-----
= 350
```

# IP Header: Version, Length, ToS




- Version number (4 bits)
  - Necessary to know what other fields to expect
  - Typically “4” (for IPv4), and sometimes “6” (for IPv6)
- Header length (4 bits)
  - Number of 32-bit words in the header
  - Typically “5” (for a 20-byte IPv4 header)
  - Can be more when “IP options” are used
- Type-of-Service (8 bits)
  - Allow different packets to be treated differently
  - Low delay for audio, high bandwidth for bulk transfer

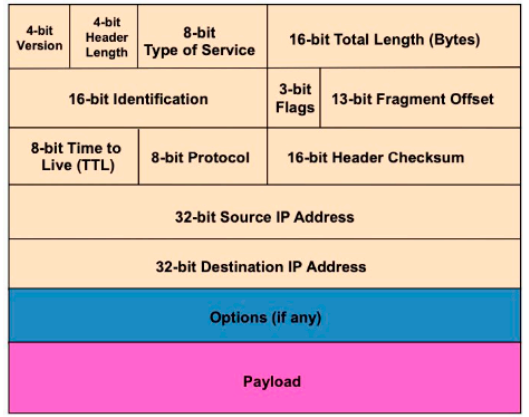
# IP Header: Length, Fragments, TTL




- Total length (16 bits)
  - Number of bytes in the packet
  - Max size is 63,535 bytes ($2^{16}$ -1)
  - … though most links impose smaller limits
- Time-To-Live (8 bits)
  - Used to identify packets stuck in forwarding loops
  - … and eventually discard them from the network
- Fragmentation information (32 bits)
  - Supports dividing a large IP packet into fragments
  - … in case a link cannot handle a large IP packet

# IP packet format

- IP header: Fragmentation
  - E.g.,
    - 576 B MTU; 20 B header; 1420 B packet data




| Fragment | Identification | Flags | Fragment Offset | Total Length |
|---|---|---|---|---|
| One | X=987 | 001 | 0 | 572 |
| Two | X=987 | 001 | 69 | 572 |
| Three | X=987 | 000 | 138 | 336 |

# Conclusion

- Best-effort global packet delivery
  - Simple end-to-end abstraction
  - Enables higher-level abstractions on top
  - Doesn't rely on much from the links below
- IP addressing and forwarding
  - Hierarchy for scalability and decentralized control
  - Allocation of IP prefixes
  - Longest prefix match forwarding
- Next time: switches & routers

CHAPTER

# 4

# *Introduction to Network Layer*

To solve the problem of delivery through several links, the network layer (or the internetwork layer, as it is sometimes called) was designed. The network layer is responsible for host-to-host delivery and for routing the packets through the routers. In this chapter, we give an introduction to the network layer to prepare readers for a more thorough coverage in Chapters 5 through 12. In this chapter we give the rationale for the need of the network layers and the issues involved. However, we need the next eight chapters to fully understand how these issues are answered. We may even need to cover the whole book before we get satisfactory answers for them.

## OBJECTIVES

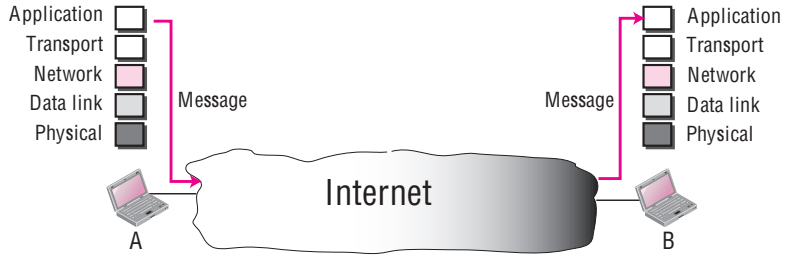
*This chapter has several objectives:*

- To introduce switching and in particular packet switching as the mechanism of data delivery in the network layer.
- To discuss two distinct types of services a packet-switch network can provide: connectionless service and connection-oriented service.
- To discuss how routers forward packets in a connectionless packet-switch network using the destination address of the packet and a routing table.
- To discuss how routers forward packets in a connection-oriented packet-switch network using the label on the packet and a routing table.
- To discuss services already provided in the network layer such as logical addressing and delivery at the source, at each router, and at the destination.
- To discuss issues or services that are not directly provided in the network layer protocol, but are sometimes provided by some auxiliary protocols or some protocols added later to the Internet.

## 4.1 INTRODUCTION

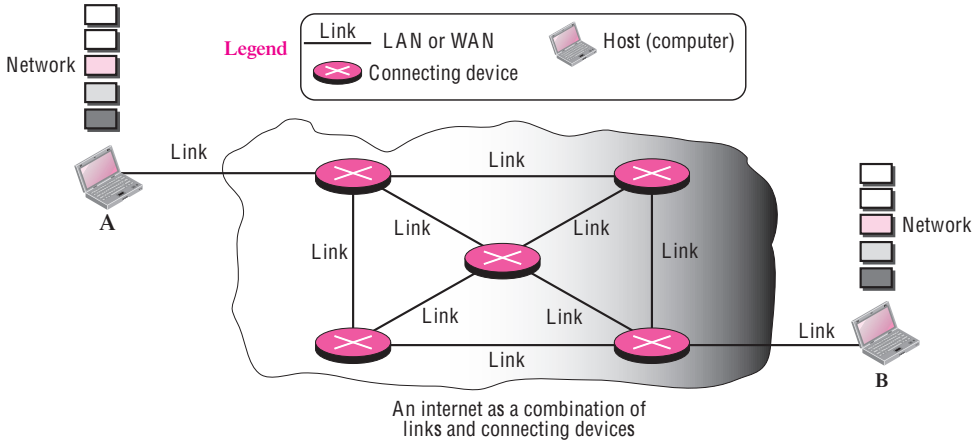
At the conceptual level, we can think of the global Internet as a black box network that connects millions (if not billions) of computers in the world together. At this level, we are only concerned that a message from the application layer in one computer reaches the application layer in another computer. In this conceptual level, we can think of communication between A and B as shown in Figure 4.1.

**Figure 4.1** *Internet as a black box*




The Internet, however, is not one single network; it is made of many networks (or links) connected together through the connecting devices. In other words, the Internet is an internetwork, a combination of LANs and WANs. To better understand the role of the network layer (or the internetwork layer), we need to move from our conceptual level and think about all of these LANs and WANs that make the Internet. Since it is impossible to show all of these LANs and WANs, we show only an imaginary small internet with a few networks and a few connecting devices, as shown in Figure 4.2.

**Figure 4.2** *Internet as a combination of LAN and WANs connected together*




An internet as a combination of links and connecting devices

In this model, a connecting device such as a router acts as a switch. When a packet arrives from one of its ports (interface), the packet is forwarded through another port to the next switch (or final destination). In other words, a process called **switching** occurs at the connecting device.

## 4.2 SWITCHING

From the previous discussion, it is clear that the passage of a message from a source to a destination involves many decisions. When a message reaches a connecting device, a decision needs to be made to select one of the output ports through which the packet needs to be send out. In other words, the connecting device acts as a switch that connects one port to another port.

### Circuit Switching

One solution to the switching is referred to as **circuit switching**, in which a physical circuit (or channel) is established between the source and destination of the message before the delivery of the message. After the circuit is established, the entire message, is transformed from the source to the destination. The source can then inform the network that the transmission is complete, which allows the network to open all switches and use the links and connecting devices for another connection. The circuit switching was never implemented at the network layer; it is mostly used at the physical layer.

**In circuit switching, the whole message is sent from the source to the destination without being divided into packets.**

#### Example 4.1

A good example of a circuit-switched network is the early telephone systems in which the path was established between a caller and a callee when the telephone number of the callee was dialed by the caller. When the callee responded to the call, the circuit was established. The voice message could now flow between the two parties, in both directions, while all of the connecting devices maintained the circuit. When the caller or callee hung up, the circuit was disconnected. The telephone network is not totally a circuit-switched network today.

### Packet Switching

The second solution to switching is called **packet switching**. The network layer in the Internet today is a packet-switched network. In this type of network, a message from the upper layer is divided into manageable packets and each packet is sent through the network. The source of the message sends the packets one by one; the destination of the message receives the packets one by one. The destination waits for all packets belonging to the same message to arrive before delivering the message to the upper layer. The connecting devices in a packet-switching network still need to decide how to route the packets to the final destination. Today, a packet-switched network can use two different

approaches to route the packets: the datagram approach and the virtual circuit approach. We discuss both approaches in the next section.

**In packet switching, the message is first divided into manageable packets at the source before being transmitted. The packets are assembled at the destination.**

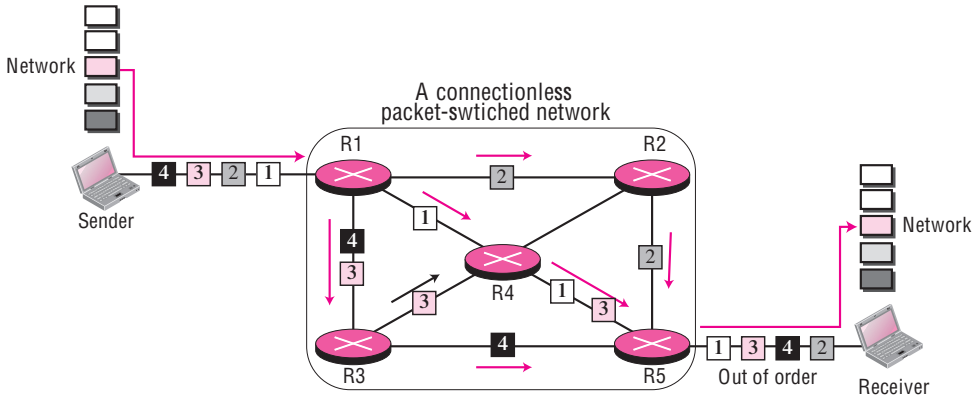
## 4.3 PACKET SWITCHING AT NETWORK LAYER

The network layer is designed as a packet-switched network. This means that the packet at the source is divided into manageable packets, normally called **datagrams**. Individual datagrams are then transferred from the source to the destination. The received datagrams are assembled at the destination before recreating the original message. The packet-switched network layer of the Internet was originally designed as a *connectionless service,* but recently there is a tendency to change this *to a connection-oriented service*. We first discuss the dominant trend and then briefly discuss the new one.

### Connectionless Service

When the Internet started, the network layer was designed to provide a **connectionless service**, in which the network layer protocol treats each packet independently, with each packet having no relationship to any other packet. The packets in a message may or may not travel the same path to their destination. When the Internet started, it was decided to make the network layer a connectionless service to make it simple. The idea was that the network layer is only responsible for delivery of packets from the source to the destination. Figure 4.3 shows the idea.

**Figure 4.3** *A connectionless packet-switched network*

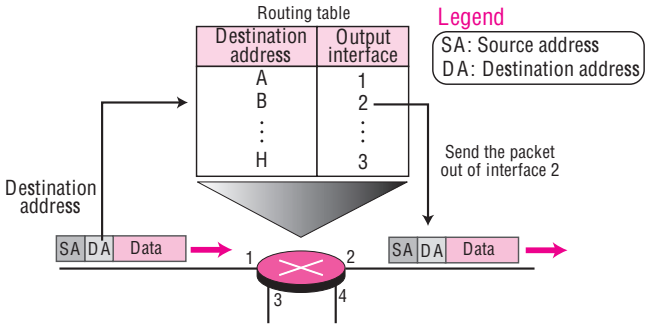



When the network layer provides a connectionless service, each packet traveling in the Internet is an independent entity; there is no relationship between packets belonging to the same message. The switches in this type of network are called *routers*. A packet

belonging to a message may be followed by a packet belonging to the same message or a different message. A packet may be followed by a packet coming from the same or from a different source.

Each packet is routed based on the information contained in its header: source and destination address. The destination address defines where it should go; the source address defines where it comes from. The router in this case routes the packet based only on the destination address. The source address may be used to send an error message to the source if the packet is discarded. Figure 4.4 shows the forwarding process in a router in this case. We have used symbolic addresses such as A and B.

**Figure 4.4** *Forwarding process in a router when used in a connectionless network*

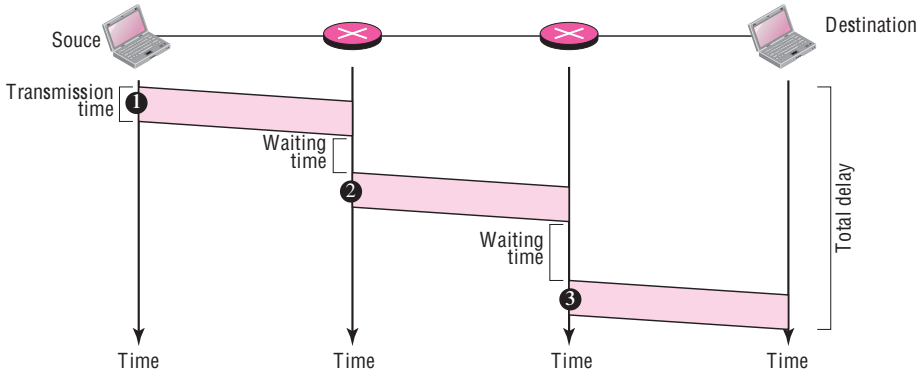



**In a connectionless packet-switched network, the forwarding decision is based on the destination address of the packet.**

### *Delay In Connectionless Network*

If we ignore the fact that the packet may be lost and resent and also the fact that the destination may be needed to wait to receive all packets, we can model the delay as shown in Figure 4.5.

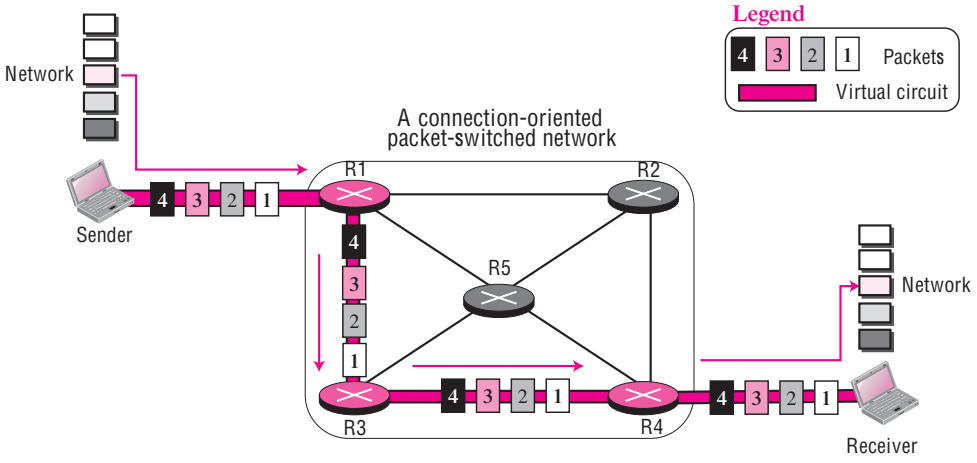
**Figure 4.5** *Delay in a connectionless network*



## Connection-Oriented Service

In a **connection-oriented service,** there is a relation between all packets belonging to a message. Before all datagrams in a message can be sent, a virtual connection should be set up to define the path for the datagrams. After connection setup, the datagrams can follow the same path. In this type of service, not only must the packet contain the source and destination addresses, it must also contain a *flow label,* a *virtual circuit identifier* that defines the virtual path the packet should follow. We will shortly show how this flow label is determined, but for the moment, we assume that the packet carries this *label.* Although it looks as though the use of the label may make the source and destination addresses useless, the parts of the Internet that use connectionless service at the network layer still keep these addresses for several reasons. One reason is that part of the packet path may still be using the connectionless service. Another reason is that the protocol at the network layer is designed with these addresses and it may take a while before they can be changed. Figure 4.6 shows the concept of connection-oriented service.

**Figure 4.6** *A connection-oriented packet switched network*



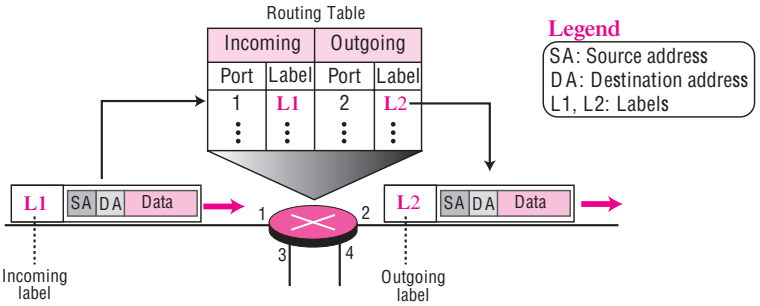

Each packet is forwarded based on the label in the packet. To follow the idea of connection-oriented design to be used in the Internet, we assume that the packet has a label when it reaches the router. Figure 4.7 shows the idea.

In this case, the forwarding decision is based on the value of the label, or virtual circuit identifier as it is sometimes called.

To create a connection-oriented service, a three-phase process is used: *setup*, *data transfer*, and *teardown*. In the setup phase, the source and destination addresses of the sender and receiver is used to make table entries for the connection-oriented service. In the teardown phase, the source and destination inform the router to delete the corresponding entries. Data transfer occurs between these two phases.

**In a connection-oriented packet switched network, the forwarding decision is based on the label of the packet.**

**Figure 4.7** *Forwarding process in a router when used in a connection-oriented network*



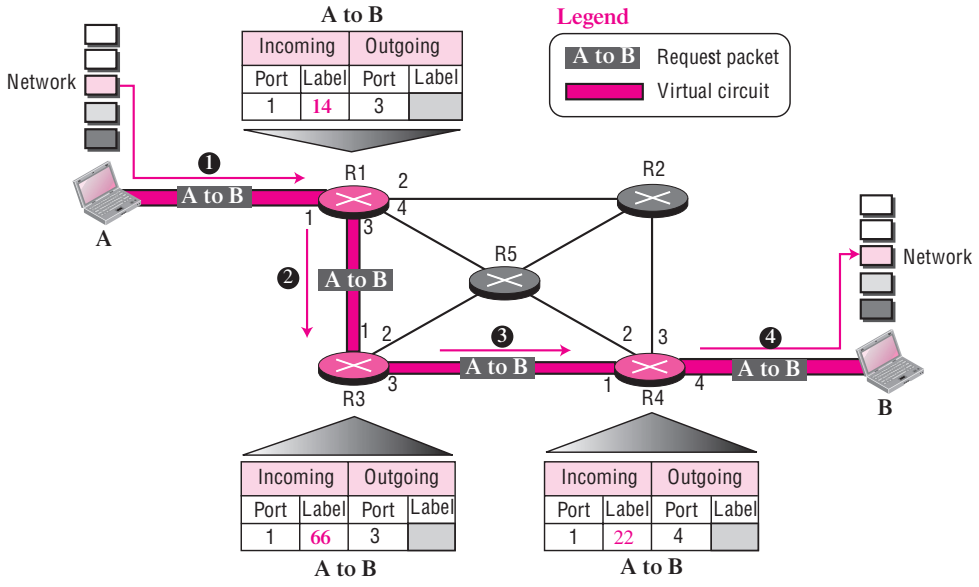

### *Setup Phase*

In the **setup phase**, a router creates an entry for a virtual circuit. For example, suppose source A needs to create a virtual circuit to destination B. Two auxiliary packets need to be exchanged between the sender and the receiver: the request packet and the acknowledgment packet.

**Request packet** A request packet is sent from the source to the destination. This auxiliary packet carries the source and destination addresses. Figure 4.8 shows the process.

1. Source A sends a request packet to router R1.
2. Router R1 receives the request packet. It knows that a packet going from A to B goes out through port 3. How the router has obtained this information is a point

**Figure 4.8** *Sending request packet in a virtual-circuit network*

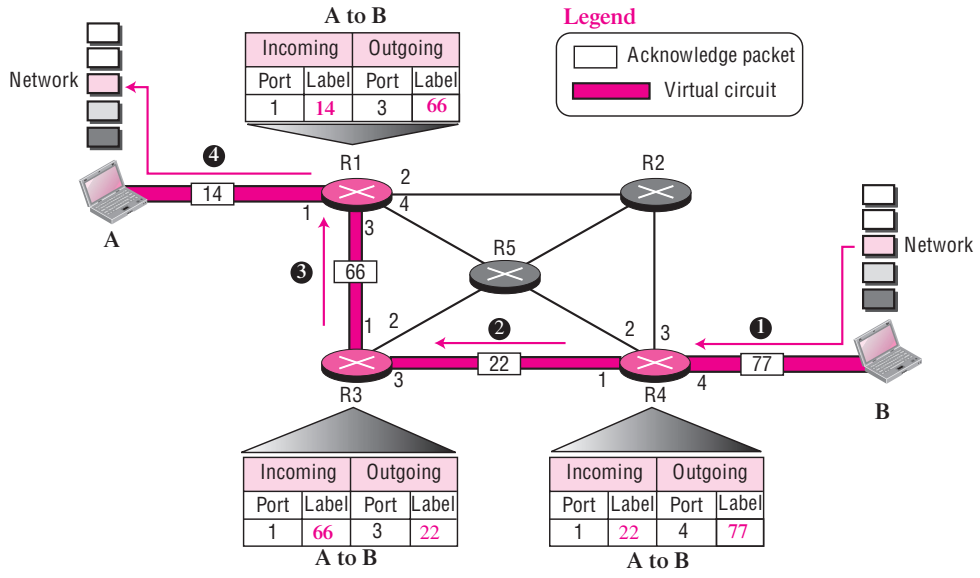


covered in future chapters. For the moment, assume that it knows the output port. The router creates an entry in its table for this virtual circuit, but it is only able to fill three of the four columns. The router assigns the incoming port (1) and chooses an available incoming label (14) and the outgoing port (3). It does not yet know the outgoing label, which will be found during the acknowledgment step. The router then forwards the packet through port 3 to router R3.

3. Router R3 receives the setup request packet. The same events happen here as at router R1; three columns of the table are completed: in this case, incoming port (1), incoming label (66), and outgoing port (2).
4. Router R4 receives the setup request packet. Again, three columns are completed: incoming port (2), incoming label (22), and outgoing port (3).
5. Destination B receives the setup packet, and if it is ready to receive packets from A, it assigns a label to the incoming packets that come from A, in this case 77. This label lets the destination know that the packets come from A, and not other sources.

**Acknowledgment Packet** A special packet, called the acknowledgment packet, completes the entries in the switching tables. Figure 4.9 shows the process.

**Figure 4.9** *Setup acknowledgment in a virtual-circuit network*



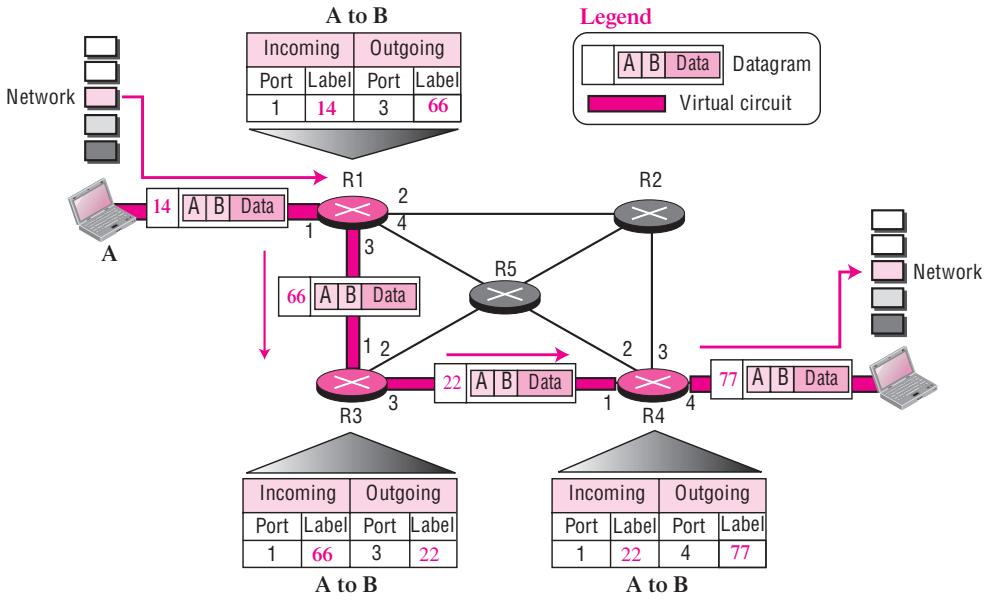

1. The destination sends an acknowledgment to router R4. The acknowledgment carries the global source and destination addresses so the router knows which entry in the table is to be completed. The packet also carries label 77, chosen by the destination as the incoming label for packets from A. Router R4 uses this label to complete the outgoing label column for this entry. Note that 77 is the incoming label for destination B, but the outgoing label for router R4.

2. Router R4 sends an acknowledgment to router R3 that contains its incoming label in the table, chosen in the setup phase. Router R3 uses this as the outgoing label in the table.
3. Router R3 sends an acknowledgment to router R1 that contains its incoming label in the table, chosen in the setup phase. Router R1 uses this as the outgoing label in the table.
4. Finally router R1 sends an acknowledgment to source A that contains its incoming label in the table, chosen in the setup phase.
5. The source uses this as the outgoing label for the data packets to be sent to destination B.

### *Data Transfer Phase*

The second phase is called the **data transfer phase**. After all routers have created their routing table for a specific virtual circuit, then the network-layer packets belonging to one message can be sent one after another. In Figure 4.10, we show the flow of one single packet, but the process is the same for 1, 2, or 100 packets. The source computer uses the label 14, which it has received from router R1 in the setup phase. Router R1 forwards the packet to router R3, but changes the label to 66. Router R3 forwards the packet to router R4, but changes the label to 22. Finally, router R4 delivers the packet to its final destination with the label 77. All the packets in the message follow the same sequence of labels to reach their destination. The packet arrives in order at the destination.

**Figure 4.10** *Flow of one packet in an established virtual circuit.*



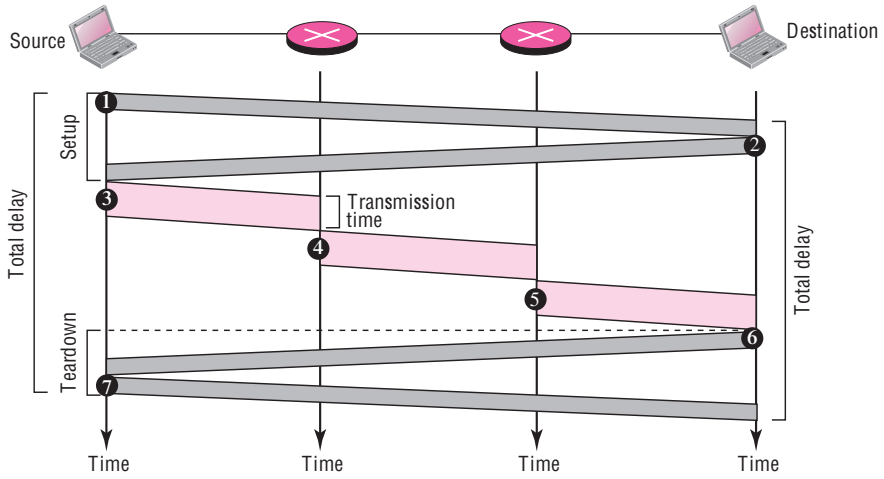

### *Teardown Phase*

In the **teardown phase,** source A, after sending all packets to B, sends a special packet called a *teardown packet*. Destination B responds with a *confirmation* packet. All routers delete the corresponding entry from their tables.

### *Delay In Connection-Oriented Network*

If we ignore the fact that the packet may be lost and resent, we can model the delay as shown in Figure 4.11.

**Figure 4.11** *Delay in a connection-oriented network*



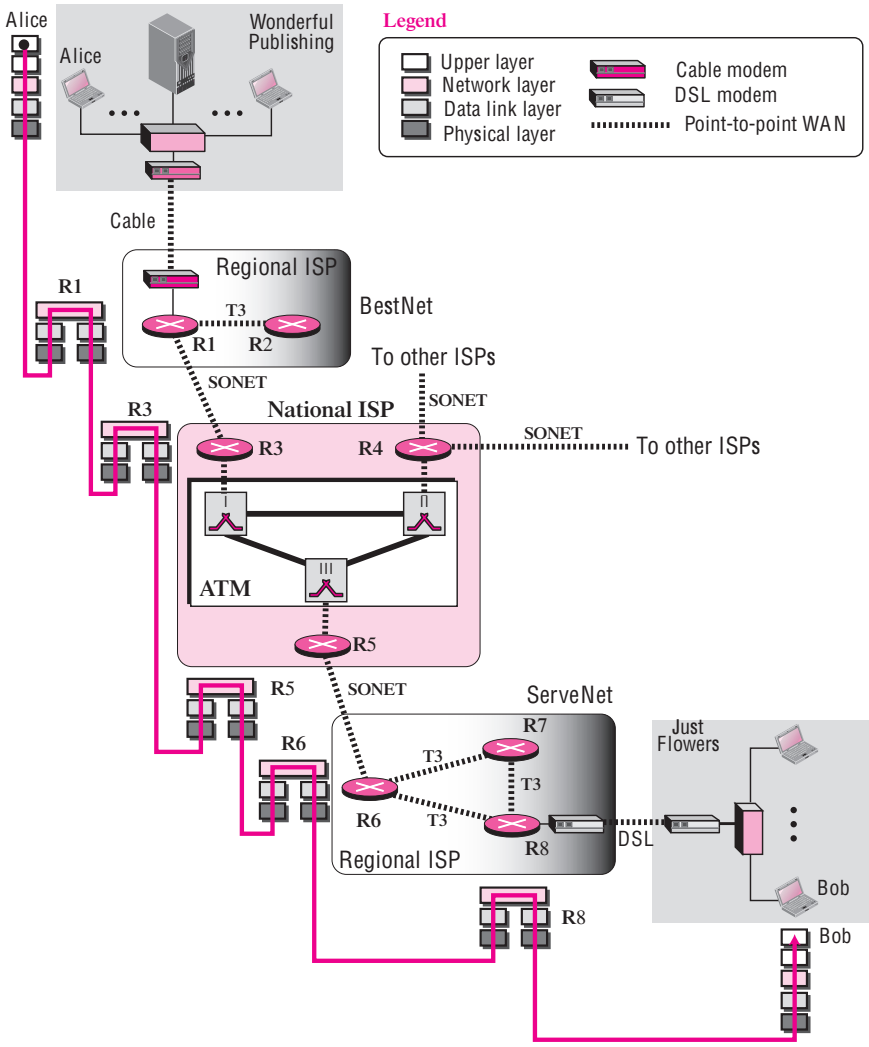

## 4.4 NETWORK LAYER SERVICES

In this section, we briefly discuss services provided by the network layer. Our discussion is mostly based on the connectionless service, the dominant service in today's Internet.

### An Example

To better understand the issues to be discussed, we give an example. In Figure 4.12, we assume Alice, who is working in a publishing company, Wonderful Publishing, needs to send a message to Bob, the manager of a flower shop, Just Flowers, to inform him that the advertising brochure for the shop has been printed and is ready to be shipped. Alice sends the message using an e-mail. Let us follow the imaginary path Alice's message takes to reach Bob. The Wonderful Publishing company uses a LAN, which is connected via a cable WAN to a regional ISP called BestNet; the Just Flowers company also uses a LAN, which is connected via a DSL WAN to another regional ISP called ServeNet. The two regional ISPs are connected through high-speed SONET WANs to a national ISP. The message that Alice sends to Bob may be split into several network-layer packets. As we will see shortly, packets may or may not follow the same path. For the sake of discussion, we follow the path of one single packet from Alice's computer to Bob's computer. We also assume that the packet passes through routers R1, R3, R5, R6, and R8 before reaching its destination. The two computers are involved in five layers; the routers are involved in three layers of the TCP/IP protocol suite.

**Figure 4.12** *An imaginary part of the Internet*



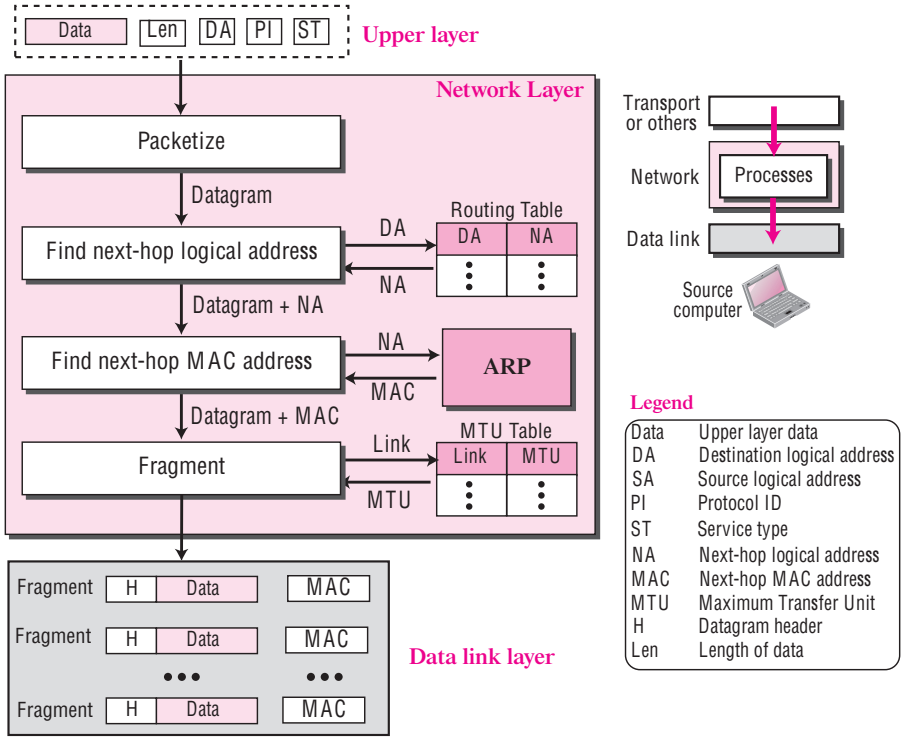

## Logical Addressing

Since the network layer provides end-to-end communication, the two computers that need to communicate with each other each need a universal identification system, referred to as network-layer address or logical address. This type of identification is provided in the network layer through a uniform and global addressing mechanism. The Internet uses an address space. Each entity that needs to use the Internet needs to be assigned a unique address from this pool. In Chapter 5 we discuss this addressing space in version 4 of the Internet; In Chapter 26, we discuss the new addressing system in version 6 (version 5 was never implemented). In Figure 4.12, Alice and Bob need two network-layer addresses to be able to communicate.

## Services Provided at the Source Computer

The network layer at the source computer provides four services: packetizing, finding the logical address of the next hop, finding the physical (MAC) address of the next hop, and fragmenting the datagram if necessary. Figure 4.13 shows these services.

**Figure 4.13** *Services provided at the source computer*




The network layer receives several pieces of information from the upper layer: data, length of data, logical destination address, protocol ID (the identifier of the protocol using the network layer), and service type (discussed later). The network layer processes these pieces of information to create a set of fragmented datagrams and the next-hop MAC address and delivered them to the data link layer. We briefly discuss each service here, but the future chapters in this part of the book explain more.

### *Packetizing*

The first duty of the network layer is to encapsulate the data coming from the upper layer in a datagram. This is done by adding a header to the data that contains the logical source and destination address of the packet, information about fragmentation, the protocol ID of the protocol that has requested the service, the data length, and possibly some options. The network layer also includes a checksum that is calculated only over the datagram header. We discuss the format of the datagram and the checksum calculation in Chapter 7. Note that the upper layer protocol only provides the logical destination

address; the logical source address comes from the network layer itself (any host needs to know its own logical address).

### *Finding Logical Address of Next Hop*

The prepared datagram contains the source and destination addresses of the packet. The datagram, as we saw before, may have to pass through many networks to reach its final destination. If the destination computer is not connected to the same network as the source, the datagram should be delivered to the next router. The source and destination address in the datagram does not tell anything about the logical address of the next hop. The network layer at the source computer needs to consult a routing table to find the logical address of the next hop.

### *Finding MAC Address of Next Hop*

The network layer does not actually deliver the datagram to the next hop; it is the duty of the data link layer to do the delivery. The data link layer needs the MAC address of the next hop to do the delivery. To find the MAC address of the next hop, the network layer could use another table to map the next-hop logical address to the MAC address. However, for the reason we discuss in Chapter 8, this task has been assigned to another auxiliary protocol called Address Resolution Protocol (ARP) that finds the MAC address of the next hop given the logical address.

### *Fragmentation*

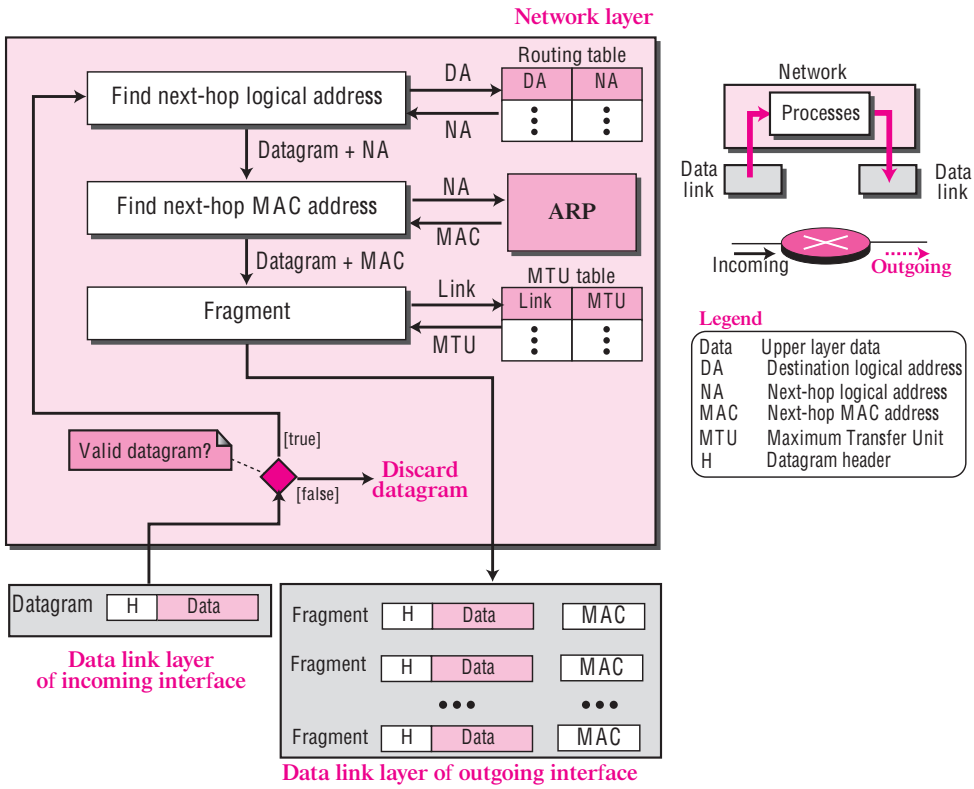
The datagram at this step may not be ready to be passed to the data link layer. As we saw in Chapter 3, most LANs and WANs have a limit on the size of the data to be carried in a frame (MTU). The datagram prepared at the network layer, may be larger than that limit. The datagram needs to be fragmented to smaller units before being passed to the data link layer. Fragmentation needs to preserve the information at the header of the datagram. In other words, although the data can be fragmented, the header needs to be repeated. In addition, some more information needs to be added to the header to define the position of the fragment in the whole datagram. We discuss fragmentation in more detail in Chapter 7.

## Services Provided at Each Router

As we have mentioned before, a router is involved with two interfaces with respect to a single datagram: the incoming interface and the outgoing interface. The network layer at the router, therefore, needs to interact with two data link layers: the data link of the incoming interface and the data link layer of the outgoing interface. The network layer is responsible to receive a datagram from the data link layer of the incoming interface, fragment it if necessary, and deliver the fragments to the data link of the outgoing interface. The router normally does not involve upper layers (with some exceptions discussed in future chapters). Figure 4.14 shows the services provides by a router at the network layer.

The three processes (finding next-hop logical address, finding next-hop MAC address, and fragmentation) here are the same as the last three processes mentioned for

**Figure 4.14** *Processing at each router*

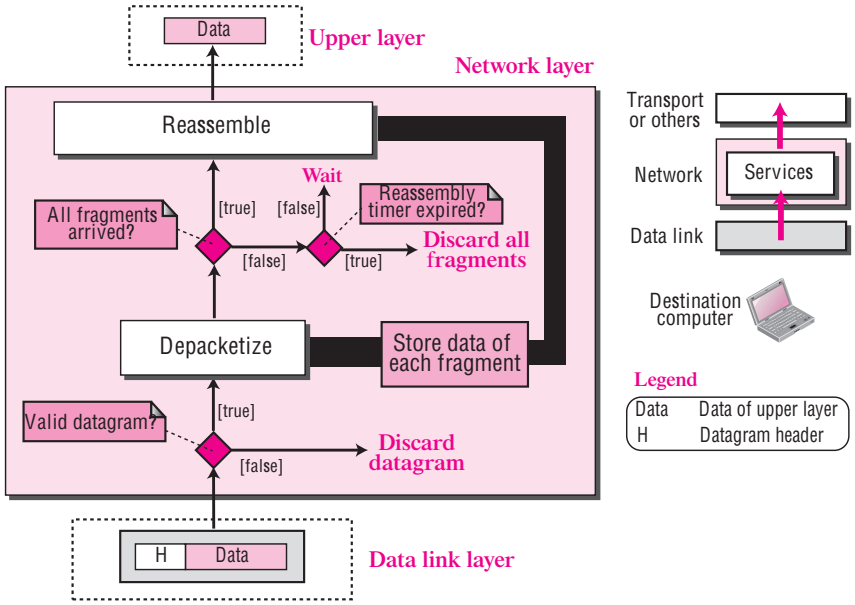



a source. Before applying these processes, however, the router needs to check the validity of the datagram using the checksum (see Chapter 7). Validation here means that the datagram header is not corrupted and the datagram is delivered to the correct router.

## Services Provided at the Destination Computer

The network layer at the destination computer is simpler. No forwarding is needed. However, the destination computer needs to assemble the fragments before delivering the data to the destination. After validating each datagram, the data is extracted from each fragment and stored. When all fragments have arrived, the data are reassembled and delivered to the upper layer. The network layer also sets a reassembly timer. If the timer is expired, all data fragments are destroyed and an error message is sent that all the fragmented datagram need to be resent. Figure 4.15 shows the process. Note that the process of fragmentation is transparent to the upper layer because the network layer does not deliver any piece of data to the upper layer until all pieces have arrived and assembled. Since a datagram may have been fragmented in the source computer as well as in any router (multilevel) fragmentation, the reassembly procedure is very delicate and complicated. We discuss the procedure in more detail in Chapter 7.

**Figure 4.15** *Processing at the destination computer*




## 4.5 OTHER NETWORK LAYER ISSUES

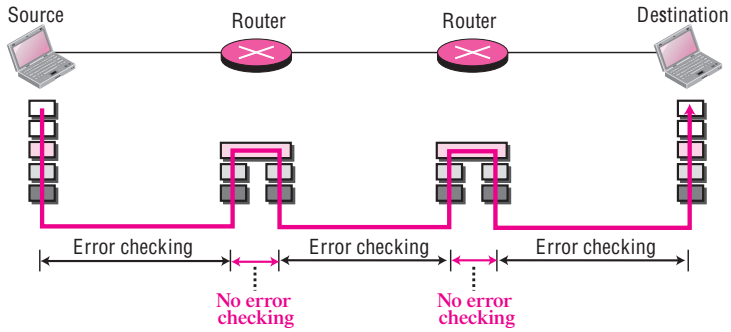
In this section we introduce some issues related to the network layer. These issues actually represent services that are normally discussed for the network layer, but they are either partially implemented at the network layer or not implemented at all. Some services are provided by some auxiliary protocols or by protocols added to the Internet later. Most of these issues resurface in future chapters.

### Error Control

**Error control** means including a mechanism for detecting corrupted, lost, or duplicate datagrams. Error control also includes a mechanism for correcting errors after they have been detected. The network layer in the Internet does not provide a real error control mechanism. At the surface level, it looks as though there is no need for error control at the network layer because each datagram passes through several networks before reaching its final destination. The data link layer that controls the behavior of these networks (LANs or WANs) use error control. In other words, if a hop-to-hop error control is already implemented at the data link layer, why do we need error control at the network layer? Although hop-to-hop error control may protect a datagram to some extent, it does not provide full protection. Figure 4.16 shows that there are some areas in the path of the datagram that some errors may occur, but never checked; the error control at the data link layer can miss any error that occurs when the datagram is being processed by the router.

The designers of the network layer wanted to make this layer operate simply and fast. They thought if there is a need for more rigorous error checking, it can be done at

**Figure 4.16** *Error checking at the data link layer*




the upper layer protocol that uses the service of the network layer. Another rationale for omitting the error checking at this layer can be related to fragmentation. Since the data is possibly fragmented at some routers and part of the network layer may be changed because of fragmentation, if we use error control, it must be checked at each router. This makes error checking at this layer very inefficient.

The designers of the network layer, however, have added a checksum field (see Chapter 7) to the datagram to control any corruption in the header, but not the whole datagram. This checksum may prevent any changes or corruptions in the header of the datagram between two hops and from end to end. For example, it prevents the delivery of the datagram to a wrong destination if the destination address has been corrupted. However, since the header may be changed in each router, the checksum needs to be calculated at the source and recalculated at each router.

We need to mention that although the network layer at the Internet does not directly provide error control, the Internet uses another protocol, ICMP, that provides some kind of error control if the datagram is discarded or has some unknown information in the header. We discuss ICMP in detail in Chapter 9.

## Flow Control

**Flow control** regulates the amount of data a source can send without overwhelming the receiver. If the upper layer at the source computer produces data faster than the upper layer at the destination computer can consume it, the receiver will be overwhelmed with data. To control the flow of data, the receiver needs to send some feedback to the sender to inform the latter it is overwhelmed with data.

The network layer in the Internet, however, does not directly provide any flow control. The datagrams are sent by the sender when they are ready without any attention to the readiness of the receiver.

**No flow control is provided for the current version of Internet network layer.**

Probably a few reasons for the lack of flow control in the design of the network layer can be mentioned. First, since there is no error control in this layer, the job of the

network layer at the receiver is so simple that it may rarely be overwhelmed. Second, the upper layers that use the service of the network layer can implement buffers to receive data from the network layer as soon as they are ready and does not have to consume the data as fast as received. Second, the flow control is provided for most of the upper layer protocols that use the services of the network layer, so another level of flow control makes the network layer more complicated and the whole system less proficient.

## Congestion Control

Another issue in a network layer protocol is **congestion control.** *Congestion* in the network layer is a situation in which too many datagrams are present in an area of the Internet. Congestion may occur if the number of datagrams sent by source computers are beyond the capacity of the network or routers. In this situation, some routers may drop some of the datagrams. However, as more datagrams are dropped, the situation may become worse because, due to the error control mechanism at the upper layers, the sender may send duplicates of the lost packets. If the congestion continues, sometimes a situation may reach a point that collapses the system and no datagram is delivered.

### *Congestion Control in a Connectionless Network*

There are several ways to control congestion in a connectionless network. One solution is referred to as signaling. In backward signaling a bit can set in the datagram moving in the direction opposite to the congested direction to inform the sender that congestion has occurred and the sender needs to slow down the sending of packets. In this case, the bit can be set in the response to a packet or in a packet that acknowledges the packet. If no feedback (acknowledgment) is used at the network layer, but the upper layer uses feedback, forward signaling can be used in which a bit is set in the packet traveling in the direction of the congestion to warn the receiver of the packet about congestion. The receiver then may inform the upper layer protocol, which in turn may inform the source. No forward or backward signaling is used in the Internet network layer.

Congestion in a connectionless network can also be implemented using a **choke packet,** a special packet that can be sent from a router to the sender when it encounters congestion. This mechanism, in fact, is implemented in the Internet network layer. The network layer uses an auxiliary protocol, ICMP, which we discuss in Chapter 9. When a router is congested, it can send an ICMP packet to the source to slow down.

Another way to ameliorate the congestion is to rank the packets by their *importance* in the whole message. A field can be used in the header of a packet to define the rank of a datagram as more important or less important, for example. If a router is congested and needs to drop some packets, the packets marked as less important can be dropped. For example, if a message represents an image, it may be divided into many packets. Some packets, the one in the corner, may be less important than the ones representing the middle part of the image. If the router is congested, it can drop these less important packets without tremendously changing the quality of the image. We talk about these issues in Chapter 25 when we discuss multimedia communication.

### *Congestion Control in a Connection-Oriented Network*

It is sometimes easier to control congestion in a connection-oriented network than in a connectionless network. One method simply creates an extra virtual circuit when there

is a congestion in an area. This, however, may create more problems for some routers. A better solution is *advanced negotiation* during the setup phase. The sender and the receiver may agree to a level of traffic when they setup the virtual circuit. The traffic level can be dictated by the routers that allow the establishment of the virtual circuits. In other words, a router can look at the exiting traffic and compare it with its maximum capacity, which allows a new virtual circuit to be created.

### Quality of Service

As the Internet has allowed new applications such as multimedia communication (in particular real-time communication of audio and video), the **quality of service (QoS)** of the communication has become more and more important. The Internet has thrived to provide better quality of service to support these applications. However, to keep the network layer untouched, these provisions are mostly implemented in the upper layer. Since QoS manifests itself more when we use multimedia communication, we discuss this issue in Chapter 25 when we discuss multimedia.

### Routing

A very important issue in the network layer is **routing;** how a router creates its routing table to help in forwarding a datagram in a connectionless service or helps in creating a virtual circuit, during setup phase, in a connection-oriented service. This can be done by *routing protocols*, that help hosts and routers make their routing table, maintain them, and update them. These are separate protocols that sometimes use the service of the network layer and sometimes the service of some transport layer protocols to help the network layer do its job. They can be grouped into two separate categories: unicast and multicast. We devote Chapter 11 to unicast routing and Chapter 12 to multicast routing. We need to assume that the routers already have created their routing protocols until we discuss these protocols in Chapters 11 and 12.

### Security

Another issue related to the communication at the network layer is security. Security was not a concern when the Internet was originally designed because it was used by a small number of users at the universities to do research activities; other people had no access to the Internet. The network layer was designed with no security provision. Today, however, security is a big concern. To provide security for a connectionless network layer, we need to have another virtual level that changes the connectionless service to a connection-oriented service. This virtual layer, called IPSec, is discussed in Chapter 30 after we discuss the general principles of cryptography and security in Chapter 29.

## 4.6 FURTHER READING

For more details about subjects discussed in this chapter, we recommend the following books: [Ste 94], [Tan 03], [Com 06], [Gar & Vid 04], and [Kur & Ros 08]. The items enclosed in brackets refer to the reference list at the end of the book.

## 4.7 KEY TERMS

choke packet
circuit switching
congestion control
connectionless service
connection-oriented service
data transfer phase
error control
flow control
packet switching
quality of service (QoS)
routing
setup phase
switching
teardown phase

## 4.8 SUMMARY

- At the conceptual level, we can think of the global Internet as a black box network. The Internet, however, is not one single network; it is made of many networks (or links) connected together through the connecting devices.
- In this Internet, a connecting device such as a router acts as a switch. Two types of switching are traditionally used in networking: circuit switching and packet switching.
- The network layer is designed as a packet-switched network. Packet-switched network can provide either a connectionless service or a connection-oriented service. When the network layer provides a connectionless service, each packet traveling in the Internet is an independent entity; there is no relationship between packets belonging to the same message. In a connection-oriented service, there is a virtual connection between all packets belonging to a message.
- In a connectionless service, the packets are forwarded to the next hop using the destination address in the packet. In a connection-oriented service, the packets are forwarded to the next hop using a label in the packet.
- In a connection-oriented network, communication occurs in three phases: setup, data transfer, and teardown. After connection setup, a virtual circuit is established between the sender and the receiver in which all packets belonging to the same message are sent through that circuit.
- We discussed existing services at the network layer in the Internet including addressing, services provided at the source computer, services provided at the destination computer, and services provided at each router.
- We also discussed some issues related to the network layer, services that are normally discussed for the network layer, but they are either partially implemented at the network layer or not implemented at all. Some of these services, such as routing and security are provided by other protocols in the Internet.

## 4.9 PRACTICE SET

### Exercises

1. Give some advantages and disadvantages of the connectionless service.
2. Give some advantages and disadvantages of the connection-oriented service.

3. If a label in a connection-oriented service is *n* bits, how many virtual circuits can be established at the same time?
4. Assume a destination computer receives messages from several computers. How can it be sure that the fragments from one source is not mixed with the fragments from another source.
5. Assume a destination computer receives several packets from a source. How can it be sure that the fragments belonging to a datagram are not mixed from the fragments belonging to another datagram.
6. Why do you think that the packets in Figure 4.7 need both addresses and labels?
7. Compare and contrast the delays in connectionless and connection-oriented services. Which service creates less delay if the message is large? Which service creates less delay if the message is small?
8. In Figure 4.13, why should the fragmentation be the last service?
9. Discuss why we need fragmentation at each router.
10. Discuss why we need to do reassembly at the final destination, not at each router.
11. In Figure 4.15, why do we need to set a timer and destroy all fragments if the timer expires? What criteria do you use in selecting the expiration duration of such a timer?

CHAPTER

# 5

# IPv4 Addresses

At the network layer, we need to uniquely identify each device on the Internet to allow global communication between all devices. In this chapter, we discuss the addressing mechanism related to the prevalent IPv4 protocol called IPv4 addressing. It is believed that IPv6 protocol will eventually supersede the current protocol, and we need to become aware of IPv6 addressing as well. We discuss IPv6 protocol and its addressing mechanism in Chapters 26 to 28.

## OBJECTIVES

*This chapter has several objectives:*

- To introduce the concept of an address space in general and the address space of IPv4 in particular.
- To discuss the classful architecture, classes in this model, and the blocks of addresses available in each class.
- To discuss the idea of hierarchical addressing and how it has been implemented in classful addressing.
- To explain subnetting and supernetting for classful architecture and show how they were used to overcome the deficiency of classful addressing.
- To discuss the new architecture, classless addressing, that has been devised to solve the problems in classful addressing such as address depletion.
- To show how some ideas borrowed from classful addressing such as subnetting can be easily implemented in classless addressing.
- To discuss some special blocks and some special addresses in each block.
- To discuss NAT technology and show how it can be used to alleviate the shortage in number of addresses in IPv4.

## 5.1 INTRODUCTION

The identifier used in the IP layer of the TCP/IP protocol suite to identify each device connected to the Internet is called the Internet address or **IP address.** An IPv4 address is a 32-bit address that *uniquely* and *universally* defines the connection of a host or a router to the Internet; an IP address is the address of the interface.

**An IPv4 address is 32 bits long.**

IPv4 addresses are *unique*. They are unique in the sense that each address defines one, and only one, connection to the Internet. Two devices on the Internet can never have the same address at the same time. However, if a device has two connections to the Internet, via two networks, it has two IPv4 addresses. The IPv4 addresses are *universal* in the sense that the addressing system must be accepted by any host that wants to be connected to the Internet.

**The IPv4 addresses are unique and universal.**

### Address Space

A protocol like IPv4 that defines addresses has an **address space.** An address space is the total number of addresses used by the protocol. If a protocol uses $b$ bits to define an address, the address space is $2^b$ because each bit can have two different values (0 or 1). IPv4 uses 32-bit addresses, which means that the address space is $2^{32}$ or 4,294,967,296 (more than four billion). Theoretically, if there were no restrictions, more than 4 billion devices could be connected to the Internet.

**The address space of IPv4 is $2^{32}$ or 4,294,967,296.**

### Notation

There are three common notations to show an IPv4 address: binary notation (base 2), dotted-decimal notation (base 256), and hexadecimal notation (base 16). The most prevalent, however, is base 256. These bases are defined in Appendix B. We also show how to convert a number from one base to another in that appendix. We recommend a review of this appendix before continuing with this chapter.

**Numbers in base 2, 16, and 256 are discussed in Appendix B.**

### *Binary Notation: Base 2*

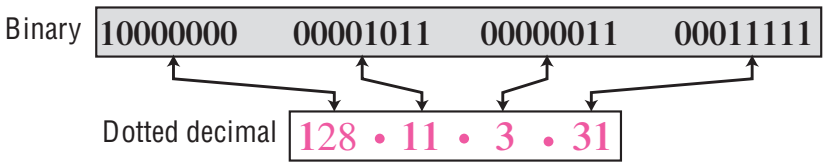
In **binary notation,** an IPv4 address is displayed as 32 bits. To make the address more readable, one or more spaces is usually inserted between each octet (8 bits). Each octet is often referred to as a byte. So it is common to hear an IPv4 address referred to as a 32-bit address, a 4-octet address, or a 4-byte address. The following is an example of an IPv4 address in binary notation:

```
01110101 10010101 00011101 11101010
```

### *Dotted-Decimal Notation: Base 256*

To make the IPv4 address more compact and easier to read, an IPv4 address is usually written in decimal form with a decimal point (dot) separating the bytes. This format is referred to as **dotted-decimal notation.** Figure 5.1 shows an IPv4 address in dotted-decimal notation. Note that because each byte (octet) is only 8 bits, each number in the dotted-decimal notation is between 0 and 255.

**Figure 5.1** *Dotted-decimal notation*




### Example 5.1

Change the following IPv4 addresses from binary notation to dotted-decimal notation.

a. **10000001 00001011 00001011 11101111**
b. **11000001 10000011 00011011 11111111**
c. **11100111 11011011 10001011 01101111**
d. **11111001 10011011 11111011 00001111**

**Solution**

We replace each group of 8 bits with its equivalent decimal number (see Appendix B) and add dots for separation:

a. 129.11.11.239
b. 193.131.27.255
c. 231.219.139.111
d. 249.155.251.15

### Example 5.2

Change the following IPv4 addresses from dotted-decimal notation to binary notation.

a. 111.56.45.78
b. 221.34.7.82

c. 241.8.56.12
d. 75.45.34.78

**Solution**

We replace each decimal number with its binary equivalent (see Appendix B):

a. `01101111 00111000 00101101 01001110`
b. `11011101 00100010 00000111 01010010`
c. `11110001 00001000 00111000 00001100`
d. `01001011 00101101 00100010 01001110`

### Example 5.3

Find the error, if any, in the following IPv4 addresses:

a. 111.56.045.78
b. 221.34.7.8.20
c. 75.45.301.14
d. 11100010.23.14.67

**Solution**

a. There should be no leading zeroes in dotted-decimal notation (045).
b. We may not have more than 4 bytes in an IPv4 address.
c. Each byte should be less than or equal to 255; 301 is outside this range.
d. A mixture of binary notation and dotted-decimal notation is not allowed.

### *Hexadecimal Notation: Base 16*

We sometimes see an IPv4 address in **hexadecimal notation.** Each hexadecimal digit is equivalent to four bits. This means that a 32-bit address has 8 hexadecimal digits. This notation is often used in network programming.

### Example 5.4

Change the following IPv4 addresses from binary notation to hexadecimal notation.

a. `10000001 00001011 00001011 11101111`
b. `11000001 10000011 00011011 11111111`

**Solution**

We replace each group of 4 bits with its hexadecimal equivalent (see Appendix B). Note that hexadecimal notation normally has no added spaces or dots; however, 0X (or 0x) is added at the beginning or the subscript 16 at the end to show that the number is in hexadecimal.

a. 0X810B0BEF or $810B0BEF_{16}$
b. 0XC1831BFF or $C1831BFF_{16}$

## Range of Addresses

We often need to deal with a range of addresses instead of one single address. We sometimes need to find the number of addresses in a range if the first and last address is given. Other times, we need to find the last address if the first address and the number of addresses in the range are given. In this case, we can perform subtraction or addition

operations in the corresponding base (2, 256, or 16). Alternatively, we can covert the addresses to decimal values (base 10) and perform operations in this base.

### Example 5.5

Find the number of addresses in a range if the first address is 146.102.29.0 and the last address is 146.102.32.255.

**Solution**

We can subtract the first address from the last address in base 256 (see Appendix B). The result is 0.0.3.255 in this base. To find the number of addresses in the range (in decimal), we convert this number to base 10 and add 1 to the result.

$$\text{Number of addresses} = (0 \times 256^3 + 0 \times 256^2 + 3 \times 256^1 + 255 \times 256^0) + 1 = 1024$$

### Example 5.6

The first address in a range of addresses is 14.11.45.96. If the number of addresses in the range is 32, what is the last address?

**Solution**

We convert the number of addresses minus 1 to base 256, which is 0.0.0.31. We then add it to the first address to get the last address. Addition is in base 256.

$$\text{Last address} = (14.11.45.96 + 0.0.0.31)_{256} = 14.11.45.127$$

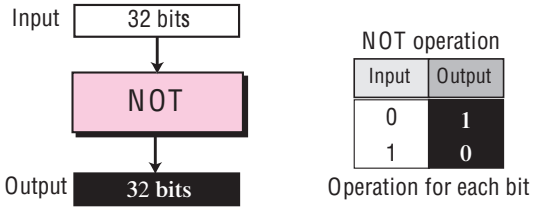
## Operations

We often need to apply some operations on 32-bit numbers in binary or dotted-decimal notation. These numbers either represent IPv4 addresses or some entities related to IPv4 addresses (such as a *mask*, which is discussed later). In this section, we introduce three operations that are used later in the chapter: NOT, AND, and OR.

### *Bitwise NOT Operation*

The bitwise NOT operation is a unary operation; it takes one input. When we apply the NOT operation on a number, it is often said that the number is complemented. The NOT operation, when applied to a 32-bit number in binary format, inverts each bit. Every 0 bit is changed to a 1 bit; every 1 bit is changed to a 0 bit. Figure 5.2 shows the NOT operation.

**Figure 5.2** *Bitwise NOT operation*



Although we can directly use the NOT operation on a 32-bit number, when the number is represented as a four-byte dotted-decimal notation, we can use a short cut; we can subtract each byte from 255.

### Example 5.7

The following shows how we can apply the NOT operation on a 32-bit number in binary.

| | | | | |
|---|---|---|---|---|
| Original number: | 00010001 | 01111001 | 00001110 | 00100011 |
| Complement: | 11101110 | 10000110 | 11110001 | 11011100 |

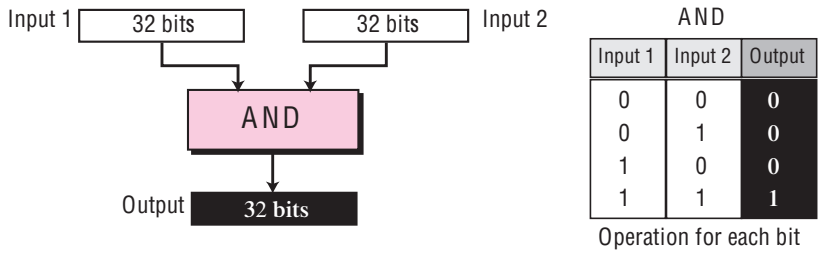
We can use the same operation using the dotted-decimal representation and the short cut.

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| Original number: | 17 | . | 121 | . | 14 | . | 35 |
| Complement: | 238 | . | 134 | . | 241 | . | 220 |

### *Bitwise AND Operation*

The bitwise AND operation is a binary operation; it takes two inputs. The AND operation compares the two corresponding bits in two inputs and selects the smaller bit from the two (or select one of them if the bits are equal). Figure 5.3 shows the AND operation.

**Figure 5.3** *Bitwise AND operation*




Although we can directly use the AND operation on the 32-bit binary representation of two numbers, when the numbers are represented in dotted-decimal notation, we can use two short cuts.

1. When at least one of the numbers is 0 or 255, the AND operation selects the smaller byte (or one of them if equal).
2. When none of the two bytes is either 0 or 255, we can write each byte as the sum of eight terms, where each term is a power of 2. We then select the smaller term in each pair (or one of them if equal) and add them to get the result.

### Example 5.8

The following shows how we can apply the AND operation on two 32-bit numbers in binary.

| | | | | |
|---|---|---|---|---|
| First number: | 00010001 | 01111001 | 00001110 | 00100011 |
| Second number: | 11111111 | 11111111 | 10001100 | 00000000 |
| Result | 00010001 | 01111001 | 00001100 | 00000000 |

We can use the same operation using the dotted-decimal representation and the short cut.

| | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| First number: | 17 | . | 121 | . | 14 | . | 35 |
| Second number: | 255 | . | 255 | . | 140 | . | 0 |
| Result: | 17 | . | 121 | . | 12 | . | 0 |

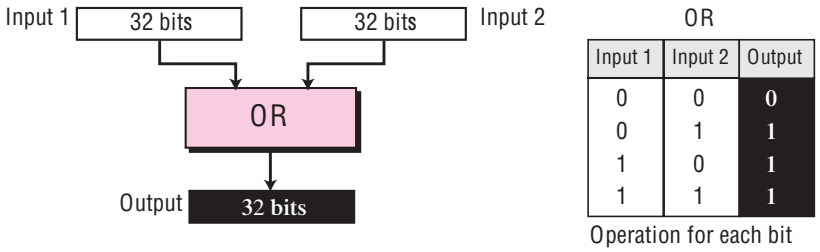
We have applied the first short cut on the first, second, and the fourth byte; we have applied the second short cut on the third byte. We have written 14 and 140 as the sum of terms and selected the smaller term in each pair as shown below.

| Powers | $2^7$ | | $2^6$ | | $2^5$ | | $2^4$ | | $2^3$ | | $2^2$ | | $2^1$ | | $2^0$ |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Byte (14) | 0 | + | 0 | + | 0 | + | 0 | + | 8 | + | 4 | + | 2 | + | 0 |
| Byte (140) | 128 | + | 0 | + | 0 | + | 0 | + | 8 | + | 4 | + | 0 | + | 0 |
| Result (12) | 0 | + | 0 | + | 0 | + | 0 | + | 8 | + | 4 | + | 0 | + | 0 |

### *Bitwise OR Operation*

The bitwise OR operation is a binary operation; it takes two inputs. The OR operation compares the corresponding bits in the two numbers and selects the larger bit from the two (or one of them if equal). Figure 5.4 shows the OR operation.

**Figure 5.4** *Bitwise OR operation*




Although we can directly use the OR operation on the 32-bit binary representation of the two numbers, when the numbers are represented in dotted-decimal notation, we can use two short cuts.

1. When at least one of the two bytes is 0 or 255, the OR operation selects the larger byte (or one of them if equal).
2. When none of the two bytes is 0 or 255, we can write each byte as the sum of eight terms, where each term is a power of 2. We then select the larger term in each pair (or one of them if equal) and add them to get the result of OR operation.

### Example 5.9

The following shows how we can apply the OR operation on two 32-bit numbers in binary.

| | | | | |
|---|---|---|---|---|
| First number: | 00010001 | 01111001 | 00001110 | 00100011 |
| Second number: | 11111111 | 11111111 | 10001100 | 00000000 |
| Result | 11111111 | 11111111 | 10001110 | 00100011 |

We can use the same operation using the dotted-decimal representation and the short cut.

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| First number: | 17 | . | 121 | . | 14 | . | 35 |
| Second number: | 255 | . | 255 | . | 140 | . | 0 |
| Result: | 255 | . | 255 | . | 142 | . | 35 |

We have used the first short cut for the first and second bytes and the second short cut for the third byte.

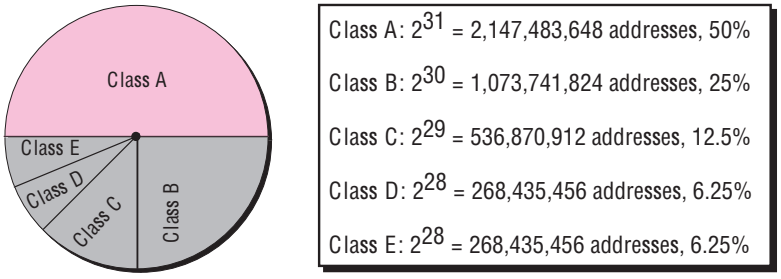
## 5.2 CLASSFUL ADDRESSING

IP addresses, when started a few decades ago, used the concept of *classes*. This architecture is called **classful addressing.** In the mid-1990s, a new architecture, called **classless addressing,** was introduced that supersedes the original architecture. In this section, we introduce classful addressing because it paves the way for understanding classless addressing and justifies the rationale for moving to the new architecture. Classless addressing is discussed in the next section.

### Classes

In classful addressing, the IP address space is divided into five **classes: A, B, C, D,** and **E.** Each class occupies some part of the whole address space. Figure 5.5 shows the class occupation of the address space.

**Figure 5.5** *Occupation of the address space*

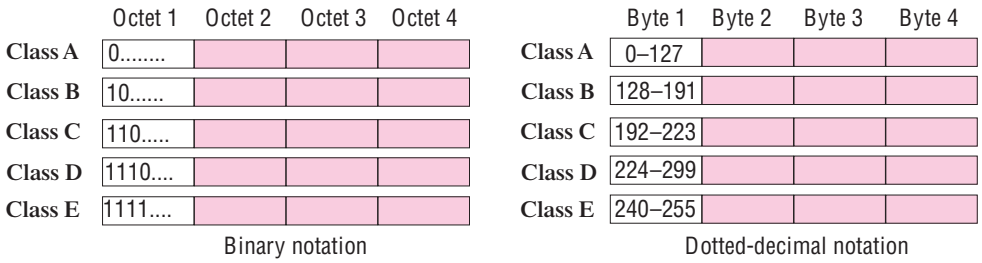



**In classful addressing, the address space is divided into five classes: A, B, C, D, and E.**

### Recognizing Classes

We can find the class of an address when the address is given either in binary or dotted-decimal notation. In the binary notation, the first few bits can immediately tell us the class of the address; in the dotted-decimal notation, the value of the first byte can give the class of an address (Figure 5.6).

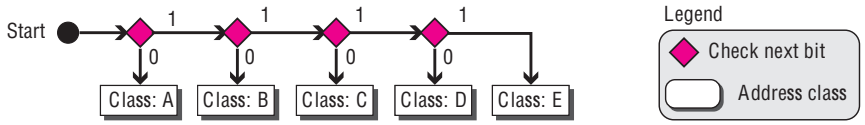
**Figure 5.6** *Finding the class of an address*




Note that some special addresses fall in class A or E. We emphasize that these special addresses are exceptions to the classification; they are discussed later in the chapter.

Computers often store IPv4 addresses in binary notation. In this case, it is very convenient to write an algorithm to use a continuous checking process for finding the address as shown in Figure 5.7.

**Figure 5.7** *Finding the address class using continuous checking*




### Example 5.10

Find the class of each address:

a. **00000001 00001011 00001011 11101111**
b. **11000001 10000011 00011011 11111111**
c. **10100111 11011011 10001011 01101111**
d. **11110011 10011011 11111011 00001111**

**Solution**

See the procedure in Figure 5.7.

a. The first bit is 0. This is a class A address.
b. The first 2 bits are 1; the third bit is 0. This is a class C address.
c. The first bit is 1; the second bit is 0. This is a class B address.
d. The first 4 bits are 1s. This is a class E address.

### Example 5.11

Find the class of each address:

a. 227.12.14.87
b. 193.14.56.22
c. 14.23.120.8
d. 252.5.15.111

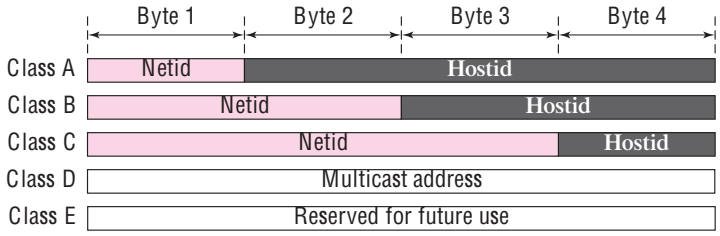
**Solution**

a. The first byte is 227 (between 224 and 239); the class is D.
b. The first byte is 193 (between 192 and 223); the class is C.
c. The first byte is 14 (between 0 and 127); the class is A.
d. The first byte is 252 (between 240 and 255); the class is E.

### *Netid and Hostid*

In classful addressing, an IP address in classes A, B, and C is divided into **netid** and **hostid.** These parts are of varying lengths, depending on the class of the address. Figure 5.8 shows the netid and hostid bytes. Note that classes D and E are not divided into netid and hostid, for reasons that we will discuss later.

**Figure 5.8** *Netid and hostid*




In class A, 1 byte defines the netid and 3 bytes define the hostid. In class B, 2 bytes define the netid and 2 bytes define the hostid. In class C, 3 bytes define the netid and 1 byte defines the hostid.

## Classes and Blocks

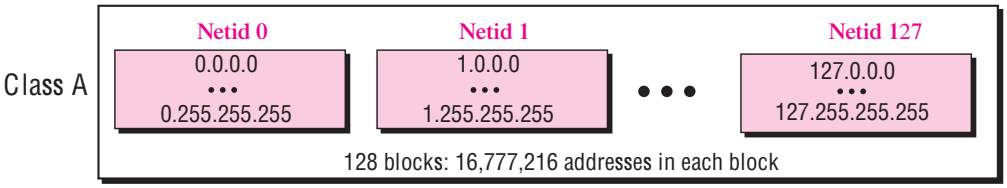
One problem with classful addressing is that each class is divided into a fixed number of blocks with each block having a fixed size. Let us look at each class.

### *Class A*

Since only 1 byte in class A defines the netid and the leftmost bit should be 0, the next 7 bits can be changed to find the number of blocks in this class. Therefore, class A is divided into $2^7 = 128$ blocks that can be assigned to 128 organizations (the number is less because some blocks were reserved as special blocks). However, each block in this class contains 16,777,216 addresses, which means the organization should be a really

large one to use all these addresses. Many addresses are wasted in this class. Figure 5.9 shows the block in class A.

**Figure 5.9** *Blocks in class A*

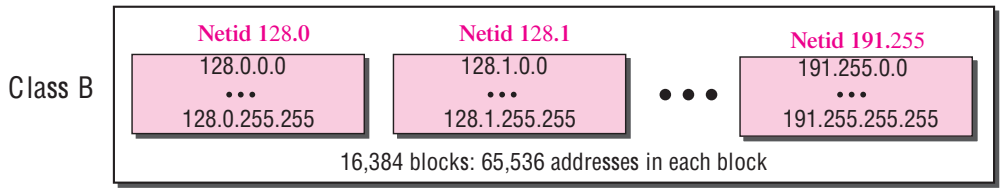



**Millions of class A addresses are wasted.**

### *Class B*

Since 2 bytes in class B define the class and the two leftmost bit should be 10 (fixed), the next 14 bits can be changed to find the number of blocks in this class. Therefore, class B is divided into $2^{14} = 16{,}384$ blocks that can be assigned to 16,384 organizations (the number is less because some blocks were reserved as special blocks). However, each block in this class contains 65,536 addresses. Not so many organizations can use so many addresses. Many addresses are wasted in this class. Figure 5.10 shows the blocks in class B.

**Figure 5.10** *Blocks in class B*

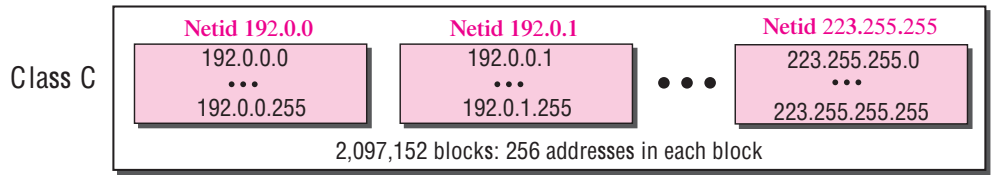



**Many class B addresses are wasted.**

### *Class C*

Since 3 bytes in class C define the class and the three leftmost bits should be 110 (fixed), the next 21 bits can be changed to find the number of blocks in this class. Therefore, class C is divided into $2^{21} = 2{,}097{,}152$ blocks, in which each block contains 256 addresses, that can be assigned to 2,097,152 organizations (the number is less because some blocks were reserved as special blocks). Each block contains 256 addresses. However, not so many organizations were so small as to be satisfied with a class C block. Figure 5.11 shows the blocks in class C.

**Figure 5.11** *Blocks in class C*

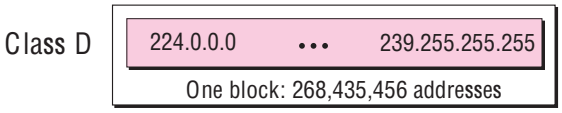



**Not so many organizations are so small to have a class C block.**

### *Class D*

There is just one block of class D addresses. It is designed for multicasting, as we will see in a later section. Each address in this class is used to define one group of hosts on the Internet. When a group is assigned an address in this class, every host that is a member of this group will have a multicast address in addition to its normal (unicast) address. Figure 5.12 shows the block.

**Figure 5.12** *The single block in Class D*

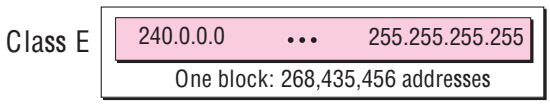



**Class D addresses are made of one block, used for multicasting.**

### *Class E*

There is just one block of class E addresses. It was designed for use as reserved addresses, as shown in Figure 5.13.

**Figure 5.13** *The single block in Class E*




**The only block of class E addresses was reserved for future purposes.**

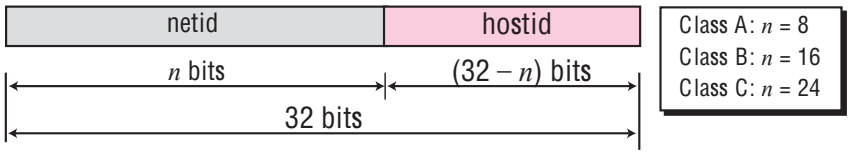
## Two-Level Addressing

The whole purpose of IPv4 addressing is to define a destination for an Internet packet (at the network layer). When classful addressing was designed, it was assumed that the whole Internet is divided into many networks and each network connects many hosts. In other words, the Internet was seen as a network of networks. A network was normally created by an organization that wanted to be connected to the Internet. The Internet authorities allocated a block of addresses to the organization (in class A, B, or C).

**The range of addresses allocated to an organization in classful addressing was a block of addresses in Class A, B, or C.**

Since all addresses in a network belonged to a single block, each address in classful addressing contains two parts: netid and hostid. The netid defines the network; the hostid defines a particular host connected to that network. Figure 5.14 shows an IPv4 address in classful addressing. If $n$ bits in the class defines the net, then $32 - n$ bits defines the host. However, the value of $n$ depends on the class the block belongs to. The value of $n$ can be 8, 16 or 24 corresponding to classes A, B, and C respectively.

**Figure 5.14** *Two-level addressing in classful addressing*




### Example 5.12

Two-level addressing can be found in other communication systems. For example, a telephone system inside the United States can be thought of as two parts: area code and local part. The area code defines the area, the local part defines a particular telephone subscriber in that area.

(626) 3581301

The area code, 626, can be compared with the netid, the local part, 3581301, can be compared to the hostid.

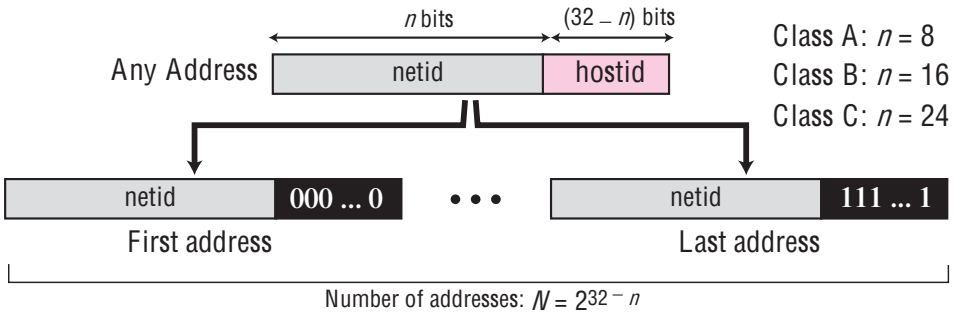
### *Extracting Information in a Block*

A block is a range of addresses. Given any address in the block, we normally like to know three pieces of information about the block: the number of addresses, the first address, and the last address. Before we can extract these pieces of information, we need to know the class of the address, which we showed how to find in the previous section. After the class of the block is found, we know the value of $n$, the length of netid in bits. We can now find these three pieces of information as shown in Figure 5.15.

1. The number of addresses in the block, $N$, can be found using $N = 2^{32-n}$.

2. To find the first address, we keep the *n* leftmost bits and set the (32 – *n*) rightmost bits all to 0s.
3. To find the last address, we keep the *n* leftmost bits and set the (32 – *n*) rightmost bits all to 1s.

**Figure 5.15** *Information extraction in classful addressing*



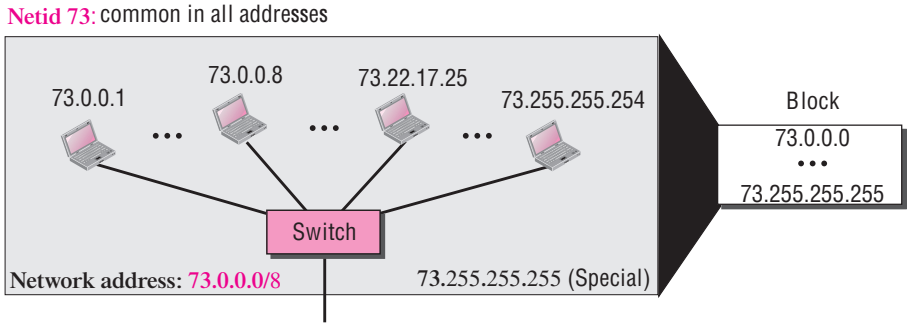

## Example 5.13

An address in a block is given as 73.22.17.25. Find the number of addresses in the block, the first address, and the last address.

### Solution

Since 73 is between 0 and 127, the class of the address is A. The value of *n* for class A is 8. Figure 5.16 shows a possible configuration of the network that uses this block. Note that we show the value of *n* in the network address after a slash. Although this was not a common practice in classful addressing, it helps to make it a habit in classless addressing in the next section.

**Figure 5.16** *Solution to Example 5.13*




1. The number of addresses in this block is $N = 2^{32-n} = 2^{24} = 16{,}777{,}216$.
2. To find the first address, we keep the leftmost 8 bits and set the rightmost 24 bits all to 0s. The first address is 73.0.0.0/8 in which 8 is the value of *n*. The first address is called the *network address* and is not assigned to any host. It is used to define the network.

3. To find the last address, we keep the leftmost 8 bits and set the rightmost 24 bits all to 1s. The last address is 73.255.255.255. The last address is normally used for a special purpose, as discussed later in the chapter.

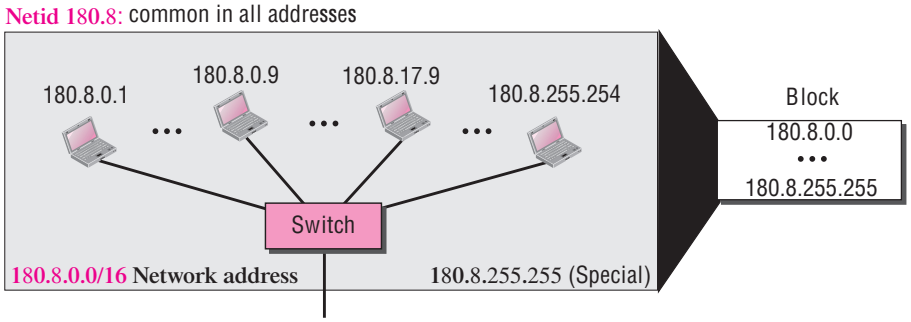
### Example 5.14

An address in a block is given as 180.8.17.9. Find the number of addresses in the block, the first address, and the last address.

#### Solution

Since 180 is between 128 and 191, the class of the address is B. The value of *n* for class B is 16. Figure 5.17 shows a possible configuration of the network that uses this block.

**Figure 5.17** *Solution to Example 5.14*




1. The number of addresses in this block is $N = 2^{32-n} = 2^{16} = 65{,}536$.
2. To find the first address, we keep the leftmost 16 bits and set the rightmost 16 bits all to 0s. The first address (network address) is 18.8.0.0/16, in which 16 is the value of *n*.
3. To find the last address, we keep the leftmost 16 bits and set the rightmost 16 bits all to 1s. The last address is 18.8.255.255.

### Example 5.15

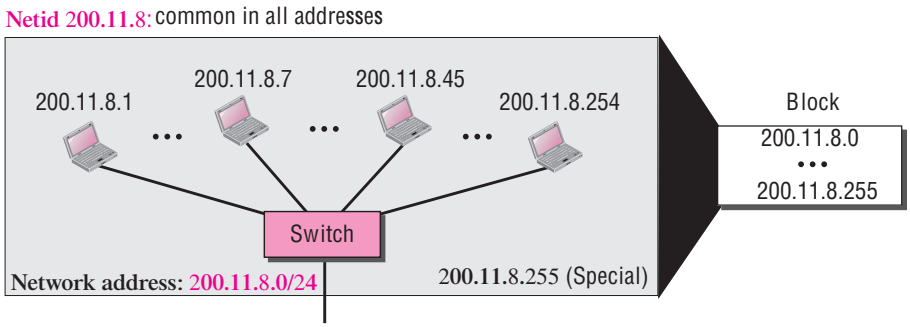
An address in a block is given as 200.11.8.45. Find the number of addresses in the block, the first address, and the last address.

#### Solution

Since 200 is between 192 and 223, the class of the address is C. The value of *n* for class C is 24. Figure 5.18 shows a possible configuration of the network that uses this block.

1. The number of addresses in this block is $N = 2^{32-n} = 2^{8} = 256$.
2. To find the first address, we keep the leftmost 24 bits and set the rightmost 8 bits all to 0s. The first address is 200.11.8.0/24. The first address is called the network address.
3. To find the last address, we keep the leftmost 24 bits and set the rightmost 8 bits all to 1s. The last address is 200.11.8.255.

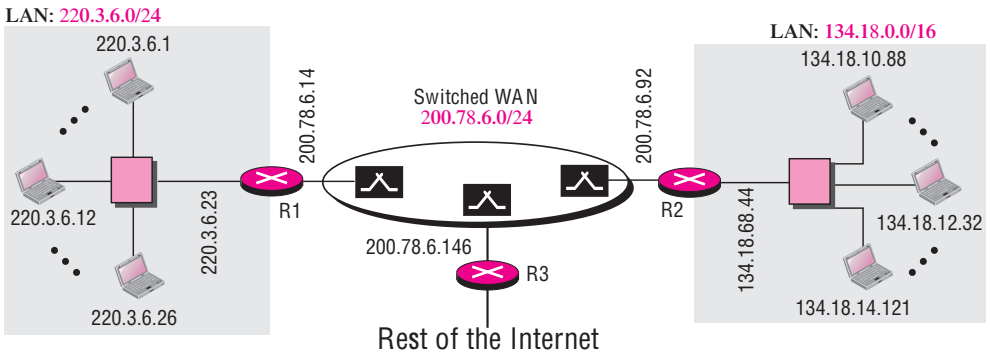
**Figure 5.18** *Solution to Example 5.15*




## An Example

Figure 5.19 shows a hypothetical part of an internet with three networks.

**Figure 5.19** *Sample internet*




We have

1. A LAN with the network address 220.3.6.0 (class C).
2. A LAN with the network address 134.18.0.0 (class B).
3. A switched WAN (class C), such as Frame Relay or ATM, that can be connected to many routers. We have shown three. One router connects the WAN to the left LAN, one connects the WAN to the right LAN, and one connects the WAN to the rest of the internet.

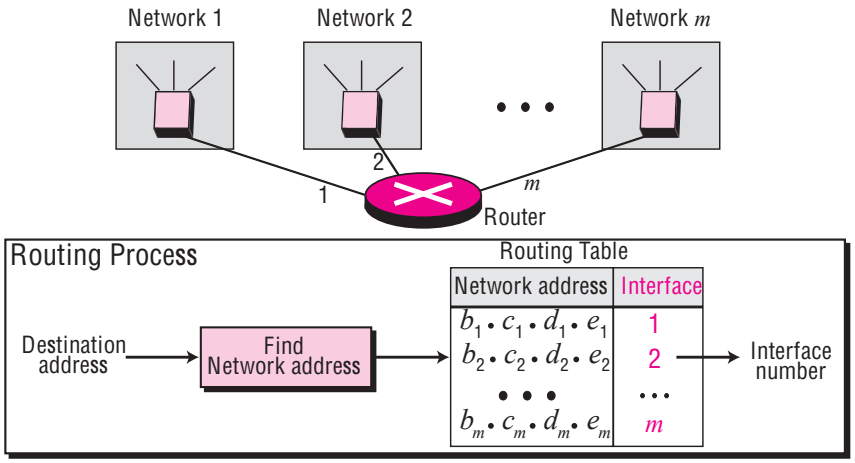
### *Network Address*

The above three examples show that, given any address, we can find all information about the block. The first address, **network address,** is particularly important because it is used in routing a packet to its destination network. For the moment, let us assume that an internet is made of *m* networks and a router with *m* interfaces. When a packet arrives at the router from any source host, the router needs to know to which network the packet

should be sent; the router needs to know from which interface the packet should be sent out. When the packet arrives at the network, it reaches its destination host using another strategy that we discuss in later chapters. Figure 5.20 shows the idea. After the network address has been found, the router consults its routing table to find the corresponding interface from which the packet should be sent out. The network address is actually the identifier of the network; each network is identified by its network address.

**The network address is the identifier of a network.**

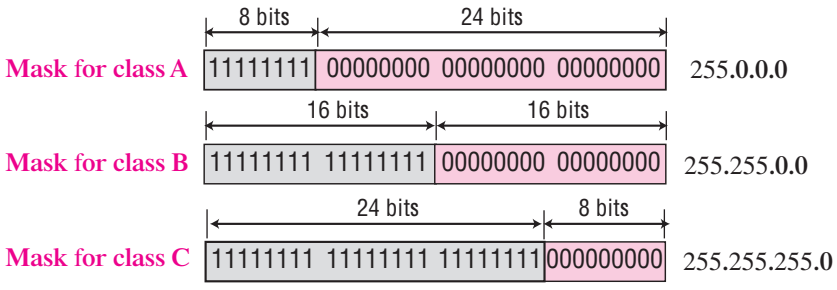
**Figure 5.20** *Network address*




### *Network Mask*

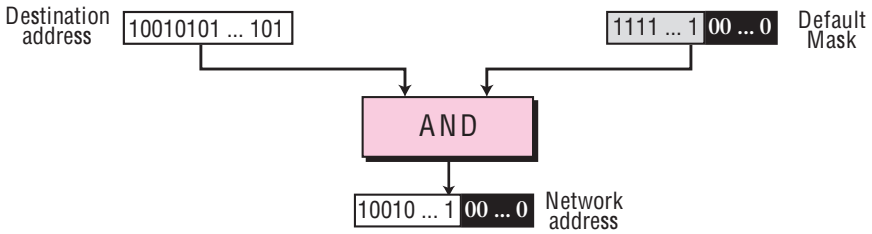
The methods we described previously for extracting the network address are mostly used to show the concept. The routers in the Internet normally use an algorithm to extract the network address from the destination address of a packet. To do this, we need a network mask. A **network mask** or a **default mask** in classful addressing is a 32-bit number with $n$ leftmost bits all set to 1s and (32 – $n$) rightmost bits all set to 0s. Since $n$ is different for each class in classful addressing, we have three default masks in classful addressing as shown in Figure 5.21.

**Figure 5.21** *Network mask*



To extract the network address from the destination address of a packet, a router uses the AND operation described in the previous section. When the destination address (or any address in the block) is ANDed with the default mask, the result is the network address (Figure 5.22). The router applies the AND operation on the binary (or hexadecimal representation) of the address and the mask, but when we show an example, we use the short cut discussed before and apply the mask on the dotted-decimal notation. The default mask can also be used to find the number of addresses in the block and the last address in the block, but we discuss these applications in classless addressing.

**Figure 5.22** *Finding a network address using the default mask*




### Example 5.16

A router receives a packet with the destination address 201.24.67.32. Show how the router finds the network address of the packet.

**Solution**

We assume that the router first finds the class of the address and then uses the corresponding default mask on the destination address, but we need to know that a router uses another strategy as we will discuss in the next chapter. Since the class of the address is B, we assume that the router applies the default mask for class B, 255.255.0.0 to find the network address.

| | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| Destination address | → | 201 | . | 24 | . | 67 | . | 32 |
| Default mask | → | 255 | . | 255 | . | 0 | . | 0 |
| Network address | → | 201 | . | 24 | . | 0 | . | 0 |

We have used the first short cut as described in the previous section. The network address is 201.24.0.0 as expected.

## Three-Level Addressing: Subnetting

As we discussed before, the IP addresses were originally designed with two levels of addressing. To reach a host on the Internet, we must first reach the network and then the host. It soon became clear that we need more than two hierarchical levels, for two reasons. First, an organization that was granted a block in class A or B needed to divide its large network into several subnetworks for better security and management. Second, since the blocks in class A and B were almost depleted and the blocks in class C were smaller than the needs of most organizations, an organization that has been granted a block in class A or B could divide the block into smaller subblocks and share them with

other organizations. The idea of splitting a block to smaller blocks is referred to as subnetting. In **subnetting,** a network is divided into several smaller subnetworks (subnets) with each subnetwork having its own subnetwork address.

### Example 5.17

Three-level addressing can be found in the telephone system if we think about the local part of a telephone number as an exchange and a subscriber connection:

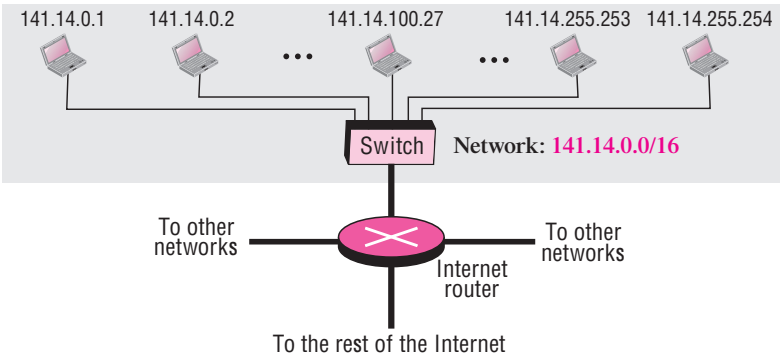
(626) 358 - 1301

in which 626 is the area code, 358 is the exchange, and 1301 is the subscriber connection.

### Example 5.18

Figure 5.23 shows a network using class B addresses before subnetting. We have just one network with almost $2^{16}$ hosts. The whole network is connected, through one single connection, to one of the routers in the Internet. Note that we have shown /16 to show the length of the netid (class B).

**Figure 5.23** *Example 5.18*




### Example 5.19

Figure 5.24 shows the same network in Figure 5.23 after subnetting. The whole network is still connected to the Internet through the same router. However, the network has used a private router to divide the network into four subnetworks. The rest of the Internet still sees only one network; internally the network is made of four subnetworks. Each subnetwork can now have almost $2^{14}$ hosts. The network can belong to a university campus with four different schools (buildings). After subnetting, each school has its own subnetworks, but still the whole campus is one network for the rest of the Internet. Note that /16 and /18 show the length of the netid and subnetids.

### *Subnet Mask*

We discussed the network mask (default mask) before. The network mask is used when a network is not subnetted. When we divide a network to several subnetworks, we need to create a subnetwork mask (or subnet mask) for each subnetwork. A subnetwork has subnetid and hostid as shown in Figure 5.25.

**Figure 5.24** *Example 5.19*

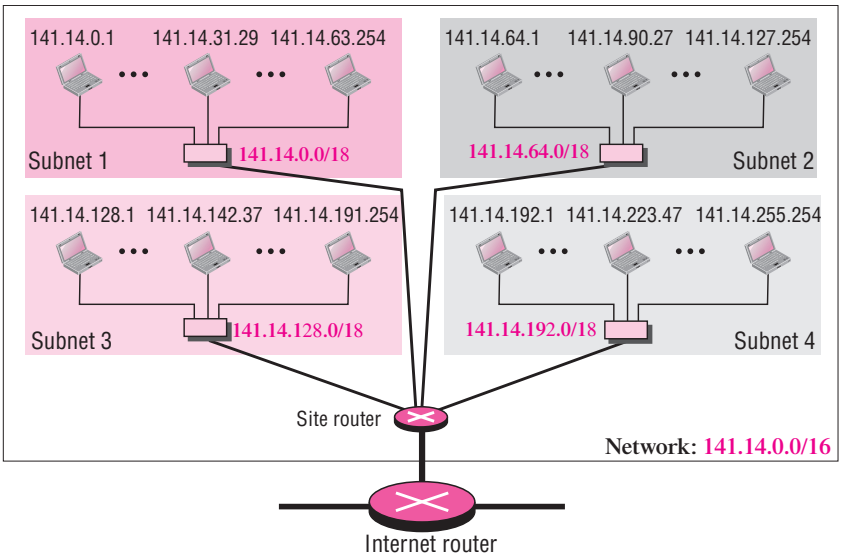
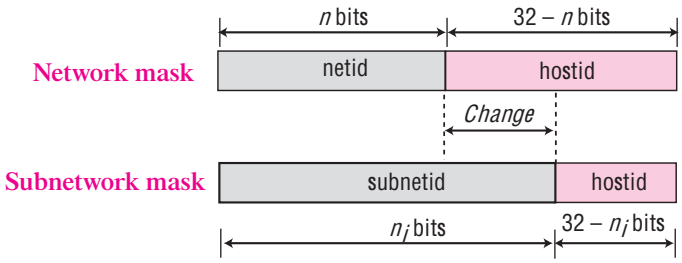



**Figure 5.25** *Network mask and subnetwork mask*




Subnetting increases the length of the netid and decreases the length of hostid. When we divide a network to $s$ number of subnetworks, each of equal numbers of hosts, we can calculate the subnetid for each subnetwork as

$$n_{sub} = n + \log_2 s$$

in which $n$ is the length of netid, $n_{sub}$ is the length of each subnetid, and $s$ is the number of subnets which must be a power of 2.

### Example 5.20

In Example 5.19, we divided a class B network into four subnetworks. The value of $n = 16$ and the value of $n_1 = n_2 = n_3 = n_4 = 16 + \log_2 4 = 18$. This means that the subnet mask has eighteen 1s and fourteen 0s. In other words, the subnet mask is 255.255.192.0 which is different from the network mask for class B (255.255.0.0).

### *Subnet Address*

When a network is subnetted, the first address in the subnet is the identifier of the subnet and is used by the router to route the packets destined for that subnetwork. Given any address in the subnet, the router can find the subnet mask using the same procedure we discussed to find the network mask: ANDing the given address with the subnet mask. The short cuts we discussed in the previous section can be used to find the subnet address.

### Example 5.21

In Example 5.19, we show that a network is divided into four subnets. Since one of the addresses in subnet 2 is 141.14.120.77, we can find the subnet address as:

| | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Address | → | 141 | . | 14 | . | 120 | . | 77 |
| Mask | → | 255 | . | 255 | . | 192 | . | 0 |
| Subnet Address | → | 141 | . | 14 | . | 64 | . | 0 |

The values of the first, second, and fourth bytes are calculated using the first short cut for AND operation. The value of the third byte is calculated using the second short cut for the AND operation.

| | | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Address (120) | 0 | + | 64 | + | 32 | + | 16 | + | 8 | + | 0 | + | 0 | + | 0 |
| Mask (192) | 128 | + | 64 | + | 0 | + | 0 | + | 0 | + | 0 | + | 0 | + | 0 |
| Result (64) | 0 | + | 64 | + | 0 | + | 0 | + | 0 | + | 0 | + | 0 | + | 0 |

### *Designing Subnets*

We show how to design a subnet when we discuss classless addressing. Since classful addressing is a special case of classless addressing, what is discussed later can also be applied to classful addressing.

## Supernetting

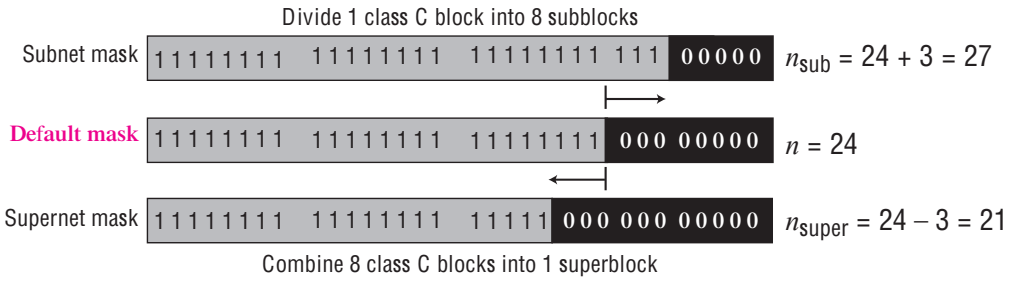
Subnetting could not completely solve address depletion problems in classful addressing because most organizations did not want to share their granted blocks with others. Since class C blocks were still available but the size of the block did not meet the requirement of new organizations that wanted to join the Internet, one solution was **supernetting.** In supernetting, an organization can combine several class C blocks to create a larger range of addresses. In other words, several networks are combined to create a supernetwork. By doing this, an organization can apply for several class C blocks instead of just one. For example, an organization that needs 1000 addresses can be granted four class C blocks.

### *Supernet Mask*

A **supernet mask** is the reverse of a subnet mask. A subnet mask for class C has more 1s than the default mask for this class. A supernet mask for class C has less 1s than the default mask for this class.

Figure 5.26 shows the difference between a subnet mask and a supernet mask. A subnet mask that divides a block into eight subblocks has three more 1s ($2^3 = 8$) than the default mask; a supernet mask that combines eight blocks into one superblock has three less 1s than the default mask.

**Figure 5.26** *Comparison of subnet, default, and supernet masks*




In supernetting, the number of class C addresses that can be combined to make a supernet needs to be a power of 2. The length of the supernetid can be found using the formula

$$n_{super} = n - \log_2 c$$

in which $n_{super}$ defines the length of the supernetid in bits and $c$ defines the number of class C blocks that are combined.

Unfortunately, supernetting provided two new problems: First, the number of blocks to combine needs to be a power of 2, which means an organization that needed seven blocks should be granted at least eight blocks (address wasting). Second, supernetting and subnetting really complicated the routing of packets in the Internet.

## 5.3 CLASSLESS ADDRESSING

Subnetting and supernetting in classful addressing did not really solve the address depletion problem and made the distribution of addresses and the routing process more difficult. With the growth of the Internet, it was clear that a larger address space was needed as a long-term solution. The larger address space, however, requires that the length of IP addresses to be increased, which means the format of the IP packets needs to be changed. Although the long-range solution has already been devised and is called IPv6 (see Chapters 26 to 28), a short-term solution was also devised to use the same address space but to change the distribution of addresses to provide a fair share to each organization. The short-term solution still uses IPv4 addresses, but it is called *classless* addressing. In other words, the class privilege was removed from the distribution to compensate for the address depletion.

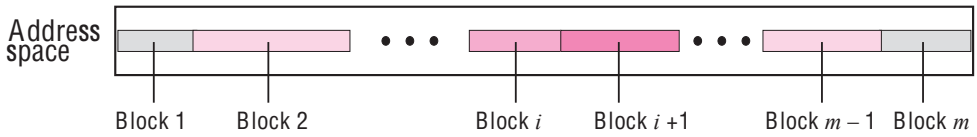
There was another motivation for classless addressing. During the 1990s, Internet service providers (ISPs) came into prominence. An ISP is an organization that provides Internet access for individuals, small businesses, and midsize organizations that do not want to create an Internet site and become involved in providing Internet services (such as e-mail services) for their employees. An ISP can provide these services. An ISP is granted a large range of addresses and then subdivides the addresses (in groups of 1, 2, 4, 8, 16, and so on), giving a range of addresses to a household or a small business. The customers are connected via a dial-up modem, DSL, or cable modem to the ISP. However, each customer needs some IPv4 addresses.

In 1996, the Internet authorities announced a new architecture called classless addressing. In classless addressing, variable-length blocks are used that belong to no classes. We can have a block of 1 address, 2 addresses, 4 addresses, 128 addresses, and so on.

## Variable-Length Blocks

In classful addressing the whole address space was divided into five classes. Although each organization was granted one block in class A, B, or C, the size of the blocks was predefined; the organization needed to choose one of the three block sizes. The only block in class D and the only block in class E were reserved for a special purpose. In classless addressing, the whole address space is divided into variable length blocks. Theoretically, we can have a block of $2^0, 2^1, 2^2, \ldots, 2^{32}$ addresses. The only restriction, as we discuss later, is that the number of addresses in a block needs to be a power of 2. An organization can be granted one block of addresses. Figure 5.27 shows the division of the whole address space into nonoverlapping blocks.

**Figure 5.27** *Variable-length blocks in classless addressing*

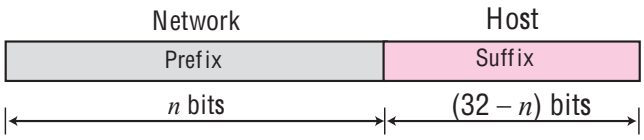



## Two-Level Addressing

In classful addressing, two-level addressing was provided by dividing an address into *netid* and *hostid*. The netid defined the network; the hostid defined the host in the network. The same idea can be applied in classless addressing. When an organization is granted a block of addresses, the block is actually divided into two parts, the **pre**fix and the **suf**fix. The prefix plays the same role as the netid; the suffix plays the same role as the hostid. All addresses in the block have the same prefix; each address has a different suffix. Figure 5.28 shows the prefix and suffix in a classless block.

**In classless addressing, the prefix defines the network and the suffix defines the host.**

**Figure 5.28** *Prefix and suffix*




In classful addressing, the length of the netid, *n*, depends on the class of the address; it can be only 8, 16, or 24. In classless addressing, the length of the prefix, *n*, depends on the size of the block; it can be 0, 1, 2, 3, . . . , 32. In classless addressing, the value of *n* is referred to as **prefix length;** the value of 32 – *n* is referred to as **suffix length.**

**The prefix length in classless addressing can be 1 to 32.**

### Example 5.22

What is the prefix length and suffix length if the whole Internet is considered as one single block with 4,294,967,296 addresses?

**Solution**

In this case, the prefix length is 0 and the suffix length is 32. All 32 bits vary to define $2^{32}$ = 4,294,967,296 hosts in this single block.

### Example 5.23

What is the prefix length and suffix length if the Internet is divided into 4,294,967,296 blocks and each block has one single address?

**Solution**

In this case, the prefix length for each block is 32 and the suffix length is 0. All 32 bits are needed to define $2^{32}$ = 4,294,967,296 blocks. The only address in each block is defined by the block itself.

### Example 5.24

The number of addresses in a block is inversely related to the value of the prefix length, *n*. A small *n* means a larger block; a large *n* means a small block.

### *Slash Notation*

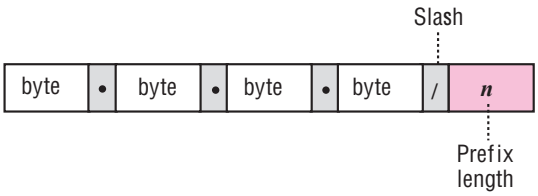
The netid length in classful addressing or the prefix length in classless addressing play a very important role when we need to extract the information about the block from a given address in the block. However, there is a difference here in classful and classless addressing.

- In classful addressing, the netid length is inherent in the address. Given an address, we know the class of the address that allows us to find the netid length (8, 16, or 24).

- In classless addressing, the prefix length cannot be found if we are given only an address in the block. The given address can belong to a block with any prefix length.

In classless addressing, we need to include the prefix length to each address if we need to find the block of the address. In this case, the prefix length, *n*, is added to the address separated by a slash. The notation is informally referred to as **slash notation.** An address in classless addressing can then be represented as shown in Figure 5.29.

**Figure 5.29** *Slash notation*




The slash notation is formally referred to as **classless interdomain routing** or **CIDR** (pronounced cider) notation.

**In classless addressing, we need to know one of the addresses in the block and the prefix length to define the block.**

### Example 5.25

In classless addressing, an address cannot per se define the block the address belongs to. For example, the address 230.8.24.56 can belong to many blocks some of them are shown below with the value of the prefix associated with that block:

| | | | | | |
|---|---|---|---|---|---|
| Prefix length:16 | → | Block: | 230.8.0.0 | to | 230.8.255.255 |
| Prefix length:20 | → | Block: | 230.8.16.0 | to | 230.8.31.255 |
| Prefix length:26 | → | Block: | 230.8.24.0 | to | 230.8.24.63 |
| Prefix length:27 | → | Block: | 230.8.24.32 | to | 230.8.24.63 |
| Prefix length:29 | → | Block: | 230.8.24.56 | to | 230.8.24.63 |
| Prefix length:31 | → | Block: | 230.8.24.56 | to | 230.8.24.57 |

### *Network Mask*

The idea of network mask in classless addressing is the same as the one in classful addressing. A network mask is a 32-bit number with the *n* leftmost bits all set to 0s and the rest of the bits all set to 1s.

### Example 5.26

The following addresses are defined using slash notations.

**a.** In the address 12.23.24.78/8, the network mask is 255.0.0.0. The mask has eight 1s and twenty-four 0s. The prefix length is 8; the suffix length is 24.

**b.** In the address 130.11.232.156/16, the network mask is 255.255.0.0. The mask has sixteen 1s and sixteen 0s.The prefix length is 16; the suffix length is 16.

**c.** In the address 167.199.170.82/27, the network mask is 255.255.255.224. The mask has twenty-seven 1s and five 0s. The prefix length is 27; the suffix length is 5.

### *Extracting Block Information*

An address in slash notation (CIDR) contains all information we need about the block: the first address (network address), the number of addresses, and the last address. These three pieces of information can be found as follows:

- The number of addresses in the block can be found as:

$$N = 2^{32-n}$$

in which $n$ is the prefix length and $N$ is the number of addresses in the block.

- The first address (network address) in the block can be found by ANDing the address with the network mask:

**First address = (any address) AND (network mask)**

Alternatively, we can keep the $n$ leftmost bits of any address in the block and set the 32– $n$ bits to 0s to find the first address.

- The last address in the block can be found by either adding the first address with the number of addresses or, directly, by ORing the address with the complement (NOTing) of the network mask:

**Last address = (any address) OR [NOT (network mask)]**

Alternatively, we can keep the $n$ leftmost bits of any address in the block and set the 32 – $n$ bits to 1s to find the last address.

### Example 5.27

One of the addresses in a block is 167.199.170.82/27. Find the number of addresses in the network, the first address, and the last address.

#### Solution

The value of $n$ is 27. The network mask has twenty-seven 1s and five 0s. It is 255.255.255.240.

**a.** The number of addresses in the network is $2^{32-n} = 2^{32-n} = 2^5 = 32$.

**b.** We use the AND operation to find the first address (network address).The first address is 167.199.170.64/27.

| | |
|---|---|
| Address in binary: | **10100111 11000111 10101010 01010010** |
| Network mask: | **11111111 11111111 11111111 11100000** |
| First address: | **10100111 11000111 10101010 01000000** |

c. To find the last address, we first find the complement of the network mask and then OR it with the given address: The last address is 167.199.170.95/27.

| | | | | |
|---|---|---|---|---|
| Address in binary: | 10100111 | 11000111 | 10101010 | 01010010 |
| Complement of network mask: | 00000000 | 00000000 | 00000000 | 00011111 |
| Last address: | 10100111 | 11000111 | 10101010 | 01011111 |

### Example 5.28

One of the addresses in a block is 17.63.110.114/24. Find the number of addresses, the first address, and the last address in the block.

#### Solution

The network mask is 255.255.255.0.

a. The number of addresses in the network is $2^{32-24} = 256$.

b. To find the first address, we use the short cut methods discussed early in the chapter.

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| Address: | 17 | . | 63 | . | 110 | . | 114 |
| Network mask: | 255 | . | 255 | . | 255 | . | 0 |
| First address (AND): | 17 | . | 63 | . | 110 | . | 0 |

The first address is 17.63.110.0/24.

c. To find the last address, we use the complement of the network mask and the first short cut method we discussed before. The last address is 17.63.110.255/24.

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| Address: | 17 | . | 63 | . | 110 | . | 114 |
| Complement of the mask (NOT): | 0 | . | 0 | . | 0 | . | 255 |
| Last address (OR): | 17 | . | 63 | . | 110 | . | 255 |

### Example 5.29

One of the addresses in a block is 110.23.120.14/20. Find the number of addresses, the first address, and the last address in the block.

#### Solution

The network mask is 255.255.240.0.

a. The number of addresses in the network is $2^{32-20} = 4096$.

b. To find the first address, we apply the first short cut to bytes 1, 2, and 4 and the second short cut to byte 3. The first address is 110.23.112.0/20.

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| Address: | 110 | . | 23 | . | 120 | . | 14 |
| Network mask: | 255 | . | 255 | . | 240 | . | 0 |
| First address (AND): | 110 | . | 23 | . | 112 | . | 0 |

c. To find the last address, we apply the first short cut to bytes 1, 2, and 4 and the second short cut to byte 3. The OR operation is applied to the complement of the mask. The last address is 110.23.127.255/20.

| | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| Address: | 110 | . | 23 | . | 120 | . | 14 |
| Network mask: | 0 | . | 0 | . | 15 | . | 255 |
| Last address (OR): | 110 | . | 23 | . | 127 | . | 255 |

## Block Allocation

The next issue in classless addressing is block allocation. How are the blocks allocated? The ultimate responsibility of block allocation is given to a global authority called the Internet Corporation for Assigned Names and Addresses (ICANN). However, ICANN does not normally allocate addresses to individual Internet users. It assigns a large block of addresses to an ISP (or a larger organization that is considered an ISP in this case). For the proper operation of the CIDR, three restrictions need to be applied to the allocated block.

1. The number of requested addresses, $N$, needs to be a power of 2. This is needed to provide an integer value for the prefix length, $n$ (see the second restriction). The number of addresses can be 1, 2, 4, 8, 16, and so on.
2. The value of prefix length can be found from the number of addresses in the block. Since $N = 2^{32 - n}$, then $n = \log_2 (2^{32}/N) = 32 - \log_2 N$. That is the reason why $N$ needs to be a power of 2.
3. The requested block needs to be allocated where there are a contiguous number of unallocated addresses in the address space. However, there is a restriction on choosing the beginning addresses of the block. The beginning address needs to be divisible by the number of addresses in the block. To see this restriction, we can show that the beginning address can be calculated as $X \times 2^{n - 32}$ in which $X$ is the decimal value of the prefix. In other words, the beginning address is $X \times N$.

### Example 5.30

An ISP has requested a block of 1000 addresses. The following block is granted.

a. Since 1000 is not a power of 2, 1024 addresses are granted ($1024 = 2^{10}$).
b. The prefix length for the block is calculated as $n = 32 - \log_2 1024 = 22$.
c. The beginning address is chosen as 18.14.12.0 (which is divisible by 1024).

The granted block is 18.14.12.0/22. The first address is 18.14.12.0/22 and the last address is 18.14.15.255/22.

### *Relation to Classful Addressing*

All issues discussed for classless addressing can be applied to classful addressing. As a matter of fact, classful addressing is a special case of the classless addressing in which the blocks in class A, B, and C have the prefix length $n_A = 8$, $n_B = 16$, and $n_C = 24$. A block in classful addressing can be easily changed to a block in class addressing if we use the prefix length defined in Table 5.1.

**Table 5.1** *Prefix length for classful addressing*

| *Class* | *Prefix length* | *Class* | *Prefix length* |
|---|---|---|---|
| A | /8 | D | /4 |
| B | /16 | E | /4 |
| C | /24 | | |

### Example 5.31

Assume an organization has given a class A block as 73.0.0.0 in the past. If the block is not revoked by the authority, the classless architecture assumes that the organization has a block 73.0.0.0/8 in classless addressing.

## Subnetting

Three levels of hierarchy can be created using subnetting. An organization (or an ISP) that is granted a range of addresses may divide the range into several subranges and assign each subrange to a **subnetwork** (or **subnet**). The concept is the same as we discussed for classful addressing. Note that nothing stops the organization from creating more levels. A subnetwork can be divided into several sub-subnetworks. A sub-subnetwork can be divided into several sub-sub-subnetworks. And so on.

### *Designing Subnets*

The subnetworks in a network should be carefully designed to enable the routing of packets. We assume the total number of addresses granted to the organization is $N$, the prefix length is $n$, the assigned number of addresses to each subnetwork is $N_{sub}$, the prefix length for each subnetwork is $n_{sub}$, and the total number of subnetworks is $s$. Then, the following steps need to be carefully followed to guarantee the proper operation of the subnetworks.

1. The number of addresses in each subnetwork should be a power of 2.
2. The prefix length for each subnetwork should be found using the following formula:

$$n_{sub} = n + \log_2 (N/N_{sub})$$

3. The starting address in each subnetwork should be divisible by the number of addresses in that subnetwork. This can be achieved if we first assign addresses to larger networks.

**The restrictions applied in allocating addresses for a subnetwork are parallel to the ones used to allocate addresses for a network.**

### *Finding Information about Each Subnetwork*

After designing the subnetworks, the information about each subnetwork, such as first and last address, can be found using the process we described to find the information about each network in the Internet.

### Example 5.32

An organization is granted the block 130.34.12.64/26. The organization needs four subnetworks, each with an equal number of hosts. Design the subnetworks and find the information about each network.

### Solution

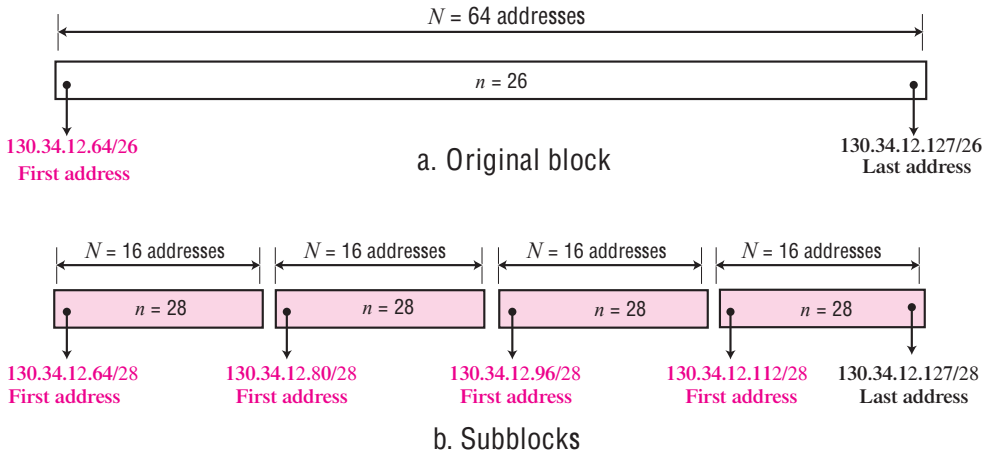
The number of addresses for the whole network can be found as $N = 2^{32-26} = 64$. Using the process described in the previous section, the first address in the network is 130.34.12.64/26 and the last address is 130.34.12.127/26. We now design the subnetworks:

1. We grant 16 addresses for each subnetwork to meet the first requirement (64/16 is a power of 2).
2. The **subnetwork** mask for each subnetwork is:

$$n_1 = n_2 = n_3 = n_4 = n + \log_2 (N/N_i) = 26 + \log_2 4 = 28$$

3. We grant 16 addresses to each subnet starting from the first available address. Figure 5.30 shows the subblock for each subnet. Note that the starting address in each subnetwork is divisible by the number of addresses in that subnetwork.

**Figure 5.30** *Solution to Example 5.32*




## Example 5.33

An organization is granted a block of addresses with the beginning address 14.24.74.0/24. The organization needs to have 3 subblocks of addresses to use in its three subnets as shown below:

- One subblock of 120 addresses.
- One subblock of 60 addresses.
- One subblock of 10 addresses.

### Solution

There are $2^{32-24} = 256$ addresses in this block. The first address is 14.24.74.0/24; the last address is 14.24.74.255/24.

a. The number of addresses in the first subblock is not a power of 2. We allocate 128 addresses. The first can be used as network address and the last as the special address. There are still 126 addresses available. The subnet mask for this subnet can be found as $n_1 = 24 + \log_2 (256/128) = 25$. The first address in this block is 14.24.74.0/25; the last address is 14.24.74.127/25.

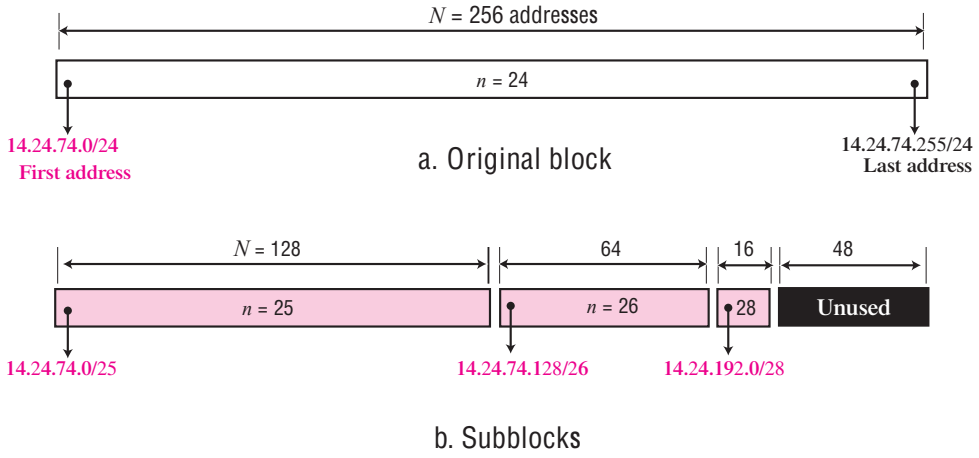
**b.** The number of addresses in the second subblock is not a power of 2 either. We allocate 64 addresses. The first can be used as network address and the last as the special address. There are still 62 addresses available. The subnet mask for this subnet can be found as $n_1 = 24 + \log_2 (256/64) = 26$. The first address in this block is 14.24.74.128/26; the last address is 14.24.74.191/26.

**c.** The number of addresses in the third subblock is not a power of 2 either. We allocate 16 addresses. The first can be used as network address and the last as the special address. There are still 14 addresses available. The subnet mask for this subnet can be found as $n_1 = 24 + \log_2 (256/16) = 28$. The first address in this block is 14.24.74.192/28; the last address is 14.24.74.207/28.

**d.** If we add all addresses in the previous subblocks, the result is 208 addresses, which means 48 addresses are left in reserve. The first address in this range is 14.24.74.209. The last address is 14.24.74.255. We don't know about the prefix length yet.

**e.** Figure 5.31 shows the configuration of blocks. We have shown the first address in each block.

**Figure 5.31** *Solution to Example 5.33*




## Example 5.34

Assume a company has three offices: Central, East, and West. The Central office is connected to the East and West offices via private, point-to-point WAN lines. The company is granted a block of 64 addresses with the beginning address 70.12.100.128/26. The management has decided to allocate 32 addresses for the Central office and divides the rest of addresses between the two other offices.

1. The number of addresses are assigned as follows:

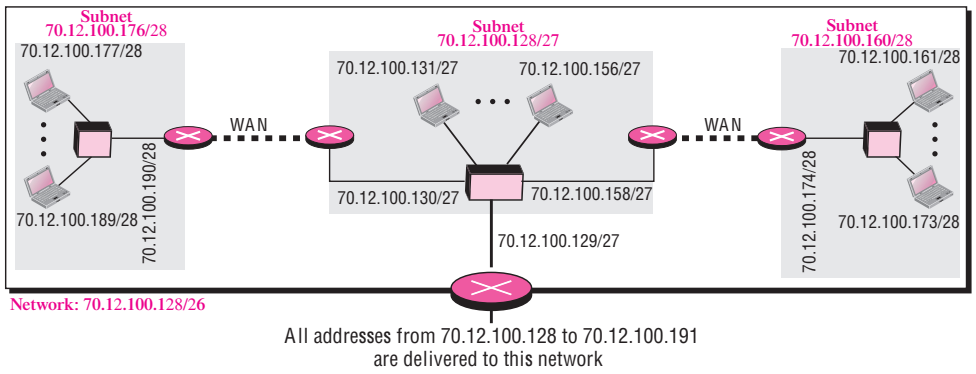
| Central office $N_c = 32$ | East office $N_e = 16$ | West office $N_w = 16$ |
|---|---|---|

2. We can find the prefix length for each subnetwork:

| $n_c = n + \log_2(64/32) = 27$ | $n_e = n + \log_2(64/16) = 28$ | $n_w = n + \log_2(64/16) = 28$ |
|---|---|---|

3. Figure 5.32 shows the configuration designed by the management. The Central office uses addresses 70.12.100.128/27 to 70.12.100.159/27. The company has used three of these addresses for the routers and has reserved the last address in the subblock. The East office uses the addresses 70.12.100.160/28 to 70.12.100.175/28. One of these addresses is used for the router and the company has reserved the last address in the subblock. The West office uses the addresses 70.12.100.160/28 to 70.12.100.175/28. One of these addresses is used for the router and the company has reserved the last address in the subblock. The company uses no address for the point-to-point connections in WANs.

**Figure 5.32** *Example 14*




### *Address Aggregation*

One of the advantages of CIDR architecture is **address aggregation.** ICANN assigns a large **block of addresses** to an ISP. Each ISP in turn divides its assigned block into smaller subblocks and grants the subblocks to its customers; many blocks of addresses are aggregated in one block and granted to one ISP.

### Example 5.35

An ISP is granted a block of addresses starting with 190.100.0.0/16 (65,536 addresses). The ISP needs to distribute these addresses to three groups of customers as follows:

- The first group has 64 customers; each needs approximately 256 addresses.
- The second group has 128 customers; each needs approximately 128 addresses.
- The third group has 128 customers; each needs approximately 64 addresses.

We design the subblocks and find out how many addresses are still available after these allocations.

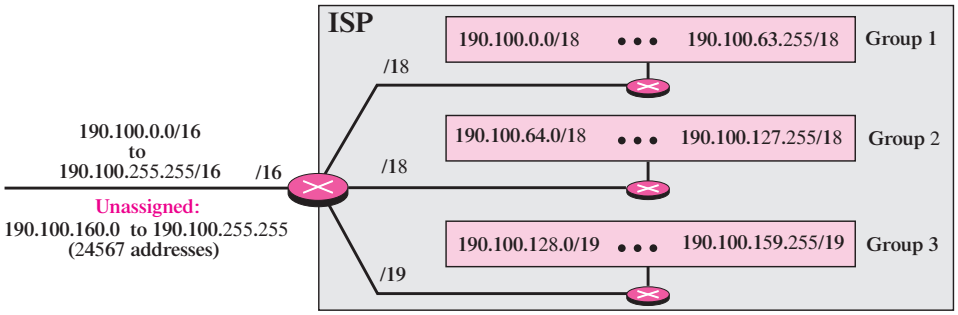
#### Solution

Let us solve the problem in two steps. In the first step, we allocate a subblock of addresses to each group. The total number of addresses allocated to each group and the prefix length for each subblock can found as

| | |
|---|---|
| Group 1: $64 \times 256 = 16{,}384$ | $n_1 = 16 + \log_2 (65536/16384) = 18$ |
| Group 2: $128 \times 128 = 16{,}384$ | $n_2 = 16 + \log_2 (65536/16384) = 18$ |
| Group 3: $128 \times 64 = 8192$ | $n_3 = 16 + \log_2 (65536/8192) = 19$ |

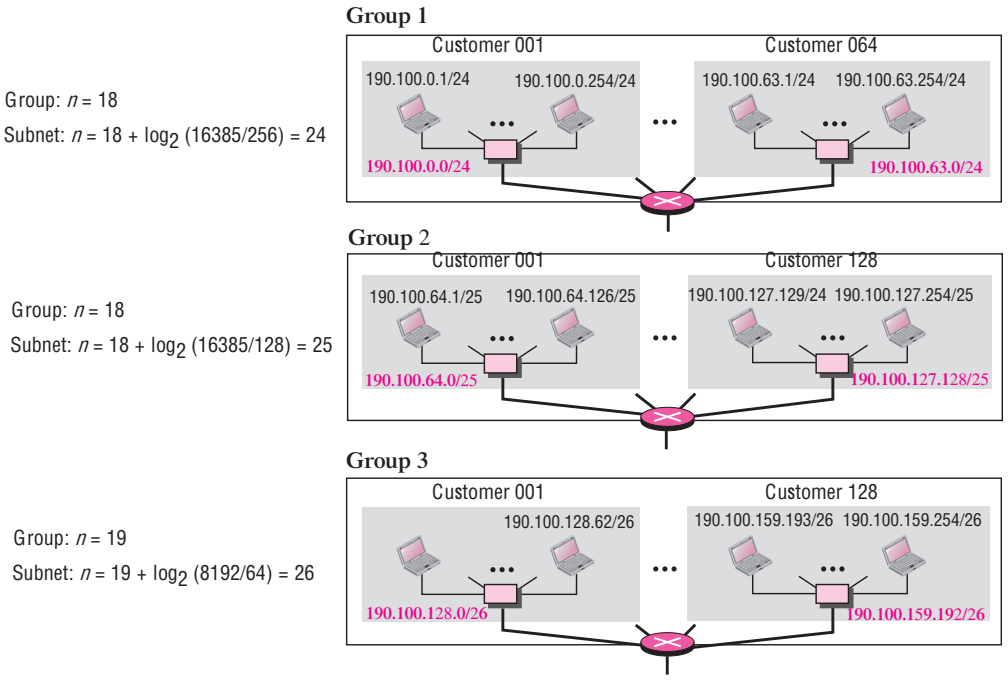
Figure 5.33 shows the design for the first hierarchical level.

**Figure 5.33** *Solution to Example 5.35: first step*




Now we can think about each group. The prefix length changes for the networks in each group depending on the number of addresses used in each network. Figure 5.34 shows the second level of the hierarchy. Note that we have used the first address for each customer as the subnet address and have reserved the last address as a special address.

**Figure 5.34** *Solution to Example 5.35: second step*



## 5.4 SPECIAL ADDRESSES

In classful addressing some addresses were reserved for special purposes. The classless addressing scheme inherits some of these special addresses from classful addressing.

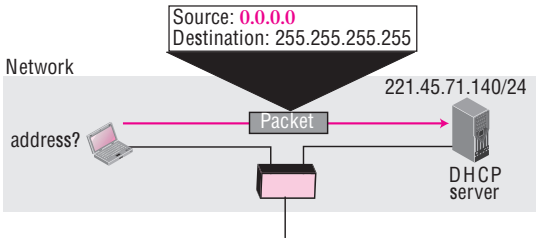
### Special Blocks

Some blocks of addresses are reserved for special purposes.

#### *All-Zeros Address*

The block 0.0.0.0/32, which contains only one single address, is reserved for communication when a host needs to send an IPv4 packet but it does not know its own address. This is normally used by a host at bootstrap time when it does not know its IPv4 address. The host sends an IPv4 packet to a bootstrap server (called DHCP server as discussed in Chapter 18) using this address as the source address and a **limited broadcast address** as the destination address to find its own address (see Figure 5.35).

**Figure 5.35** *Examples of using the all-zeros address*




#### *All-Ones Address: Limited Broadcast Address*

The block 255.255.255.255/32, which contains one single address, is reserved for limited broadcast address in the current network. A host that wants to send a message to every other host can use this address as a destination address in an IPv4 packet. However, a router will block a packet having this type of address to confine the broadcasting to the local network. In Figure 5.36, a host sends a datagram using a destination IPv4 address consisting of all 1s. All devices on this network receive and process this datagram.

#### *Loopback Addresses*

The block 127.0.0.0/8 is used for the **loopback address,** which is an address used to test the software on a machine. When this address is used, a packet never leaves the machine; it simply returns to the protocol software. It can be used to test the IPv4 software. For example, an application such as "ping" can send a packet with a loopback address as the destination address to see if the IPv4 software is capable of receiving and processing a packet. As another example, the loopback address can be used by a *client process* (a running application program) to send a message to a server process on the same machine. Note that this can be used only as a destination address in an IPv4 packet (see Figure 5.37).

**Figure 5.36** *Example of limited broadcast address*

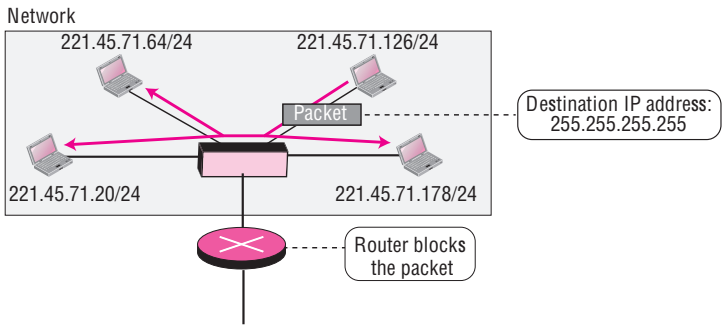
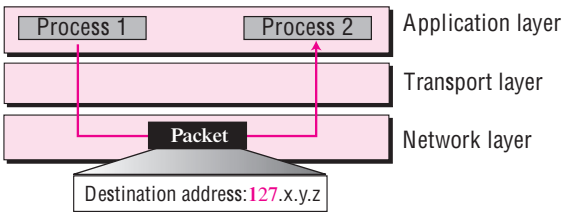



**Figure 5.37** *Example of loopback address*




### *Private Addresses*

A number of blocks are assigned for private use. They are not recognized globally. These blocks are depicted in Table 5.2. These addresses are used either in isolation or in connection with network address translation techniques (see NAT section later in this chapter).

**Table 5.2** *Addresses for private networks*

| *Block* | *Number of addresses* | *Block* | *Number of addresses* |
|---|---|---|---|
| 10.0.0.0/8 | 16,777,216 | 192.168.0.0/16 | 65,536 |
| 172.16.0.0/12 | 1,047,584 | 169.254.0.0/16 | 65,536 |

### *Multicast Addresses*

The block 224.0.0.0/4 is reserved for multicast communication. We discuss multicasting in Chapter 12 in detail.

## Special Addresses in Each block

It is not mandatory, but it is recommended, that some addresses in a block be used for special addresses. These addresses are not assigned to any host. However, if a block (or subblock) is so small, we cannot afford to use part of the addresses as special addresses.

### *Network Address*

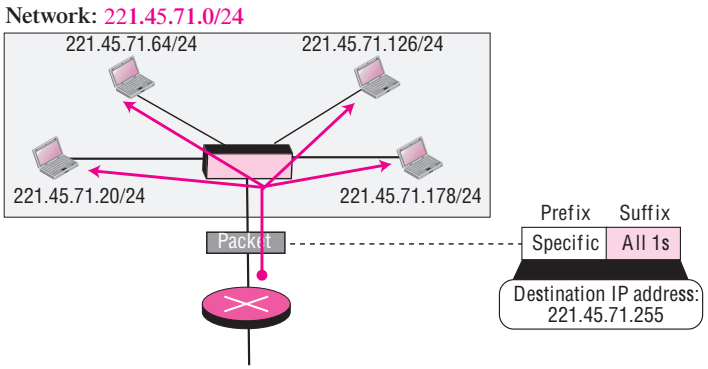
We have already discussed network addresses. The first address (with the suffix set all to 0s) in a block defines the network address. It actually defines the network itself

(cabling) and not any host in the network. Of course, the first address in a subnetwork is called the subnetwork address and plays the same role.

### *Direct Broadcast Address*

The last address in a block or subblock (with the suffix set all to 1s) can be used as a **direct broadcast address.** This address is usually used by a router to send a packet to all hosts in a specific network. All hosts will accept a packet having this type of destination address. Note that this address can be used only as a destination address in an IPv4 packet. In Figure 5.38, the router sends a datagram using a destination IPv4 address with a suffix of all 1s. All devices on this network receive and process the datagram.

**Figure 5.38** *Example of a direct broadcast address*



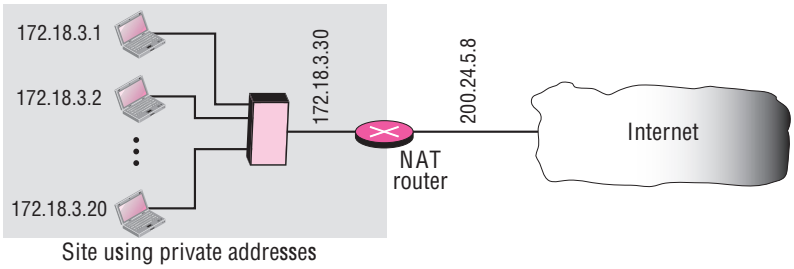

## 5.5 NAT

The distribution of addresses through ISPs has created a new problem. Assume that an ISP has granted a small range of addresses to a small business or a household. If the business grows or the household needs a larger range, the ISP may not be able to grant the demand because the addresses before and after the range may have already been allocated to other networks. In most situations, however, only a portion of computers in a small network need access to the Internet simultaneously. This means that the number of allocated addresses does not have to match the number of computers in the network. For example, assume a small business with 20 computers in which the maximum number of computers that access the Internet simultaneously is only 5. Most of the computers are either doing some task that does not need Internet access or communicating with each other. This small business can use the TCP/IP protocol for both internal and universal communication. The business can use 20 (or 25) addresses from the private block addresses discussed before for internal communication; five addresses for universal communication can be assigned by the ISP.

A technology that can provide the mapping between the private and universal addresses, and at the same time, support virtual private networks that we discuss in Chapter 30, is **network address translation (NAT).** The technology allows a site to use a set of private addresses for internal communication and a set of global Internet addresses

(at least one) for communication with the rest of the world. The site must have only one single connection to the global Internet through a NAT-capable router that runs NAT software. Figure 5.39 shows a simple implementation of NAT.

**Figure 5.39** *NAT*

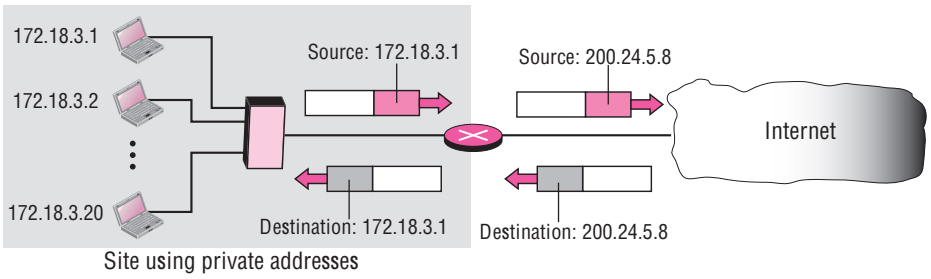



As the figure shows, the private network uses private addresses. The router that connects the network to the global address uses one private address and one global address. The private network is transparent to the rest of the Internet; the rest of the Internet sees only the NAT router with the address 200.24.5.8.

## Address Translation

All of the outgoing packets go through the NAT router, which replaces the *source address* in the packet with the global NAT address. All incoming packets also pass through the NAT router, which replaces the *destination address* in the packet (the NAT router global address) with the appropriate private address. Figure 5.40 shows an example of address translation.

**Figure 5.40** *Address translation*




## Translation Table

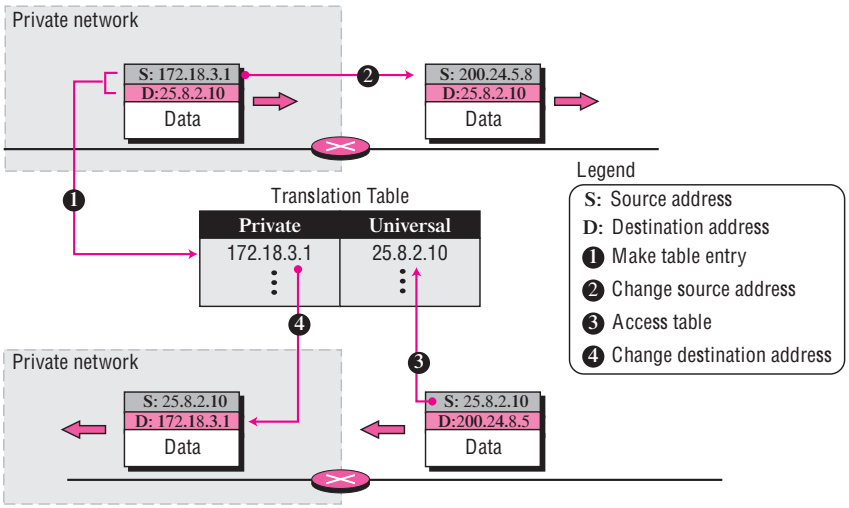
The reader may have noticed that translating the source addresses for an outgoing packet is straightforward. But how does the NAT router know the destination address for a packet coming from the Internet? There may be tens or hundreds of private IP

addresses, each belonging to one specific host. The problem is solved if the NAT router has a **translation table.**

### *Using One IP Address*

In its simplest form, a translation table has only two columns: the private address and the external address (destination address of the packet). When the router translates the source address of the outgoing packet, it also makes note of the destination address—where the packet is going. When the response comes back from the destination, the router uses the source address of the packet (as the external address) to find the private address of the packet. Figure 5.41 shows the idea.

**Figure 5.41** *Translation*




In this strategy, communication must always be initiated by the private network. The NAT mechanism described requires that the private network start the communication. As we will see, NAT is used mostly by ISPs that assign one single address to a customer. The customer, however, may be a member of a private network that has many private addresses. In this case, communication with the Internet is always initiated from the customer site, using a client program such as HTTP, TELNET, or FTP to access the corresponding server program. For example, when e-mail that originates from a non-customer site is received by the ISP e-mail server, it is stored in the mailbox of the customer until retrieved with a protocol such as POP.

A private network cannot run a server program for clients outside of its network if it is using NAT technology.

### *Using a Pool of IP Addresses*

Using only one global address by the NAT router allows only one private-network host to access the same external host. To remove this restriction, the NAT router can use a

pool of global addresses. For example, instead of using only one global address (200.24.5.8), the NAT router can use four addresses (200.24.5.8, 200.24.5.9, 200.24.5.10, and 200.24.5.11). In this case, four private-network hosts can communicate with the same external host at the same time because each pair of addresses defines a connection. However, there are still some drawbacks. No more than four connections can be made to the same destination. No private-network host can access two external server programs (e.g., HTTP and TELNET) at the same time. And, likewise, two private-network hosts cannot access the same external server program (e.g., HTTP or TELNET) at the same time.

### *Using Both IP Addresses and Port Addresses*

To allow a many-to-many relationship between private-network hosts and external server programs, we need more information in the translation table. For example, suppose two hosts inside a private network with addresses 172.18.3.1 and 172.18.3.2 need to access the HTTP server on external host 25.8.3.2. If the translation table has five columns, instead of two, that include the source and destination port addresses and the transport layer protocol, the ambiguity is eliminated. Table 5.3 shows an example of such a table.

**Table 5.3** *Five-column translation table*

| *Private Address* | *Private Port* | *External Address* | *External Port* | *Transport Protocol* |
|---|---|---|---|---|
| 172.18.3.1 | 1400 | 25.8.3.2 | 80 | TCP |
| 172.18.3.2 | 1401 | 25.8.3.2 | 80 | TCP |
| … | … | … | … | … |

Note that when the response from HTTP comes back, the combination of source address (25.8.3.2) and destination port address (1400) defines the private network host to which the response should be directed. Note also that for this translation to work, the ephemeral port addresses (1400 and 1401) must be unique.

## 5.6 FURTHER READING

For more details about subjects discussed in this chapter, we recommend the following books and RFCs. The items enclosed in brackets refer to the reference list at the end of the book.

### Books

Several books give thorough coverage of materials discussed in this chapter. We recommend [Com 06], [Tan 03], [Koz 05], [Ste 95].

### RFCs

Several RFCs deal with IPv4 addressing including RFC 917, RFC 927, RFC 930, RFC 932, RFC 940, RFC 950, RFC 1122, and RFC 1519.

## 5.7 KEY TERMS

address aggregation
address space
binary notation
block of addresses
class A address
class B address
class C address
class D address
class E address
classful addressing
classless addressing
classless interdomain routing (CIDR)
default mask
direct broadcast address
dotted-decimal notation
hexadecimal notation
hostid
IP address
limited broadcast address
loopback address
netid
network address
Network Address Translation (NAT)
network mask
prefix
prefix length
slash notation
subnet
subnet mask
subnetting
subnetwork
suffix
suffix length
supernet mask
supernetting
translation table

## 5.8 SUMMARY

- The identifier used in the IP layer of the TCP/IP protocol suite is called the Internet address or IP address. An IPv4 address is 32 bits long. An address space is the total number of addresses used by the protocol. The address space of IPv4 is $2^{32}$ or 4,294,967,296.
- In classful addressing, the IPv4 address space is divided into five classes: A, B, C, D, and E. An organization is granted a block in one of the three classes, A, B, or C. Classes D and E is reserved for special purposes. An IP address in classes A, B, and C is divided into netid and hostid.
- In classful addressing, the first address in the block is called the network address. It defines the network to which an address belongs. The network address is used in routing a packet to its destination network.
- A network mask or a default mask in classful addressing is a 32-bit number with $n$ leftmost bits all set to 1s and $(32 - n)$ rightmost bits all set to 0s. It is used by a router to find the network address from the destination address of a packet.
- The idea of splitting a network into smaller subnetworks is called subnetting. A subnetwork mask, like a network mask, is used to find the subnetwork address when a destination IP address is given. In supernetting, an organization can combine several class C blocks to create a larger range of addresses.
- In 1996, the Internet authorities announced a new architecture called classless addressing or CIDR that allows an organization to have a block of addresses of any size as long as the size of the block is a power of two.

- The address in classless addressing is also divided into two parts: the prefix and the suffix. The prefix plays the same role as the netid; the suffix plays the same role as the hostid. All addresses in the block have the same prefix; each address has a different suffix.
- Some of the blocks in IPv4 are reserved for special purposes. In addition, some addresses in a block are traditionally used for special addresses. These addresses are not assigned to any host.
- To improve the distribution of addresses, NAT technology has been created to allow the separation of private addresses in a network from the global addresses used in the Internet. A translation table can translate the private addresses, selected from the blocks allocated for this purpose, to global addresses. The translation table also translates the IP addresses as well as the port number for mapping from the private to global addresses and vice versa.

## 5.9 PRACTICE SET

### Exercises

1. What is the address space in each of the following systems?
   a. a system with 8-bit addresses
   b. a system with 16-bit addresses
   c. a system with 64-bit addresses
2. An address space has a total of 1,024 addresses. How many bits are needed to represent an address?
3. An address space uses three symbols: 0, 1, and 2 to represent addresses. If each address is made of 10 symbols, how many addresses are available in this system?
4. Change the following IP addresses from dotted-decimal notation to binary notation:
   a. 114.34.2.8
   b. 129.14.6.8
   c. 208.34.54.12
   d. 238.34.2.1
5. Change the following IP addresses from dotted-decimal notation to hexadecimal notation:
   a. 114.34.2.8
   b. 129.14.6.8
   c. 208.34.54.12
   d. 238.34.2.1
6. Change the following IP addresses from hexadecimal notation to binary notation:
   a. 0x1347FEAB
   b. 0xAB234102

c. 0x0123A2BE
d. 0x00001111

7. How many hexadecimal digits are needed to define the netid in each of the following classes?
   a. Class A
   b. Class B
   c. Class C
8. Change the following IP addresses from binary notation to dotted-decimal notation:
   a. `01111111 11110000 01100111 01111101`
   b. `10101111 11000000 11111000 00011101`
   c. `11011111 10110000 00011111 01011101`
   d. `11101111 11110111 11000111 00011101`
9. Find the class of the following IP addresses:
   a. 208.34.54.12
   b. 238.34.2.1
   c. 242.34.2.8
   d. 129.14.6.8
10. Find the class of the following IP addresses:
   a. `11110111 11110011 10000111 11011101`
   b. `10101111 11000000 11110000 00011101`
   c. `11011111 10110000 00011111 01011101`
   d. `11101111 11110111 11000111 00011101`
11. Find the netid and the hostid of the following IP addresses:
   a. 114.34.2.8
   b. 132.56.8.6
   c. 208.34.54.12
   d. 251.34.98.5
12. Find the number of addresses in the range if the first address is 14.7.24.0 and the last address is 14.14.34.255.
13. If the first address in a range is 122.12.7.0 and there are 2048 addresses in the range, what is the last address?
14. Find the result of each operation:
   a. NOT (22.14.70.34)
   b. NOT (145.36.12.20)
   c. NOT (200.7.2.0)
   d. NOT (11.20.255.255)
15. Find the result of each operation:
   a. (22.14.70.34) AND (255.255.0.0)
   b. (12.11.60.12) AND (255.0.0.0)

**c.** (14.110.160.12) AND (255.200.140.0)

**d.** (28.14.40.100) AND (255.128.100.0)

**16.** Find the result of each operation:

**a.** (22.14.70.34) OR (255.255.0.0)

**b.** (12.11.60.12) OR (255.0.0.0)

**c.** (14.110.160.12) OR (255.200.140.0)

**d.** (28.14.40.100) OR (255.128.100.0)

**17.** In a class A subnet, we know the IP address of one of the hosts and the subnet mask as given below:

| IP Address: 25.34.12.56 | Subnet mask: 255.255.0.0 |
|---|---|

What is the first address (subnet address)? What is the last address?

**18.** In a class B subnet, we know the IP address of one of the hosts and the subnet mask as given below:

| IP Address: 131.134.112.66 | Subnet mask: 255.255.224.0 |
|---|---|

What is the first address (subnet address)? What is the last address?

**19.** In a class C subnet, we know the IP address of one of the hosts and the subnet mask as given below:

| IP Address: 202.44.82.16 | Subnet mask: 255.255.255.192 |
|---|---|

What is the first address (subnet address)? What is the last address?

**20.** Find the subnet mask in each case:

**a.** 1024 subnets in class A

**b.** 256 subnets in class B

**c.** 32 subnets in class C

**d.** 4 subnets in class C

**21.** In a block of addresses, we know the IP address of one host is 25.34.12.56/16. What is the first address (network address) and the last address (limited broadcast address) in this block?

**22.** In a block of addresses, we know the IP address of one host is 182.44.82.16/26. What is the first address (network address) and the last address (limited broadcast address) in this block?

**23.** In fixed-length subnetting, find the number of 1s that must be added to the mask if the number of desired subnets is _____.

**a.** 2

**b.** 62

**c.** 122

**d.** 250

24. An organization is granted the block 16.0.0.0/8. The administrator wants to create 500 fixed-length subnets.
    a. Find the subnet mask.
    b. Find the number of addresses in each subnet.
    c. Find the first and the last address in the first subnet.
    d. Find the first and the last address in the last subnet (subnet 500).
25. An organization is granted the block 130.56.0.0/16. The administrator wants to create 1024 subnets.
    a. Find the subnet mask.
    b. Find the number of addresses in each subnet.
    c. Find the first and the last address in the first subnet.
    d. Find the first and the last address in the last subnet (subnet 1024).
26. An organization is granted the block 211.17.180.0/24. The administrator wants to create 32 subnets.
    a. Find the subnet mask.
    b. Find the number of addresses in each subnet.
    c. Find the first and the last address in the first subnet.
    d. Find the first and the last address in the last subnet (subnet 32).
27. Write the following mask in slash notation (/*n*):
    a. 255.255.255.0
    b. 255.0.0.0
    c. 255.255.224.0
    d. 255.255.240.0
28. Find the range of addresses in the following blocks:
    a. 123.56.77.32/29
    b. 200.17.21.128/27
    c. 17.34.16.0/23
    d. 180.34.64.64/30
29. In classless addressing, we know the first and the last address in the block. Can we find the prefix length? If the answer is yes, show the process and give an example.
30. In classless addressing, we know the first address and the number of addresses in the block. Can we find the prefix length? If the answer is yes, show the process and give an example.
31. In classless addressing, can two blocks have the same prefix length? Explain.
32. In classless addressing, we know the first address and one of the addresses in the block (not necessarily the last address). Can we find the prefix length? Explain.
33. An ISP is granted a block of addresses starting with 150.80.0.0/16. The ISP wants to distribute these blocks to 2600 customers as follows:
    a. The first group has 200 medium-size businesses; each needs approximately 128 addresses.

**b.** The second group has 400 small businesses; each needs approximately 16 addresses.

**c.** The third group has 2000 households; each needs 4 addresses.

Design the subblocks and give the slash notation for each subblock. Find out how many addresses are still available after these allocations.

**34.** An ISP is granted a block of addresses starting with 120.60.4.0/20. The ISP wants to distribute these blocks to 100 organizations with each organization receiving 8 addresses only. Design the subblocks and give the slash notation for each subblock. Find out how many addresses are still available after these allocations.

**35.** An ISP has a block of 1024 addresses. It needs to divide the addresses to 1024 customers. Does it need subnetting? Explain your answer.