ECE 363
Communication Networks

Transport Layer

Transport Layer Services

- Transport layer: communication between processes
  - Relies on and enhances network layer services
- Network layer: communication between hosts

- Goal: turn the less-than-desirable properties of the underlying network into the high level of service required by application programs
- Provide end-to-end communications between processes on hosts
  - Connection-oriented and reliable (bytestream)
  - Connectionless and unreliable (messages)

Transport Layer Versus Network Layer

- The transport layer code runs entirely on user machines, but the network layer code mostly runs on routers which are usually operated by ISPs and institutions
- Network layer has problems (losing packets, routers crashing, links failing …)
- The transport layer improves the QoS of the network layer
- The transport service is more reliable than the network service
- Application programmers can write code according to a standard set of transport service primitives and have these programs work on a wide variety of networks

End-to-End Protocols

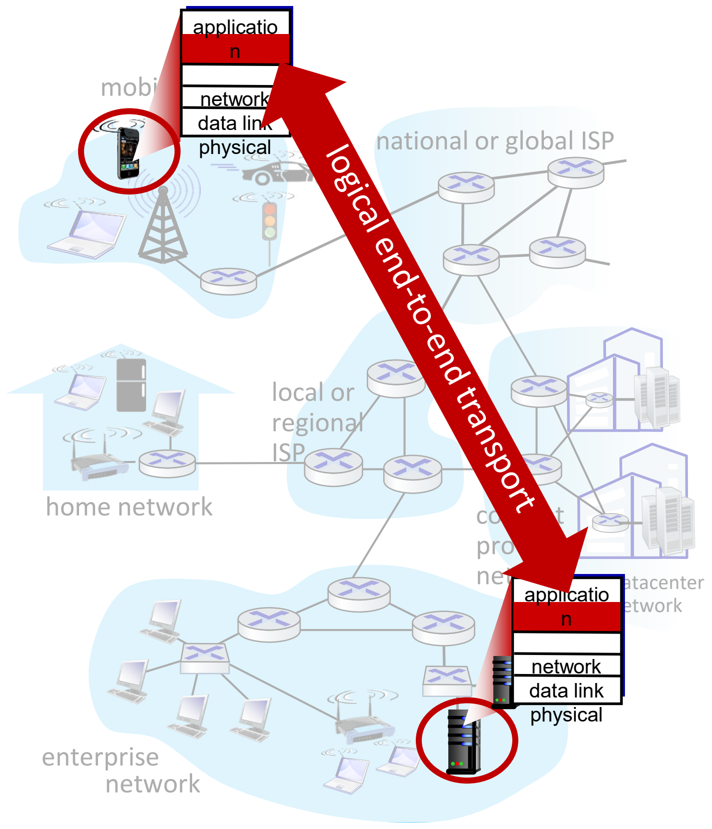
- Common properties that a transport protocol can be expected to provide
  - Guarantee segment delivery
  - Deliver segments in the same order they were sent
  - Deliver at most one copy of each segment
  - Support arbitrarily large segments
  - Support synchronization between the sender and receiver
  - Allow the receiver to apply flow control to the sender
  - Support multiple application processes on each host

End-to-End Protocols

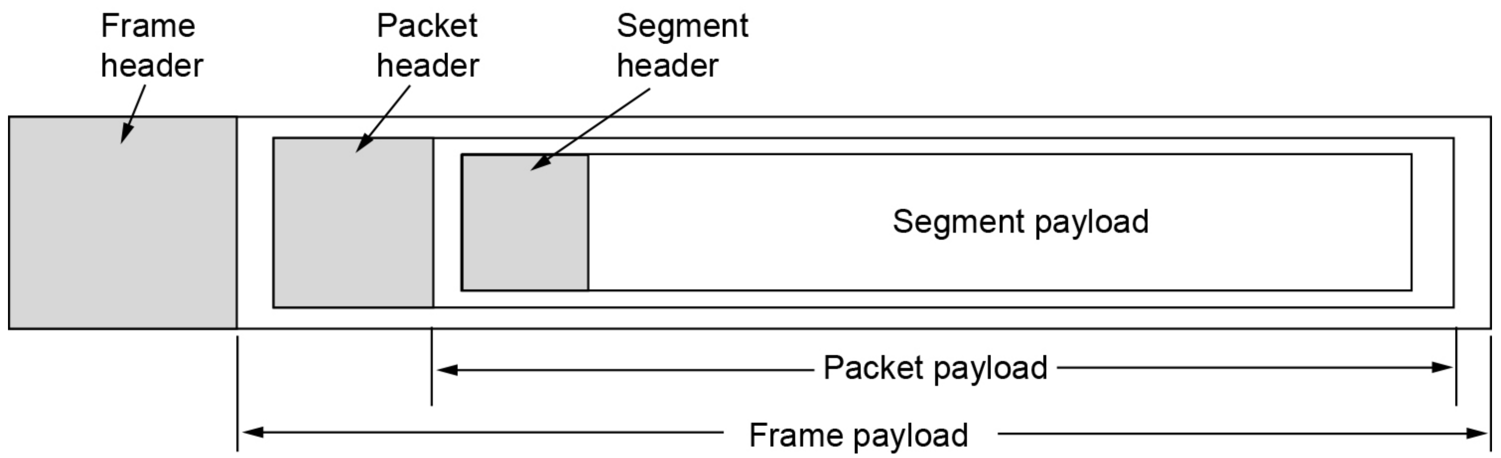
- Typical limitations of the network on which the transport protocol operates
  - Dropped segments
  - Reordered segments
  - Duplicate copies of a segment
  - Limited segment sizes
  - Segments delivered after an arbitrarily long delay

Transport Services and Protocols

- Provide logical communication between application processes running on different hosts
- Transport protocol actions in end systems
  - Sender: break application messages into segments and pass them to the network layer
  - Receiver: reassemble segments into messages and pass them to the application layer
- Two transport protocols available to Internet applications
  - TCP, UDP



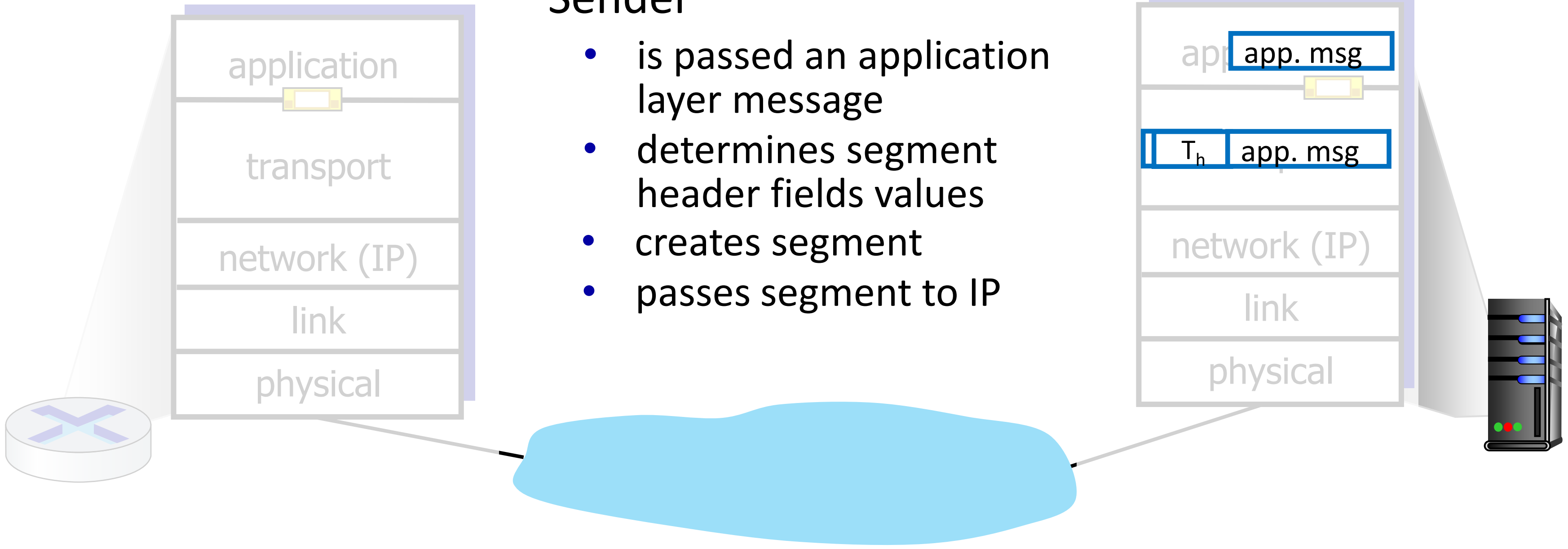
Nesting of Segments, Packets, and Frames



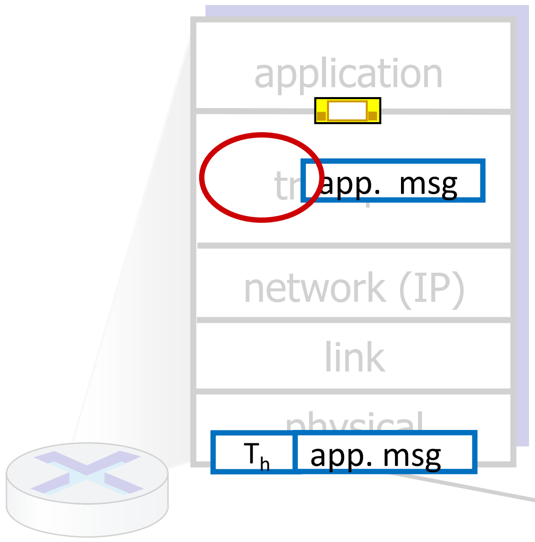
Transport Layer

Sender

- is passed an application layer message
- determines segment header fields values
- creates segment
- passes segment to IP

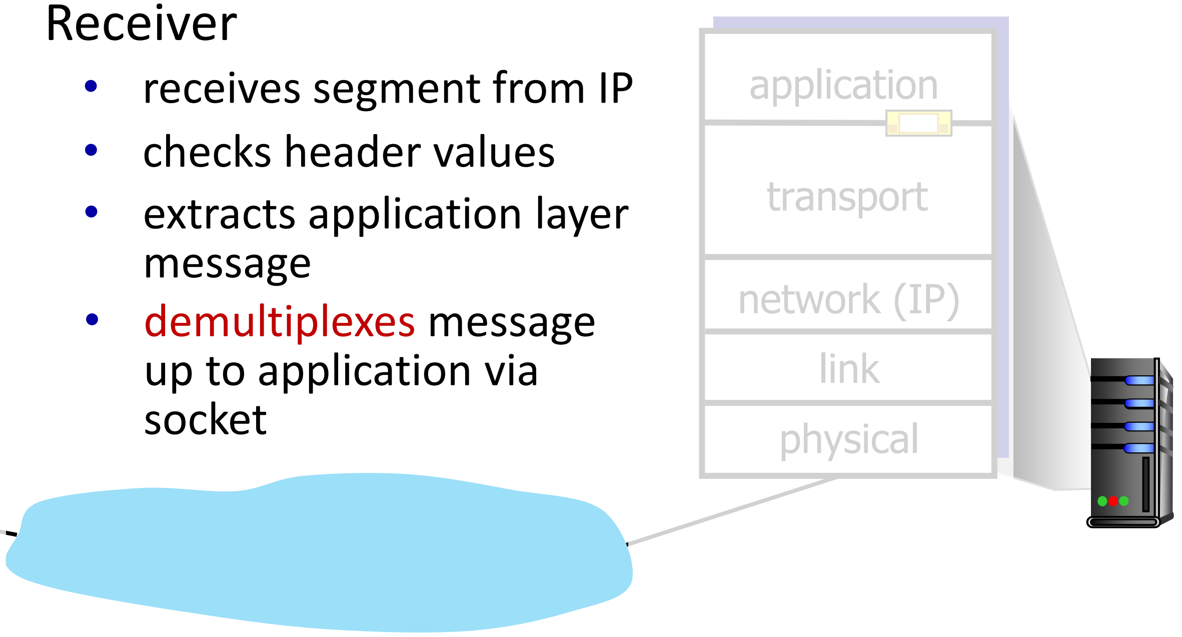


Transport Layer



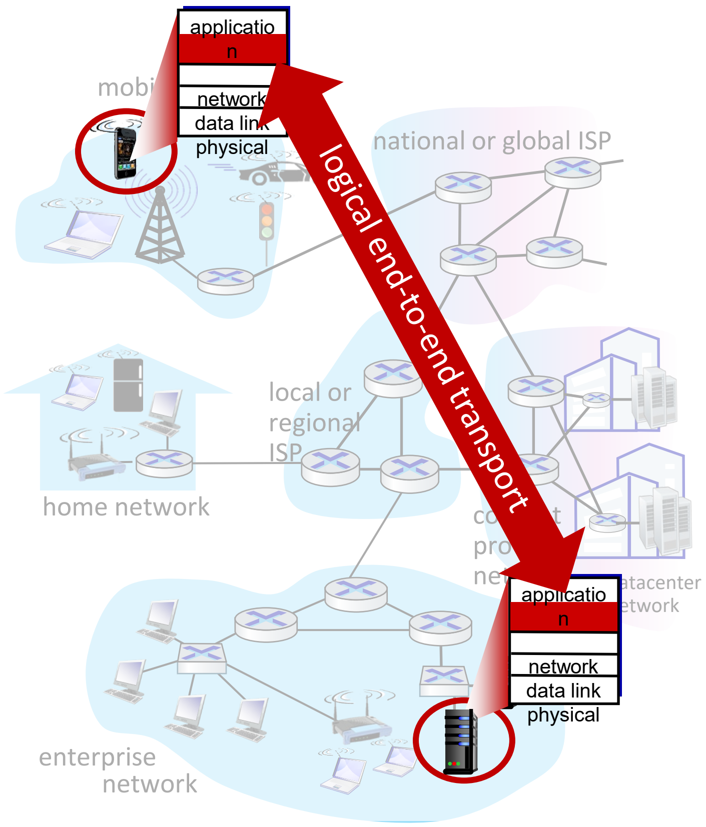
Receiver

- receives segment from IP
- checks header values
- extracts application layer message
- demultiplexes message up to application via socket



Two Principal Internet Transport Protocols

- TCP: Transmission Control Protocol
  - reliable, in-order delivery
  - congestion control
  - flow control
  - connection-oriented
- UDP: User Datagram Protocol
  - unreliable, unordered delivery
  - no-frills extension of best-effort IP
  - connectionless
- Services not provided
  - delay guarantees
  - bandwidth guarantees



Comparison of Internet Transport Protocols

TCP is full-featured, UDP is a glorified packet

TCP (Streams)	UDP (Datagrams)
Connection-oriented	Connectionless
Bytes are delivered, once, reliably, and in order	Messages may be lost, reordered, or duplicated
Arbitrary length content	Limited message size
Flow control matches sender to receiver	Can send regardless of receiver state
Congestion control matches sender to network	Can send regardless of network state
HTTP, HTTPS, FTP, SSH	DNS, VoIP, Streaming, Gaming

Berkeley Socket Primitives

Primitive	Meaning
SOCKET	Create a new communication endpoint
BIND	Associate a local address with a socket
LISTEN	Announce willingness to accept connections; give queue size
ACCEPT	Passively establish an incoming connection
CONNECT	Actively attempt to establish a connection
SEND	Send some data over the connection
RECEIVE	Receive some data from the connection
CLOSE	Release the connection

Socket API

TCP

- Server
  - SOCKET, BIND, LISTEN
- Client
  - SOCKET, CONNECT
- Server
  - ACCEPT
- Client and Server
  - SEND, RECEIVE, CLOSE

UDP

- Server
  - SOCKET, BIND
- Client
  - SOCKET
- Client and Server
  - SENDTO, RECEIVEFROM, CLOSE

Port Numbers

- Port number (16 bits)
  - Source and destination port numbers
  - Used for multiplexing and demultiplexing
- Port number allocation (iana.org)
  - Well-known port numbers 0-1023 (privileged)
    - 80: http, 443: https, 53 DNS
  - Registered port numbers 1024-49151
    - MySQL: 3306
  - Dynamic /ephemeral port numbers 49152-65535

Some Assigned Ports

Port	Protocol	Use
20, 21	FTP	File transfer
22	SSH	Remote login, replacement for Telnet
25	SMTP	Email
80	HTTP	World Wide Web
110	POP-3	Remote email access
143	IMAP	Remote email access
443	HTTPS	Secure Web (HTTP over SSL/TLS)
543	RTSP	Media player control
631	IPP	Printer sharing

Multiplexing/Demultiplexing

- IP only delivers data between end systems identified with a unique IP address
- IP does not deliver data between application processes
- Demultiplexing: delivering the data in a transport layer segment to the correct application process
- Multiplexing: gathering data at the source host from different application processes, enveloping data with header information to create segments, and passing the segments to the network layer

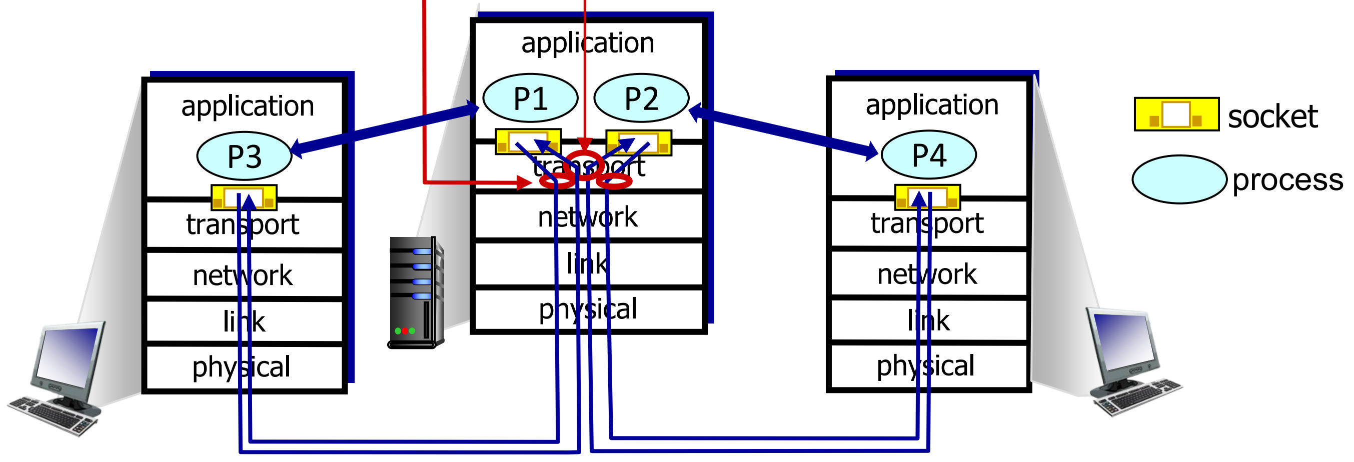
Multiplexing/Demultiplexing

Multiplexing as sender

Handle data from multiple sockets, add transport headers (later used for demultiplexing)

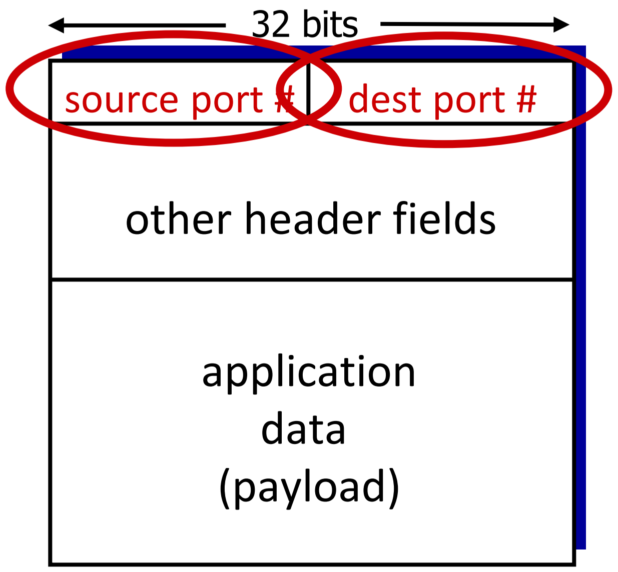
Demultiplexing as receiver

Use header information to deliver received segments to the correct socket



How Demultiplexing Works

- Host receives IP packets
  - Each packet has source IP address, destination IP address
  - Each packet carries one transport layer segment
  - Each segment has source port number, destination port number
- Host uses IP addresses and port numbers to direct segments to the appropriate socket

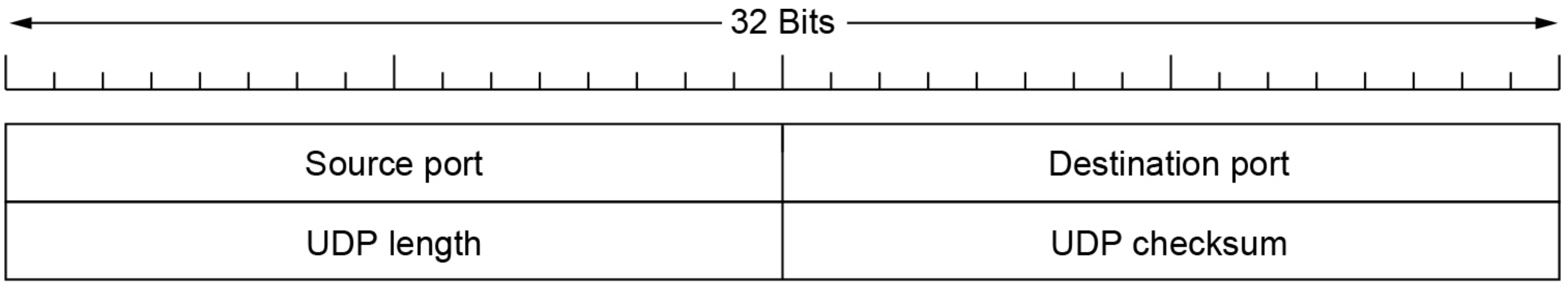


TCP/UDP segment format

UDP: User Datagram Protocol

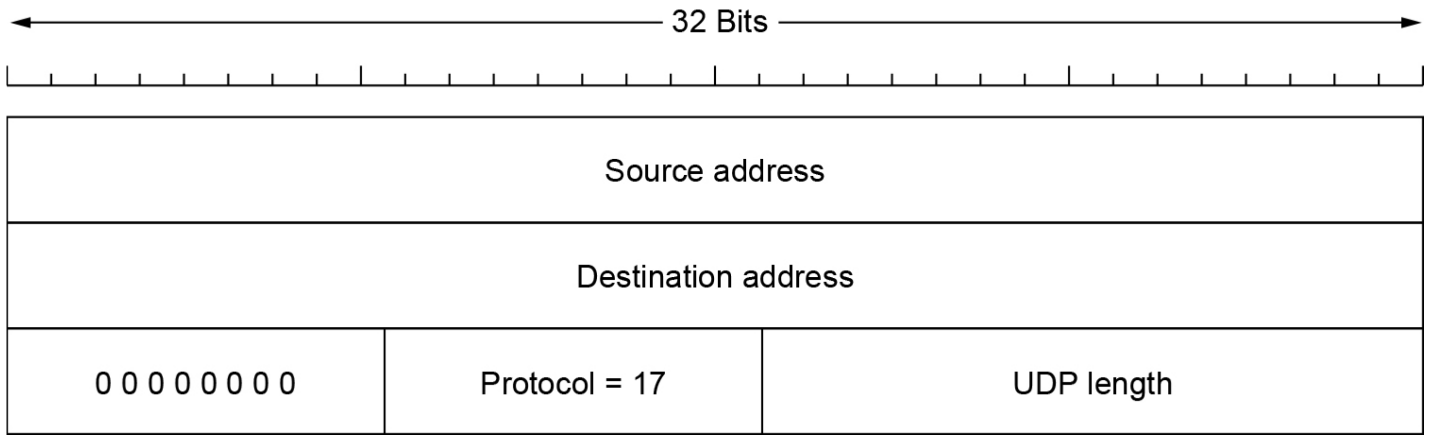
- Extends host-to-host delivery service of the underlying network into a process-to-process communication service
- Adds a level of demultiplexing which allows multiple application processes on each host to share the network
- UDP is used for
  - Streaming multimedia applications
    - loss tolerant, rate sensitive
  - DNS, SNMP, DHCP

UDP Header

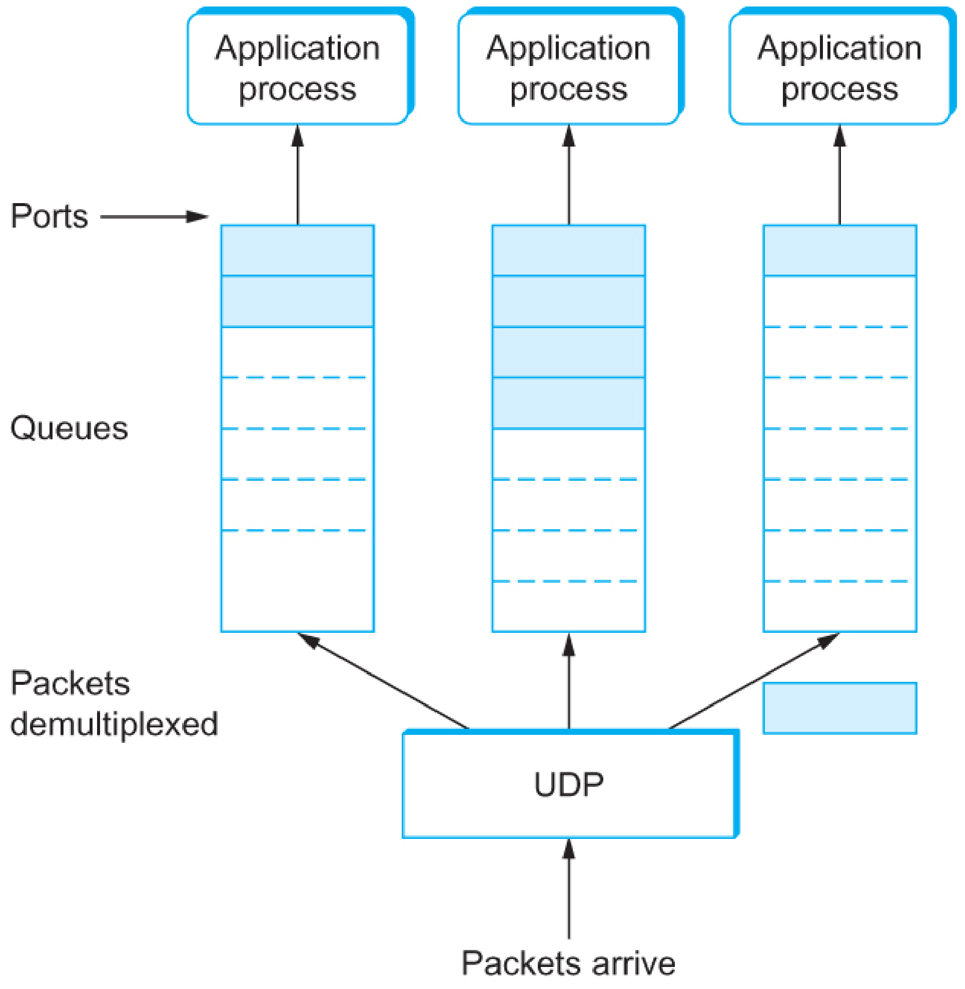


- UDP length, in bytes, of the UDP segment including the header
- UDP checksum is the 1's complement sum (similar to IP) of the UDP header, payload, and a pseudo-header derived from the IP addresses

IPv4 Pseudoheader Included in the UDP Checksum



UDP Buffering



UDP: User Datagram Protocol

- UDP is a no frills protocol
  - Segments may be lost, delivered out of order
  - Best effort service: send and hope for the best
- UDP has its plusses
  - No setup/handshaking needed (no RTT incurred)
  - Can function when network service is degraded
  - Provides some reliability (checksum)
- Can build additional functionality on top of UDP in the application layer, e.g. HTTP/3
  - Used with high-latency networks, mobile roaming scenarios, and packet-loss environments
  - It is particularly beneficial for video streaming, large content delivery, and mobile-heavy applications

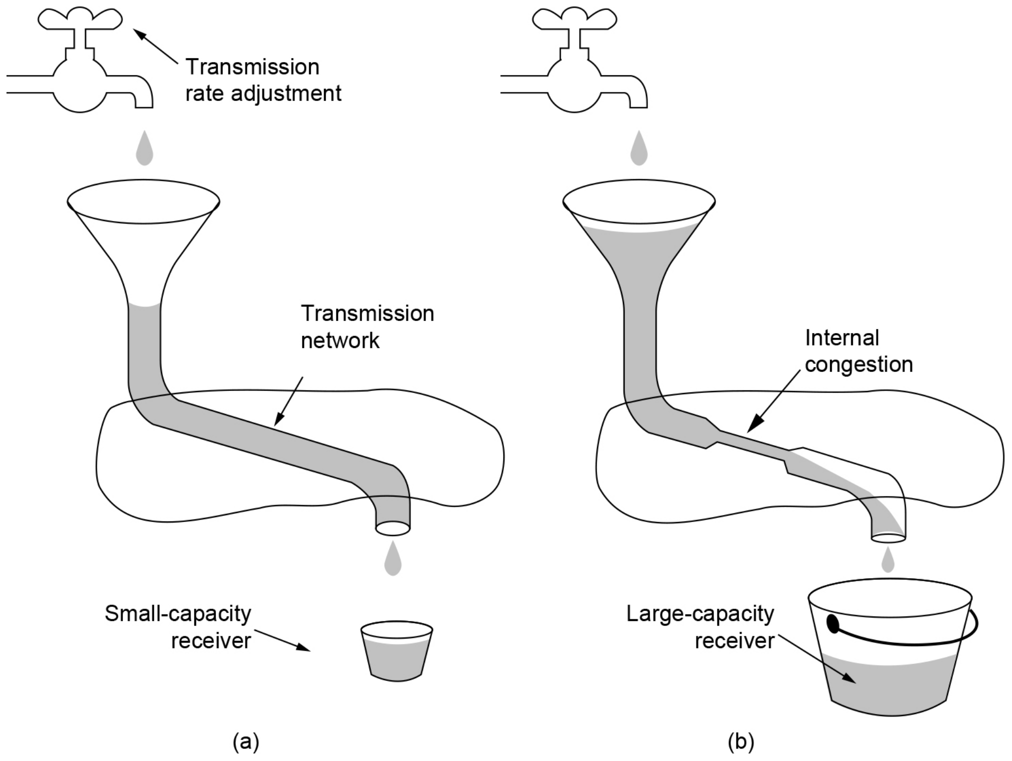
Internet Transmission Control Protocol (TCP)

- In contrast to UDP, TCP offers a reliable, connection-oriented, bytestream service with
  - Addressing
  - Multiplexing
  - Connection establishment and release
    - Problem: Delayed and duplicate packets
    - Solution: Three-way handshake
  - Error control and flow control
  - Congestion control

Flow Control Versus Congestion Control

- Flow control involves preventing senders from overrunning the capacity of the receivers
- Congestion control involves preventing too much data from being injected into the network, thereby causing routers or links to become overloaded

Flow Control Versus Congestion Control



(a) A fast network with a low-capacity receiver.

(b) A slow network with a high-capacity receiver.

End-to-End Issues

- At the heart of TCP is the sliding window algorithm (similar to the Link Layer)
- As TCP runs over the Internet rather than a point-to-point link, the following issues need to be addressed by the sliding window algorithm
  - TCP supports logical connections between processes that are running on two different computers in the Internet
  - TCP connections may have widely different RTT times
  - Segments can arrive out of order
  - Segments can be delayed
  - Retransmissions may include different byte ranges than the original transmission

TCP Connection Identifier

- A TCP connection is uniquely defined by

(Source IP, Source Port, Destination IP, Destination Port)

- This is often called the socket pair or 4-tuple
- In rare situations
    - e.g. NAT, packet filtering, and firewalls

  a 5-tuple is often used

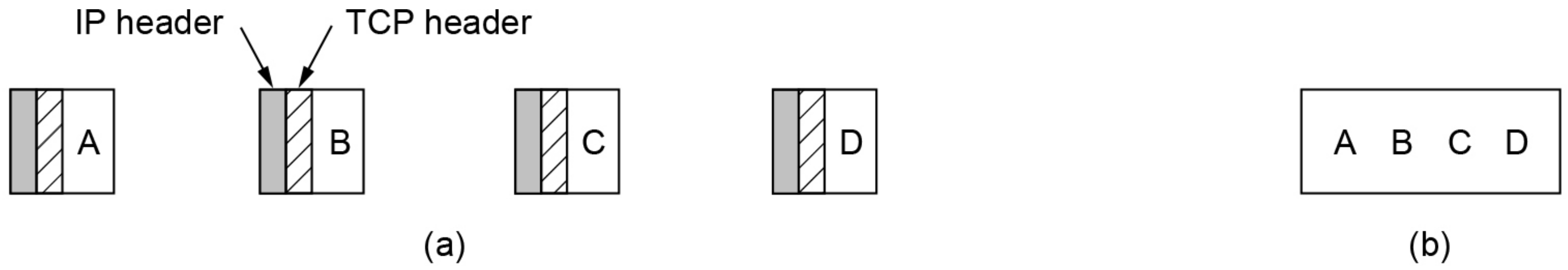
(Protocol, Source IP, Source Port, Destination IP, Destination Port)

- The additional field (Protocol) distinguishes TCP from UDP
- For TCP alone, the protocol is implicit

Some Assigned Ports

Port	Protocol	Use
20, 21	FTP	File transfer
22	SSH	Remote login, replacement for Telnet
25	SMTP	Email
80	HTTP	World Wide Web
110	POP-3	Remote email access
143	IMAP	Remote email access
443	HTTPS	Secure Web (HTTP over SSL/TLS)
543	RTSP	Media player control
631	IPP	Printer sharing

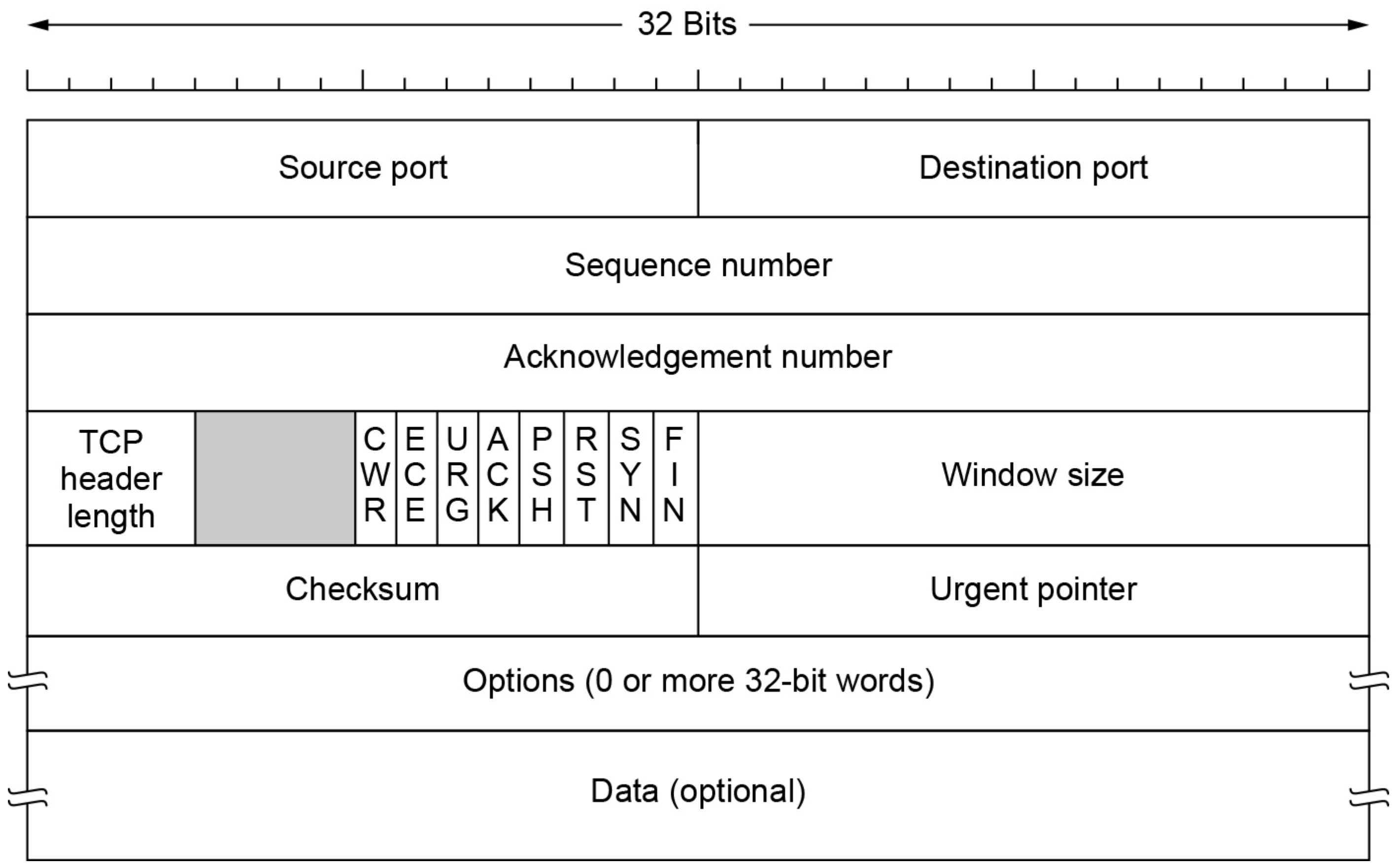
The TCP Service Model



(a) Four 512-byte segments sent as separate IP datagrams.

(b) The 2048 bytes of data delivered to the application in a single READ call.

TCP Segment Header



TCP Segment Header

- The Source Port and Destination Port fields identify the source and destination ports, respectively.
- The Sequence Number, Acknowledgement Number, and Window Size fields are used in the TCP sliding window algorithm
- Each data byte corresponds to a sequence number
  - Sequence Number is the sequence number for the first byte of data carried in that segment
    - Exception: SYN/FIN sequence numbers
  - Acknowledgment Number is the next in-order byte expected
- Window Size is used for flow control

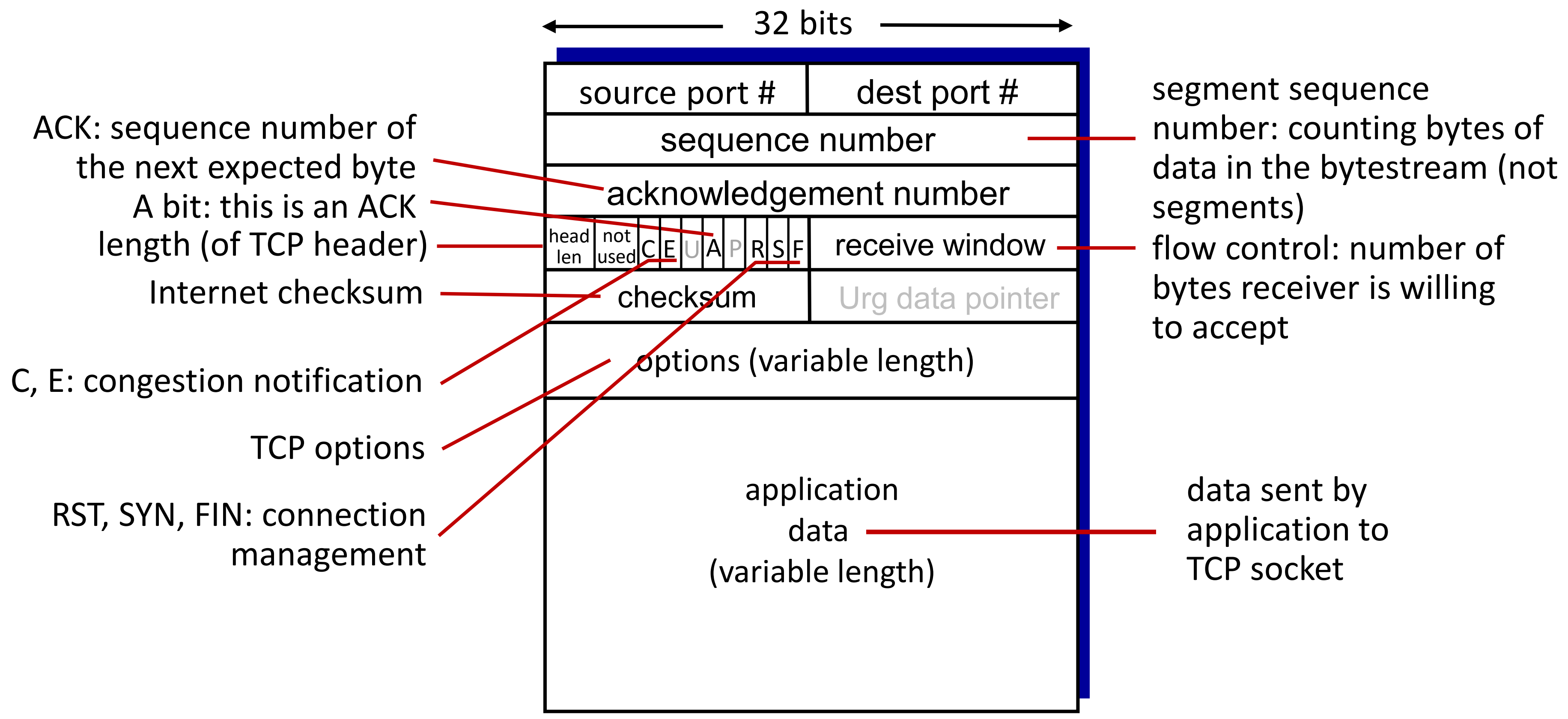
TCP Segment Header

- There are eight Flag bits CWE, ECE, SYN, FIN, RST, PSH, URG, and ACK
- The SYN and FIN flags are used when establishing and terminating a TCP connection, respectively
- The ACK flag is set when the Acknowledgment field is valid
- The URG flag signifies that this segment contains urgent data. When this flag is set the Urgent Pointer indicates a byte offset from the current sequence number
- The PSH flag signifies that the sender invoked the push operation, which indicates to the receiver TCP that it should notify the receiving process of this fact
- The RST flag signifies that the receiver has become confused, e.g. it received a segment it did not expect to receive and so wants to abort the connection
- THE CWE and ECE flag bits are used for congestion control

TCP Segment Header

- The Checksum is similar to that for UDP
- It is computed over the TCP header, the TCP data, and the pseudoheader
- The pseudoheader is made up of the
  - Source IP address
  - Destination IP address
  - Protocol = 6
  - TCP segment length = TCP header length + TCP payload length
- The checksum is mandatory

TCP Segment Structure

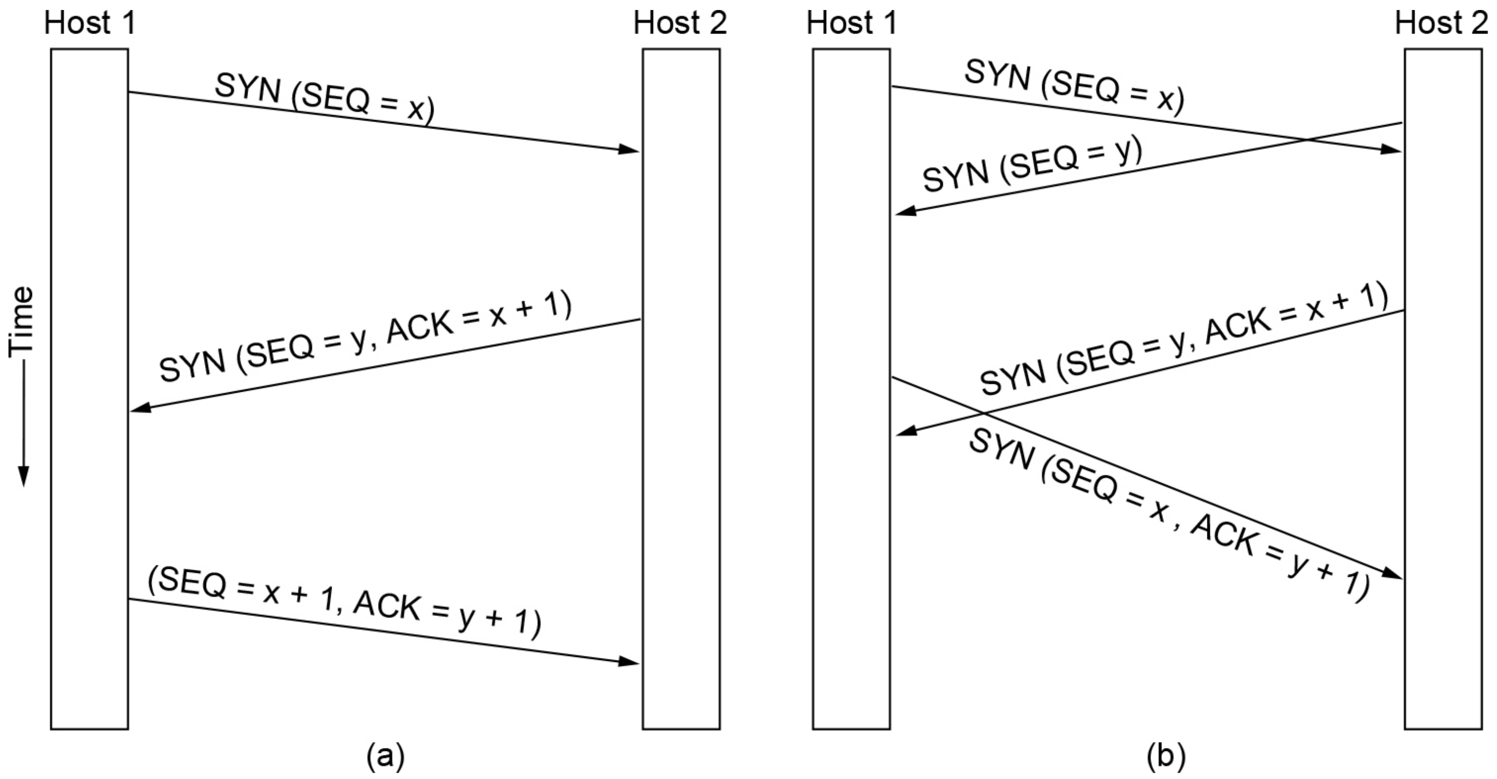


32 bits
source port #
dest port #
sequence number
acknowledgement number
head len
not used
C
E
U
A
P
R
S
F
receive window
checksum
Urg data pointer
options (variable length)
application data (variable length)
ACK: sequence number of the next expected byte
A bit: this is an ACK
length (of TCP header)
Internet checksum
C, E: congestion notification
TCP options
RST, SYN, FIN: connection management
segment sequence number: counting bytes of data in the bytestream (not segments)
flow control: number of bytes receiver is willing to accept
data sent by application to TCP socket

TCP Sequence and Acknowledgement Numbers

- Sequence number is 32 bits
  - Sequence number for the first byte in the payload
    - Exception: SYN/FIN sequence numbers
  - Random initial sequence numbers
    - Exchanged during connection establishment
- Acknowledgment number is 32 bits
  - Sequence number for the next byte to expect

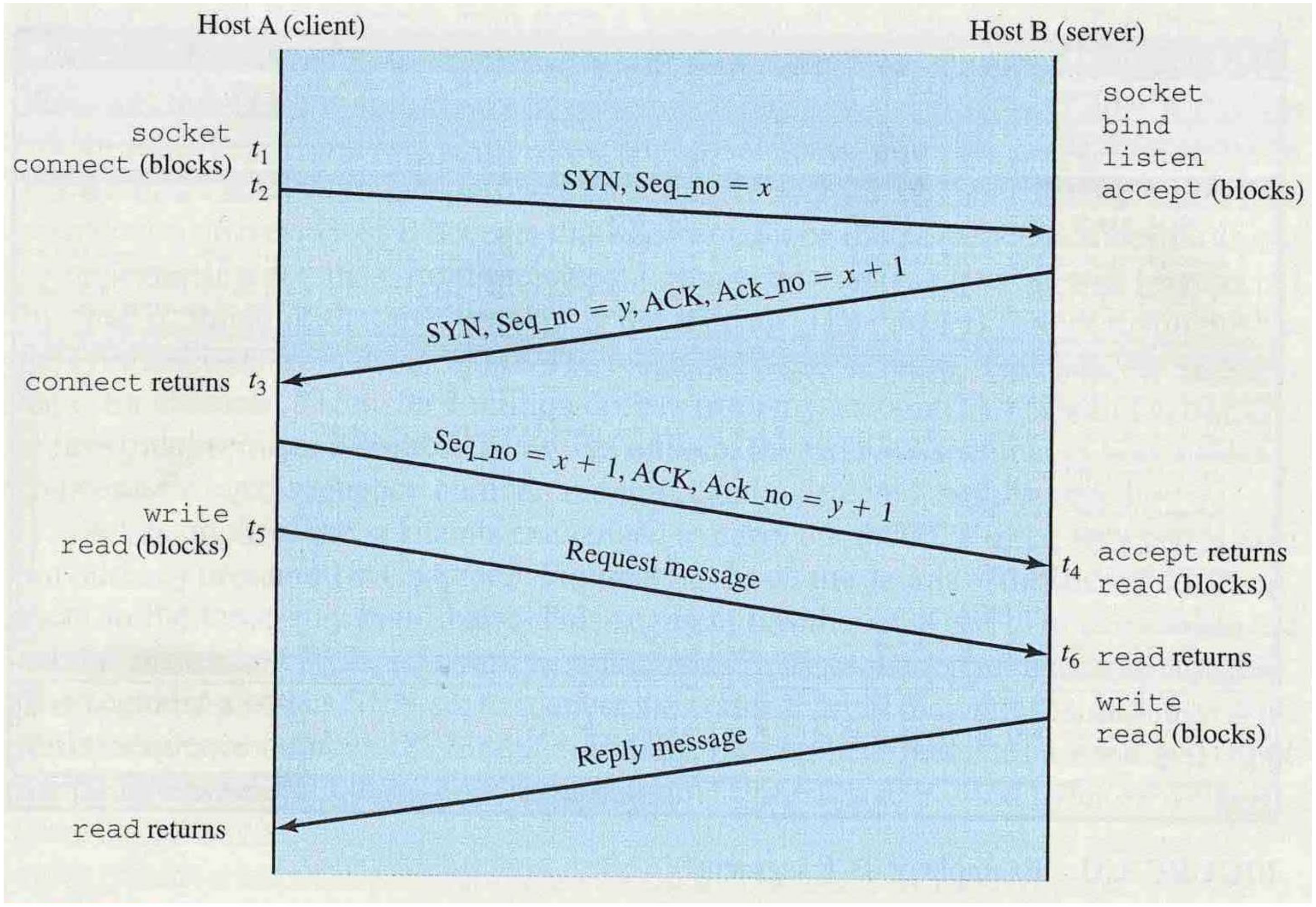
TCP Connection Establishment


Host 1
Host 2
SYN (SEQ = x)
SYN (SEQ = y, ACK = x + 1)
(SEQ = x + 1, ACK = y + 1)
Time
(a)
Host 1
Host 2
SYN (SEQ = x)
SYN (SEQ = y)
SYN (SEQ = y, ACK = x + 1)
SYN (SEQ = x , ACK = y + 1)
(b)

(a) TCP connection establishment in the normal case.

(b) Simultaneous connection establishment on both sides.

TCP Connections



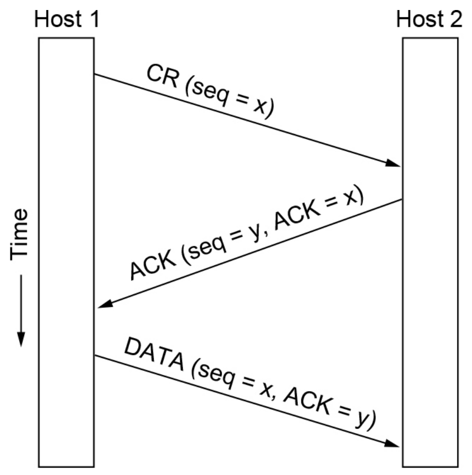
TCP Connection Establishment

Three protocol scenarios for establishing a connection using a three-way handshake. CR denotes CONNECTION REQUEST

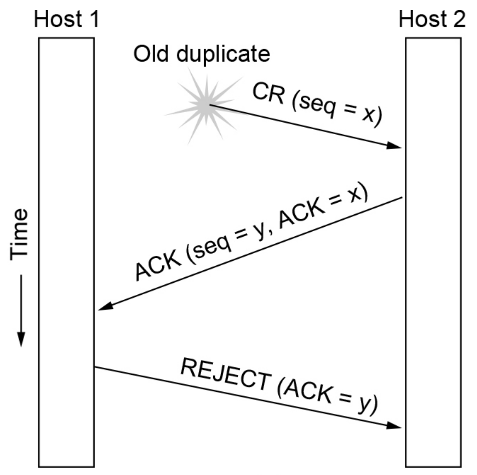
(a) Normal operation

(b) Old duplicate CONNECTION REQUEST appearing out of nowhere

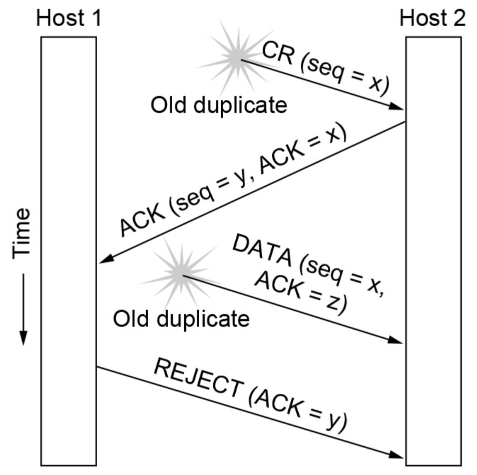
(c) Duplicate CONNECTION REQUEST and duplicate ACK



(a)

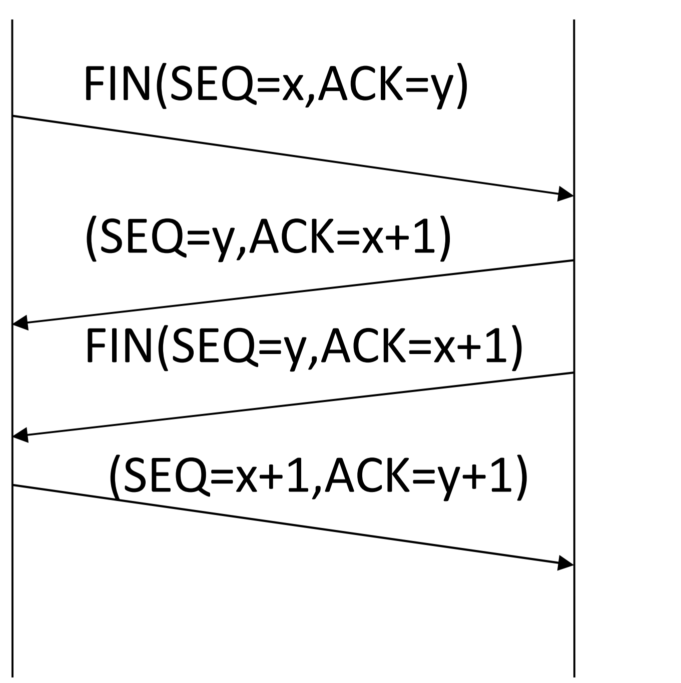


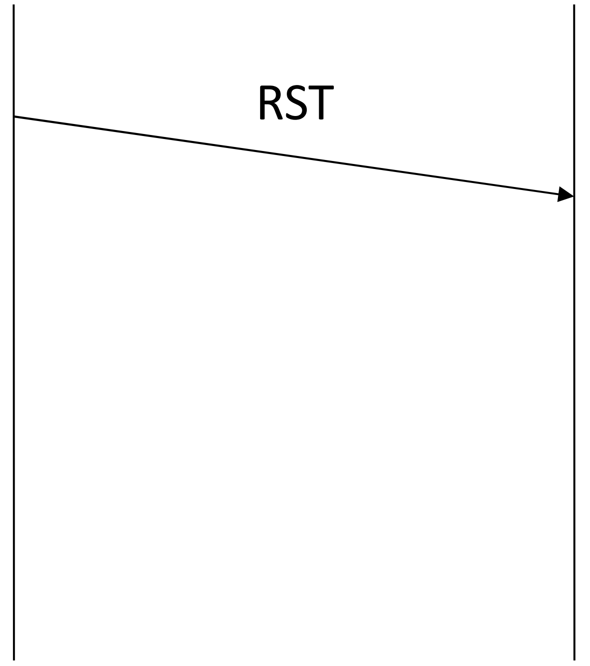
(b)



(c)

TCP Connection Release





(a) 2-way handshake in each direction

(b) Connection reset

TCP Connection Release

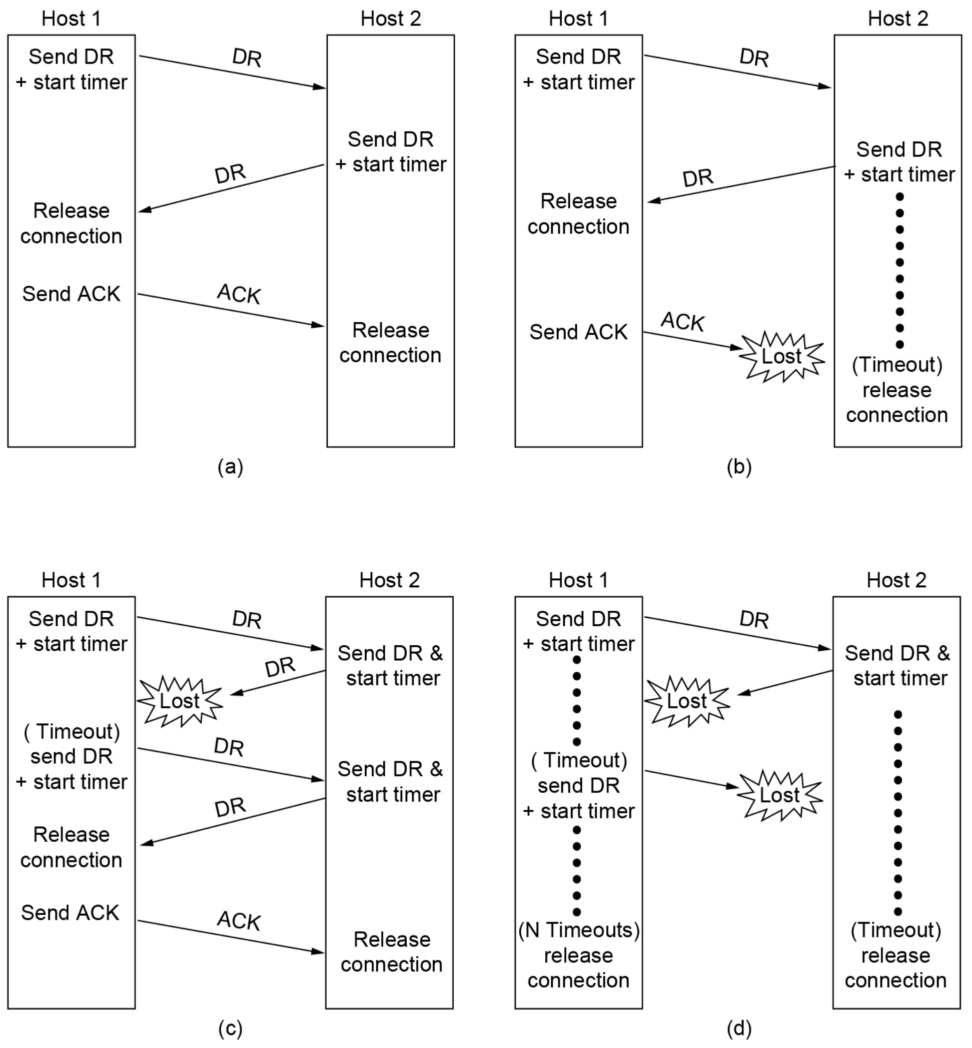
Four protocol scenarios for releasing a connection. DR denotes DISCONNECTION REQUEST

(a) Normal case of three-way handshake

(b) Final ACK lost

(c) Response lost

(d) Response lost and subsequent DRs lost



TCP Sliding Window

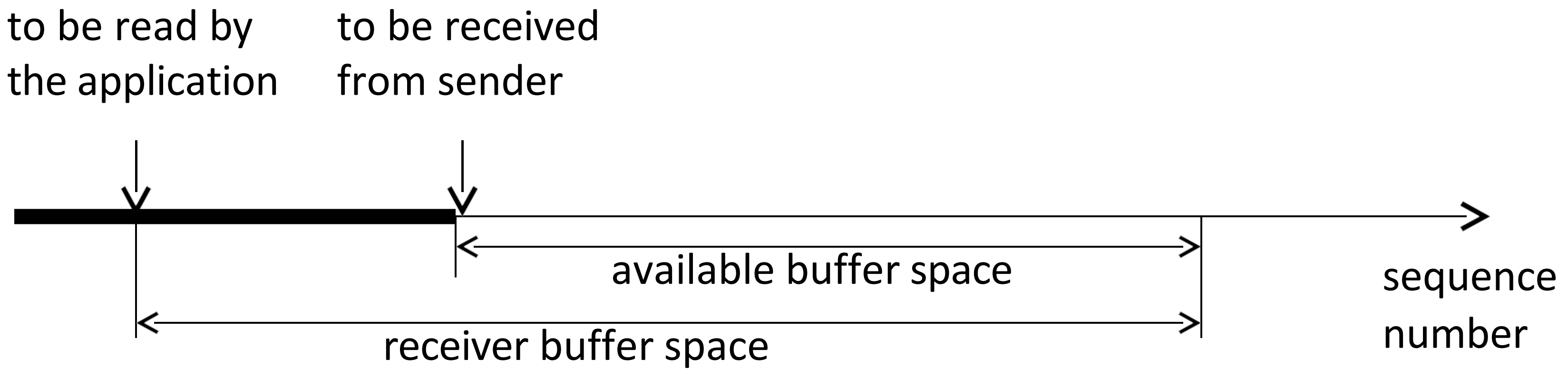
- TCP uses byte pointers
- Guarantees reliable delivery of data
- Ensures data is delivered in order
- Enforces flow control between sender and receiver
  - So the sender does not overrun the receiver’s buffer

Receiver Advertised Window

- The sliding window size at the receiver is not fixed
- The receiver advertises an adjustable window size in the Window field (16 bits) of the TCP header
    - The amount of free space (bytes) available in the receiver’s buffer
- The sender is limited to having no more than Window bytes of unACKed data at any time

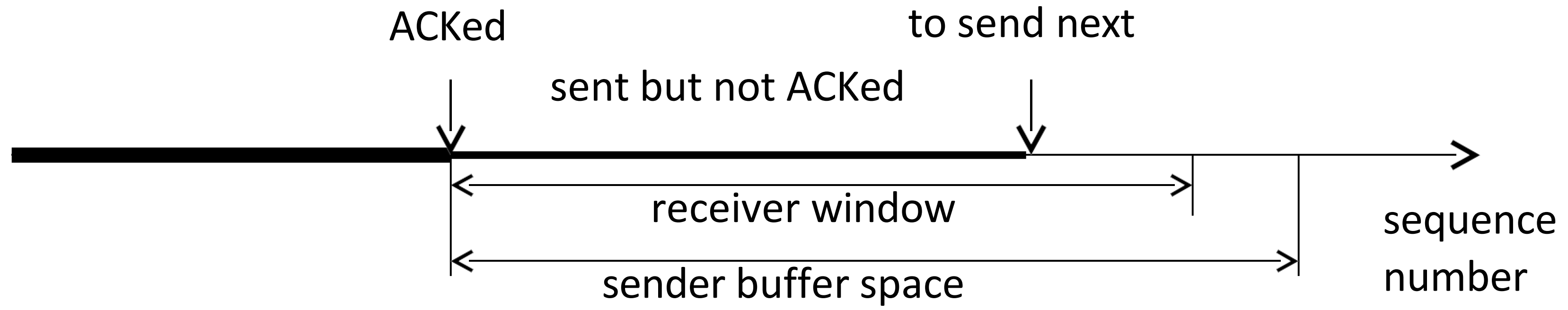
Receiver Window

- Sequence space
- The acknowledgment number is the next continuous byte to be received from the sender
- Receiver window is the available buffer space starting from this number

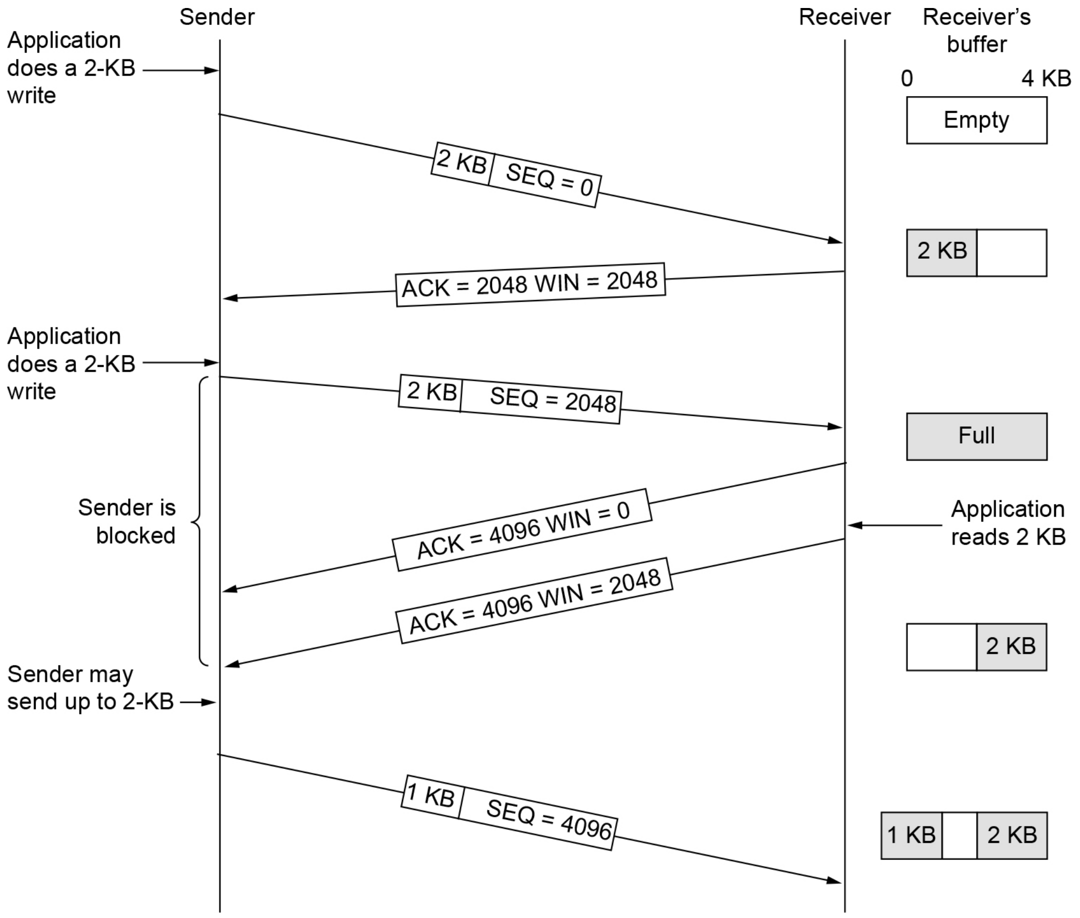


Sender Window

- Sequence number is the first byte in the payload
- Sender window
  - min{receiver window, sender buffer space}



TCP Sliding Window

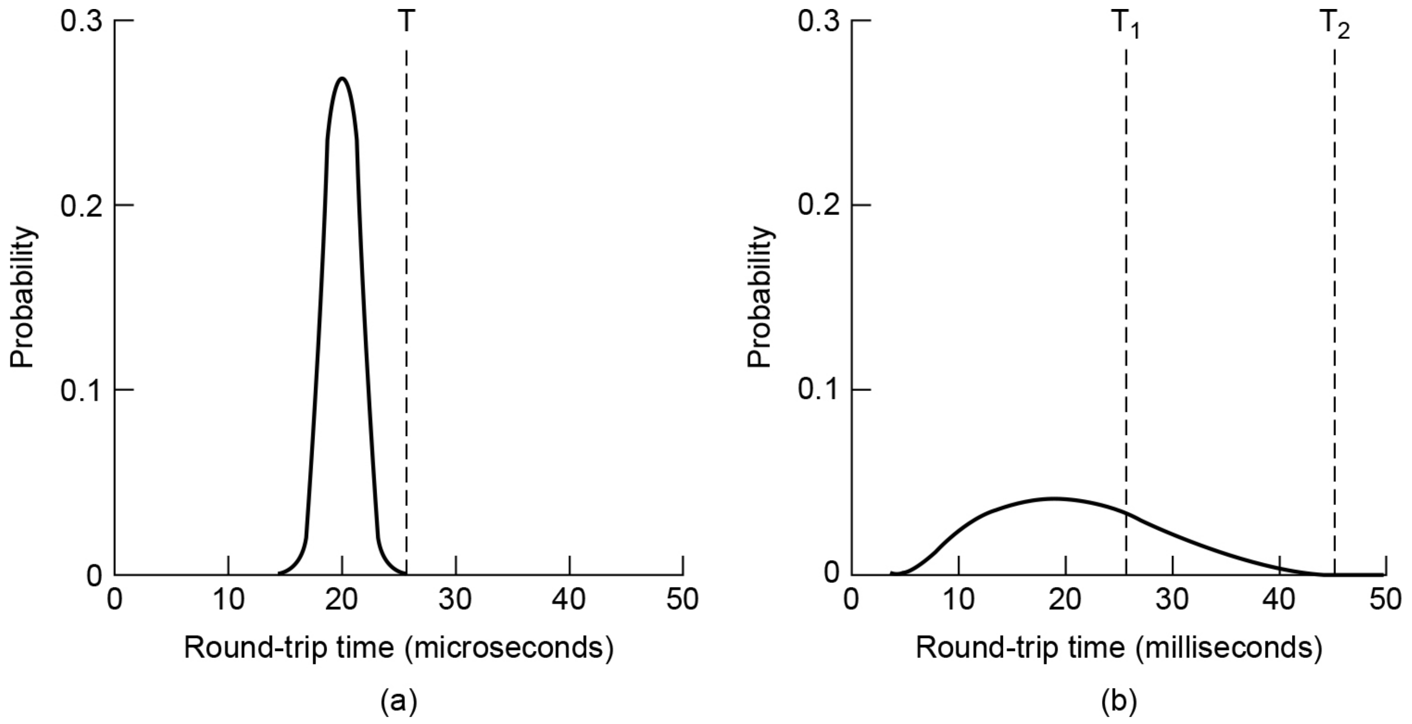


Triggering Transmission

TCP has three mechanisms to trigger the transmission of a segment

1. TCP maintains a variable MSS and sends a segment as soon as it has collected MSS bytes from the sending process
   - MSS is usually set to the size of the largest segment TCP can send without causing IP fragmentation
   - MTU of directly connected network (including TCP header and IP header)
2. Sending process explicitly asks TCP to send it
   - TCP push operation PSH
3. A timeout
   - The resulting segment contains bytes currently buffered for transmission

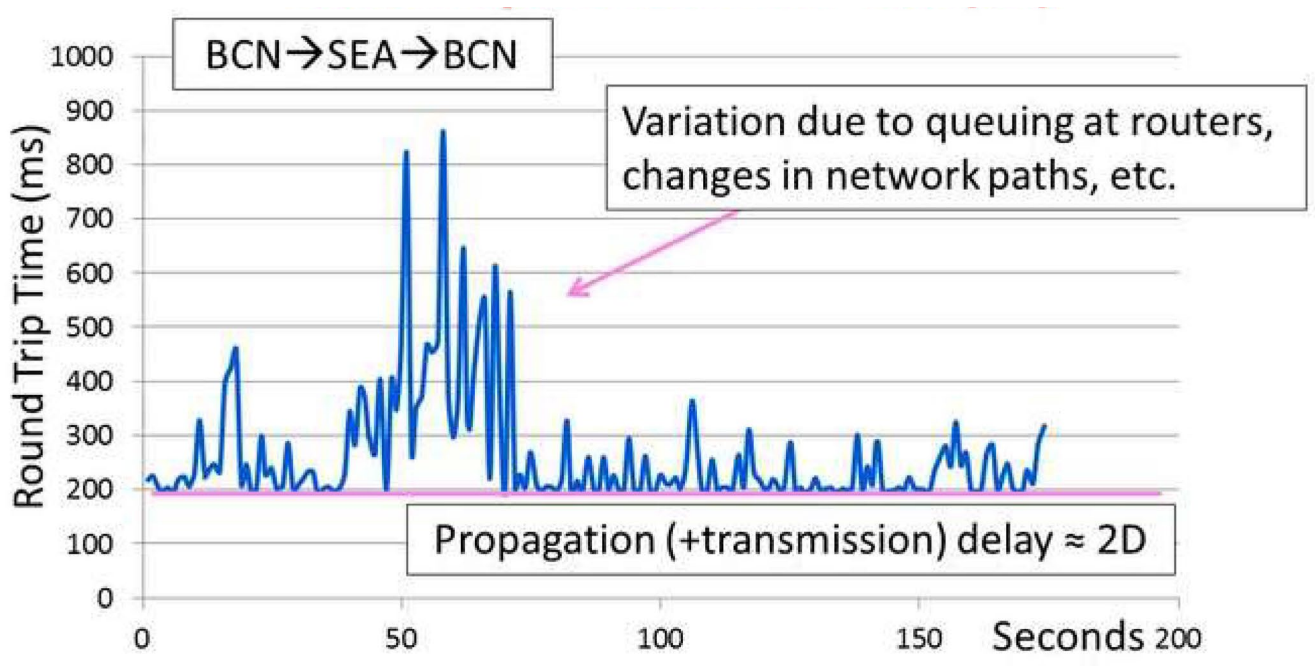
TCP Timeouts



(a) Probability density of acknowledgement arrival times in the data link layer.

(b) Probability density of acknowledgement arrival times for TCP.

Barcelona-Seattle-Barcelona RTT



Adaptive Timeout

- Smoother Round Trip Time

  $SRTT_{N+1} = \alpha \times SRTT_N + (1-\alpha) \times RTT_{N+1}$

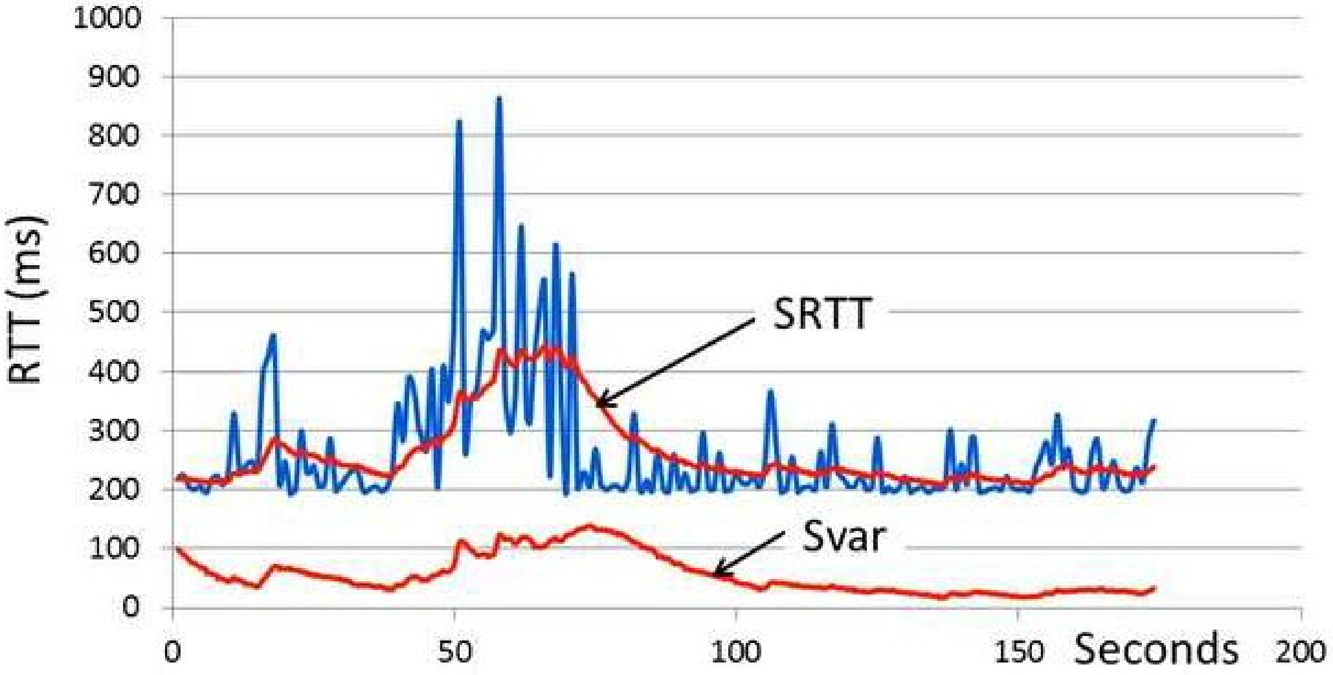
- Round trip time variation

  $Svar_{N+1} = \beta \times Svar_N + (1-\beta) \times |RTT_{N+1} - SRTT_{N+1}|$

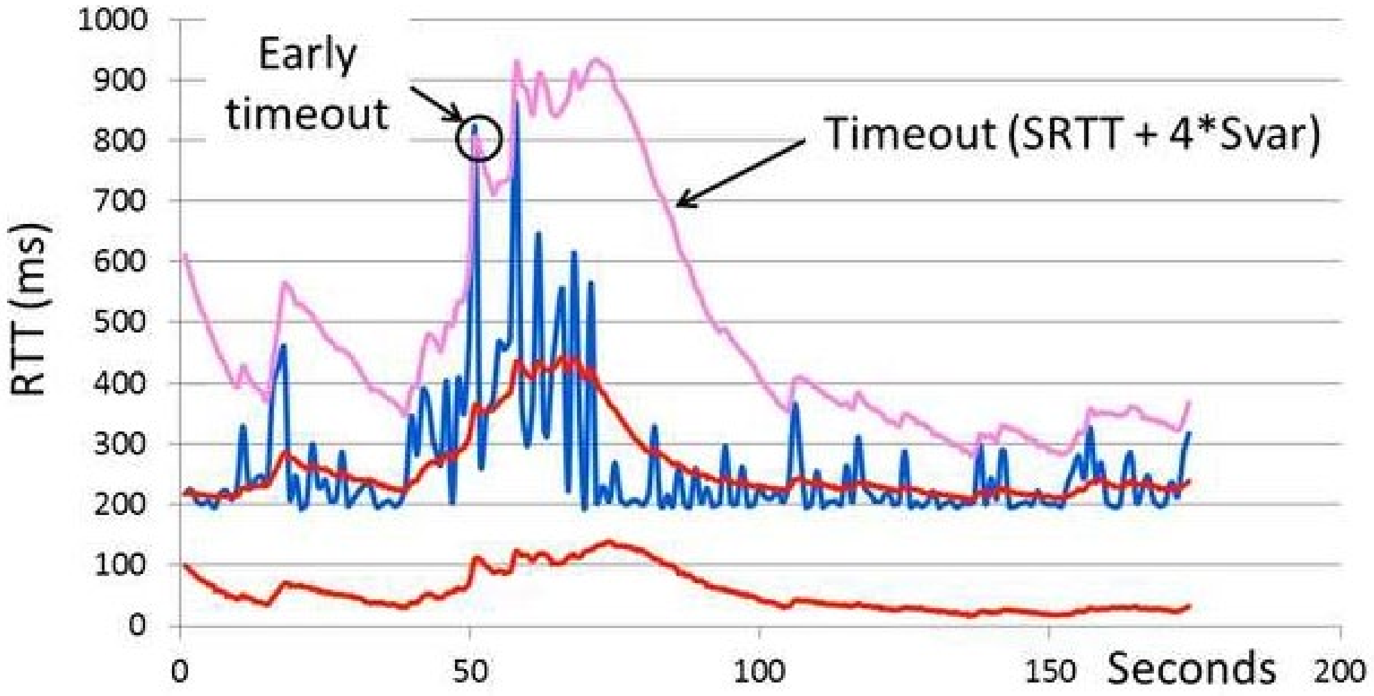
- TCP retransmission timeout

  $RTO_N = SRTT_N + 4 \times Svar_N$

SRTT α = 0.9 and Svar β = 0.9



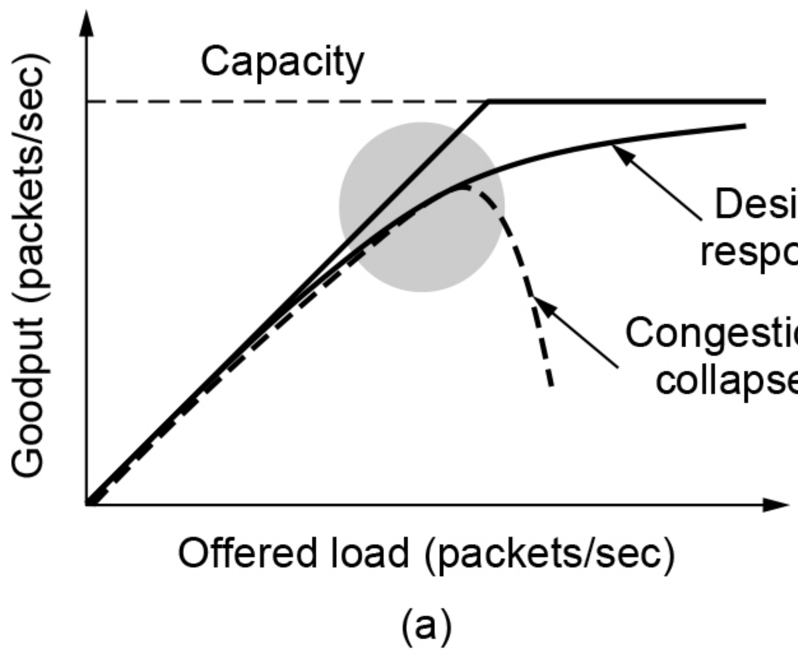
TCP Retransmission Timeout

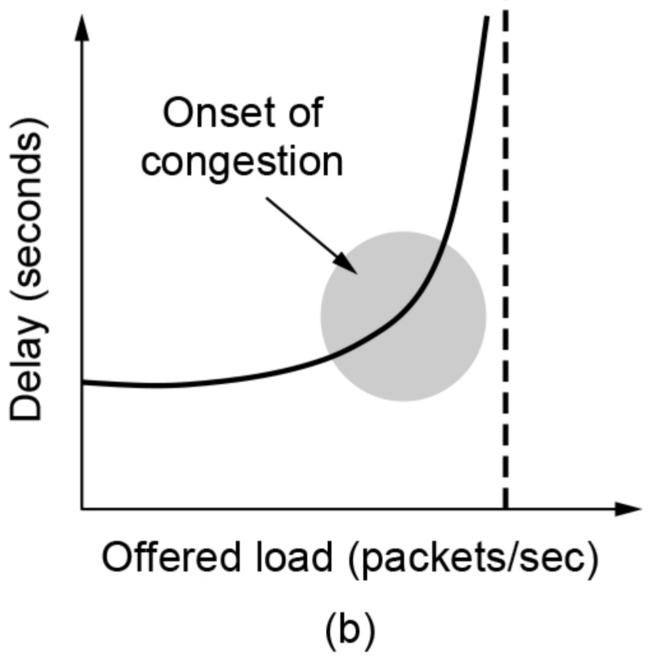


Goodput

- Goodput is defined as the rate of useful data delivered by the network
- It reflects actual user experience because it shows how much usable data is delivered
  - Does not include retransmissions
- Goodput decreases when there is
  - Packet loss
  - Congestion

Congestion





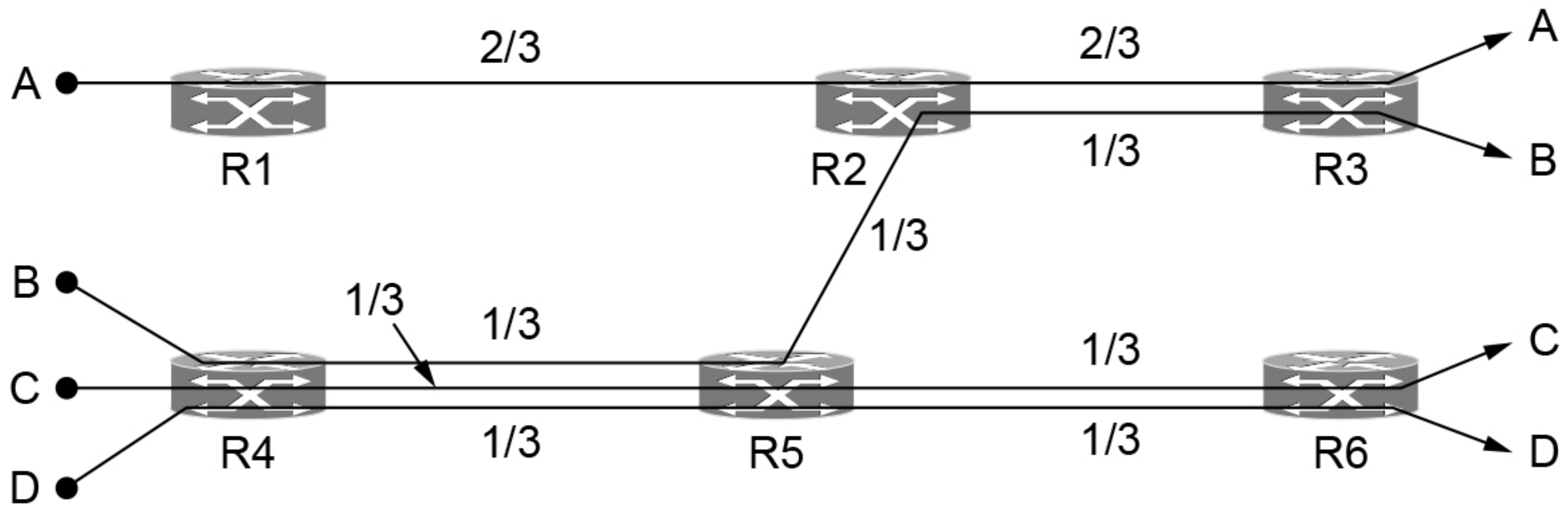
(a) Goodput

(b) Delay

Max-Min Fair Allocation

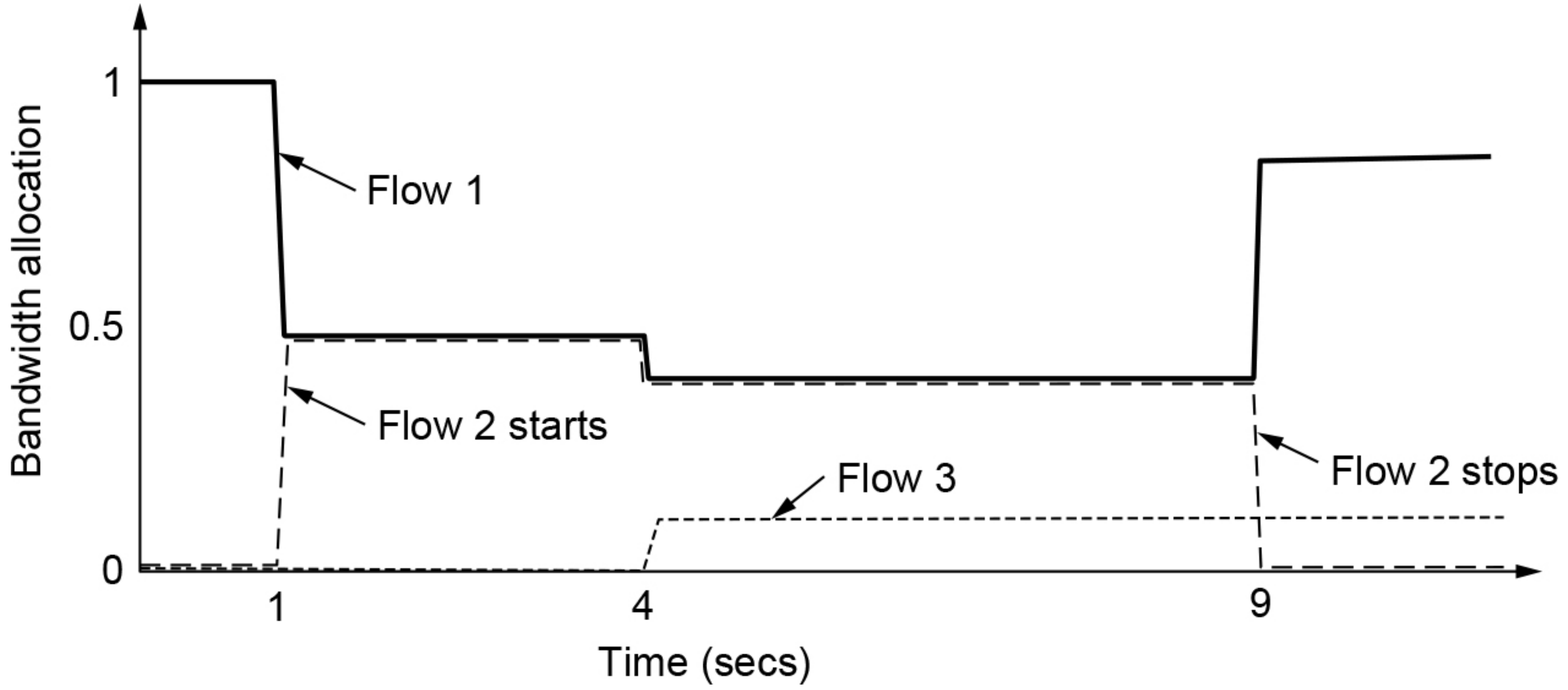
1. Start with all flows at 0
2. Increase the flows equally until there is a new bottleneck in the network
3. Hold fixed the rate of the flows that are bottlenecked
4. Go to step 2 for any remaining flows

Max-Min Fair Allocation



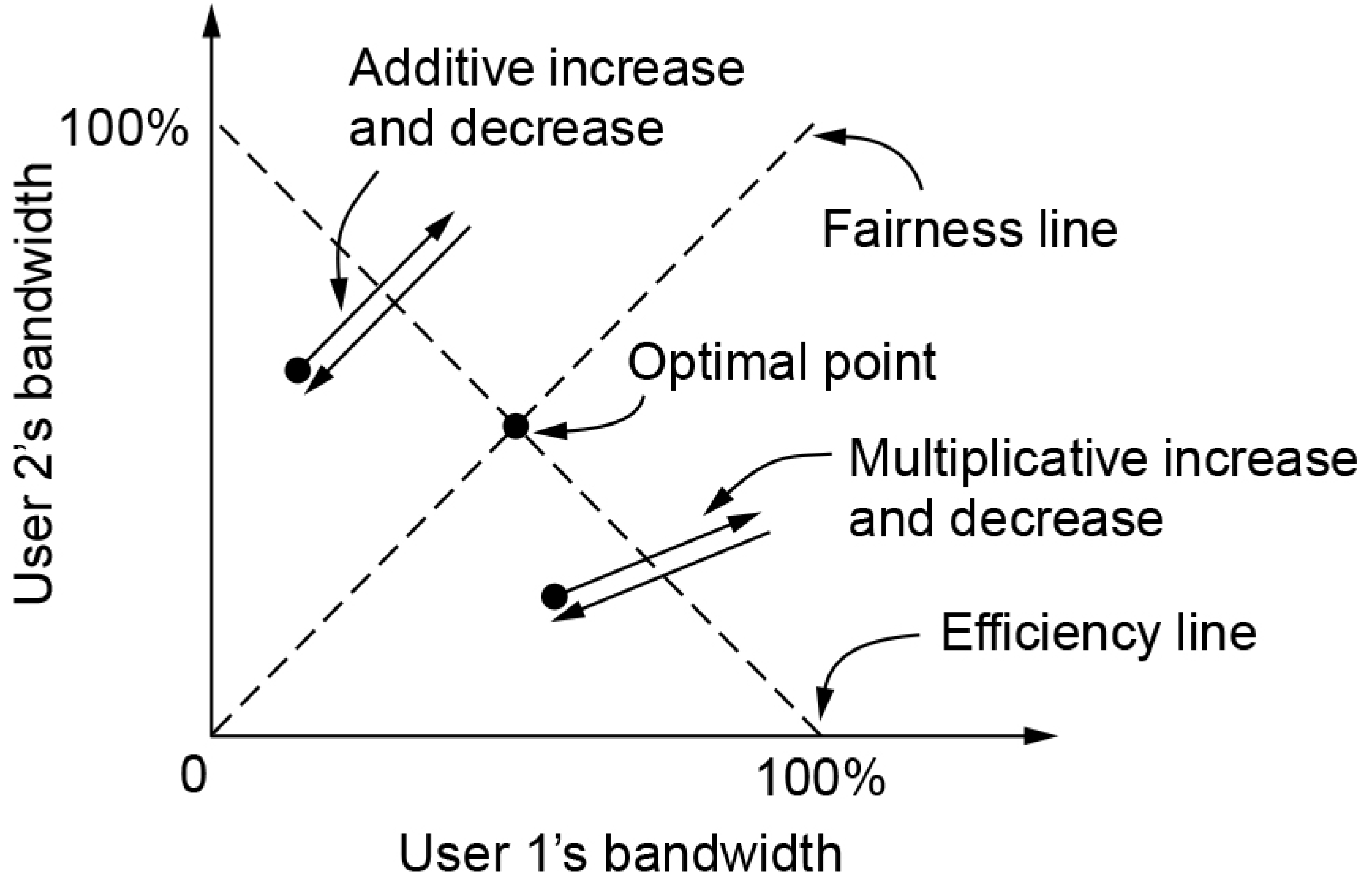
Max-min bandwidth allocation for four flows

Desirable Bandwidth Allocation



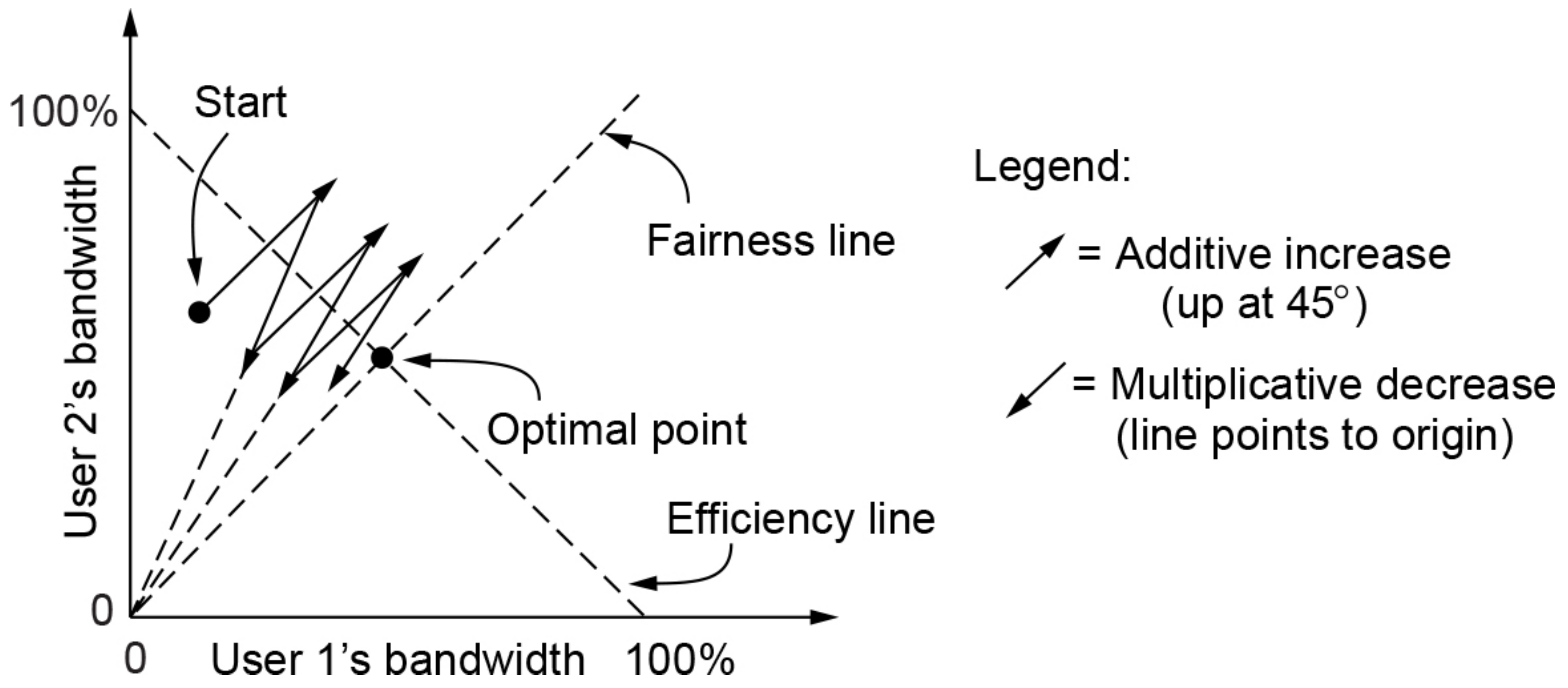
Changing bandwidth allocation over time

Regulating the Sending Rate



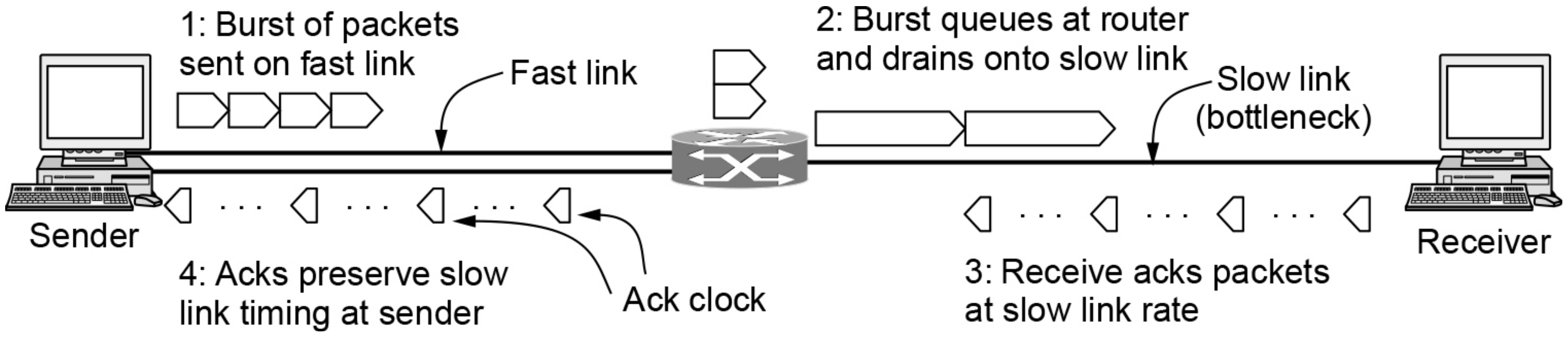
Additive and multiplicative bandwidth adjustments

Regulating the Sending Rate



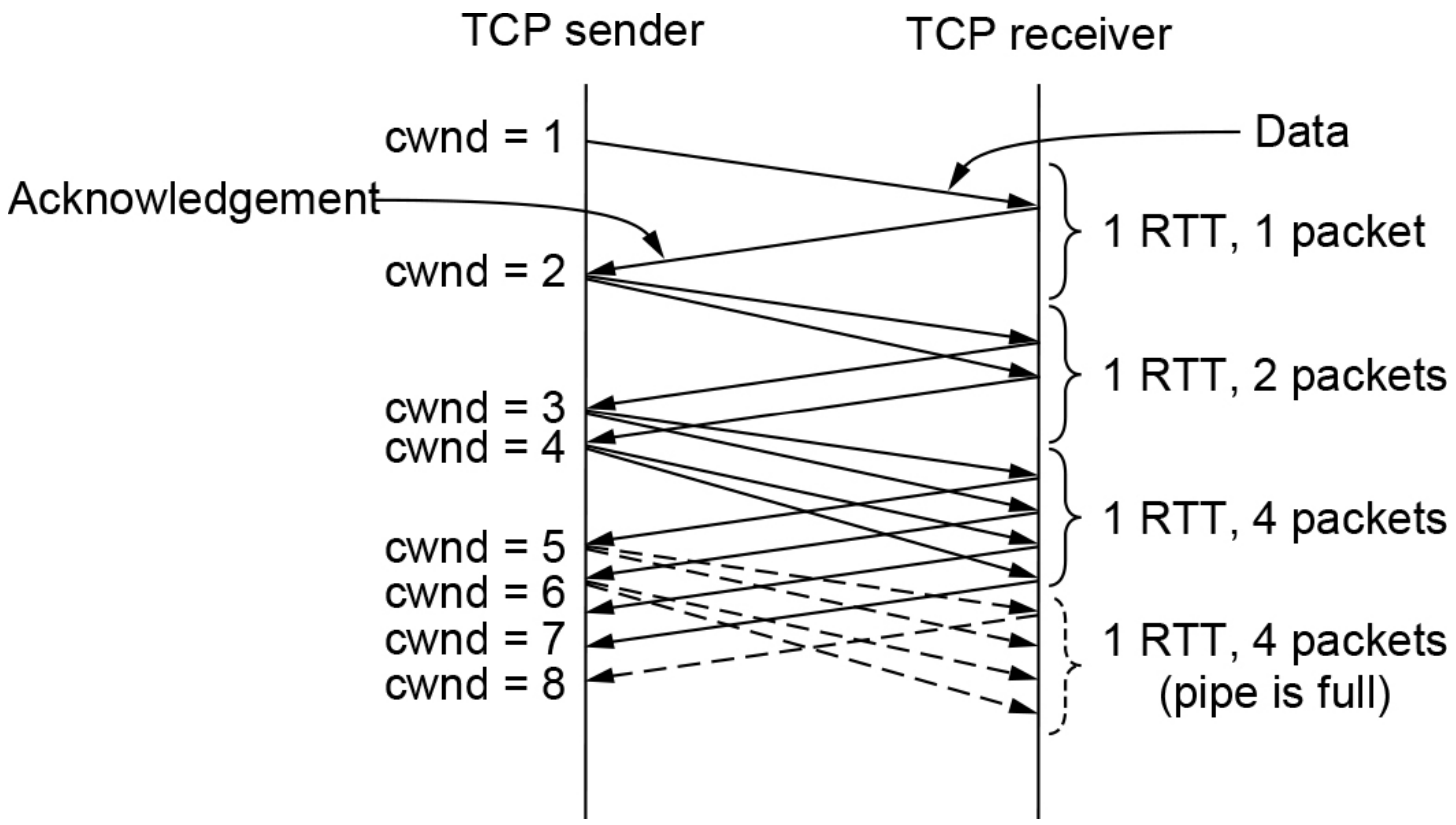
Additive Increase Multiplicative Decrease (AIMD) control law

TCP Congestion Control


1: Burst of packets sent on fast link
Fast link
2: Burst queues at router and drains onto slow link
Slow link (bottleneck)
Sender
Receiver
4: Acks preserve slow link timing at sender
Ack clock
3: Receive acks packets at slow link rate

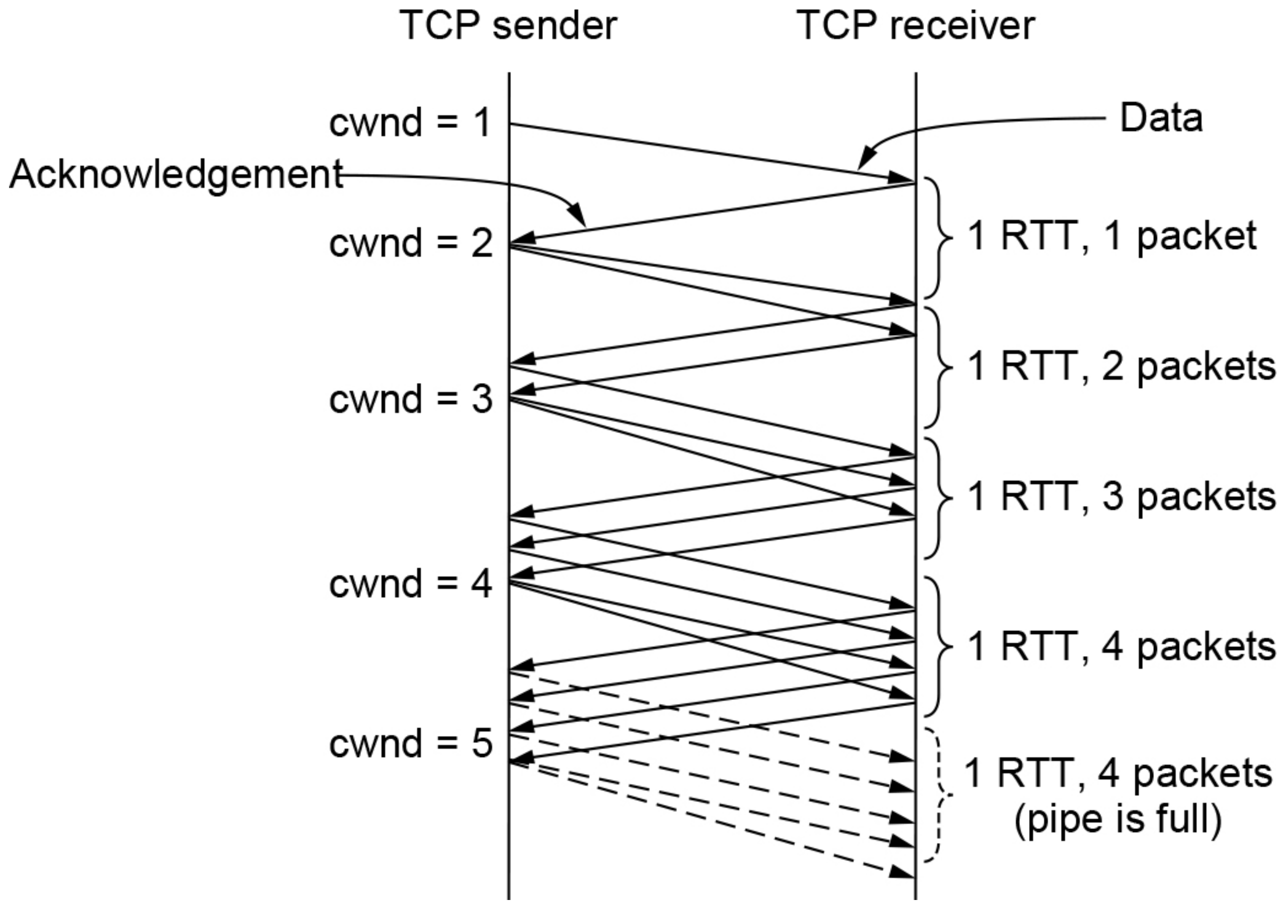
A burst of packets from a sender and the returning ACK clock

TCP Congestion Control



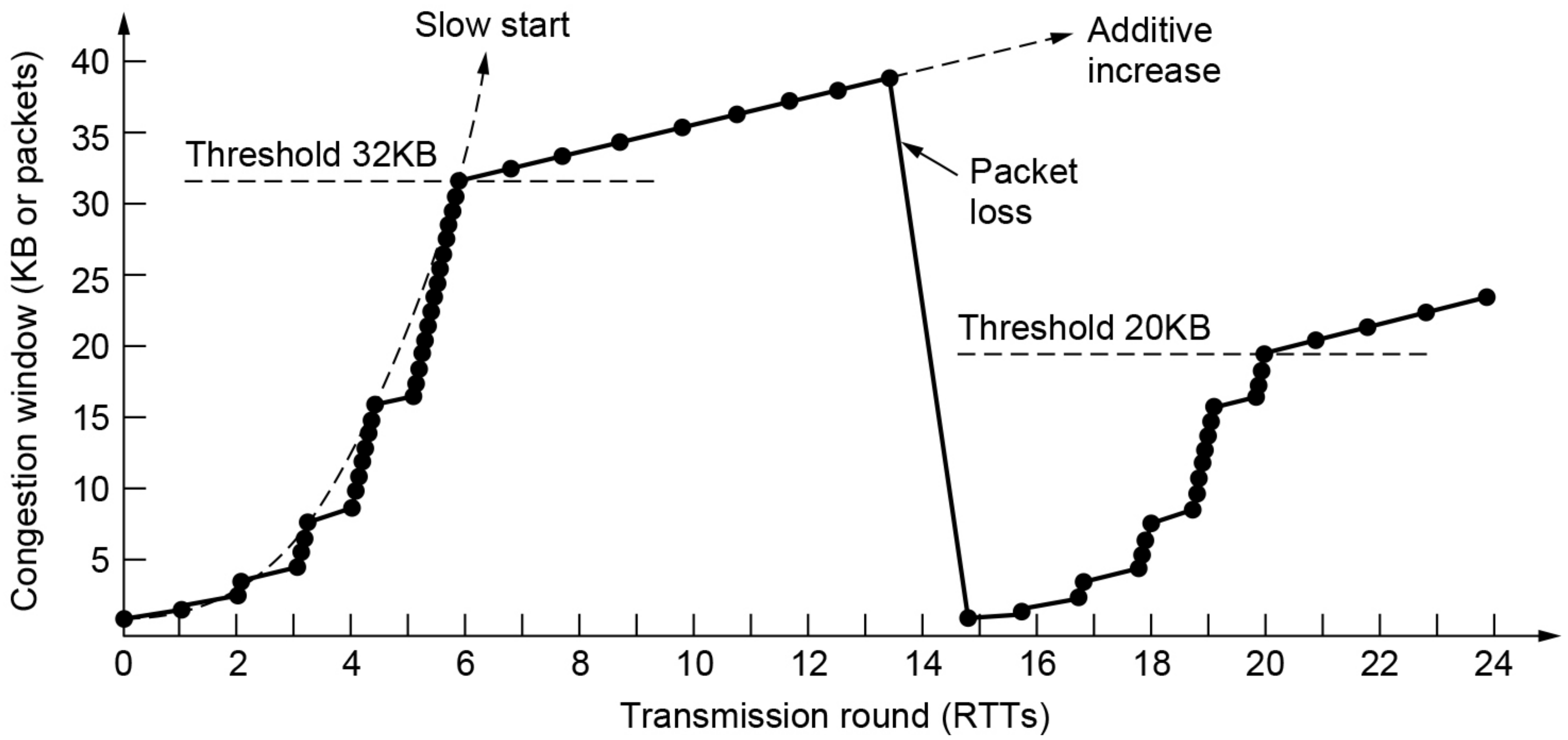
Slow start from an initial congestion window of one segment

TCP Congestion Control



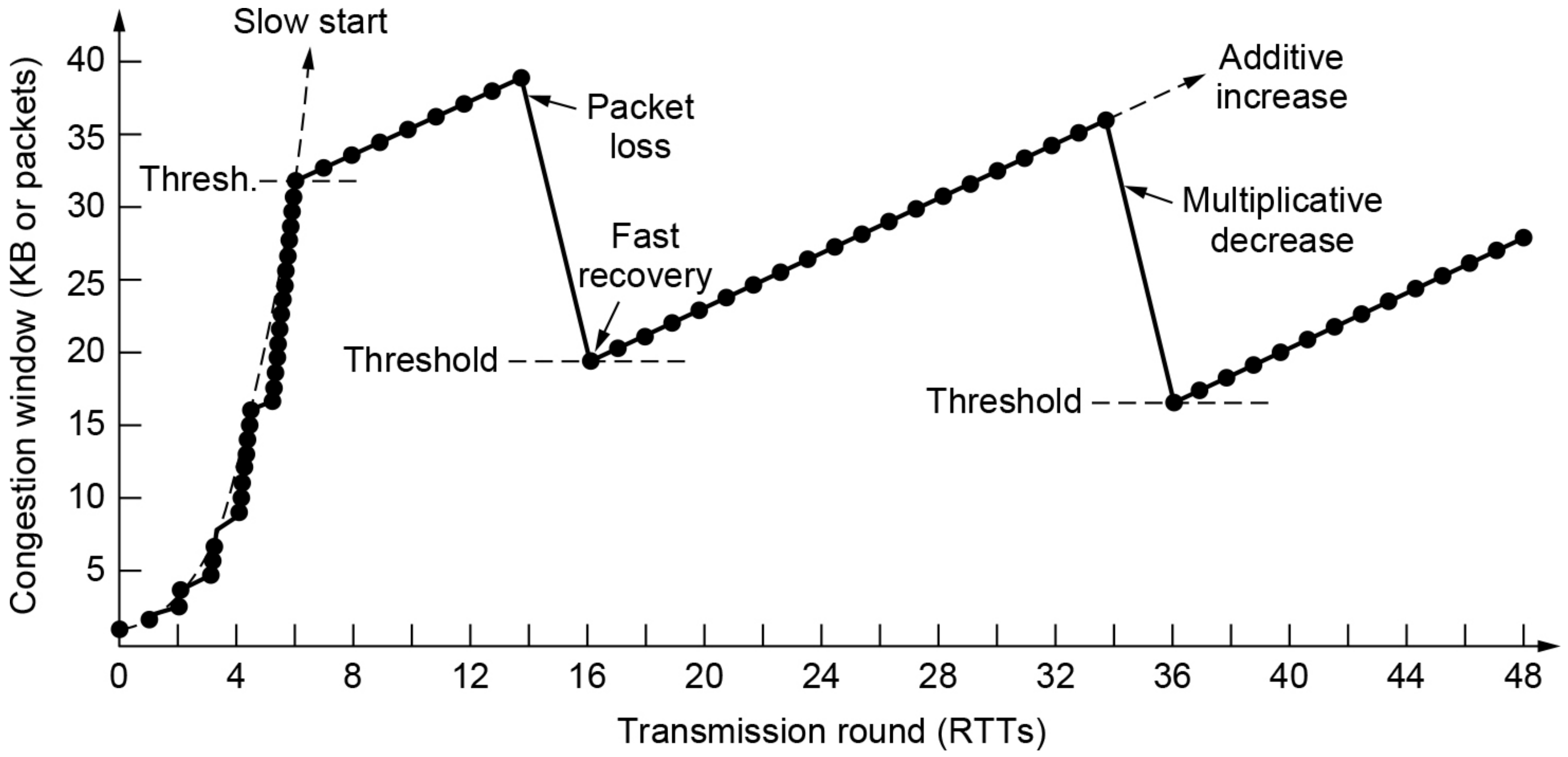
Additive increase from an initial congestion window of one segment

TCP Congestion Control



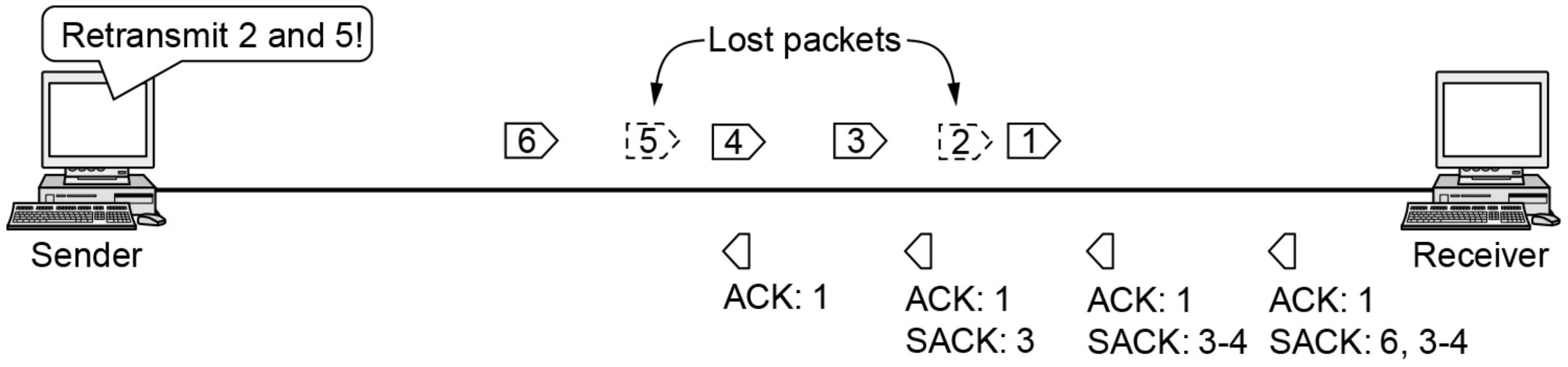
Slow start followed by additive increase in TCP Tahoe

TCP Congestion Control



Fast recovery and the sawtooth pattern of TCP Reno

TCP Congestion Control

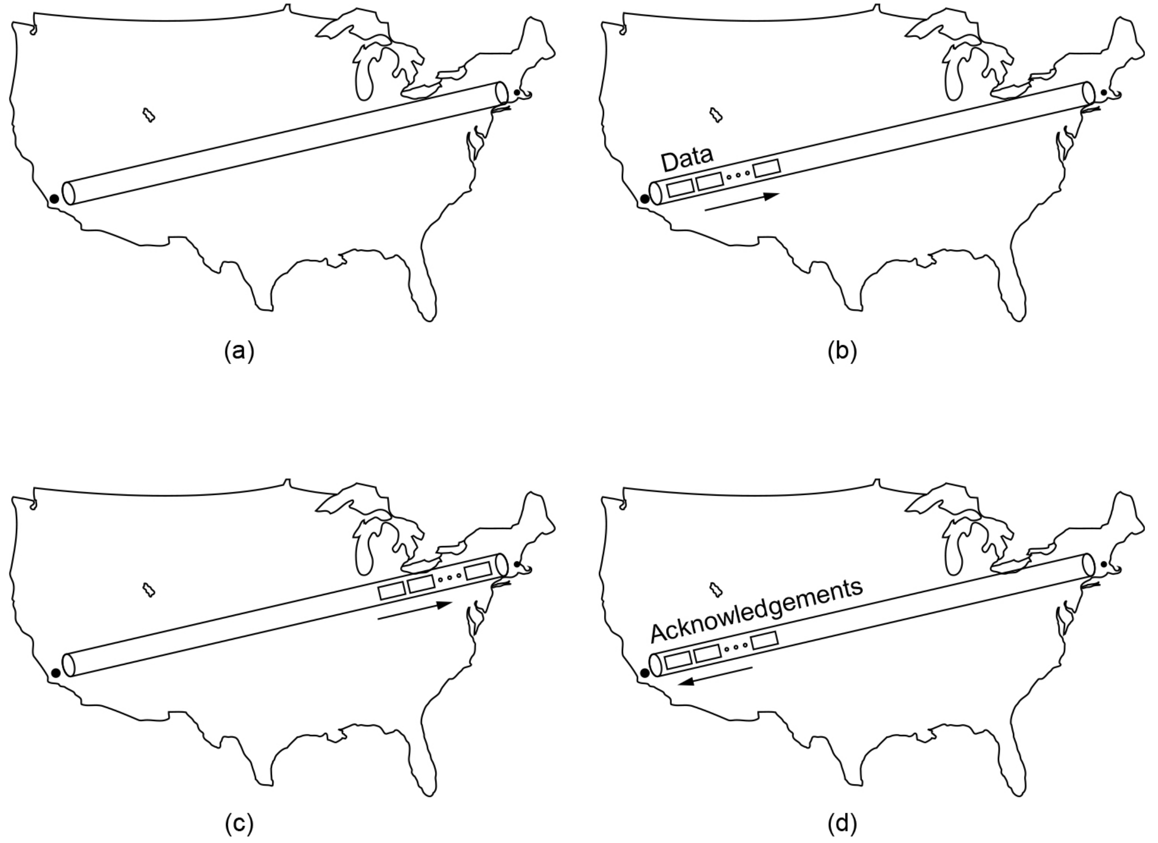


Selective acknowledgements (SACKs)

Network Performance

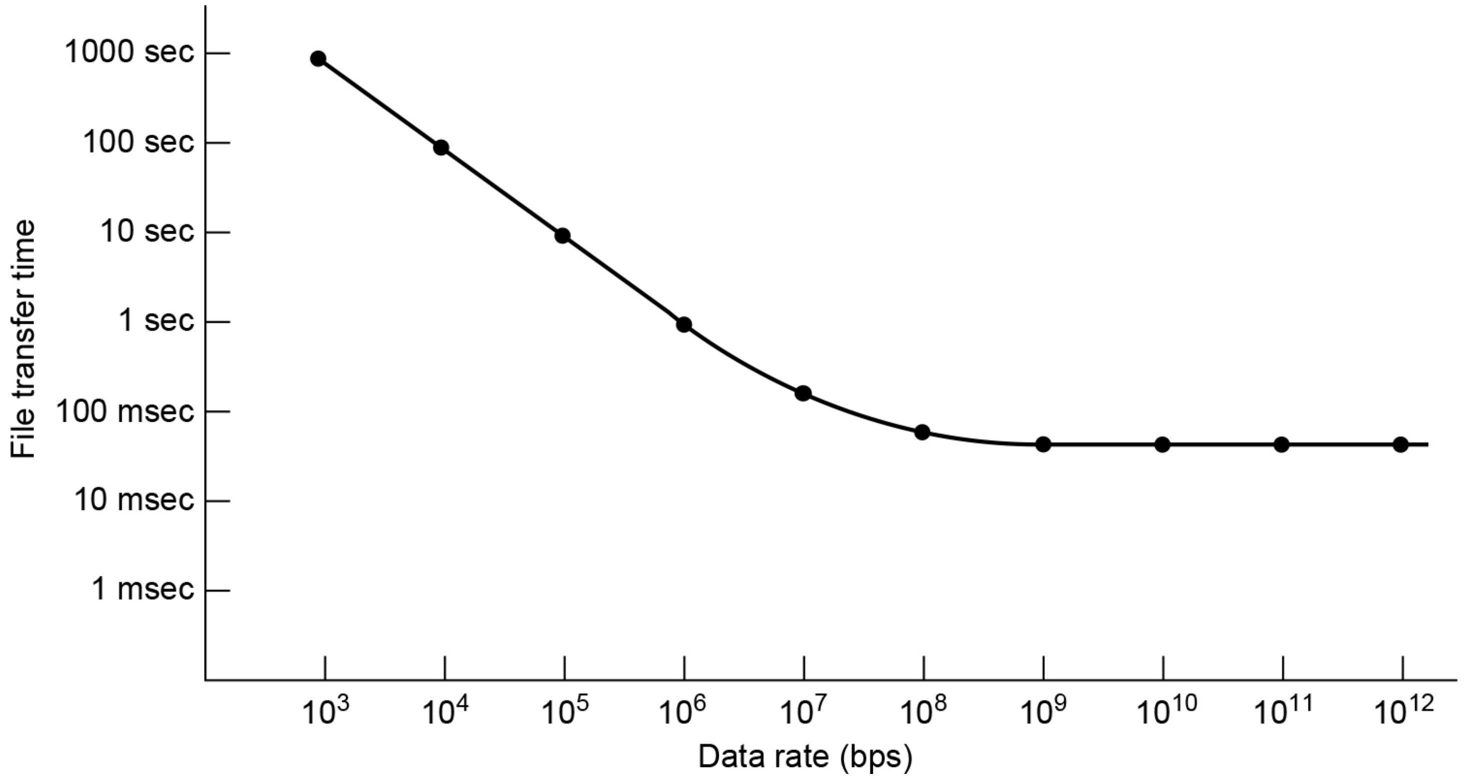
- Measuring speed has become more challenging
  - Filling the pipe requires more data
  - Under-test client/server bottlenecks move elsewhere in the network
- Speed is less relevant to network performance
  - Move beyond simple speed tests based on passive observation of network traffic
  - Quality of experience or application performance is more important
- Use multiple parallel TCP connections to fill the capacity of the link

Protocols for Long Fat Networks



The state of transmitting 1 Mbit from San Diego to Boston (a) at t = 0 (b) after 500 µsec (c) after 20 msec (d) after 40 msec

Protocols for Long Fat Networks



Time to transfer and acknowledge a 1 Mbit file over a 4000 km line