

Network Routing

Aashish Paliwal

C-STAR

International Institute of Information Technology, Hyderabad

September 26, 2024

Outline

- 1 Introduction
- 2 Desirable properties of routing
- 3 Design parameters
- 4 Routing strategies

- 1 Introduction
- 2 Desirable properties of routing
- 3 Design parameters
- 4 Routing strategies

Introduction

- Routing is a process by which the switches construct their routing table.
- Any sufficiently large network (e.g. Internet) can have hundreds or even thousands of paths to reach from node A to node B. As network scale, routing becomes more important and complex in packet switched network design.
- Routing consists a huge chunk of network layers responsibility .

Challenges in Routing?

- Distributed information: Each node only knows about their immediate neighbours in the network.
- Efficiency: The path found by network should be reasonably good.
- Failures: Links and nodes usually fail and recover arbitrarily. Network should be able to find a path, if one exists, even after failures.

Routing vs Forwarding

- Routing
 - Computes path that packets will follow.
 - Routers share data among themselves to create forwarding table.
- Forwarding
 - a table lookup.
 - Directing a packet to particular link.
 - Uses forwarding tables extensively.

Routing Game

- Require 30 volunteers.
- Adapted rules from Scott Shenker with some modification.
- Rules:
 - Each slip says Your node ID = ? and nodes you are connect to.
 - Your job: Find path from source to destination.
 - You are directly connected to two or three nodes.
 - You may not: Leave your spot or pass your slip (anything a router would not do).
 - You may: ask for advice, speak to other participants (anything).
 - You must: Try (if you're a volunteer), help or provide advice (everyone), not cheat (acting like a different node).

Outline

- 1 Introduction
- 2 Desirable properties of routing**
- 3 Design parameters
- 4 Routing strategies

Desirable properties

- Correctness
- Simplicity
- Robustness
- Stability
- Efficiency

Outline

- 1 Introduction
- 2 Desirable properties of routing
- 3 Design parameters**
- 4 Routing strategies

- **Performance Criteria:** number of hops, cost, delay, latency
- **Decision time:** per packet (Datagram), per session (Virtual Circuit)
- **Decision place:** each node (distributed), central node (centralised), originated node (source)
- **Network information source:** none, local, adjacent node, nodes along route, all nodes
- **Network information update timing:** continuous, periodic, major load change, topology change

Outline

- 1 Introduction
- 2 Desirable properties of routing
- 3 Design parameters
- 4 Routing strategies**

Routing strategies

- a substantial number of routing strategies have developed over decades
- Some known routing strategies includes:
 - Static routing
 - Flooding
 - Random routing
 - Flow based routing
 - Dynamic routing

Static Routing

- a central routing matrix is created based on least cost path and is stored at network control center
- the matrix holds the next node on the route for each source destination pair
- Advantages
 - Simple
 - Works well in reliable networks with stable load
- Disadvantages
 - Not robust - contains single point of failure
 - Does not respond to failures
 - Lacks flexibility

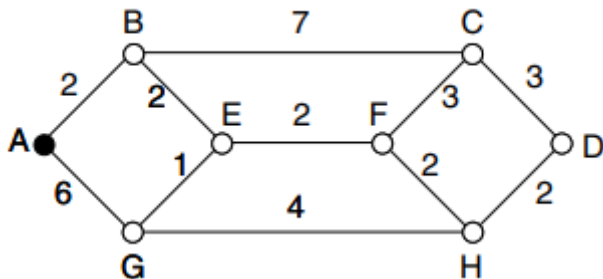
Least Cost Path

- Cost associated with each link e.g. - minimum hop, latency, queue length etc
- For each pair of attached nodes, the least cost path is looked for
- Several algorithms exist to find optimal path
 - Dijkstra's algorithm
 - Bellman-Ford algorithm

Dijkstra's algorithm

- Dijkstra's algorithm finds the shortest paths from a given source node to all other nodes in order of increasing path length
- The algorithm converges under static conditions of topology and cost
- Basic steps
 - Initialization
 - Set cost of source node to 0
 - Set cost of all other nodes to ∞
 - while unknown node left in the graph
 - Select an unknown node b with the lowest cost
 - Mark b as known
 - For each node a adjacent to b
 - a 's cost = $\min(a$'s old cost, b 's cost + cost of (b, a))

Dijkstra's algorithm example



- Does not require any network information
- Every incoming packet is forwarded to every outgoing link
- Advantages
 - Simple
 - Packet reaches every connected node
 - Highly robust (can be used for emergency messages)
 - At least one packet pass through shortest path
- Disadvantages
 - Generates huge amount of duplicate data
 - Require damping mechanism to be used in practise eg - hop count

- for every incoming packet, node selects one output link on random
- Advantages
 - Has simplicity and robustness of flooding while generation much less traffic
 - Does not need network information
- Disadvantages
 - Actual route will usually be much more expensive than least cost path

Flow Based Routing

- utilised information about topology and load information for routing
- If capacity and average flow is known for a give line, we can calculate mean packet delay
- From mean delay of all lines, we can calculate flow weighted average to get mean delay of subnet
- Routing algorithm then optimised path based on minimum average delay

Dynamic/Adaptive Routing

- Network state information must be exchanged among nodes
- Typical metrics - Number of hops, time delay, total number of packets queued in the path
- How much information is required?
 - More information exchange \implies better routing \implies more overhead
 - More frequent \implies better routing \implies more overhead
- Popular approaches
 - Distance Vector Routing
 - Link State Routing

Distance Vector Routing

- Can be divided into three steps
 - Identify neighbours
 - Send advertisements to neighbours consisting of tuples as
$$[(dest1, cost1), (dest2, cost2), (dest3, cost3).....]$$
 - Integration step uses Bellman-Ford algorithm to find least cost path
$$updated_cost = \min(current_cost, link_cost(source, advertiser) + advertised_cost(advertiser, destination))$$
- Advantages
 - Simple
 - Works well for small networks
 - Only advertises best cost to each destination
- Disadvantages
 - Works only for small networks
 - Cannot determine topology of network

Link State Routing

- Can be divided into three steps
 - Identify neighbours
 - Send link-state advertisements (LSA) periodically
 $[seq\#, (nbhr1, linkcost1), (nbhr2, linkcost2), \dots]$
 - Integration
 - if $seq\#$ in incoming LSA \neq $seq\#$ in saved LSA for source node: Update LSA for node with new $seq\#$, neighbour list rebroadcast LSA to neighbour (Flooding)
 - Remove saved LSA if $seq\#$ is very old
 - Result: Each node has complete current map of network
- Uses Dijkstra's algorithm at each node to compute shortest path
- Advantages
 - Scalable
 - Fast network convergence
- Disadvantages
 - High network overhead
 - Require more memory and processing power

IP Protocol Stack: Key Abstractions



- **Transport layer is where we:**
 - Provide applications with good abstractions
 - Without support or feedback from the network

Transport Protocols

- **Logical communication between processes**
 - Sender divides a message into segments
 - Receiver reassembles segments into message
- **Transport services**
 - (De)multiplexing packets
 - Detecting corrupted data
 - Optionally: reliable delivery, flow control, ...

User Datagram Protocol (UDP)

- **Lightweight communication between processes**
 - Send and receive messages
 - Avoid overhead of ordered, reliable delivery
 - No connection setup delay, no in-kernel connection state
- **Used by popular apps**
 - Query/response for DNS
 - Real-time data in VoIP

8 byte header

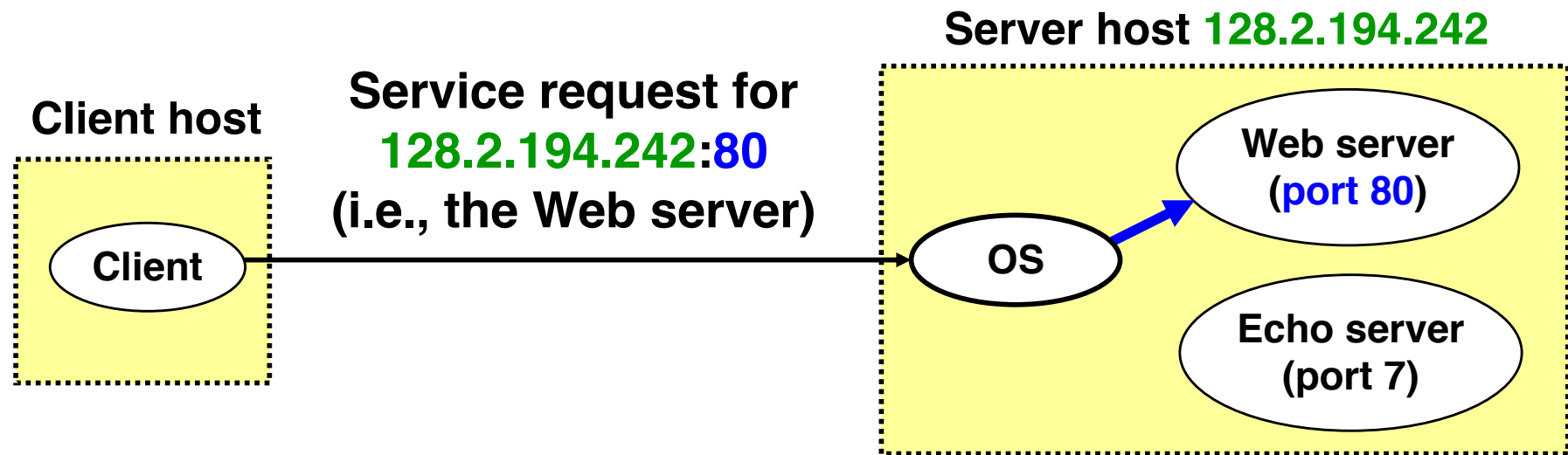
SRC port	DST port
checksum	length
DATA	

Advantages of UDP

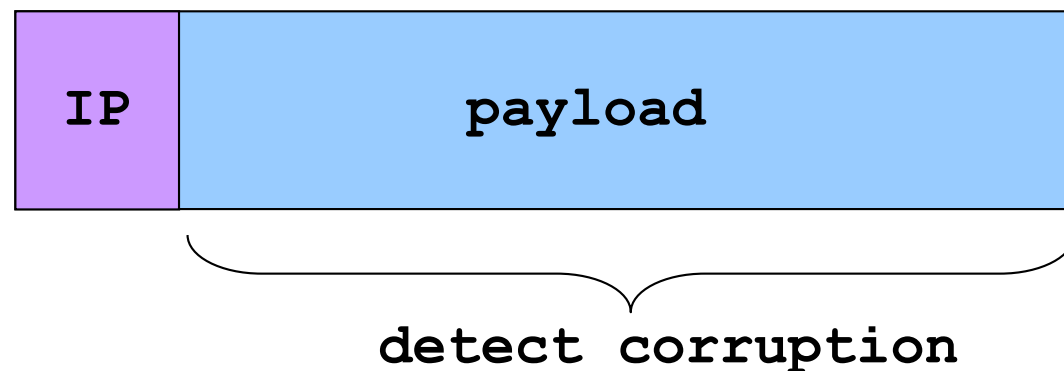
- **Fine-grain control**
 - UDP sends as soon as the application writes
- **No connection set-up delay**
 - UDP sends without establishing a connection
- **No connection state in host OS**
 - No buffers, parameters, sequence #s, etc.
- **Small header overhead**
 - UDP header is only eight-bytes long

Two Basic Transport Features

- **Demultiplexing: port numbers**



- **Error detection: checksums**



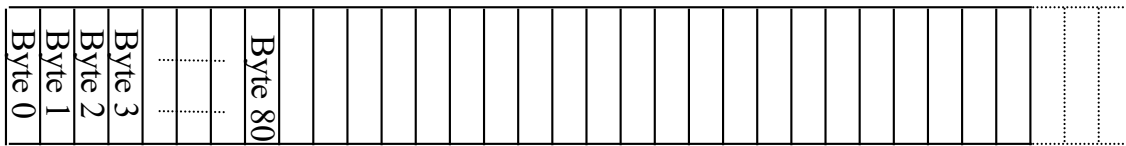
Transmission Control Protocol (TCP)

- **Stream-of-bytes service**
 - Sends and receives a stream of bytes
- **Reliable, in-order delivery**
 - Corruption: checksums
 - Detect loss/reordering: sequence numbers
 - Reliable delivery: acknowledgments and retransmissions
- **Connection oriented**
 - Explicit set-up and tear-down of TCP connection
- **Flow control**
 - Prevent overflow of the receiver's buffer space
- **Congestion control**
 - Adapt to network congestion for the greater good

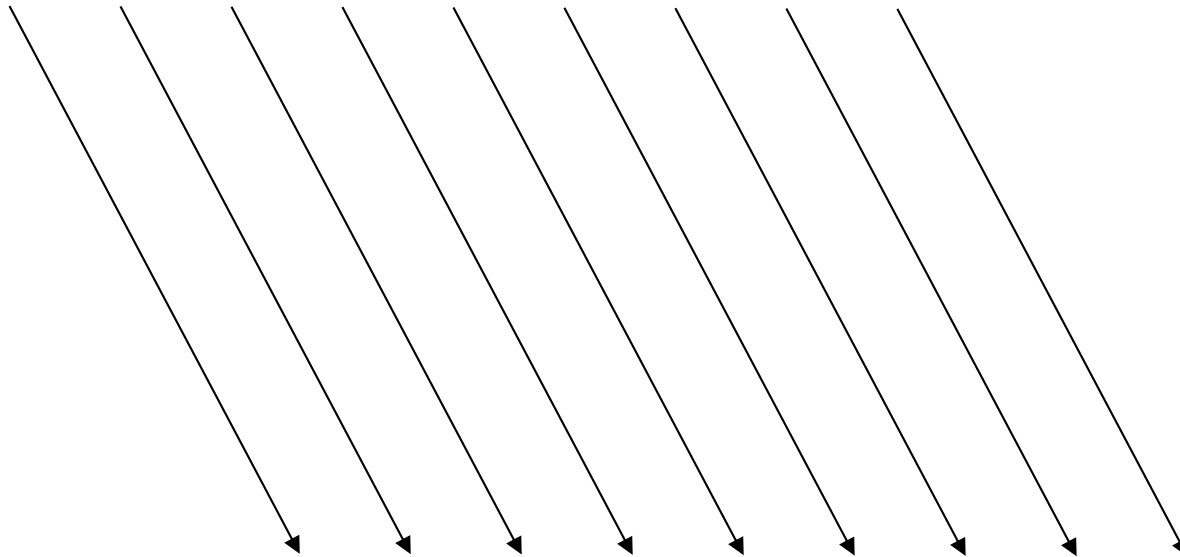
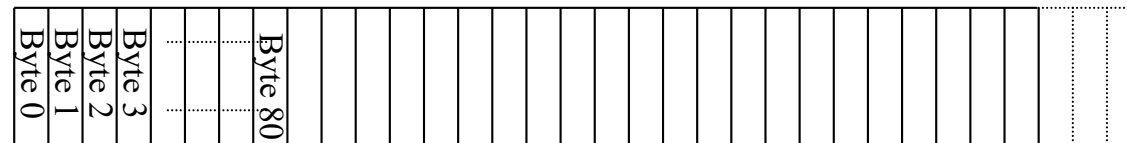
Breaking a Stream of Bytes into TCP Segments

TCP “Stream of Bytes” Service

Host A

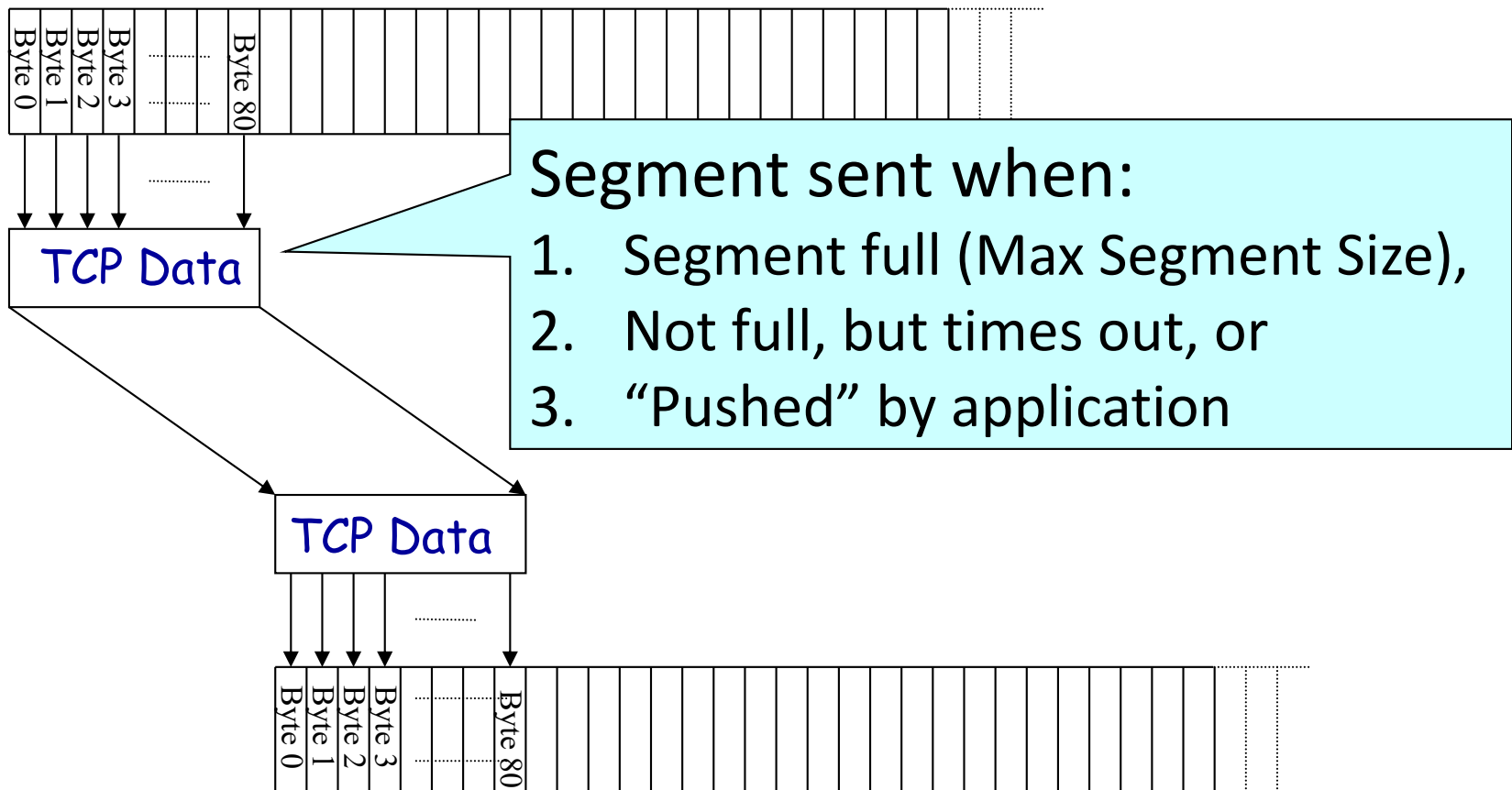


Host B



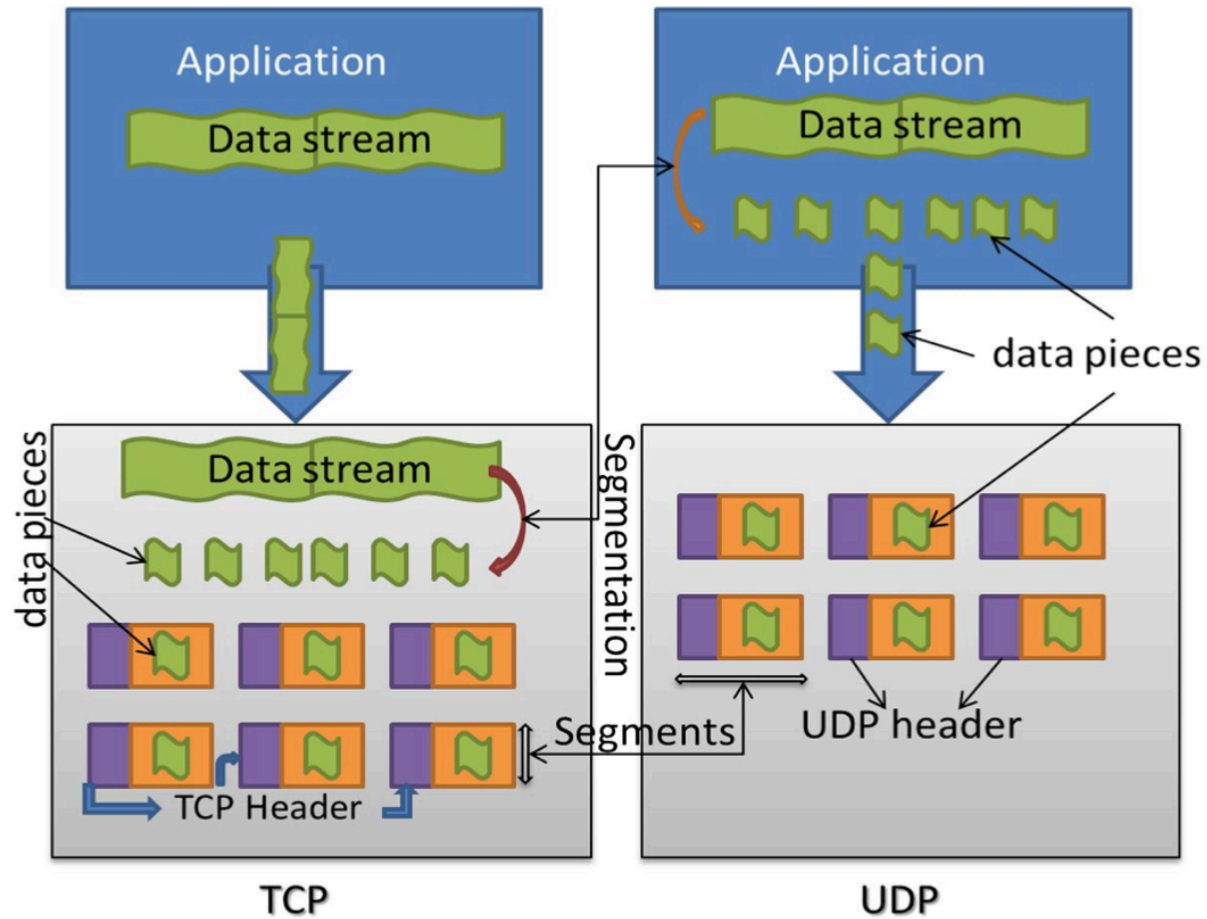
...Emulated Using TCP “Segments”

Host A

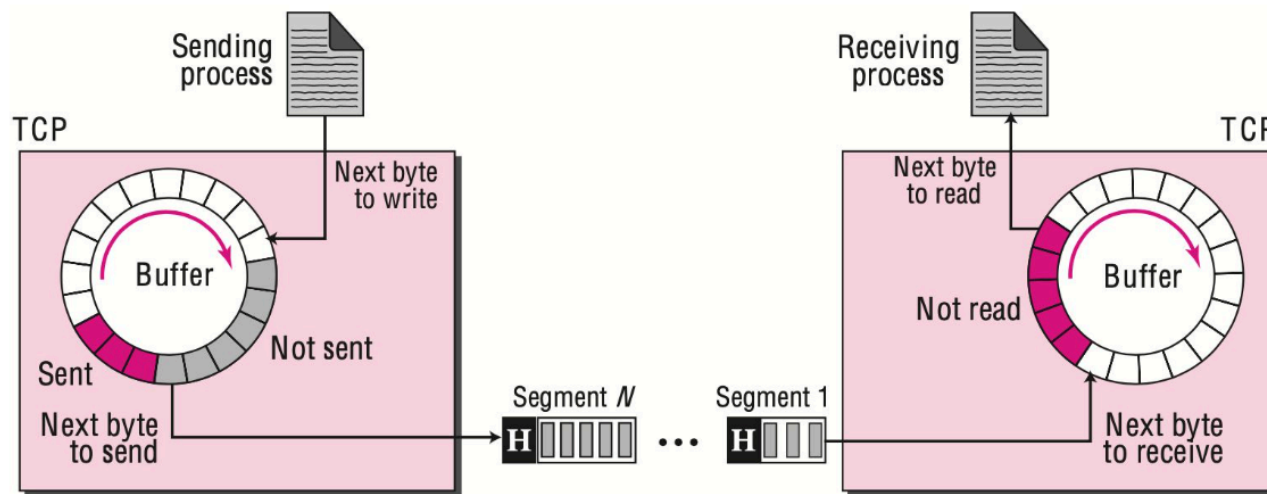


Host B

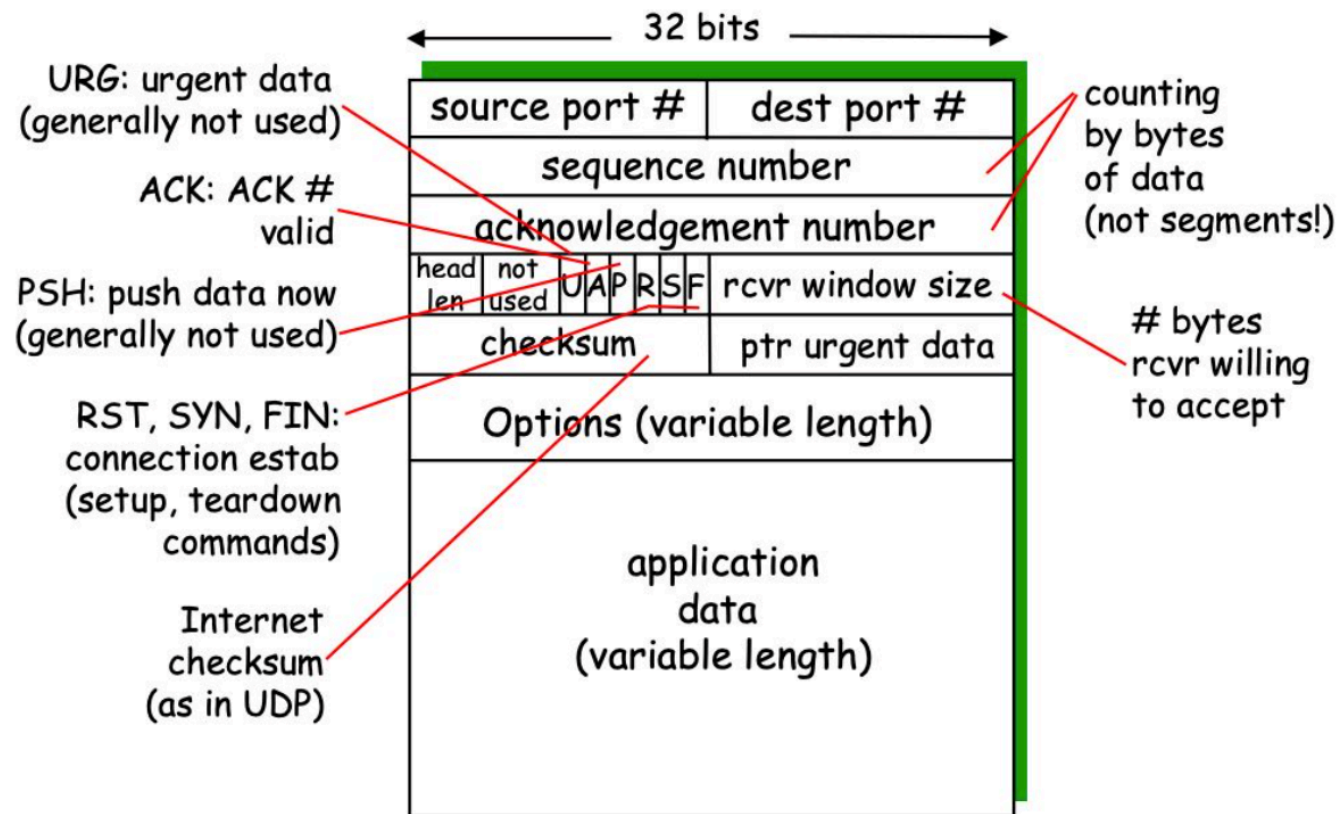
- TCP stream-of-bytes service
 - TCP supports segmentation while UDP does not



- TCP stream-of-bytes service
 - Segments and buffers

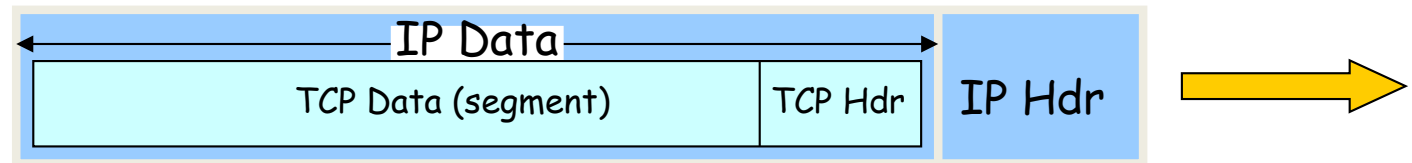


- TCP segment format



TCP Segment

- IP packet



- No bigger than Maximum Transmission Unit (MTU)
- E.g., up to 1500 bytes on an Ethernet link

- TCP packet

- IP packet with a TCP header and data inside
- TCP header is typically 20 bytes long

- TCP segment

- No more than Maximum Segment Size (MSS) bytes
- E.g., up to 1460 consecutive bytes from the stream:
MTU (1500) - IP header (20) - TCP header (20)

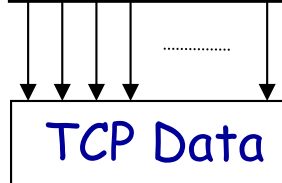
Sequence Number

Host A

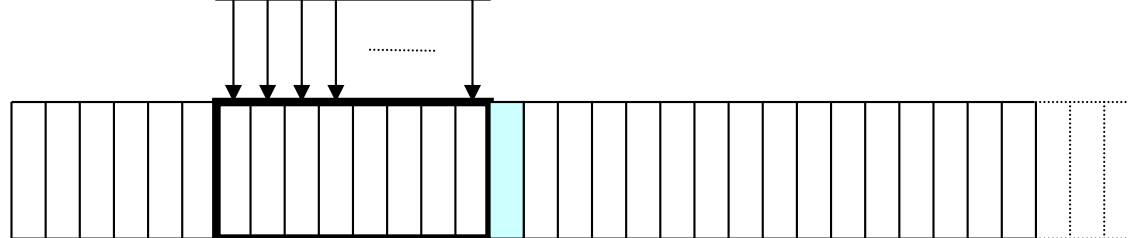
ISN (initial sequence number)



Sequence number = 1st byte



Host B



- **Sequence number**

- **Example**

- **TCP connection is transferring a file of 6000 bytes**

- **First byte is numbered 10010**

- **Data is sent in 5 segments**

- **Segments 1-4 carrying 1000 bytes, segment 5 carrying 2000 bytes**

- **What are SqN for each segment?**

- **Segment 1: SqN: 10,010 (10,010 to 11,009)**

- **Segment 2: SqN: 11,010 (11,010 to 12,009)**

- **Segment 3: SqN: 12,010 (12,010 to 13,009)**

- **Segment 4: SqN: 13,010 (13,010 to 14,009)**

- **Segment 5: SqN: 14,010 (14,010 to 16,009)**

Reliable Delivery on a Lossy Channel With Bit Errors

Challenges of Reliable Data Transfer

- Over a perfectly reliable channel: Done
- Over a channel with bit errors
 - Receiver detects errors and requests retransmission
- Over a lossy channel with bit errors
 - Some data missing, others corrupted
 - Receiver cannot easily detect loss
- Over a channel that may reorder packets
 - Receiver cannot easily distinguish loss vs. out-of-order

An Analogy

- **Alice and Bob are talking**

- What if Alice couldn't understand Bob?
- Bob asks Alice to repeat what she said



- **What if Bob hasn't heard Alice for a while?**

- Is Alice just being quiet? Has she lost reception?
- How long should Bob just keep on talking?
- Maybe Alice should periodically say “uh huh”
- ... or Bob should ask “Can you hear me now?”

Take-Aways from the Example

- **Acknowledgments from receiver**
 - Positive: “okay” or “uh huh” or “ACK”
 - Negative: “please repeat that” or “NACK”
- **Retransmission by the sender**
 - After *not* receiving an “ACK”
 - After receiving a “NACK”
 - You can use both (as TCP does implicitly)
- **Timeout by the sender (“stop and wait”)**
 - Don’t wait forever without some acknowledgment

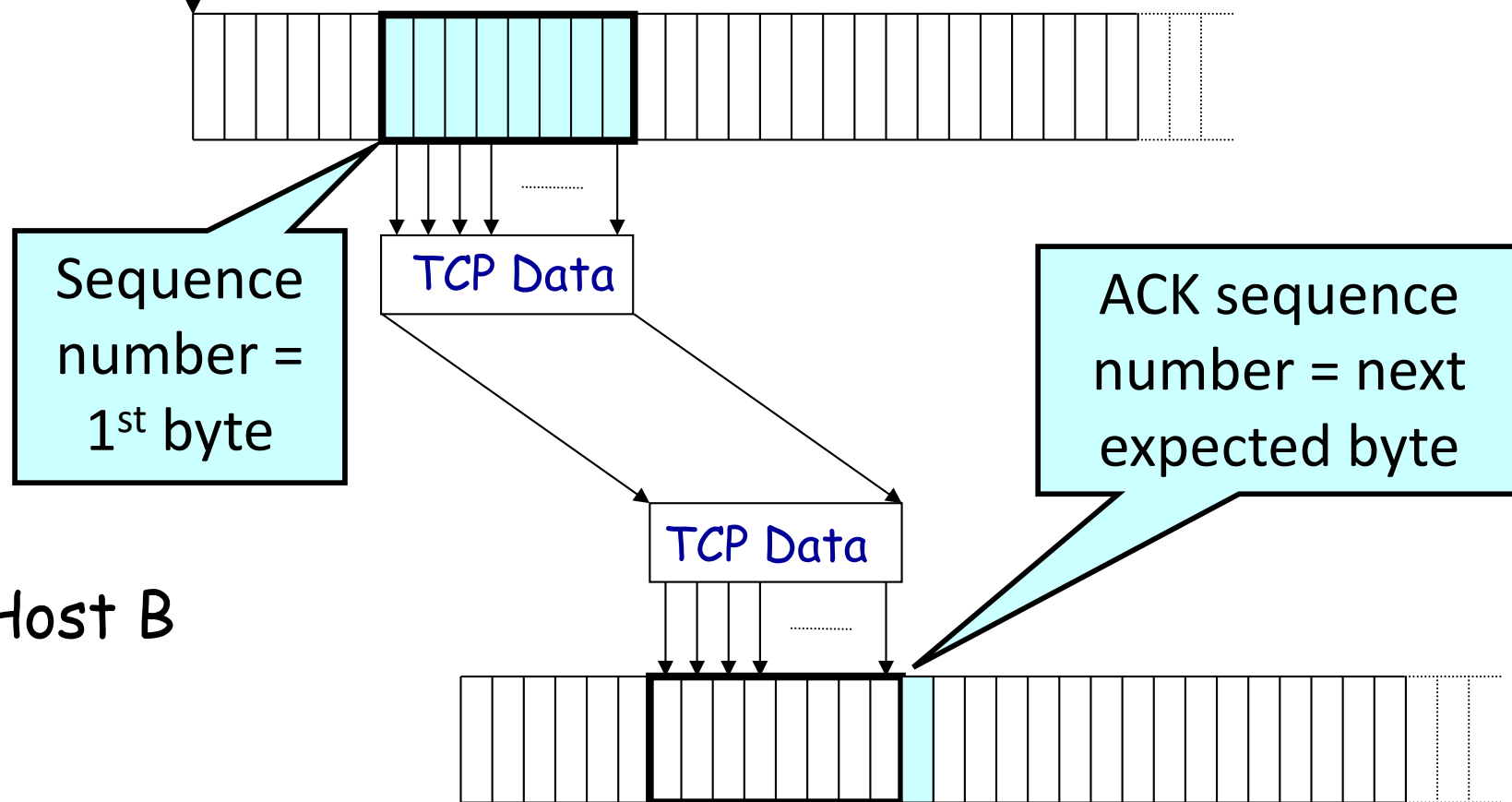
TCP Support for Reliable Delivery

- **Detect bit errors: checksum**
 - Used to detect corrupted data at the receiver
 - ...leading the receiver to drop the packet
- **Detect missing data: sequence number**
 - Used to detect a gap in the stream of bytes
 - ... and for putting the data back in order
- **Recover from lost data: retransmission**
 - Sender retransmits lost or corrupted data
 - Two main ways to detect lost packets

TCP Acknowledgments

Host A

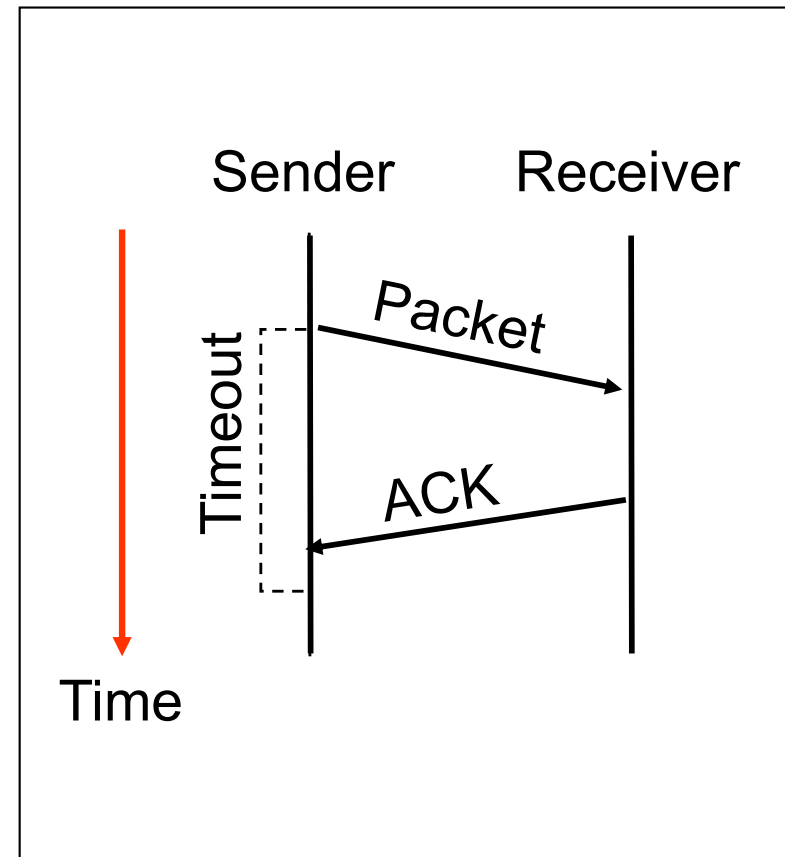
ISN (initial sequence number)



Host B

Automatic Repeat reQuest (ARQ)

- **ACK and timeouts**
 - Receiver sends ACK when it receives packet
 - Sender waits for ACK and times out
- **Simplest ARQ protocol**
 - Stop and wait
 - Send a packet, stop and wait until ACK arrives



Initial Sequence Number (ISN)

- Sequence number for the very first byte
 - E.g., Why not a de facto ISN of 0?
- Practical issue: reuse of port numbers
 - Port numbers must (eventually) get used again
 - ... and an old packet may still be in flight
 - ... and associated with the new connection
- So, TCP must change the ISN over time
 - Set from a 32-bit clock that ticks every 4 microsec
 - ... which wraps around once every 4.55 hours!

Quick TCP Math

- Initial Seq No = 501. Sender sends 4500 bytes successfully acknowledged. Next sequence number to send is:

(Y) 5000 (M) 5001 (C) 5002

- Next 1000 byte TCP segment received. Receiver acknowledges with ACK number:

(Y) 5001 (M) 6000 (C) 6001

Quick TCP Math

- Initial Seq No = 501. Sender sends 4500 bytes successfully acknowledged. Next sequence number to send is:

(Y) 5000 (M) 5001 (C) 5002

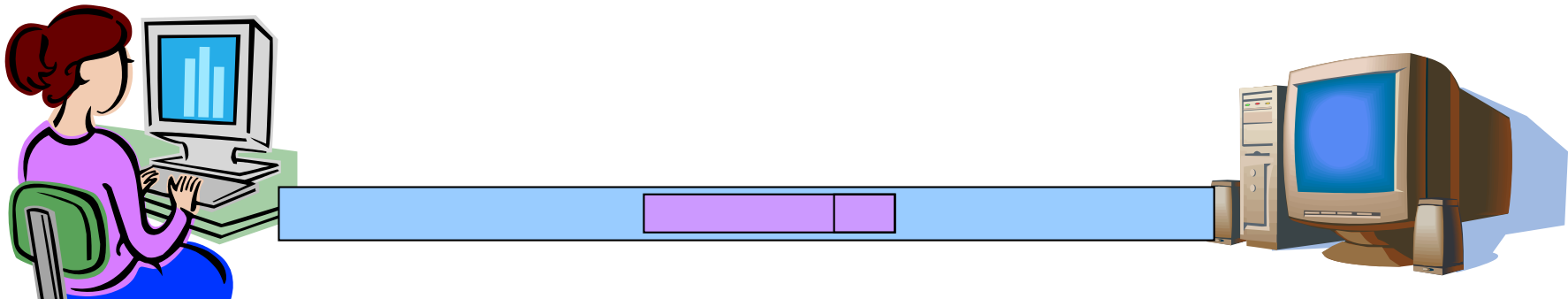
- Next 1000 byte TCP segment received. Receiver acknowledges with ACK number:

(Y) 5001 (M) 6000 (C) 6001

Flow Control: TCP Sliding Window

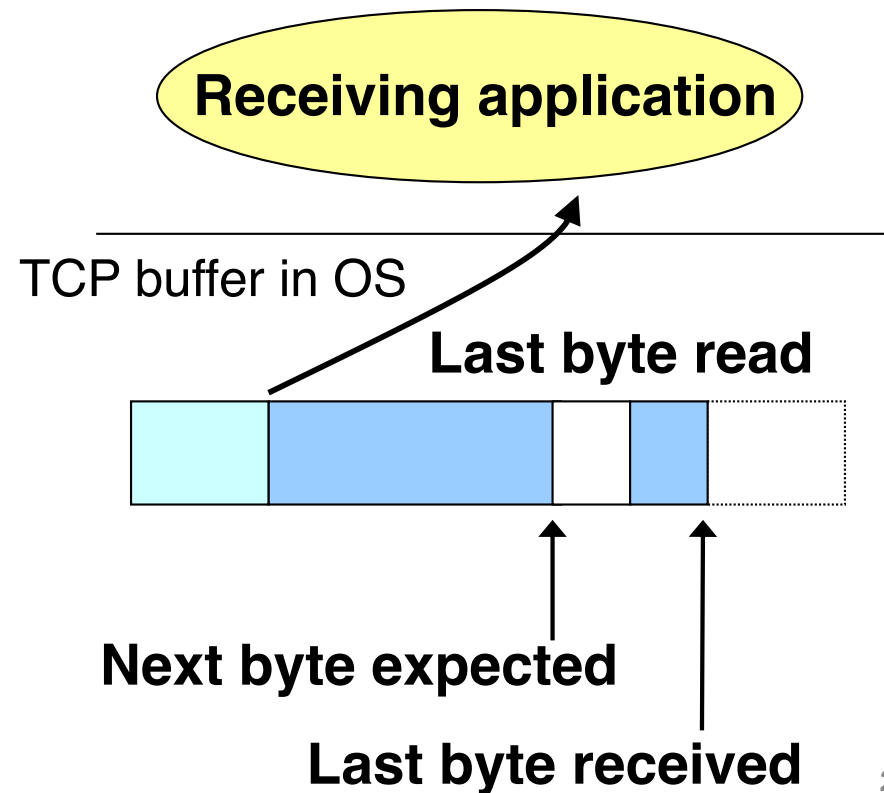
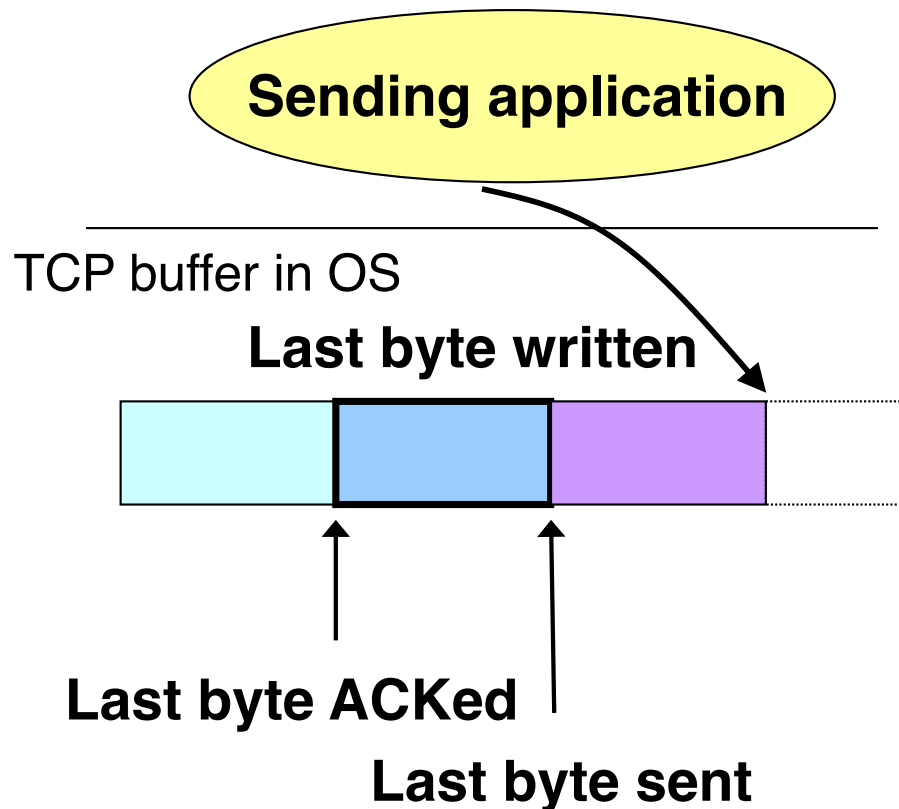
Motivation for Sliding Window

- Stop-and-wait is inefficient
 - Only one TCP segment is “in flight” at a time
- Consider: 1.5 Mbps link with 50 ms round-trip-time (RTT)
 - Assume TCP segment size of 1 KB (8 Kbits)
 - 8 Kbits/segment at 50 msec/segment → 160 Kbps
 - That’s 11% of the capacity of 1.5 Mbps link



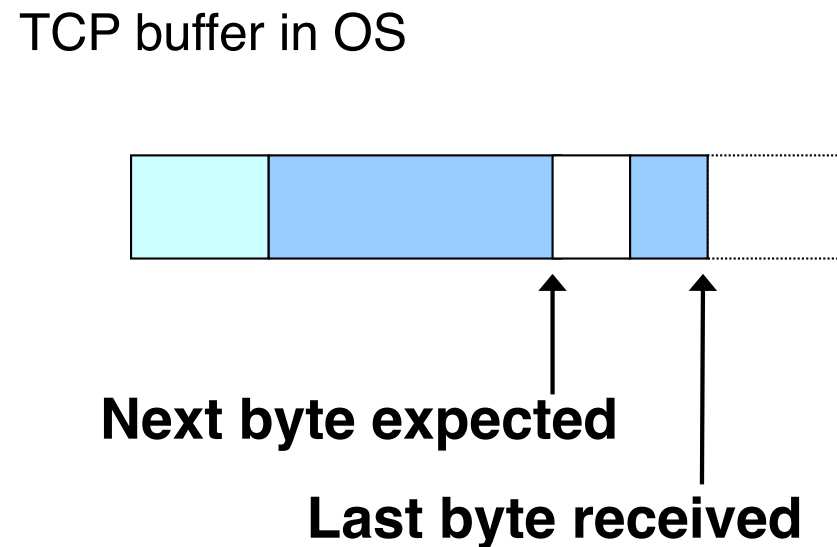
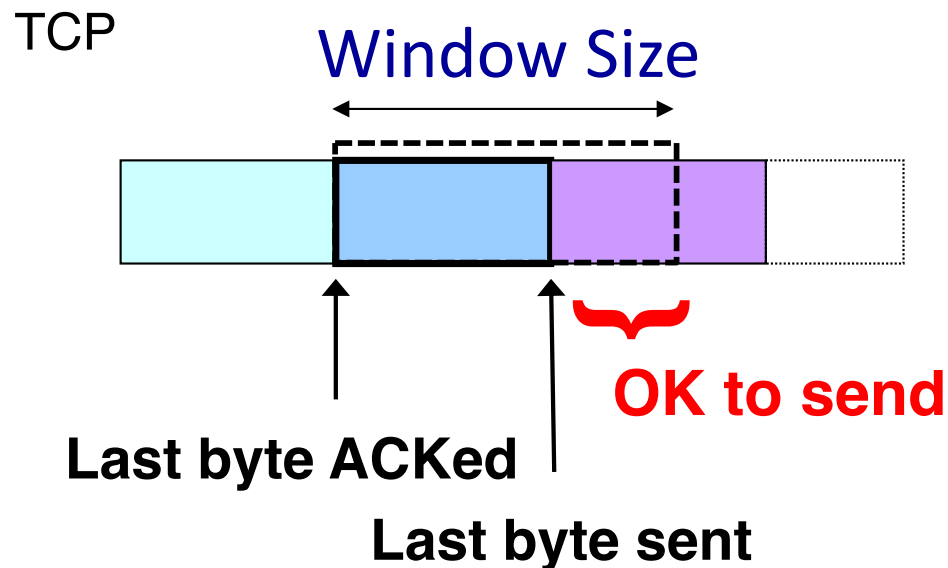
Sliding Window

- Allow a larger amount of data “in flight”
 - Allow sender to get ahead of the receiver
 - ... though not too far ahead



Sliding Window

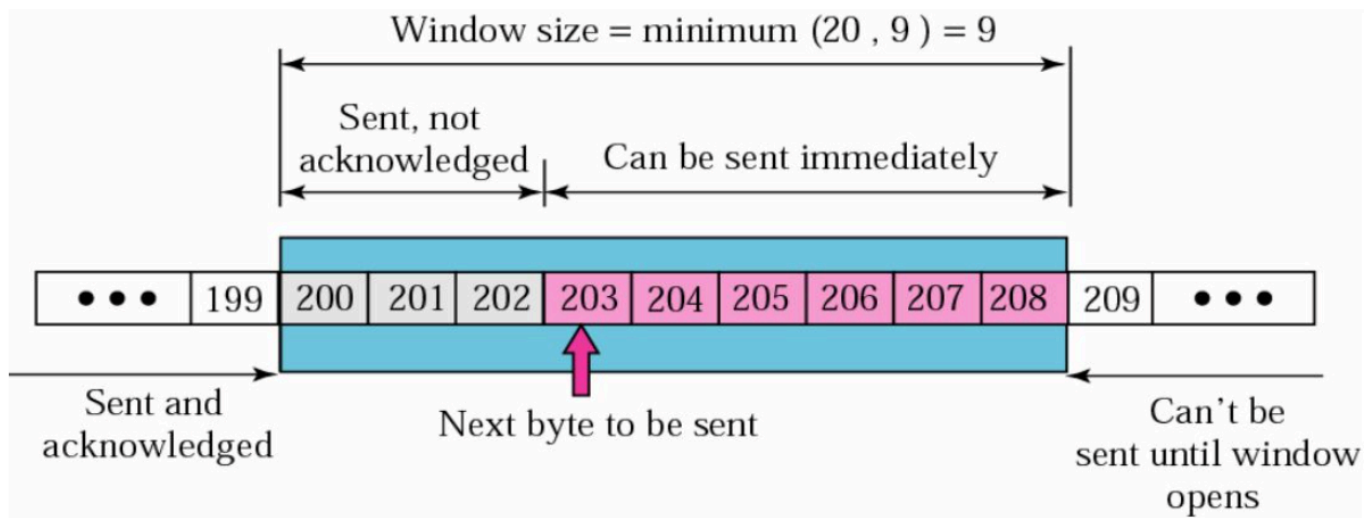
- **Receive window size**
 - Amount that can be sent without acknowledgment
 - Receiver must be able to store this amount of data
- **Receiver tells the sender the window**
 - Tells the sender the amount of free space left



- TCP sliding window

- Example

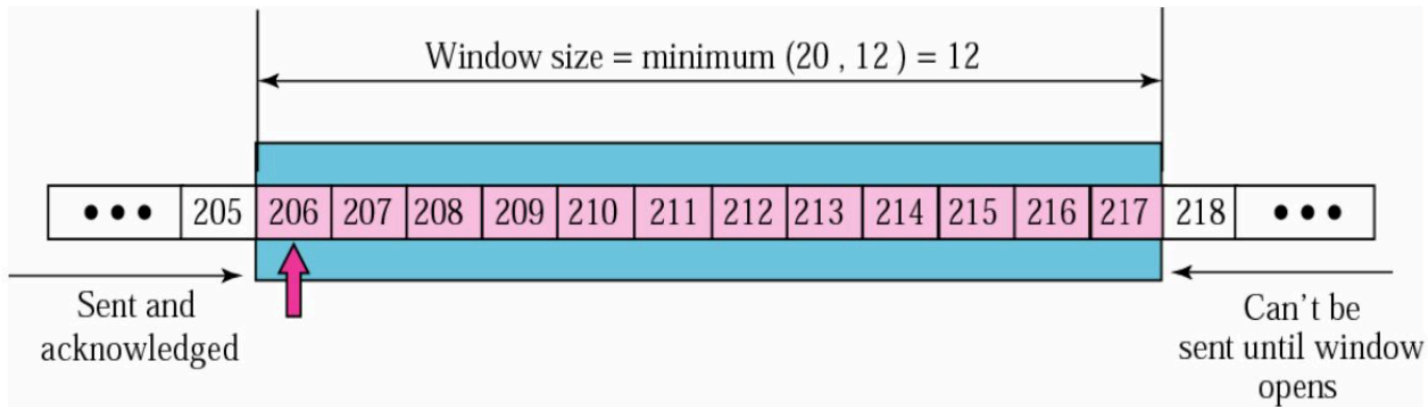
- Sender sent bytes up to 202
 - $cwnd = 20$
 - Receiver sent an ACK (ACK number = 200, $rwnd = 9$)
 - Depict sliding window



- TCP sliding window

- Example

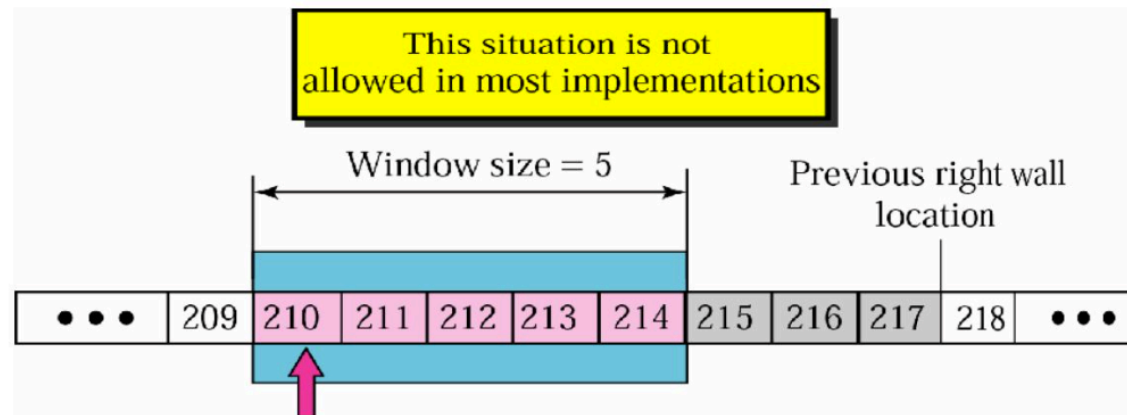
- Server receives a packet (ACK number = 206, $rwnd = 12$)
- $cwnd = 20$ (still)
- Sender has not sent any new bytes
- Depict new window



- TCP sliding window

- Example

- Server receives a packet (ACK number = 210, $rwnd = 5$)
 - $cwnd = 20$ (still)
 - Sender host has sent bytes 206, 207, 208, and 209
 - Depict new window

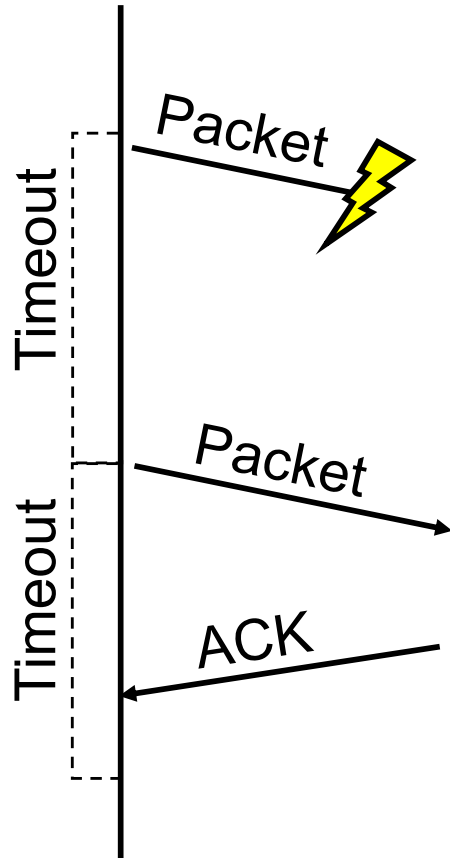


- TCP sliding window
 - Silly window syndrome
 - Problem in sliding window operation
 - Sending process creates data slowly
 - Or, receiving process consume data slowly
 - Or, both
 - Results in sending of data in very small segment
 - Reduces network efficiency
 - E.g., send a one byte segment, results in overhead of 41/1
 - Assuming TCP header is 20 bytes + IP header is 20 bytes

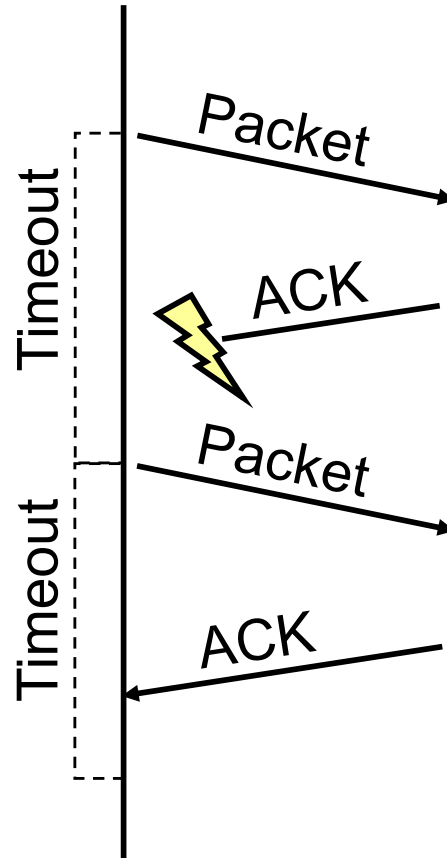
- TCP sliding window
 - Silly window syndrome
 - Syndrome created by sender
 - Nagle's algorithm
 - Syndrome created by receiver
 - Clark's solution
 - Delayed ACK

Optimizing Retransmissions

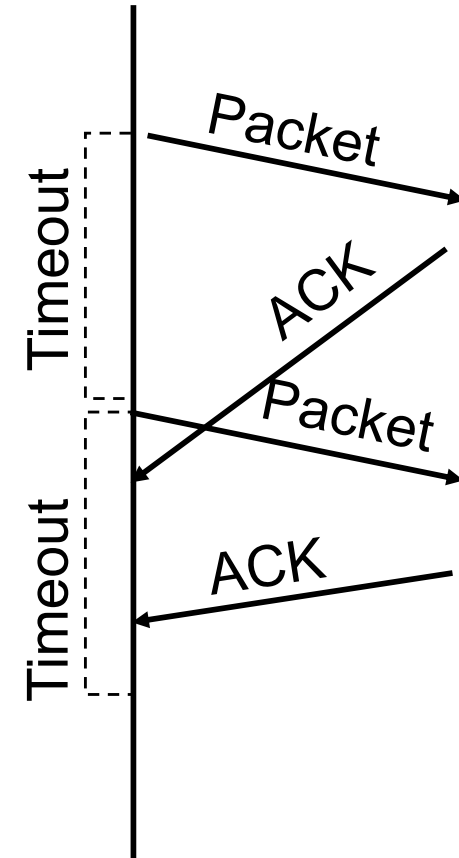
Reasons for Retransmission



Packet lost



**ACK lost
DUPLICATE
PACKET**



**Early timeout
DUPLICATE
PACKETS**

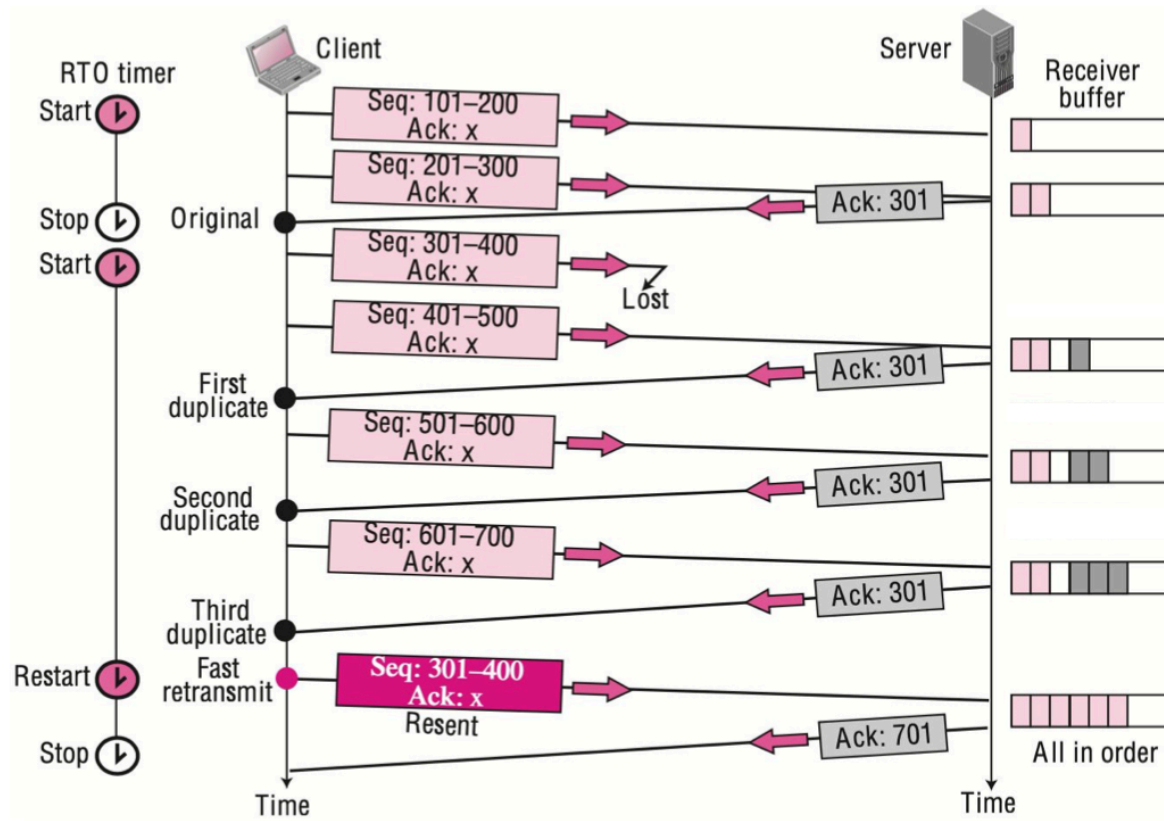
How Long Should Sender Wait?

- **Sender sets a timeout to wait for an ACK**
 - Too short: wasted retransmissions
 - Too long: excessive delays when packet lost
- **TCP sets timeout as a function of the RTT**
 - Expect ACK to arrive after an “round-trip time”
 - ... plus a fudge factor to account for queuing
- **But, how does the sender know the RTT?**
 - Running average of delay to receive an ACK

Still, timeouts are slow (\approx RTT)

- When packet n is lost...
 - ... packets $n+1$, $n+2$, and so on may get through
- Exploit the ACKs of these packets
 - ACK says receiver is still awaiting n th packet
 - Duplicate ACKs suggest later packets arrived
 - Sender uses “duplicate ACKs” as a hint
- Fast retransmission
 - Retransmit after “triple duplicate ACK”

- Fast retransmission



Effectiveness of Fast Retransmit

- **When does Fast Retransmit work best?**
 - High likelihood of many packets in flight
 - Long data transfers, large window size, ...
- **Implications for Web traffic**
 - Many Web transfers are short (e.g., 10 packets)
 - So, often there aren't many packets in flight
 - Making fast retransmit is less likely to “kick in”
 - Forcing users to click “reload” more often...

Effectiveness of Fast Retransmit

- When does Fast Retransmit work best?
 - (A) Short data transfers
 - (B) Large window size
 - (C) Small RTT networks

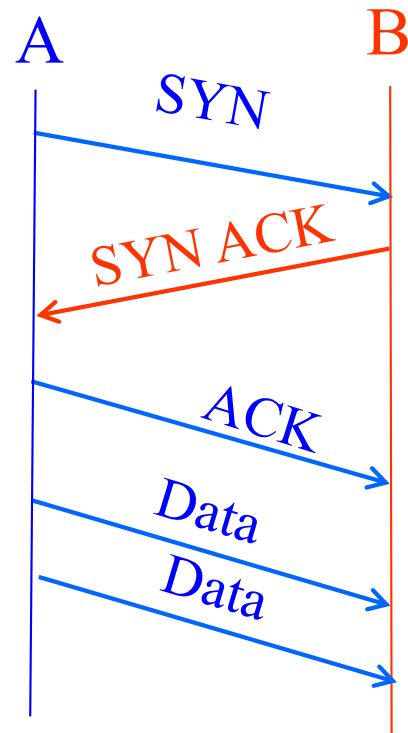
Effectiveness of Fast Retransmit

- When does Fast Retransmit work best?
 - (A) Short data transfers
 - (B) Large window size
 - (C) Small RTT networks

Starting and Ending a Connection: TCP Handshakes

- Phantom/ghost bytes
 - SYN segment
 - Cannot carry data, but consumes **one** SqN
 - SYN + ACK segment
 - Cannot carry data, but consumes **one** SqN
 - ACK segment
 - If carrying no data, consumes **no** SqN
 - FIN segment
 - Consumes **one** SqN, if it does not carry data
 - FIN + ACK segment
 - Consumes **one** SqN, if it does not carry data

Establishing a TCP Connection



Each host tells its ISN to the other host.

- Three-way handshake to establish connection
 - Host A sends a **SYN** (open) to the host B
 - Host B returns a SYN acknowledgment (**SYN ACK**)
 - Host A sends an **ACK** to acknowledge the SYN ACK

TCP Header

Flags: SYN
FIN
RST
PSH
URG
ACK

Source port		Destination port	
Sequence number			
Acknowledgment			
HdrLen	0	Flags	Advertised window
Checksum		Urgent pointer	
Options (variable)			
Data			

Step 1: A's Initial SYN Packet

Flags: SYN
FIN
RST
PSH
URG
ACK

A's port		B's port	
A's Initial Sequence Number			
Acknowledgment			
20	0	Flags	Advertised window
Checksum		Urgent pointer	
Options (variable)			

A tells B it wants to open a connection...

Step 2: B's SYN-ACK Packet

Flags: SYN
FIN
RST
PSH
URG
ACK

B's port		A's port	
B's Initial Sequence Number			
A's ISN plus 1			
20	0	Flags	Advertised window
Checksum		Urgent pointer	
Options (variable)			

**B tells A it accepts, and is ready to hear the next byte...
... upon receiving this packet, A can start sending data**

Step 3: A's ACK of the SYN-ACK

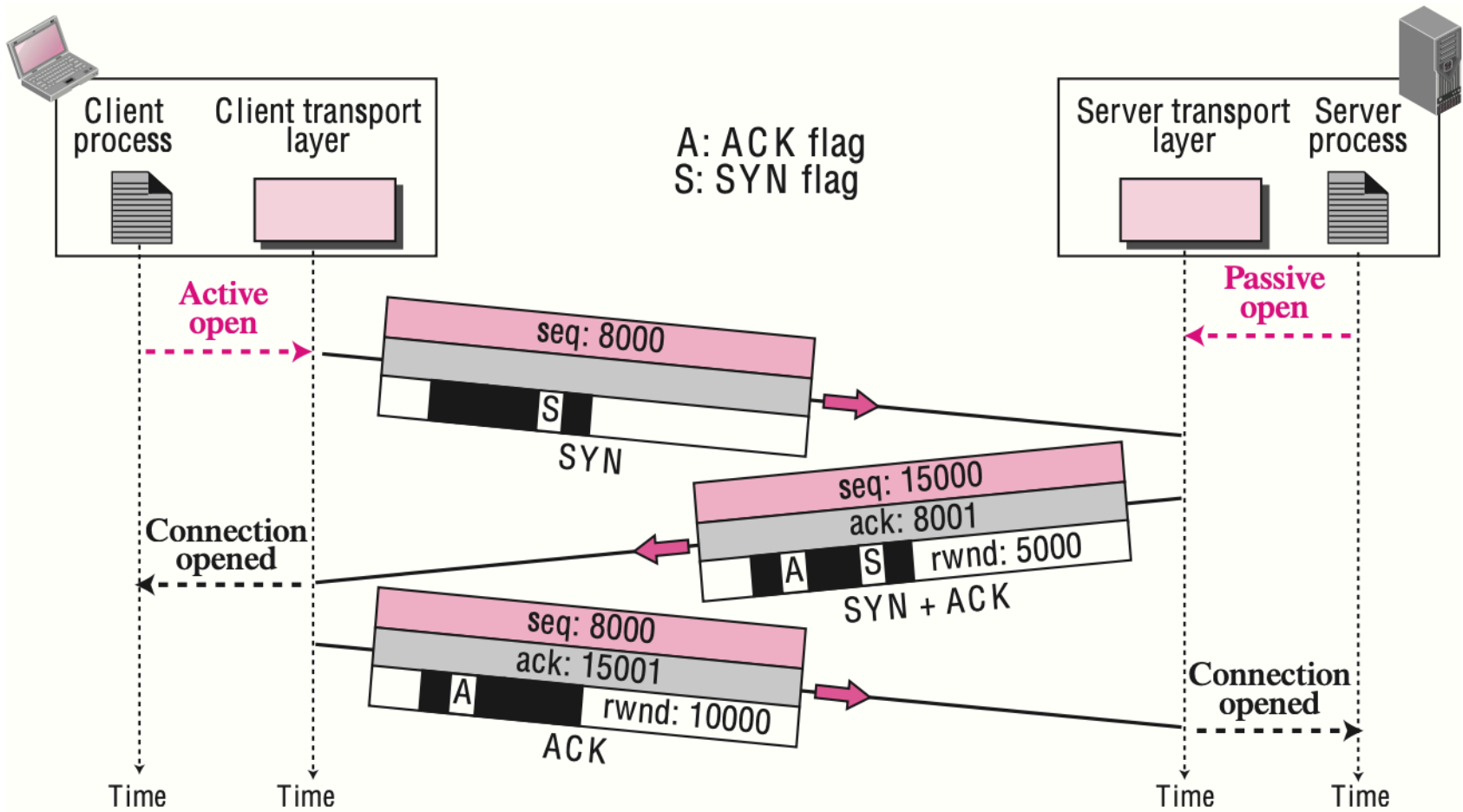
Flags: SYN
FIN
RST
PSH
URG
ACK

A's port		B's port	
Sequence number			
B's ISN plus 1			
20	0	Flags	Advertised window
Checksum		Urgent pointer	
Options (variable)			

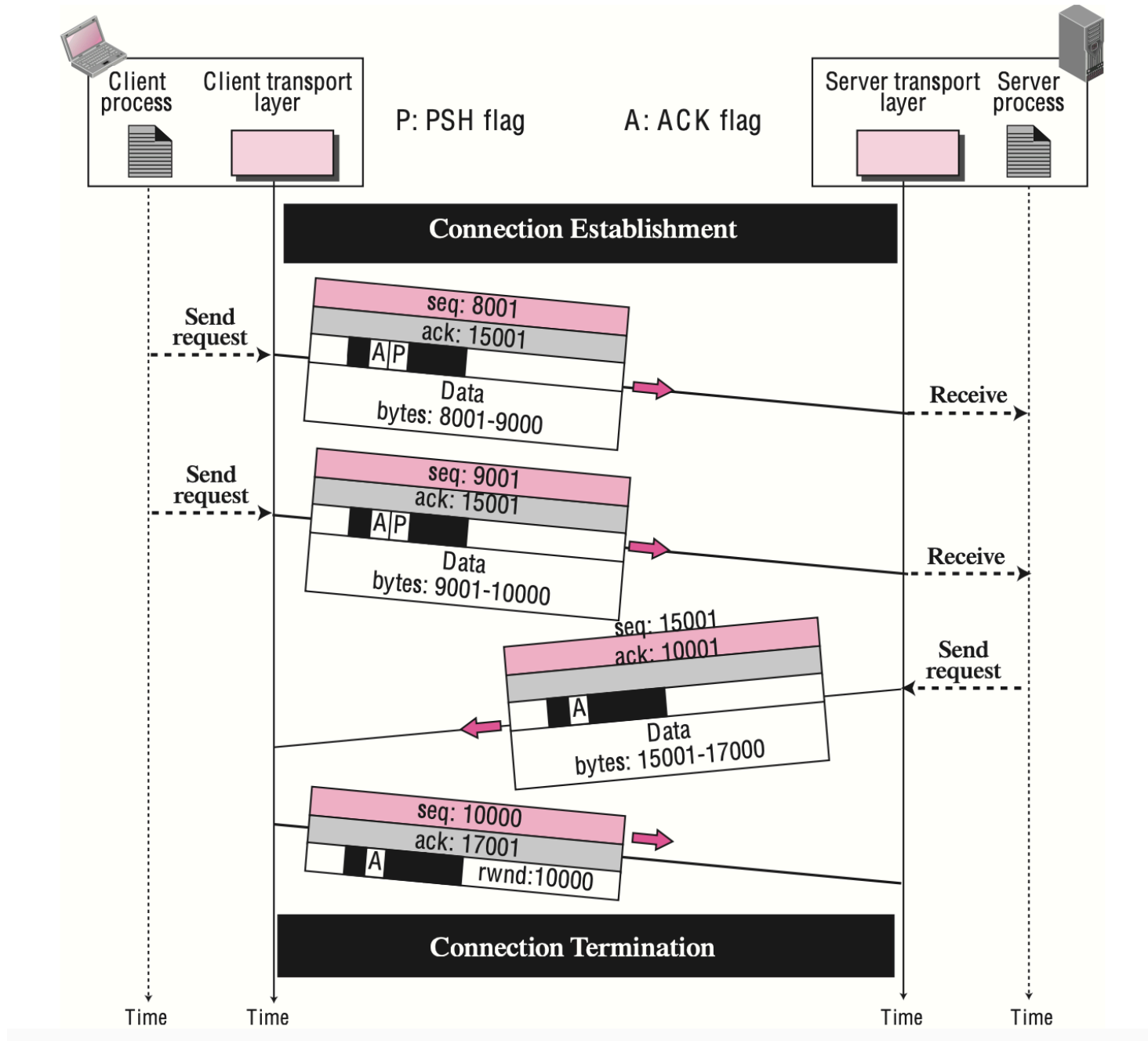
A tells B it is okay to start sending

... upon receiving this packet, B can start sending data

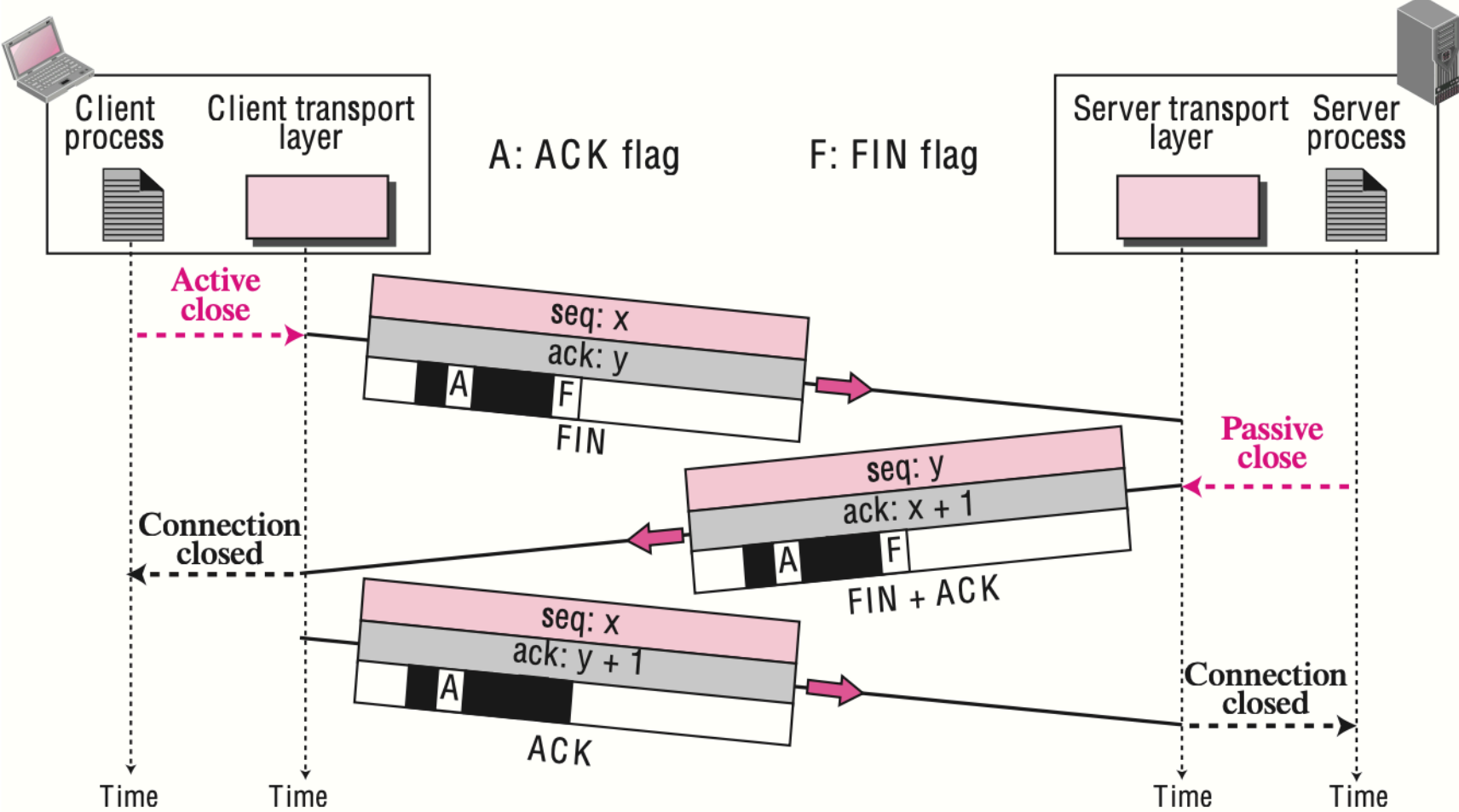
3-way handshake



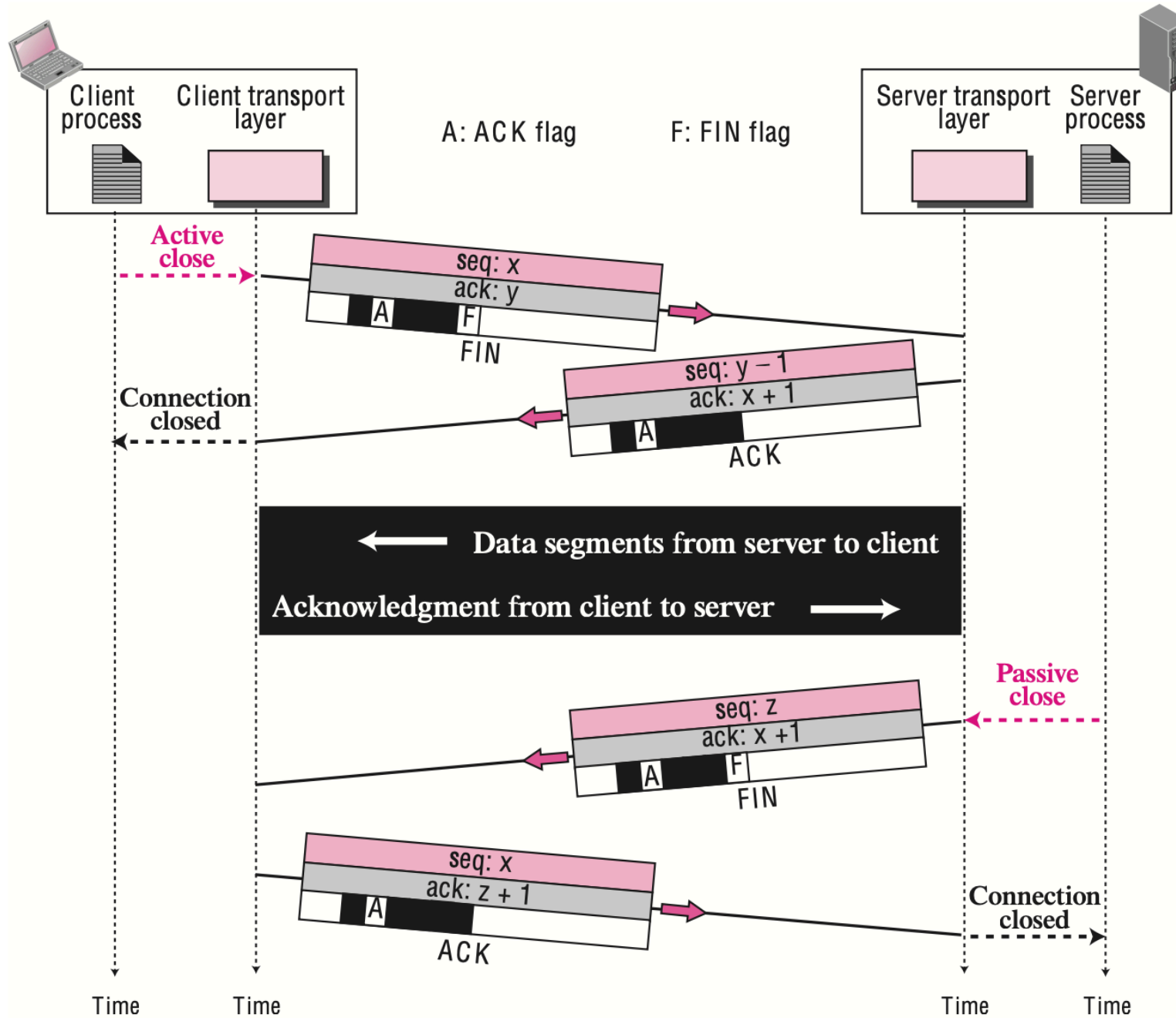
Data transfer



3-way termination



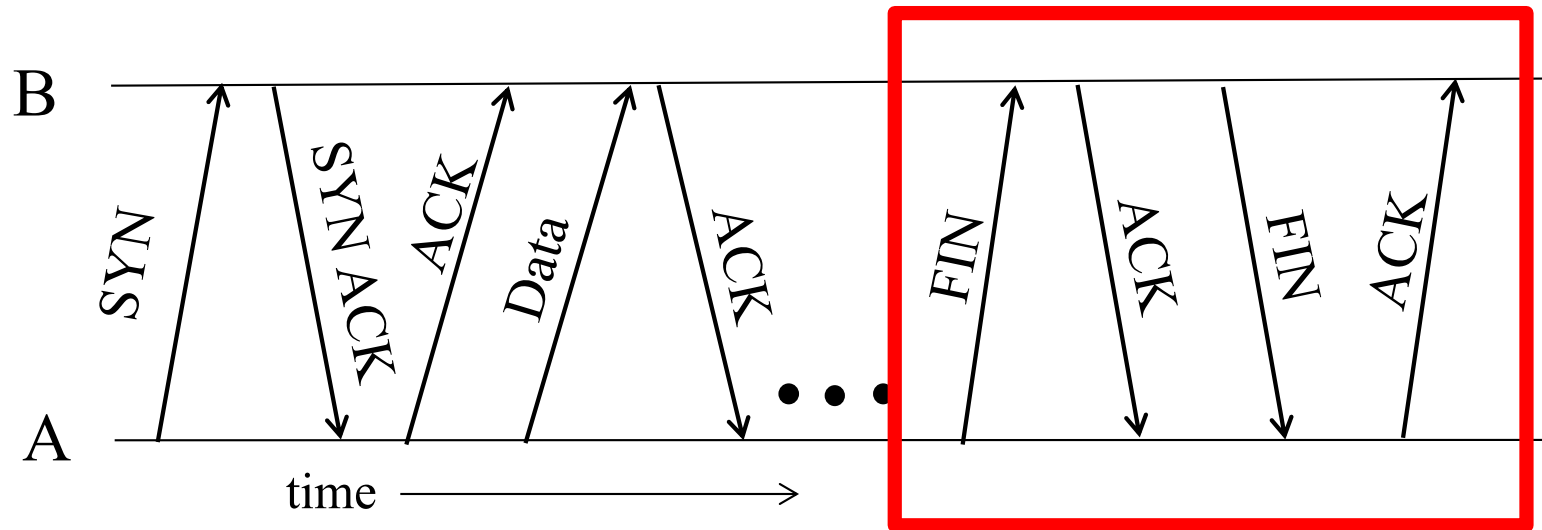
4-way termination or half close



SYN Loss and Web Downloads

- Upon sending SYN, sender sets a timer
 - If SYN lost, timer expires before SYN-ACK received
 - Sender retransmits SYN
- How should the TCP sender set the timer?
 - No idea how far away the receiver is
 - Some TCPs use default of 3 or 6 seconds
- Implications for web download
 - User gets impatient and hits reload
 - ... Users aborts connection, initiates new socket
 - Essentially, forces a fast send of a new SYN!

Tearing Down the Connection



- **Closing (each end of) the connection**
 - Finish (FIN) to close and receive remaining bytes
 - And other host sends a FIN ACK to acknowledge
 - Reset (RST) to close and not receive remaining bytes

Sending/Receiving the FIN Packet

- **Sending a FIN: close()**
 - Process is done sending data via socket
 - Process invokes “close()”
 - Once TCP has sent all the outstanding bytes...
 - ... then TCP sends a FIN
- **Receiving a FIN: EOF**
 - Process is reading data from socket
 - Eventually, read call returns an EOF

Conclusions

- **Transport protocols**
 - Multiplexing and demultiplexing
 - Checksum-based error detection
 - Sequence numbers
 - Retransmission
 - Window-based flow control