Romain Fontugne

2023 Fall Semester

Information Network Systems

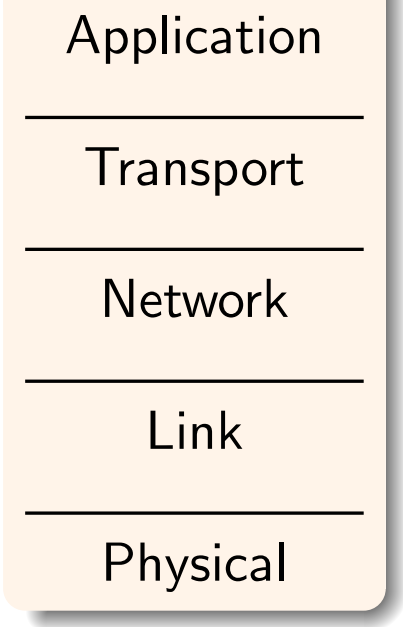
The Transport Layer

Last 2 lectures:

The network layer

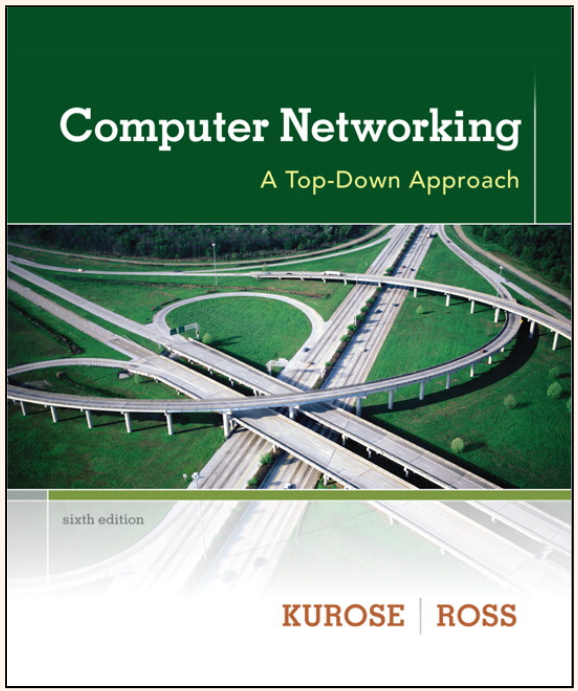
- Internet Protocol
  - IPv4
  - IPv6
  - addressing
- ICMP
- Routing Protocols
  - Link State Algorithms
  - Distance Vector Algorithms
  - Routing Hierarchy

IP Stack



Today's Lecture: Transport layer

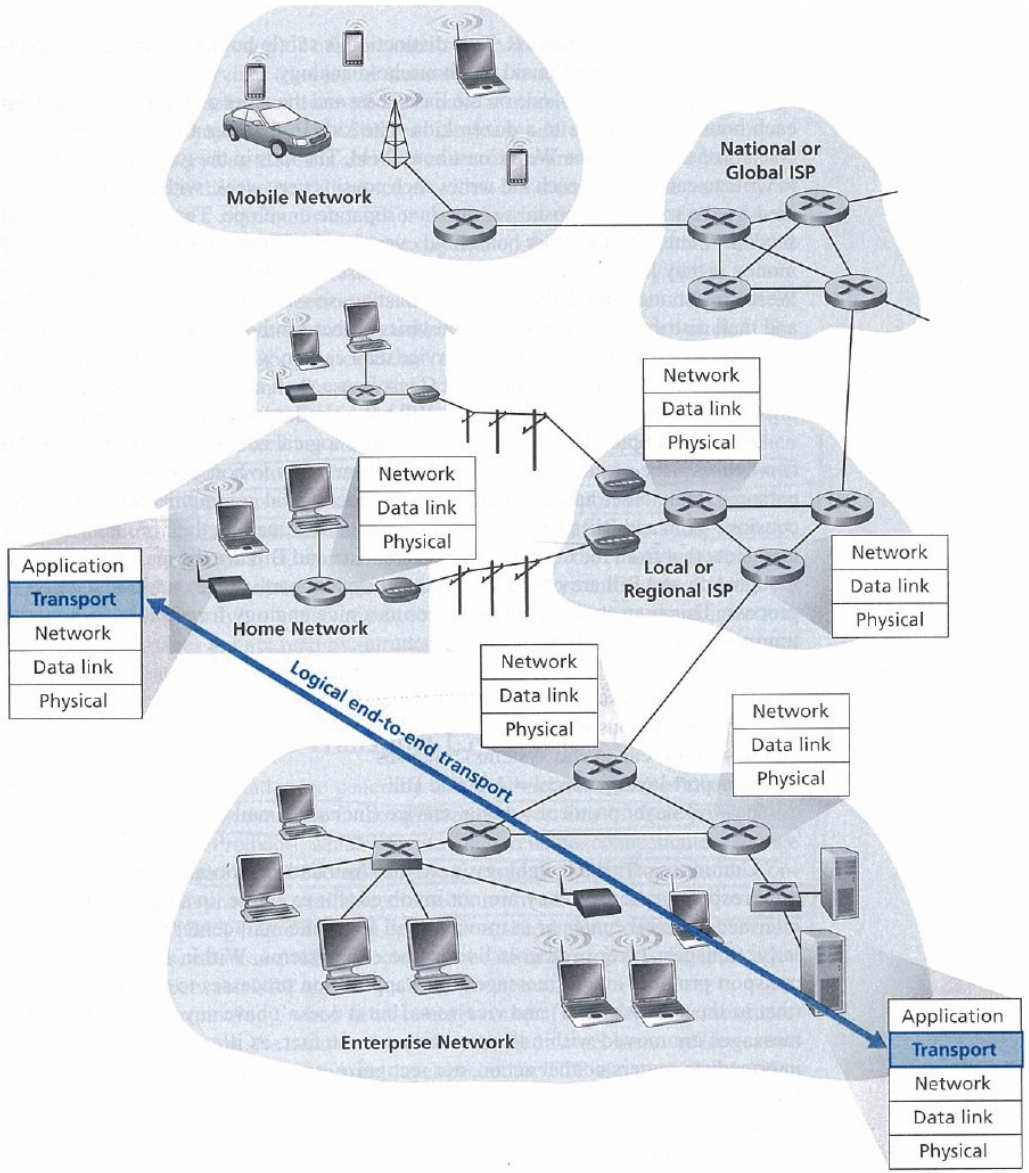
1. Transport Layer Services
2. Multiplexing and Demultiplexing
3. Connectionless transport: UDP
4. Connection-oriented transport: TCP
5. Congestion control



- Chapter 3

Transport services and protocols

- Provide logical communication between app processes running on different hosts
- Transport protocols run in end systems
  - send side: breaks app messages into segments, passes to network layer
  - rcv side: reassembles segments into messages, passes to app layer
- More than one transport protocol available to apps



Transport vs. network layer

Network Layer

- logical communication between hosts

Transport layer:

- logical communication between processes
  - relies on, and, enhances, network layer services

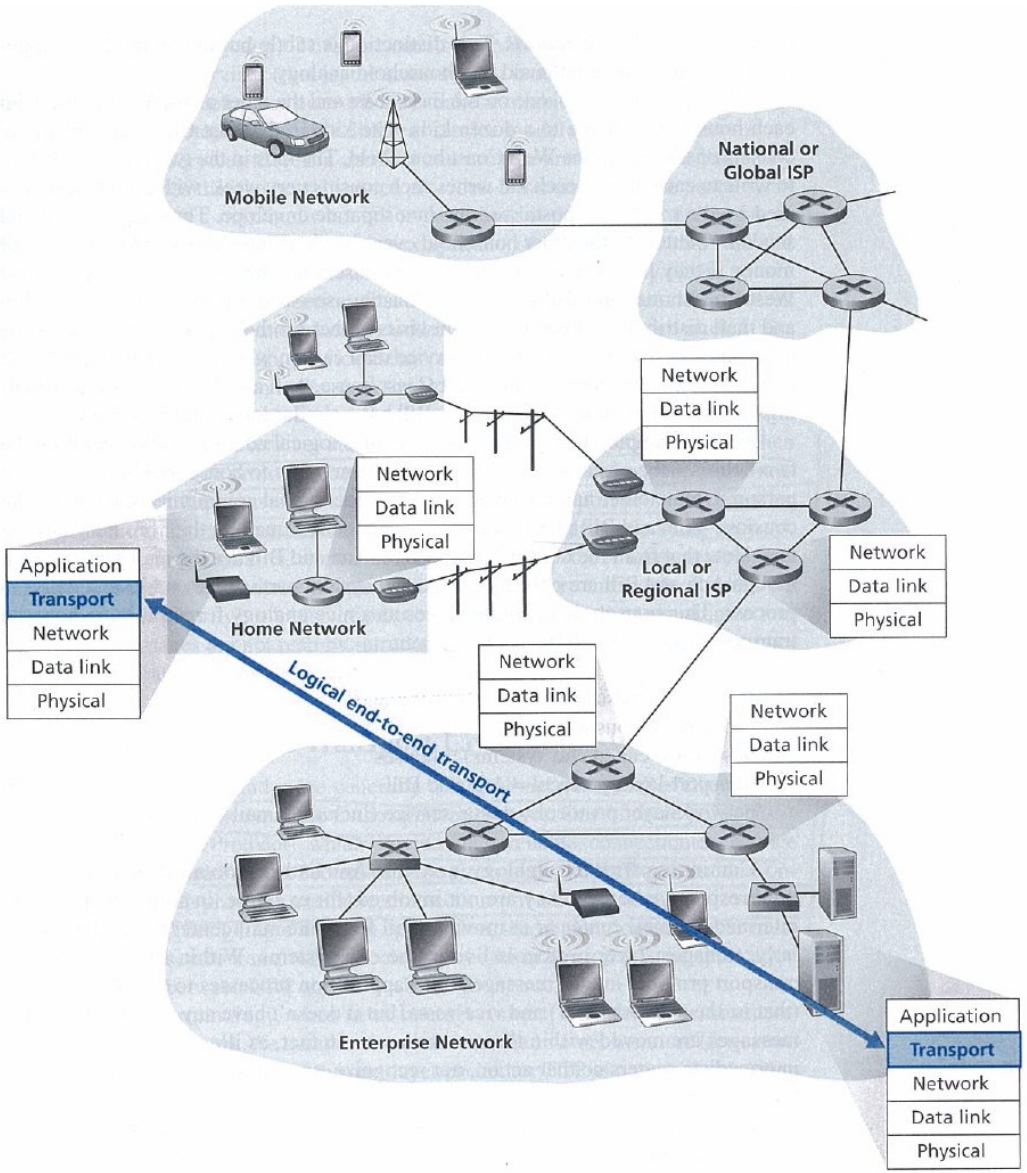
Household Analogy:

12 kids in Ann's house sending letters to 12 kids in Bill's house:

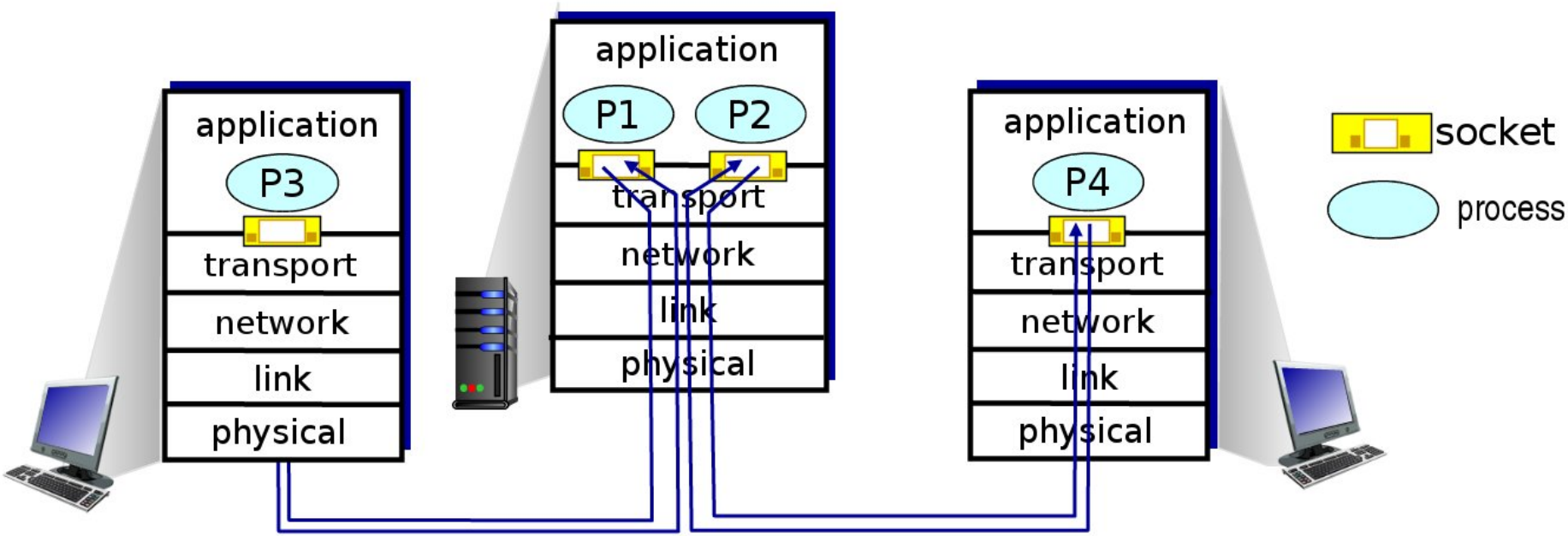
- hosts = houses
- processes = kids
- app messages = letters in envelopes
- transport protocol = Ann and Bill who demux to in-house siblings
- network-layer protocol = postal service

Internet Transport-Layer Protocols

- Unreliable, unordered delivery: UDP
  - no-frills extension of "best-effort" IP
- Reliable, in-order delivery: TCP
  - congestion control
  - flow control
  - connection setup
- Services not available:
  - delay guarantees
  - bandwidth guarantees



Multiplexing/demultiplexing



Multiplexing at sender:

handle data from multiple sockets, add transport

How demultiplexing works

- Host receives IP datagrams
  - each datagram has source IP address, destination IP address
  - each datagram carries one transport-layer segment
  - each segment has source, destination port number
- Host uses IP addresses and port numbers to direct segment to appropriate socket

Connectionless Demultiplexing

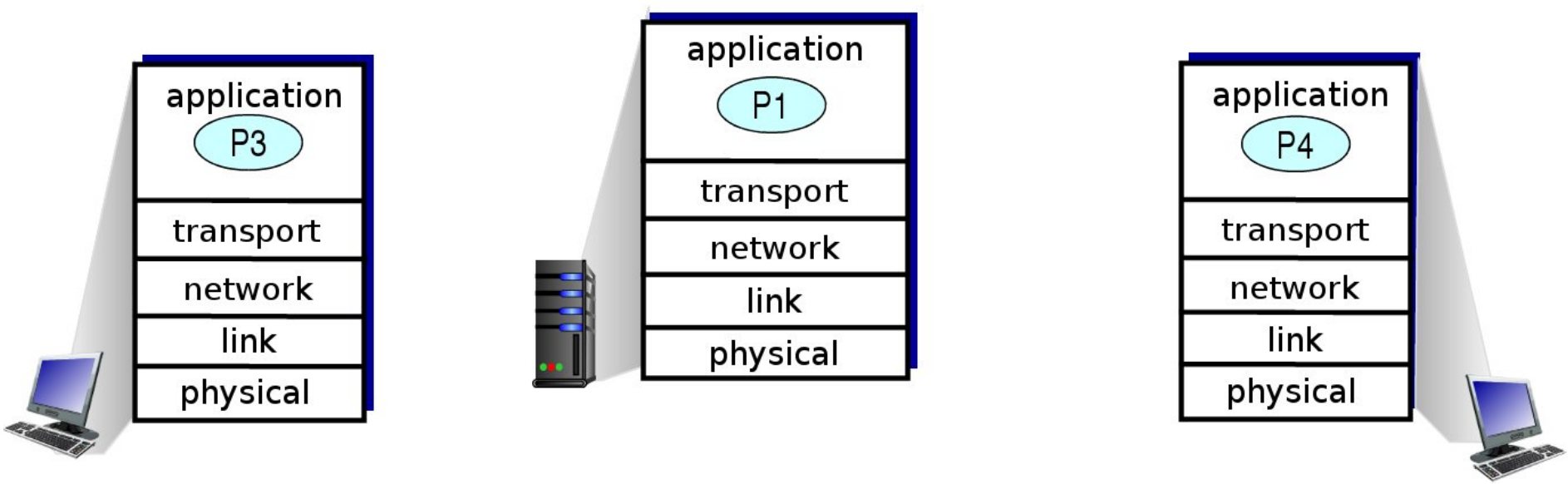
Sending UDP segment

- Create a UDP socket, must specify local port number
  Ex. in Java: *DatagramSocket mySocket1 = new DatagramSocket(12534);*
- When sending data into UDP socket, must specify:
  - destination IP address
  - destination port number

When host receives UDP segment

- Checks destination port number in segment
- Directs UDP segment to socket with that port number
- $\rightarrow$ IP datagrams with same dest. port number, but different source IP addresses and/or source port numbers will be directed to same socket at dest

Connectionless Demultiplexing: Example




```
DatagramSocket serverSocket
= new DatagramSocket
```

Connection-oriented Demultiplexing

TCP socket identified by 4-tuple:

- Source IP address
- Source port number
- Destination IP address
- Destination port number

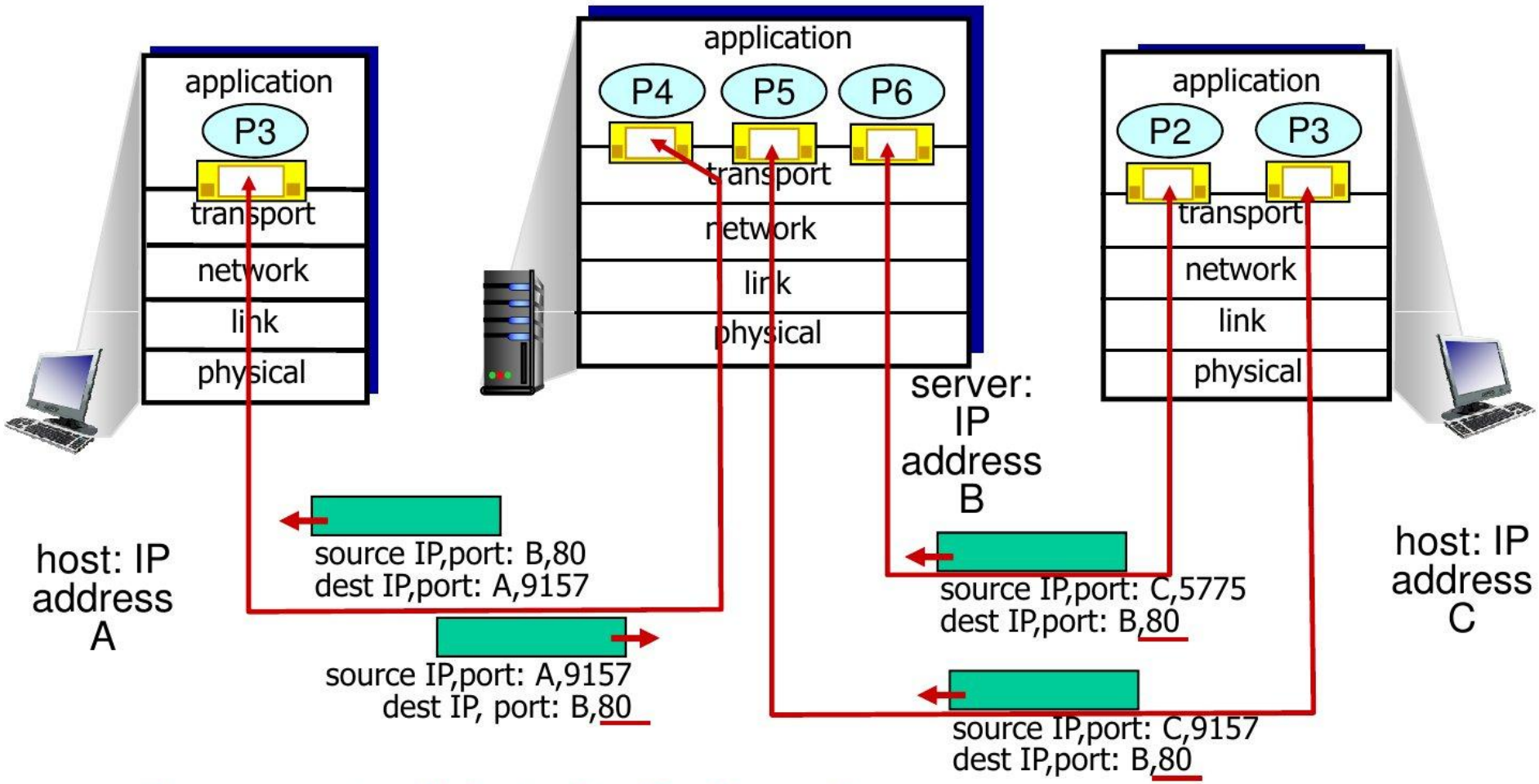
Demultiplexing

- Receiver uses all four values to direct segment to appropriate socket

Server host may support many simultaneous TCP sockets:

- Each socket identified by its own 4-tuple
- e.g.: Web servers have different sockets for each connecting client

Connection-oriented Demultiplexing



three segments, all destined to IP address: B,
dest port: 80 are demultiplexed to *different* sockets

UDP: User Datagram Protocol

Services

- "bare bone" Internet transport protocol
- "best effort" service, UDP segments may be:
  - lost
  - delivered out-of-order to application
  - connectionless:
    - no handshaking between UDP sender, receiver
    - each UDP segment handled independently of others

UDP use:

- streaming multimedia apps (loss tolerant, rate sensitive)
- DNS, DHCP

Reliable transfer over UDP:

- add reliability at application layer
- application-specific error recovery!

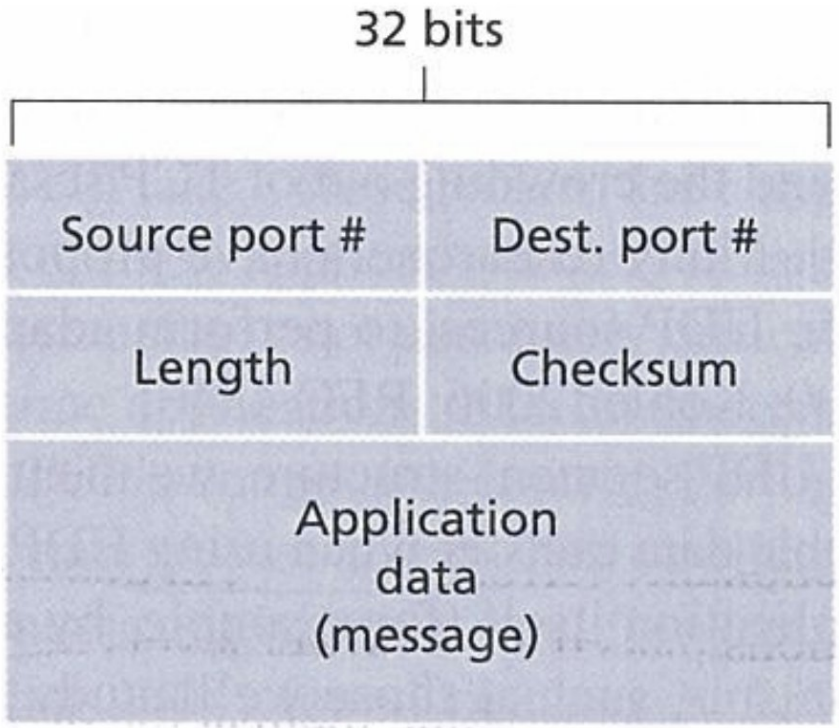
UDP: Segment Header

Four fields:

- Source port number
- Destination port number
- Length (size of the segment including header)
- Checksum (of the segment)

Port number:

- 65536 distinct port numbers
- 0 to 1023 are reserved for specific application protocols e.g. 53 for dns



Why is there a UDP?

Advantages of UDP

- Low overhead: simple, small header, no transmission error checking/correction
- no connection establishment (which can add delay)
- no connection state at sender, receiver
- no congestion control: UDP can blast away as fast as desired

Suitable for:

- time-sensitive applications / real-time systems
- bootstrapping such as the DHCP protocol
- works well in unidirectional communication with a very large number of clients (such as streaming media applications)

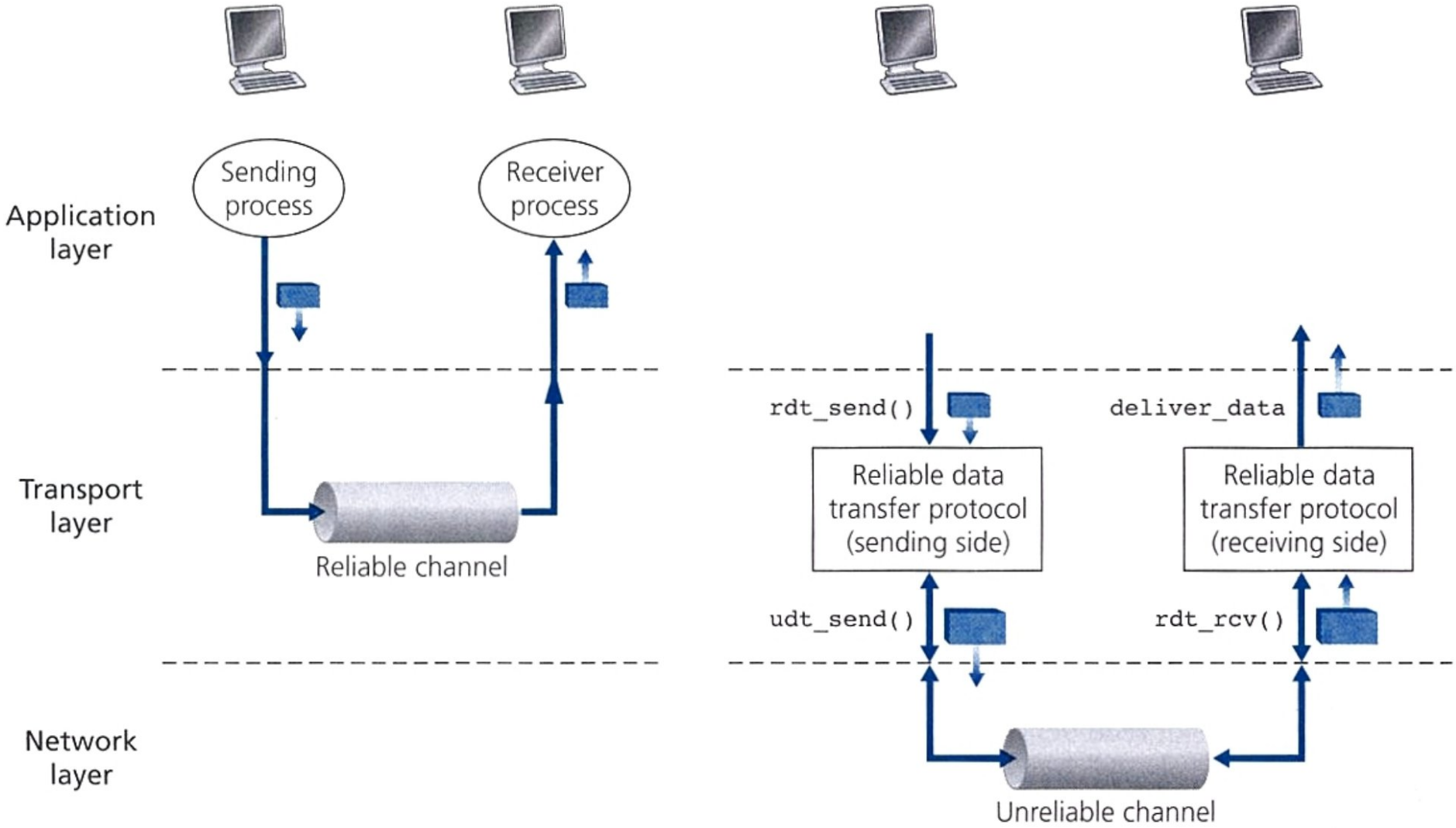
TCP: Transmission Control Protocol

Overview

- Connection-oriented:
  - handshaking (exchange of control msgs) inits sender, receiver state before data exchange
- Point-to-point:
  - one sender, one receiver
- Reliable, in-order byte stream:
  - Handle IP packets lost, duplicated, or delivered out of order
- Full duplex data:
  - bi-directional data flow in same connection
- Congestion control
  - minimize network congestion to avoid other problems (i.e. packet lost)
- Flow controlled:
  - sender will not overwhelm receiver

Reliable Data Transfer

Reliable data transfer using unreliable connection

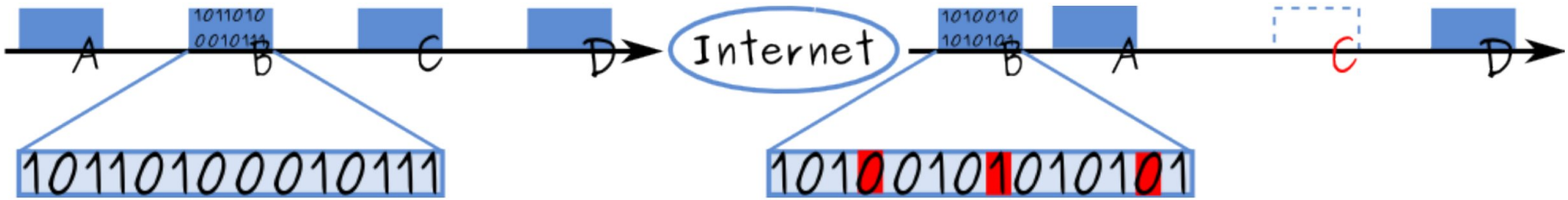


Reliable Data Transfer

Concerns:

- Corrupted bits
- Packet lost
- Delayed packets

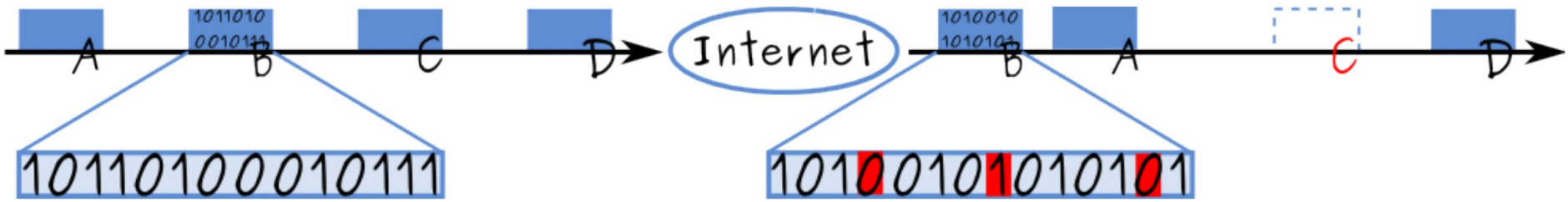
$\rightarrow$ the receiver need to provide feedback to the sender



Reliable Data Transfer

Strategies to establish a reliable data transfer

- Error detection Detect bit errors
- Acknowledgement For correctly received packets
- Neg. Acknowledgement For incorrectly received packets
- Timer Timeout/retransmit a packet
- Sequence numbers Sequential numbering of packets
- Window/pipelining Transmission of multiple packets for speedup



Sequence numbers

In error-prone channel, also NAK and ACK packets might be corrupted.
A simple solution is the use of sequence numbers

Sequence numbers

- Data packet will utilize a special field to hold a sequence number
- Receiver checks sequence number to learn whether packets are lost
- The ACK and NAK packets can hold the sequence numbers of the respective packets they refer to
- TCP: 32 bit sequence number that counts bytes in the byte stream rather than packets

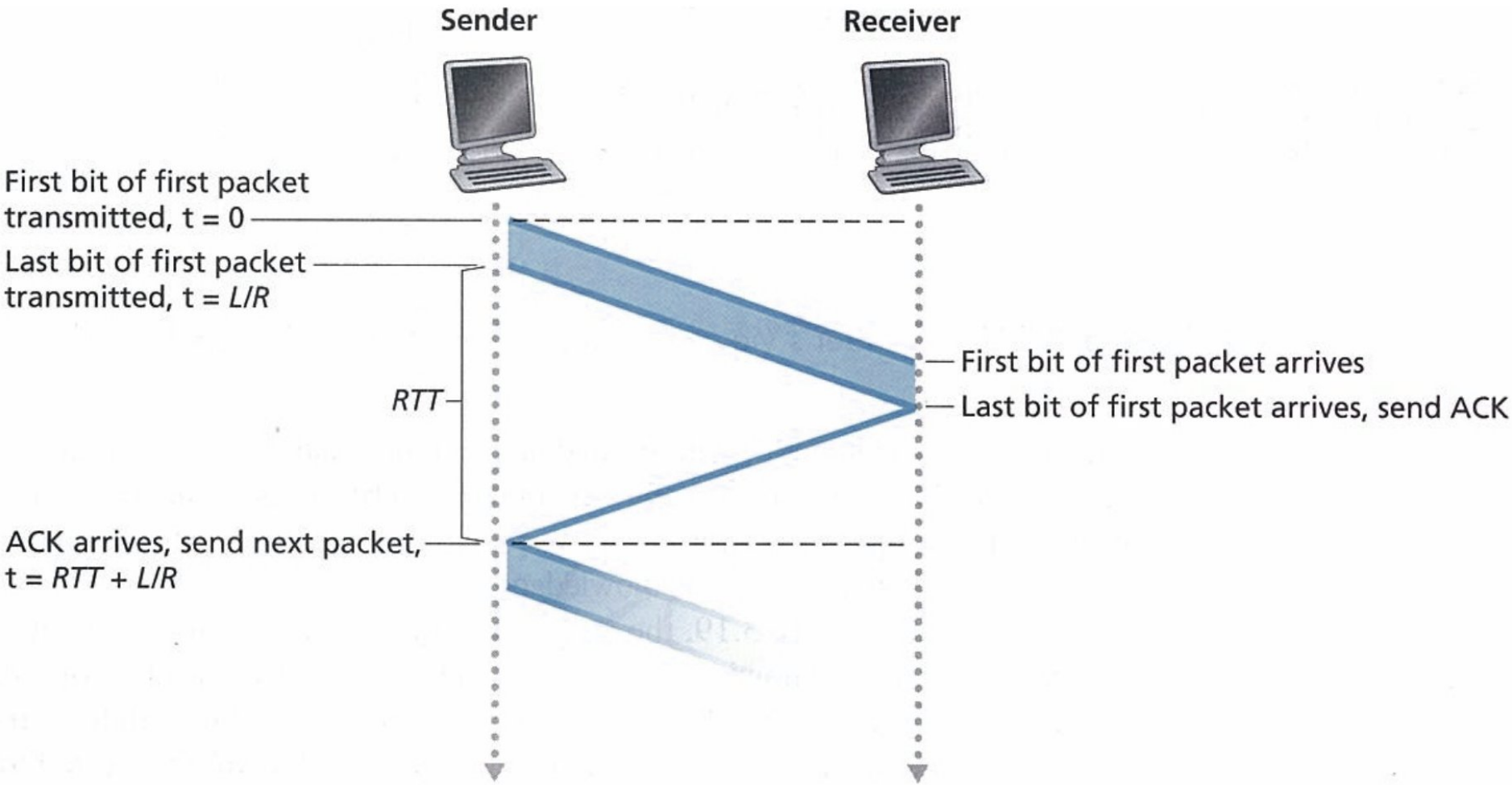
Retransmission

Packet retransmission

- The sender, after not receiving an ACK for a defined time interval, retransmits the respective packet
- The waiting time is crucial to the performance and is in practice chosen judiciously so that a packet loss can be assumed
- A too short waiting time unnecessarily congests the link and produces overhead at the receiver

Pipelining

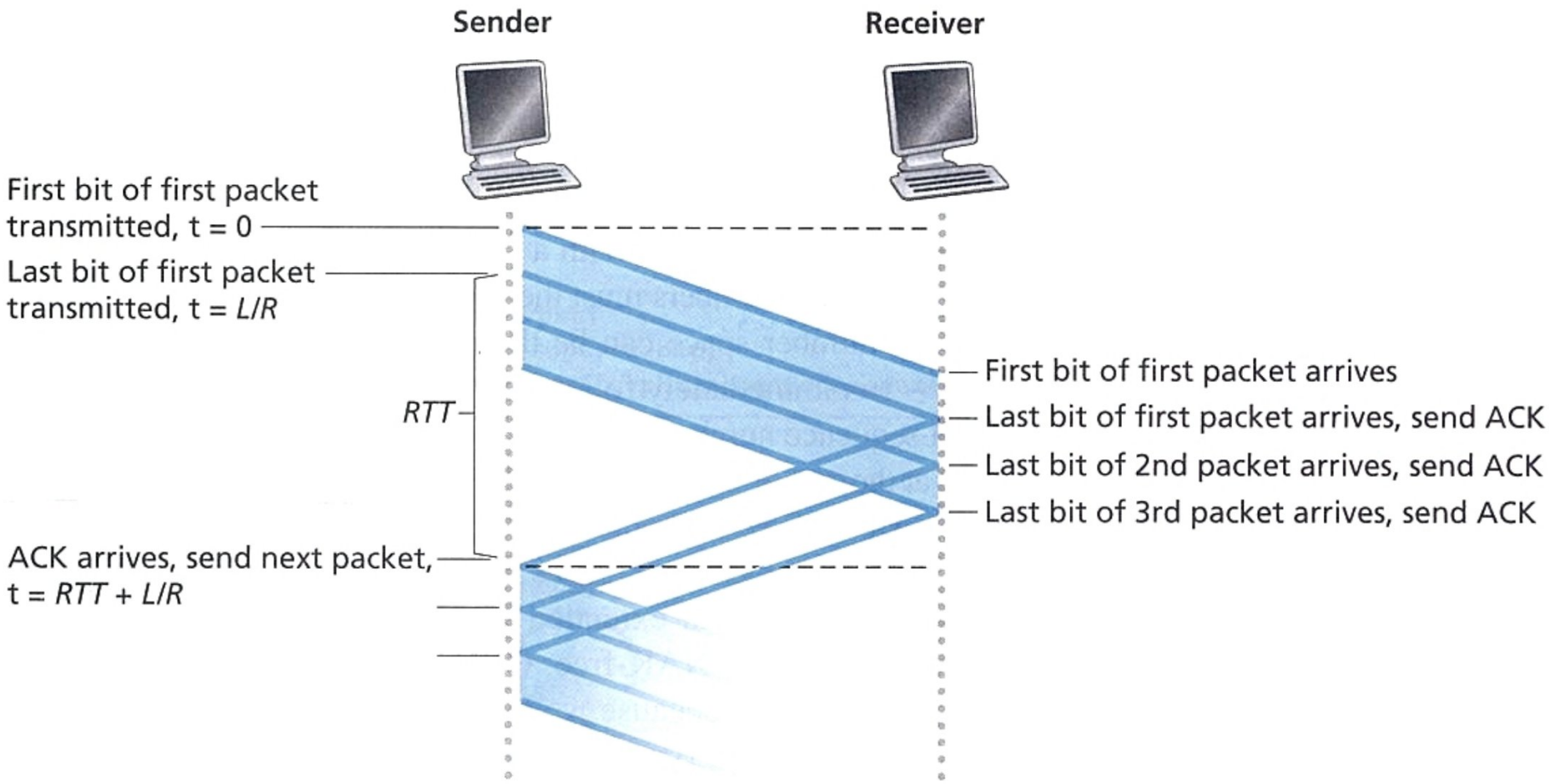
The waiting time for packet retransmission can induce serious delays.



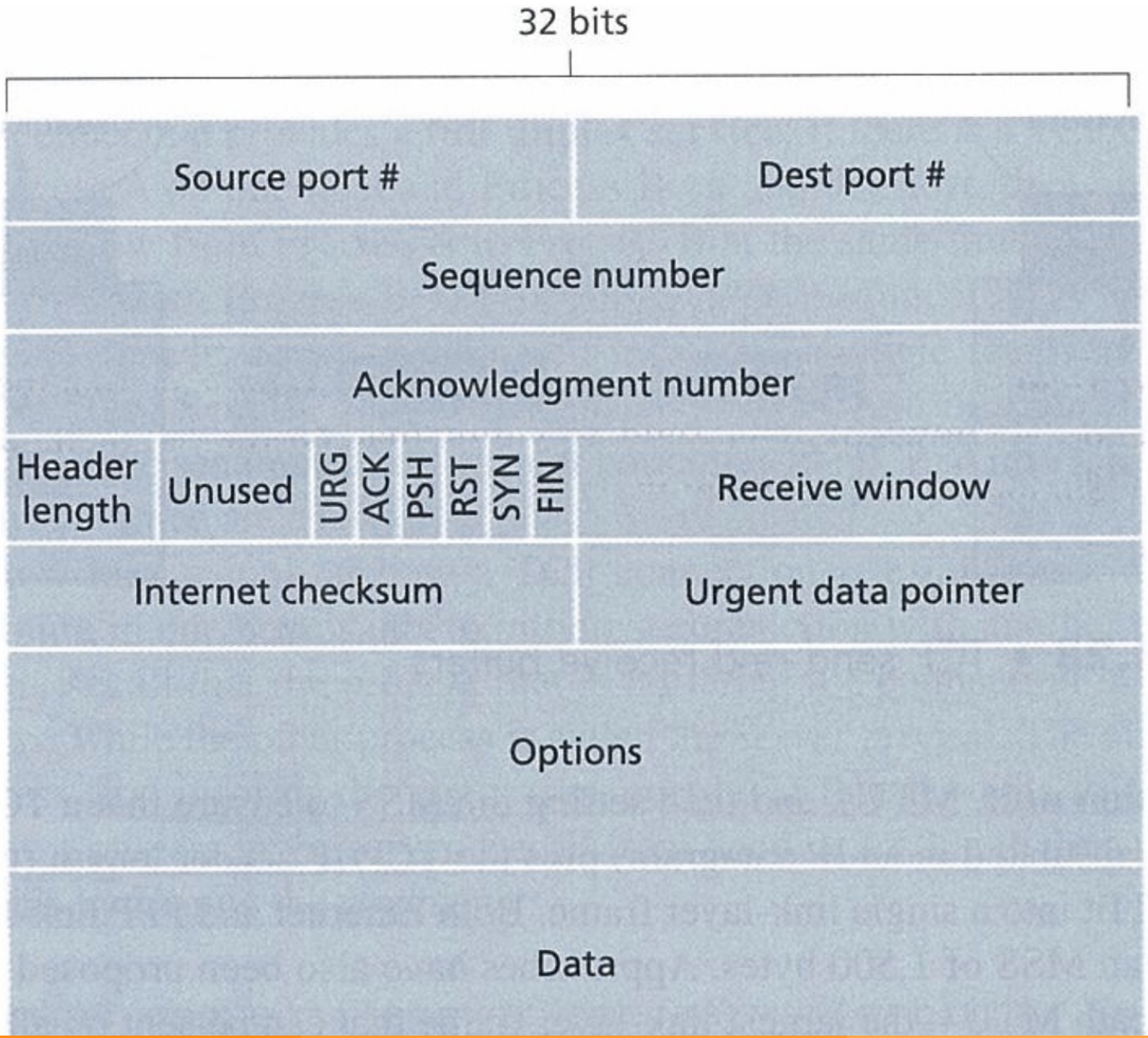
Pipelining

Packet pipelining

- The sender transmits a set of packets in a row
- Require send and receive buffers



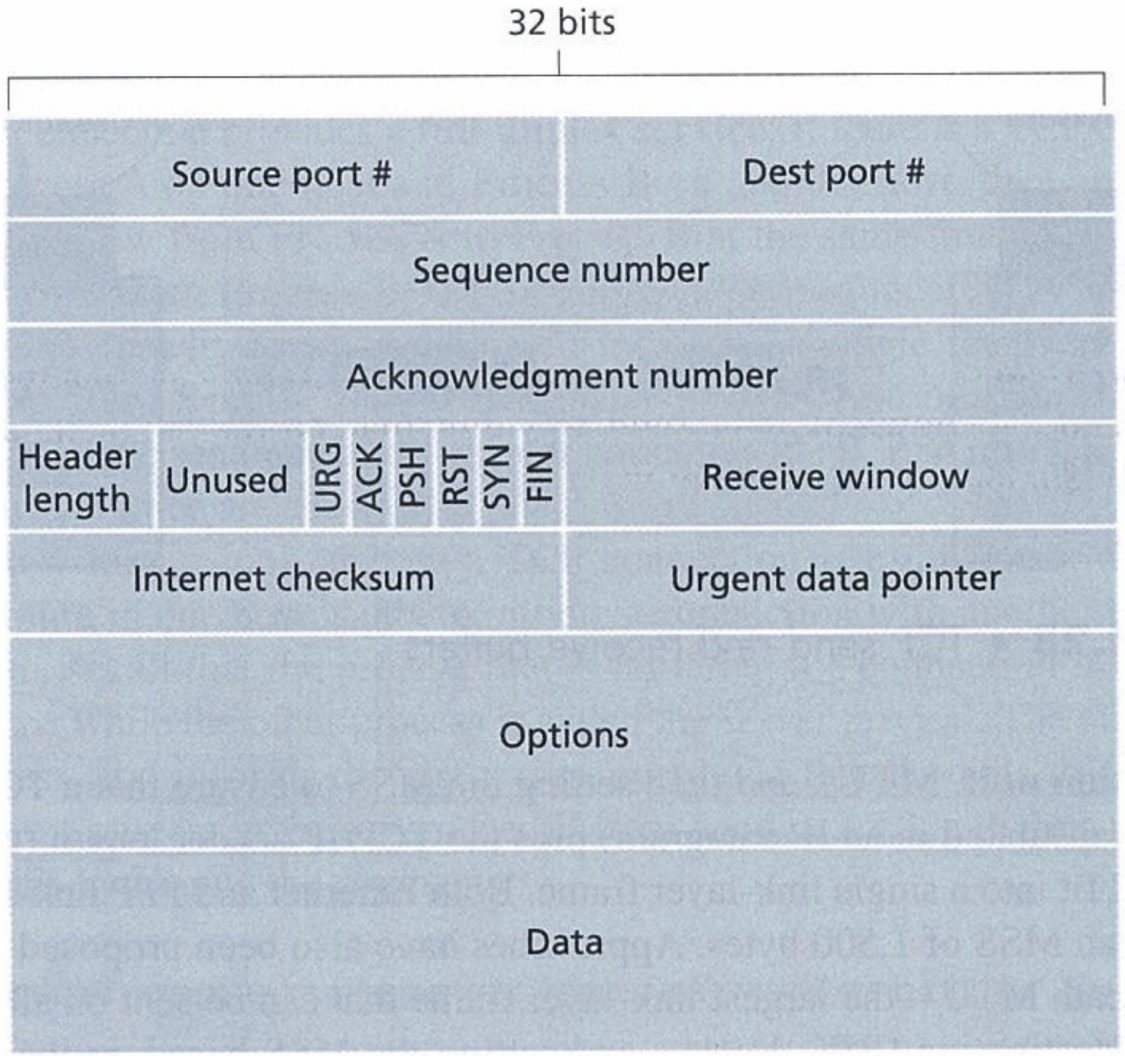
TCP: Header (20 bytes)



TCP: Header

Port numbers

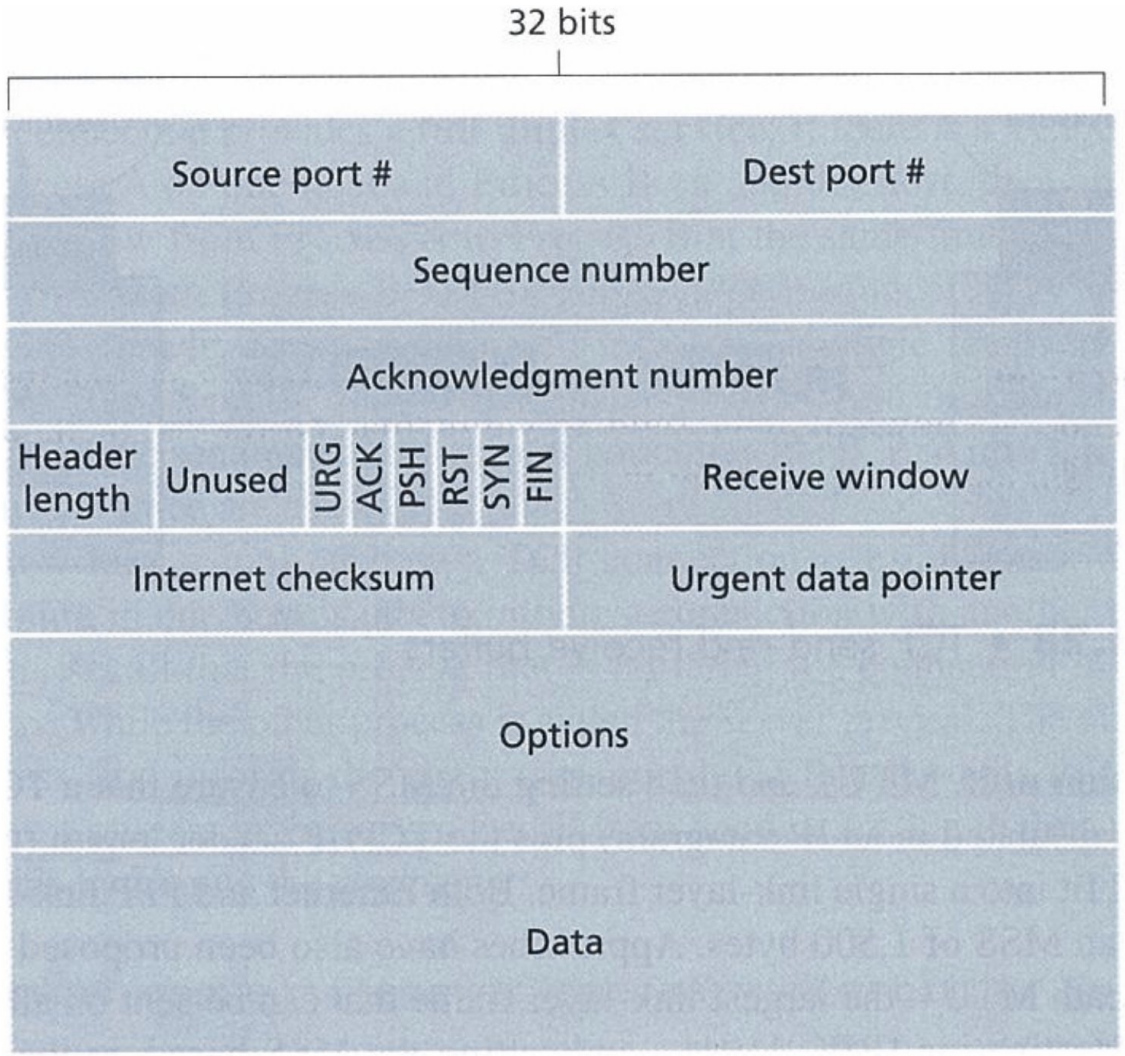
- Multiplexing & demultiplexing
- Source and destination
- 16 bits each



TCP: Header

Sequence number (32 bits)

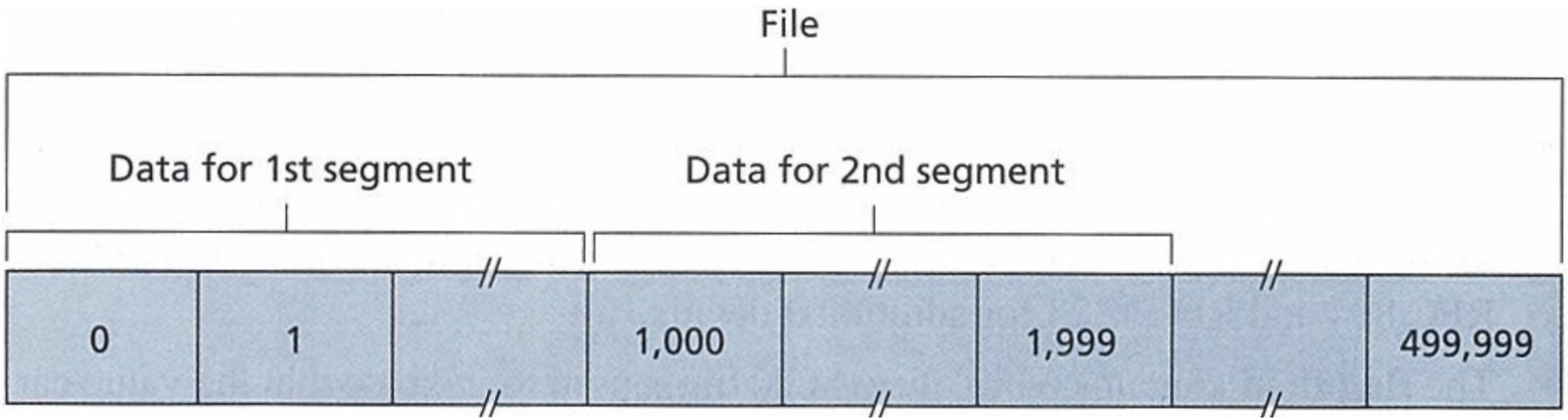
- Used for reliable data transfer



TCP: sequence number

Data segmentation and sequence number

- Sequence numbers in TCP are counting bytes of data (not segments!)
- Assume a Maximum Segment Size of 1000 bytes
- For a 500000 byte file, TCP constructs 500 segments out of the data stream.
- first segment, seq.num.=x+0; second segment, seq.num.=x+1000
  x is a random number set when the connection is established



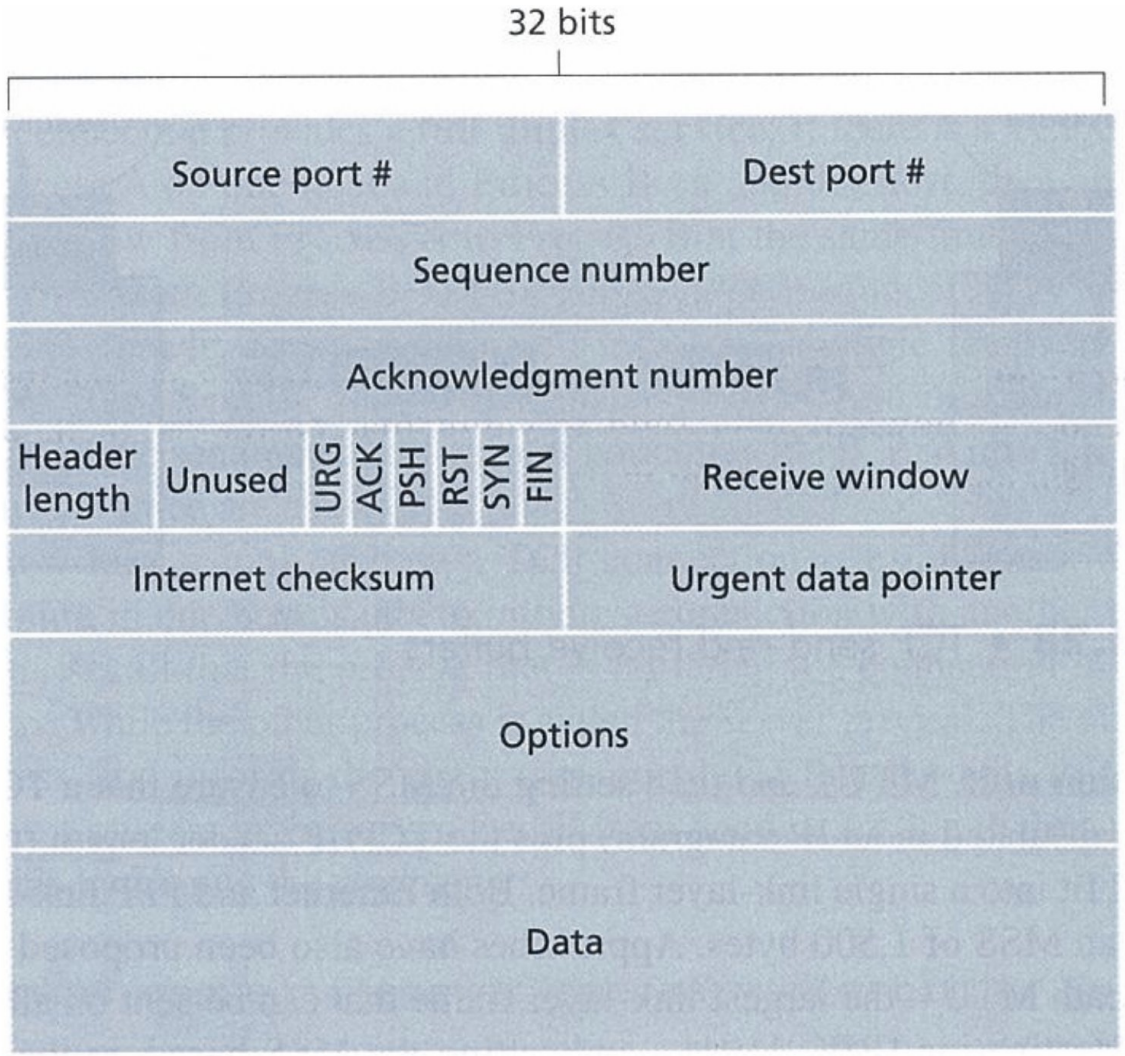
TCP: Header

Header length (4 bit)

Specifies the header length which can vary due to TCP options fields

Flag (6 bits)

aka Control bits



TCP: Header

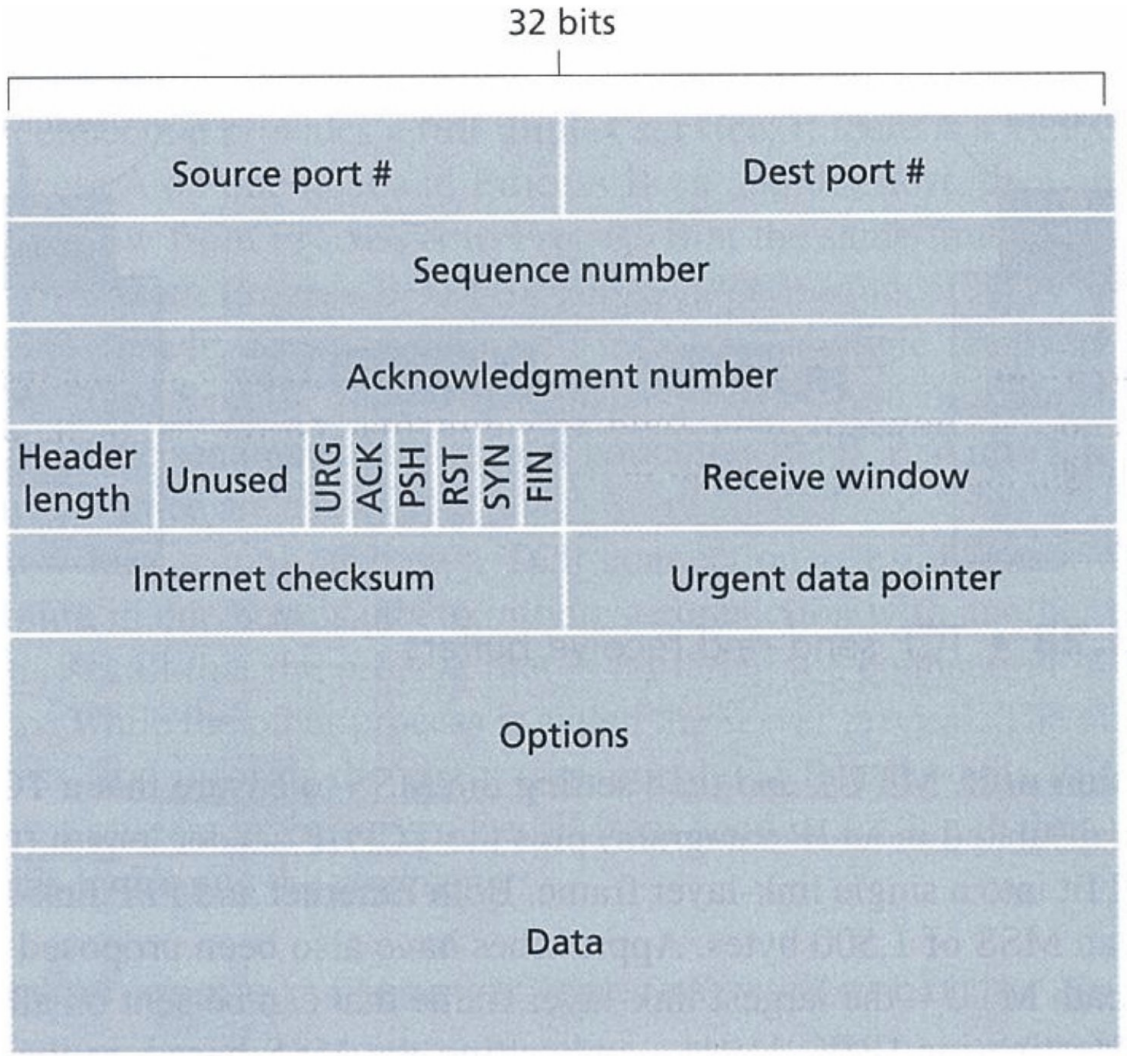
Flag (6 bit)

- ACK segment contains valid acknowledgement number
- RST,SYN,FIN Used for connection setup and closing
- PSH Indicates that receiver should pass to higher layer immediately
- URG Indicates urgent data (generally not used)

TCP: Header

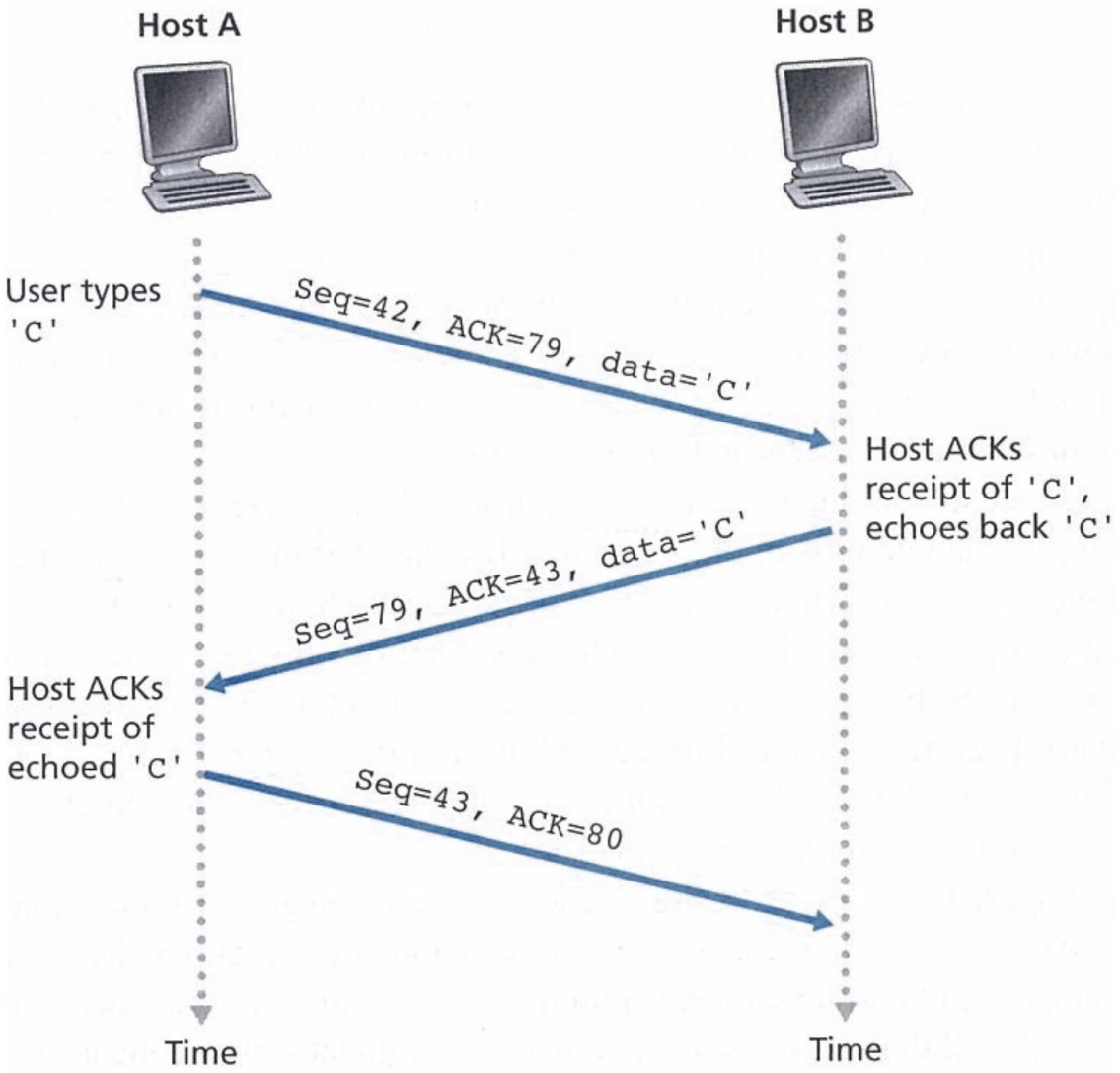
ACK number

- The acknowledgment number is the next byte expected by the receiver
- This is referred to as cumulative acknowledgment
- The receiver keeps out-of-order packets in its buffer and wait for missing packets



TCP: Header

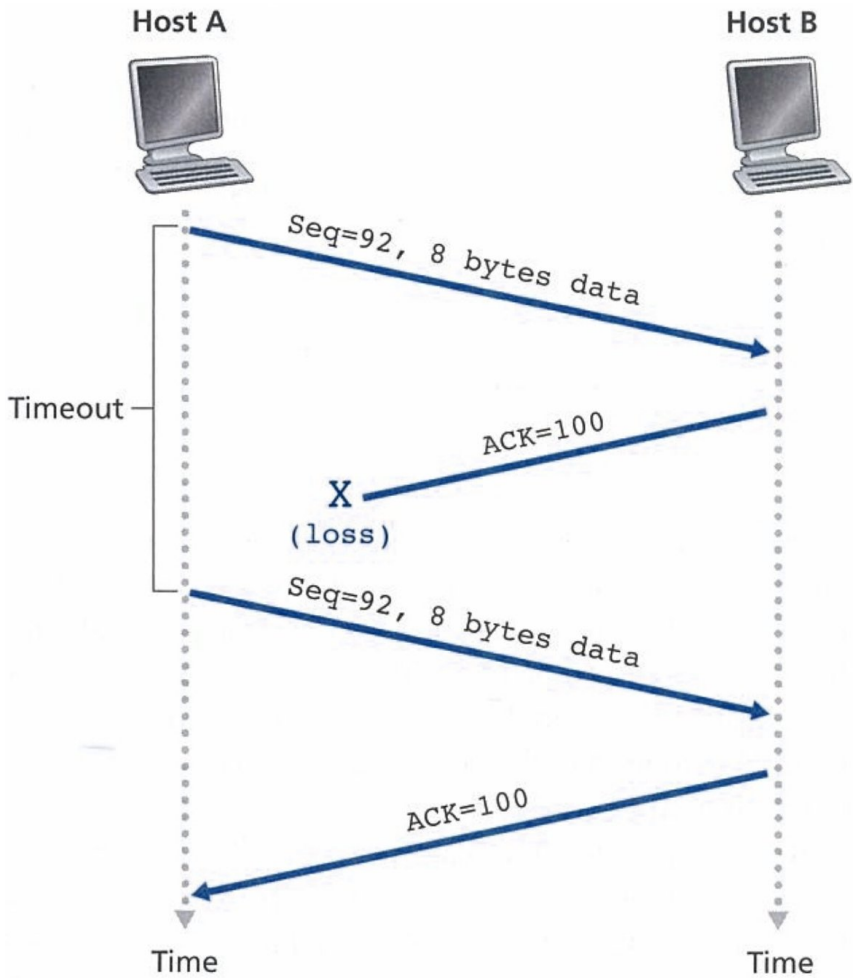
Example: ACK Number and sequence number



TCP – Timeout and retransmission

Retransmission

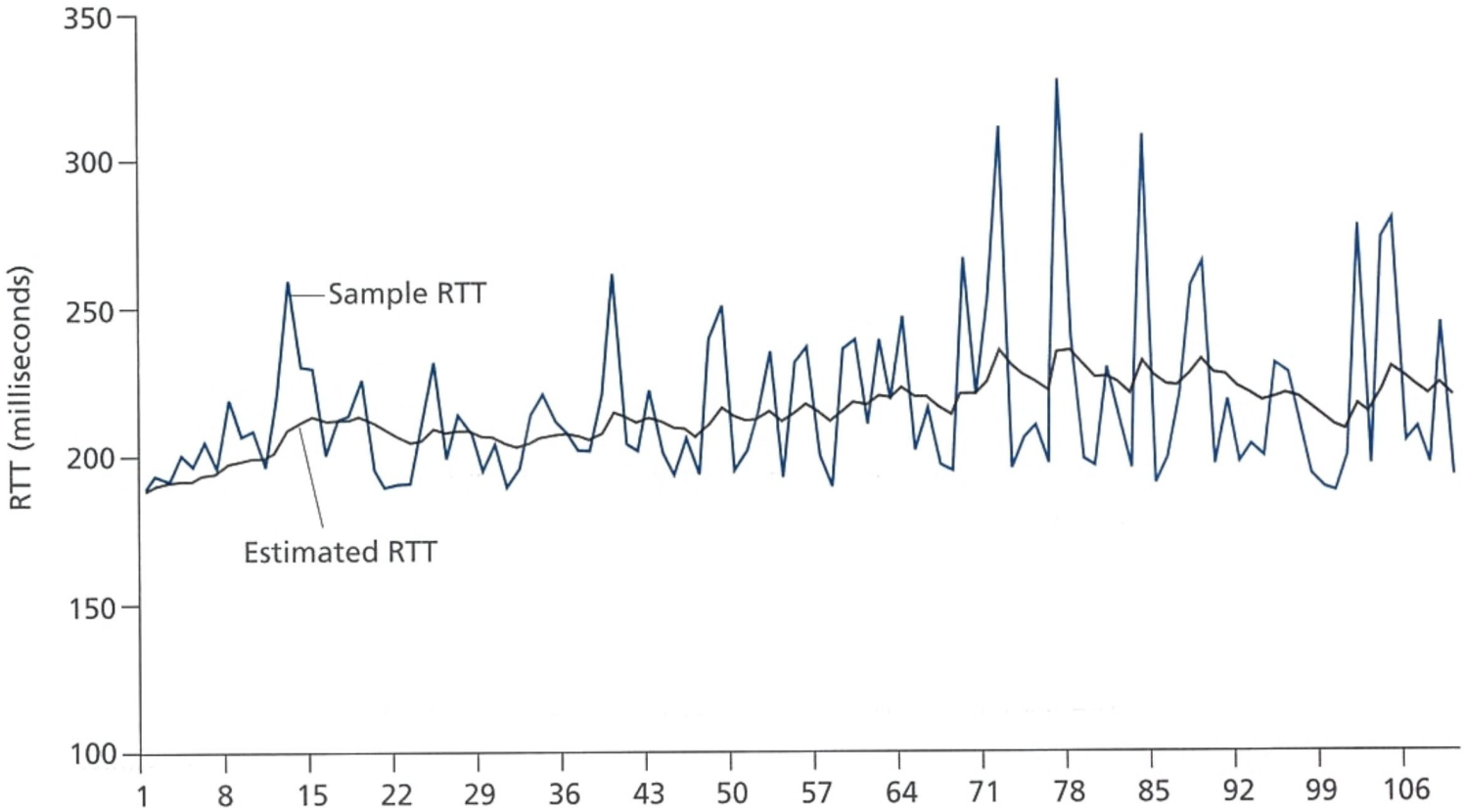
- If an ACK for a segment is not received after a certain timeout, the segment is retransmitted and timeout is doubled
- how to set TCP timeout value?
  - longer than RTT, but RTT varies...
  - too short: premature timeout, unnecessary retransmissions
  - too long: slow reaction to segment loss



TCP - Timeout and retransmission

Estimating the Round-Trip Time

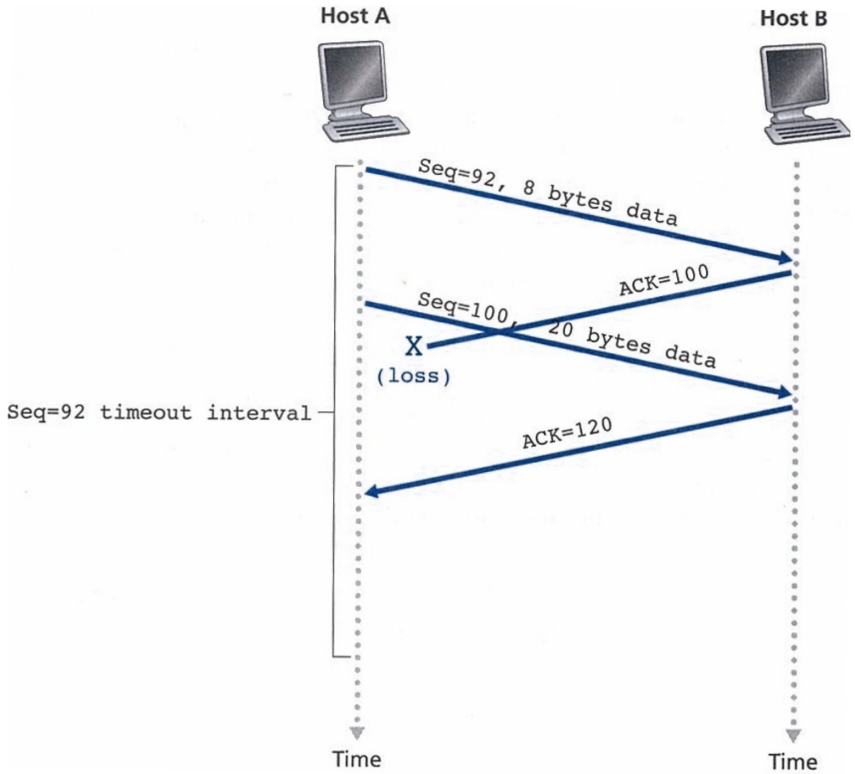
- Sample RTT of packets previously sent
- Exponential weighted moving average:
  $EstimatedRTT = (1 - \alpha)EstimatedRTT + \alpha SampleRTT$



TCP – Timeout and retransmission

Retransmission

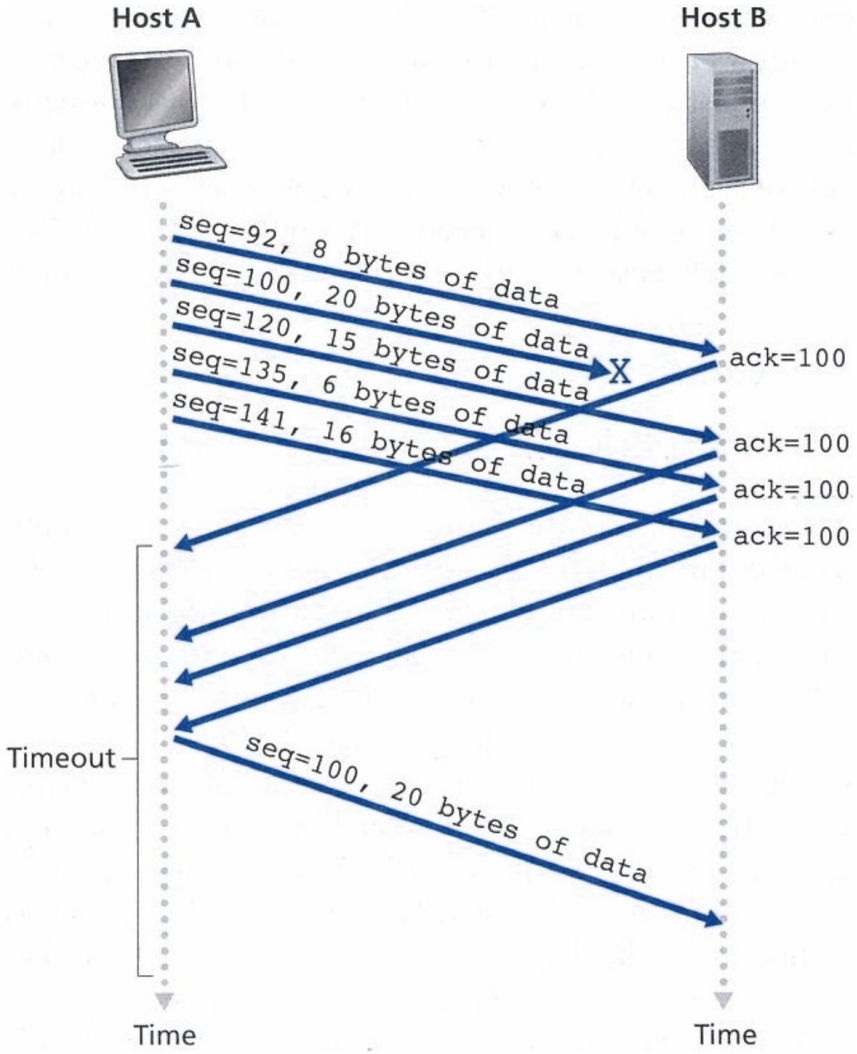
Due to cummulative acknowledgements, the transmitter does not have to retransmit when intermediate acknowledgements are lost



TCP – Timeout and retransmission

Fast Retransmission

- In order to improve data throughput, TCP performs a fast retransmission technique
- After receiving 3 duplicate acknowledgments, TCP will retransmit a segment even before timeout



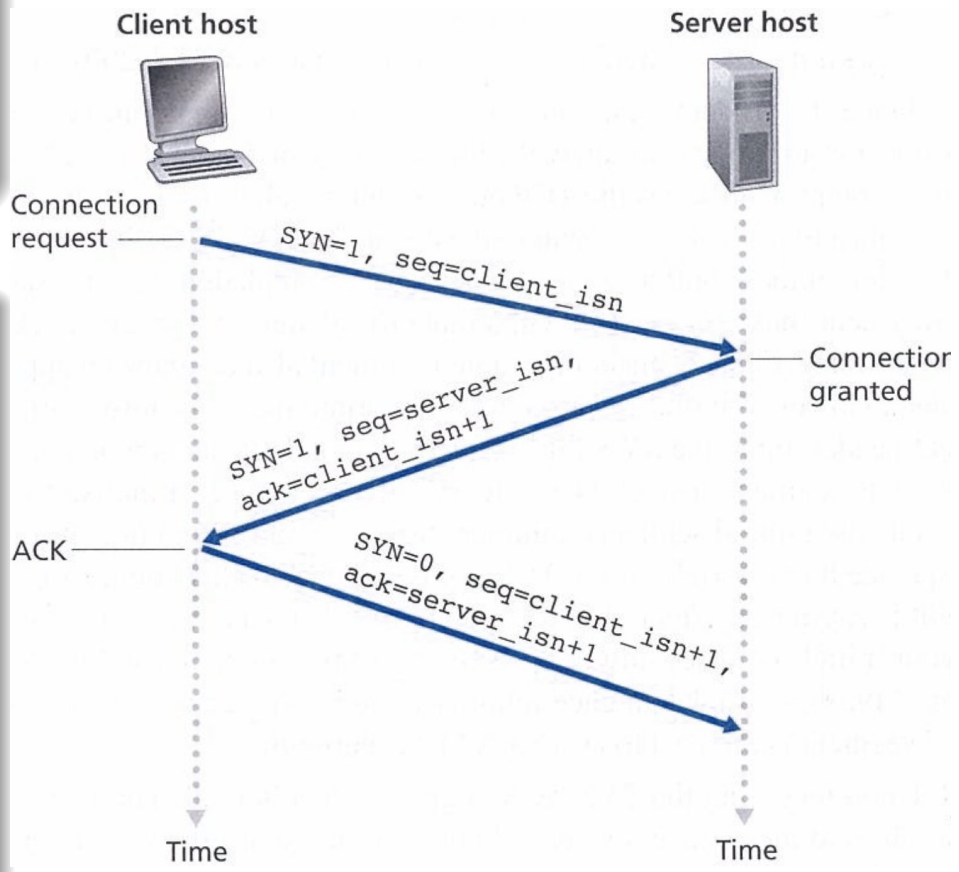
TCP – Connection establishment

Three-way handshake

- TCP set up connection in 3 steps

Step 1 (SYN segment)

- Client side sends special TCP segment with no application-layer data but with the SYN bit set to 1
- Also, the client adds a random initial sequence number (client_isn) to this segment



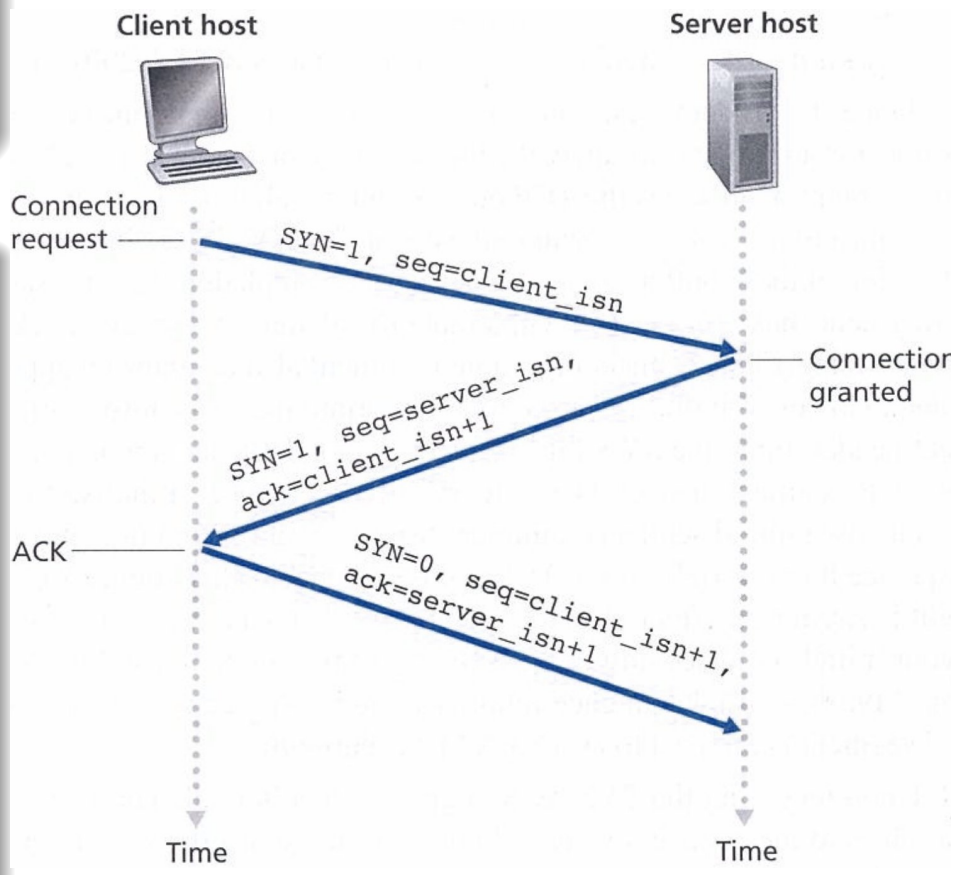
TCP – Connection establishment

Three-way handshake

- TCP set up connection in 3 steps

Step 2 (SYNACK segment)

- At receiving this SYN segment, server allocates TCP buffers and
- sends a connection granted segment back
  - SYN and ACK bits set to 1
  - acknowledgment field set to client_isn + 1
  - Sequence number chosen randomly (as client_isn)



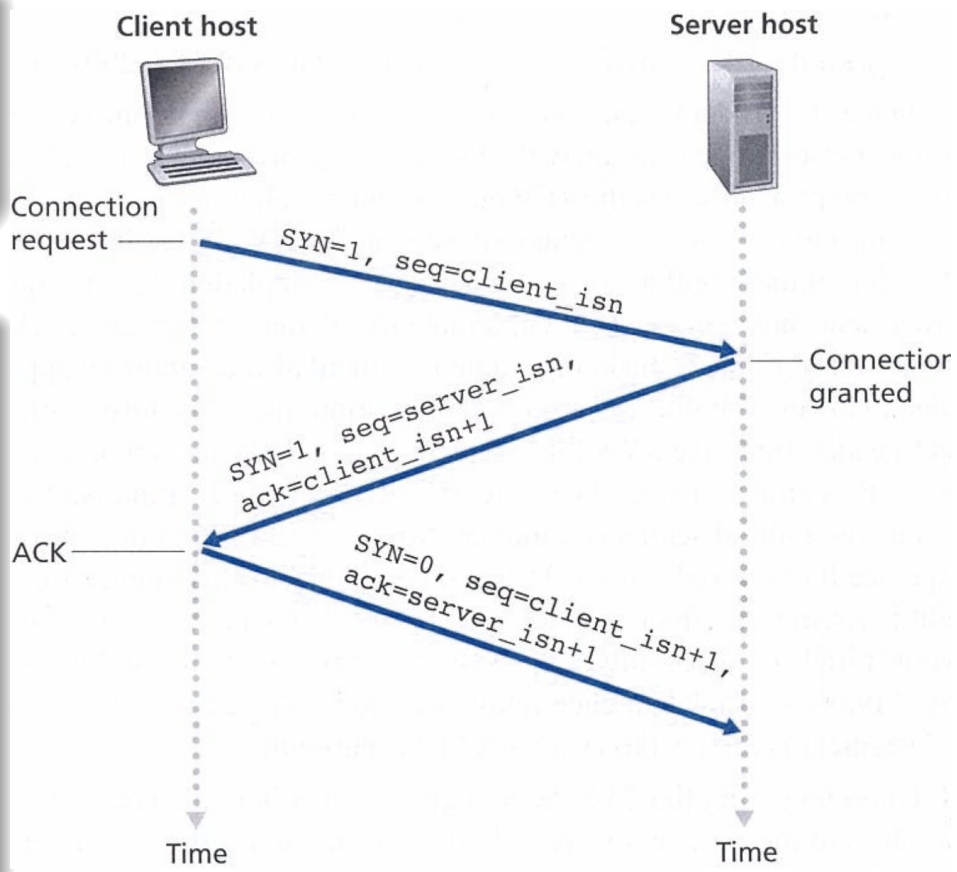
TCP – Connection establishment

Three-way handshake

- TCP set up connection in 3 steps

Step 3

- At receiving the SYNACK segment, the client allocates buffers for the connection
- Client sends ACK segment
  - SYN bit set to 0
  - acknowledgment field set to server_isn + 1

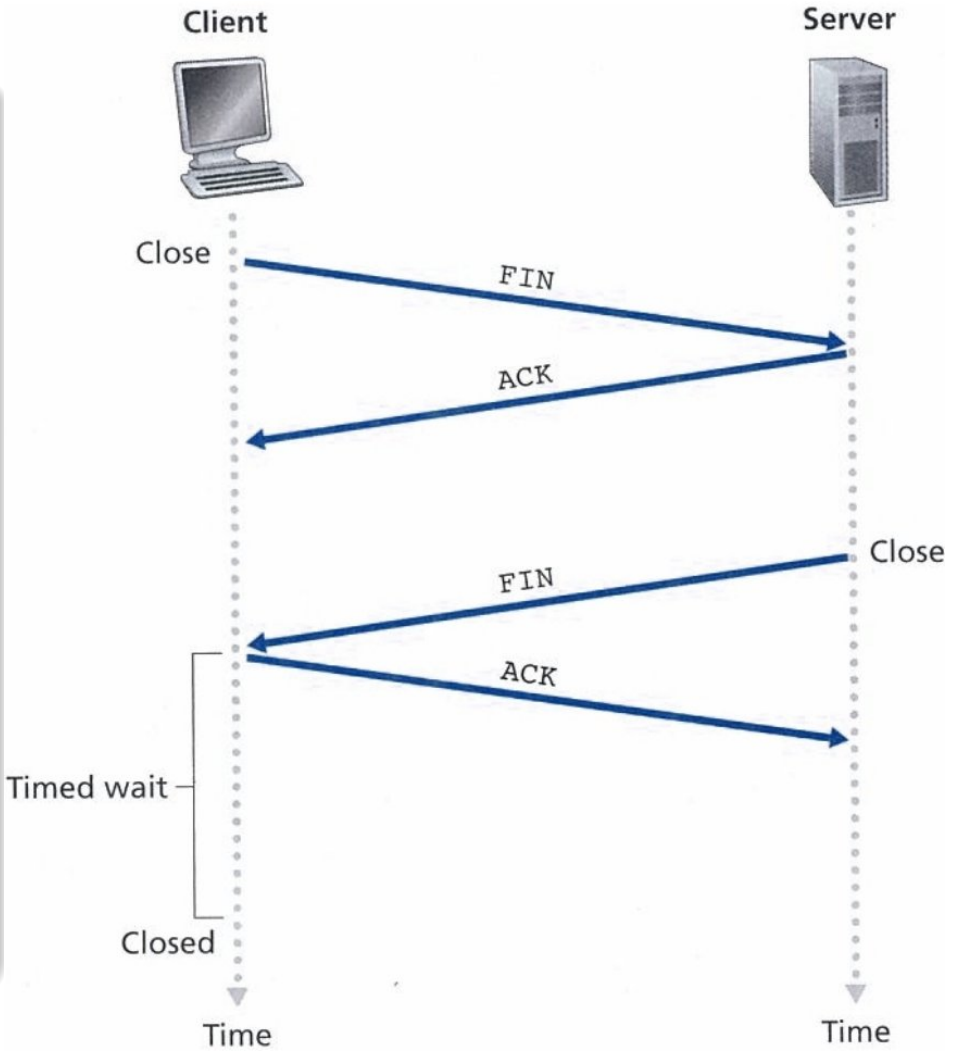


TCP – Closing a connection

To close a connection

1. TCP client sends special segment with FIN bit set to 1
2. Server sends an acknowledgment segment back
3. Server sends its own shutdown segment (FIN bit set to 1)
4. This is acknowledged by the client

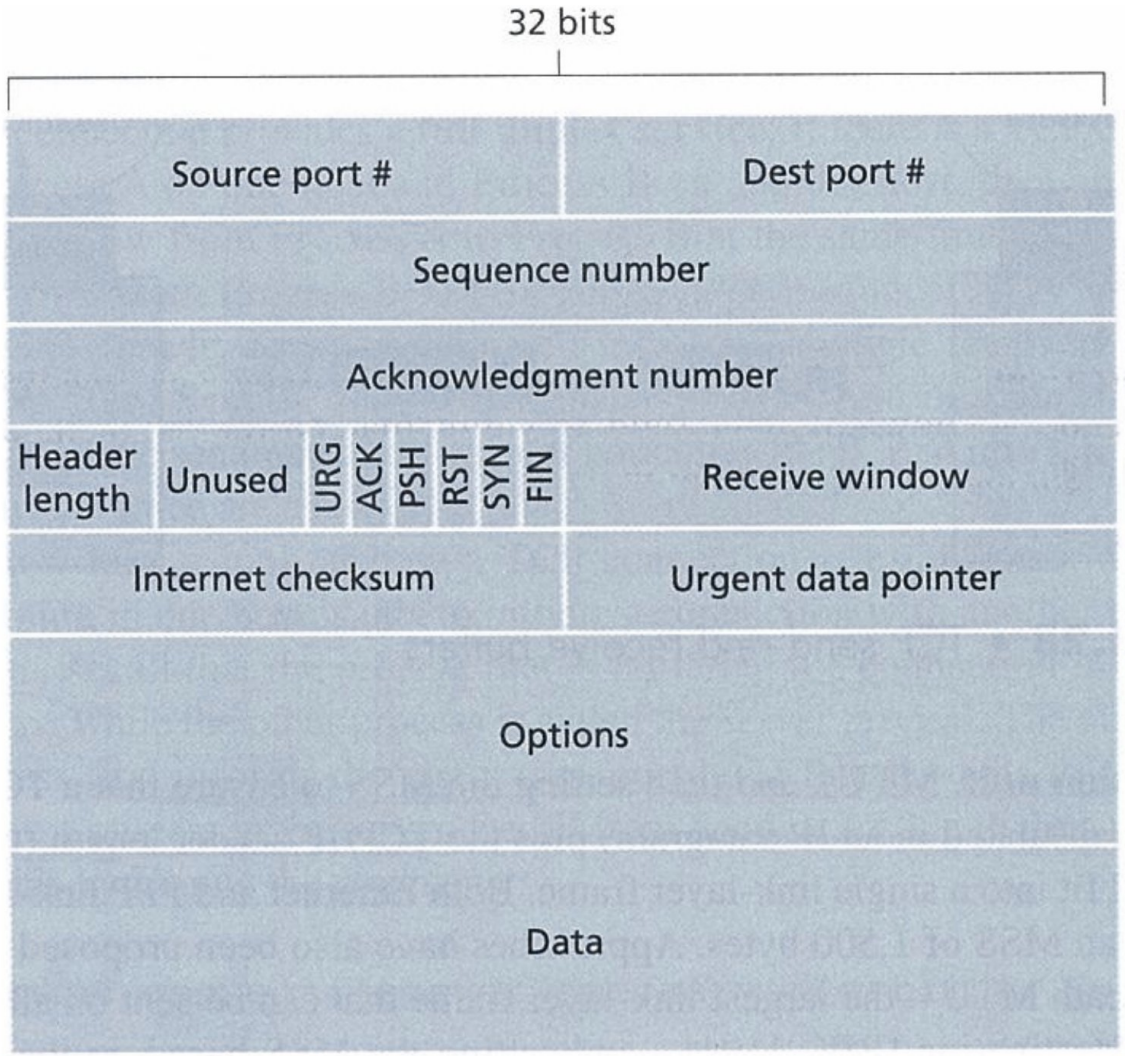
All resources of both parties are released (i.e. buffer&port number).



TCP: Header

Receive window (16 bits)

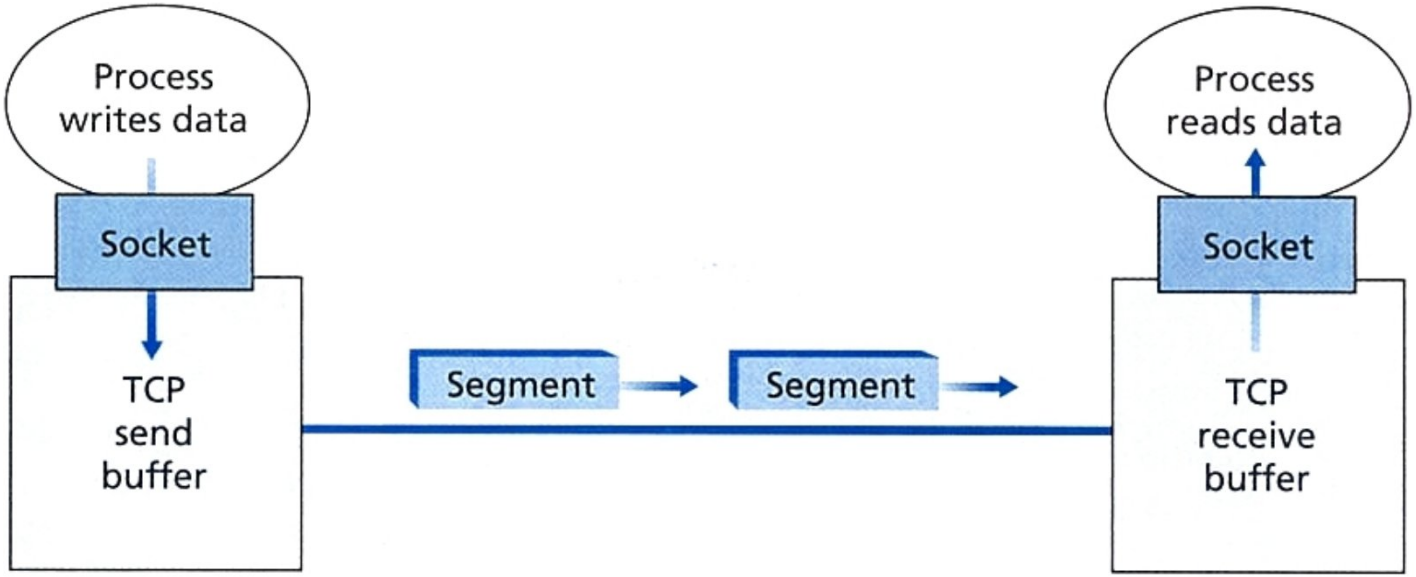
Indicates the number of bytes a receiver can accept (**Flow control**)



Flow Control

TCP Flow Control

- Hosts on each side of a TCP connection allocate a buffer
- Received data is placed in the buffer
- The buffer is read by the associated application process
- Sending data quicker than the app. reads causes buffer overflows
- TCP implements a flow-control service to prevent this event
- Field "receive window" informs the peer on the local buffer status



Congestion Control

Congestion Control

- Avoid overloading the network (congestion)
  (Remember congestion is the cause of delay and packet lost)
- While maximizing the throughput

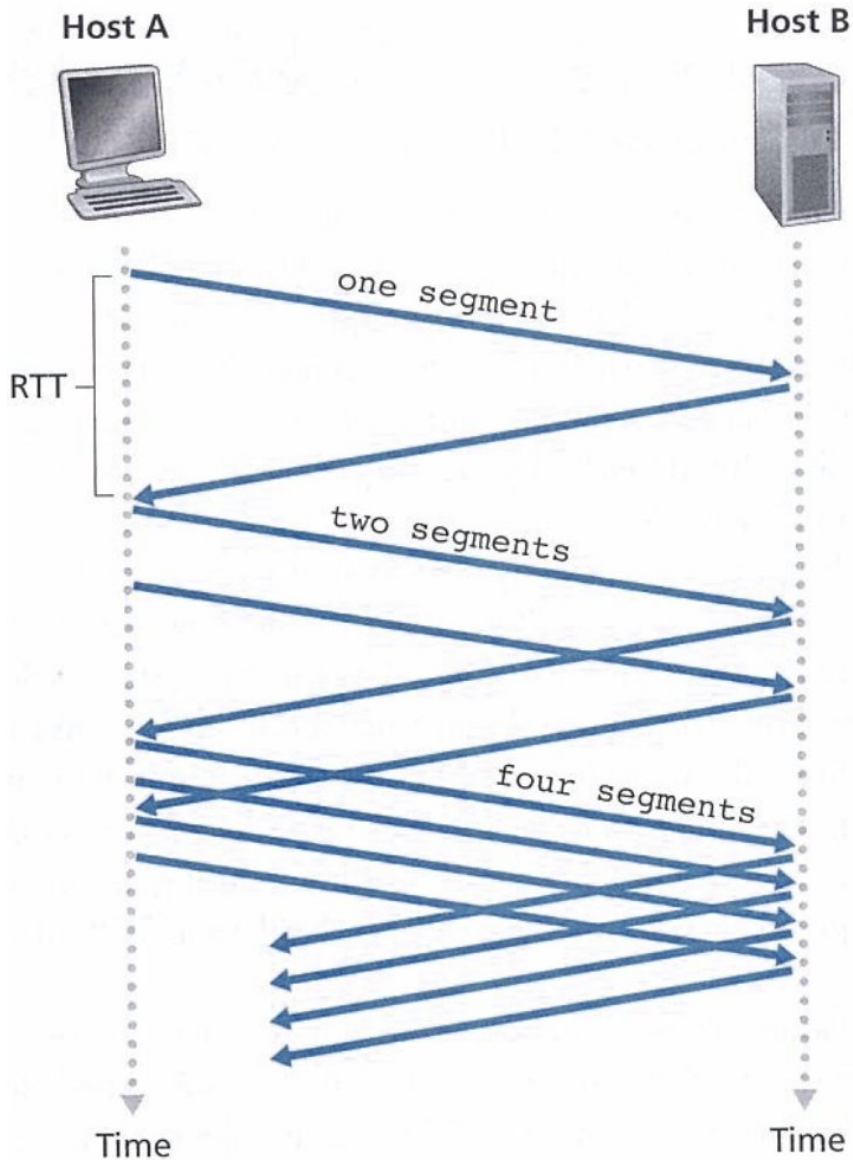
TCP guesses the amount of available bandwidth using:

- Slow start
- Congestion avoidance
- Fast recovery

TCP – Congestion control

Slow start

1. Initially, sending rate is 1 MSS (Maximum Segment Size) per RTT
2. If the transmission is successful, the sending rate is doubled each RTT (exponential increase)
3. If timeout occur, a threshold T is set to half of the current sending rate and start again at 1
4. When it reaches the threshold, TCP enters **congestion avoidance mode**



TCP – Congestion control

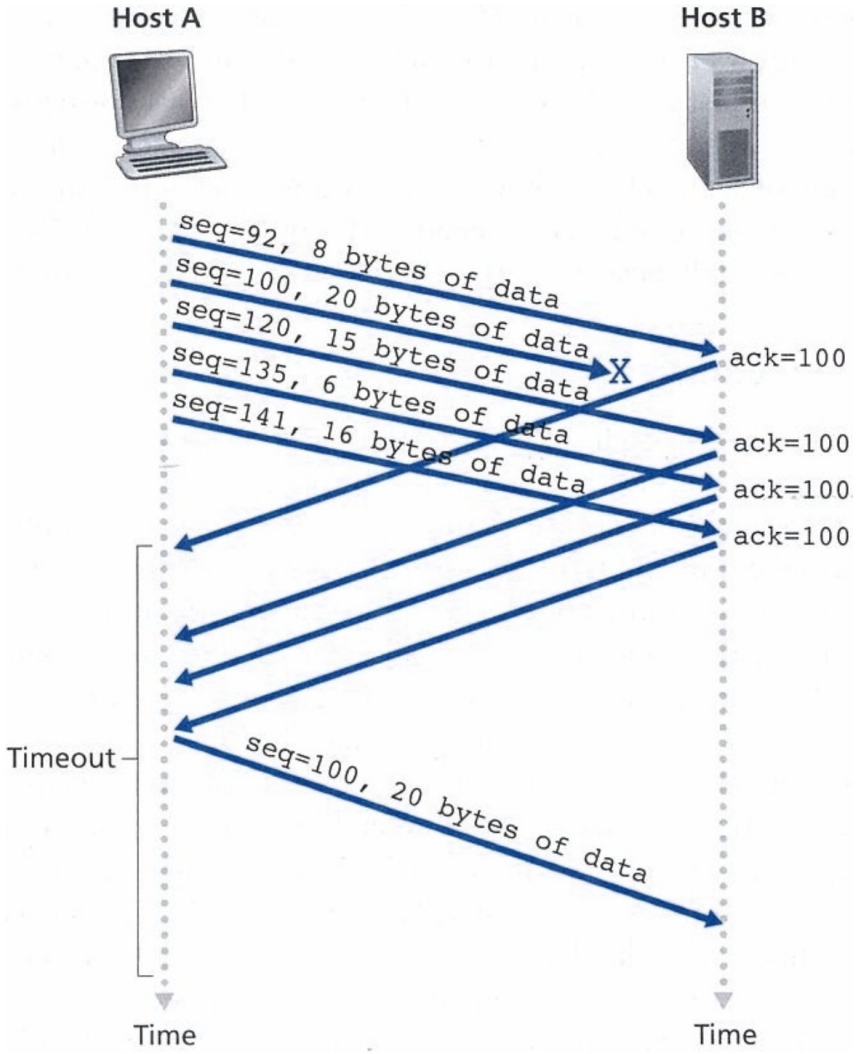
Congestion Avoidance Mode

1. The initial sending rate in congestion avoidance mode is close to congestion
2. Unlike slow start, linear increase of the sending rate
3. The sending rate is increased by one MSS every RTT
4. If a timeout occurs, the sending rate is again halved and TCP enters the **fast recovery state**

TCP – Congestion control

Fast Recovery State

1. The sending rate is increased by 1 MSS for each duplicate ACK received for the missing segment that caused TCP to enter fast recovery mode.
2. If timeout occurs, TCP starts over with slow start



TCP vs. UDP

TCP

- **Reliable**: No packet lost
- **Ordered**: Data arrives in the same order it was sent
- **Heavyweight**: Retransmission, buffering, +20 bytes header
- **Sending rate**: Congestion control throttles the sender
- **Streaming**: segments represent a flow of data
- Example: HTTP (port 80), FTP (port 21), SSH (port 22), SMTP (Email, port 25)

UDP

- **Unreliable**: no guarantee
- **No ordered**: messages are delivered as they arrive
- **Lightweight**: No retransmission, maintain any connection state, 8 bytes header
- **Sending rate**: Data sent as soon as available
- **Datagram-like**: Packets are sent individually
- Example: DNS (port 53), DHCP, IPTV, Voice over IP (VoIP), RIP

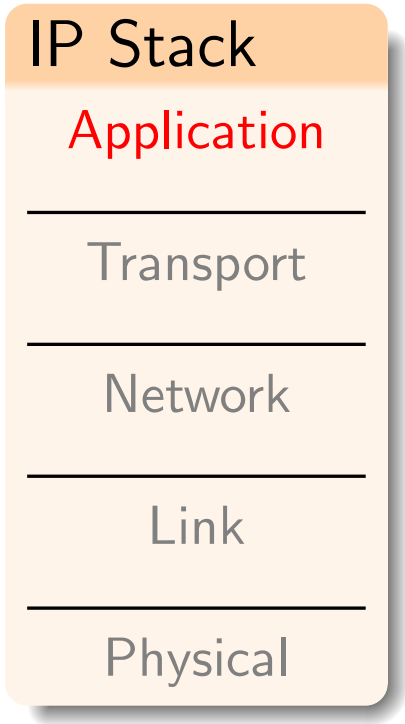
The Transport Layer: Summary

Today's lecture covered:

- Transport Layer Services
- UDP
- TCP

In the next lecture

We'll see the application layer: HTTP, FTP, Email, DNS



Today's important points

- TCP
- TCP
- differences between TCP and UDP