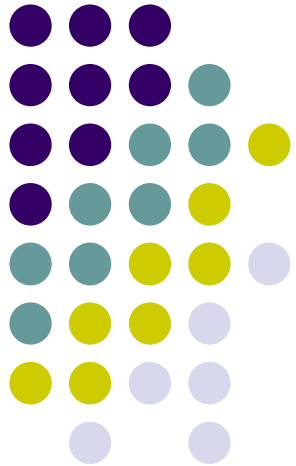
Chapter 3

Transport layer

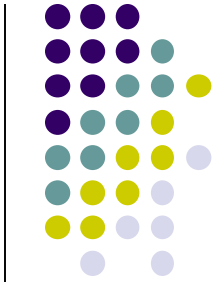
Liping Shen 申丽萍

lpshen@sjtu.edu.cn



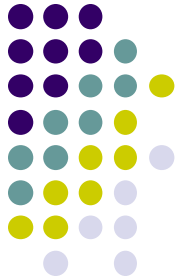
These slides are based upon the exceptional slides provided by Kurose and Ross

Chapter 3 outline



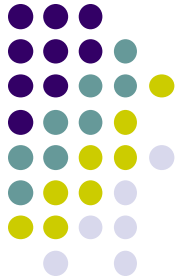
- 3.1 Transport-layer services
- 3.2 Multiplexing and demultiplexing
- 3.3 Connectionless transport: UDP
- 3.4 Principles of reliable data transfer
- 3.5 Connection-oriented transport: TCP
  - segment structure
  - connection management
  - reliable data transfer
  - flow control
  - Timer management
- 3.6 Principles of congestion control
- 3.7 TCP congestion control

TCP: Overview RFCs: 793, 1122, 1323, 2018, 2581



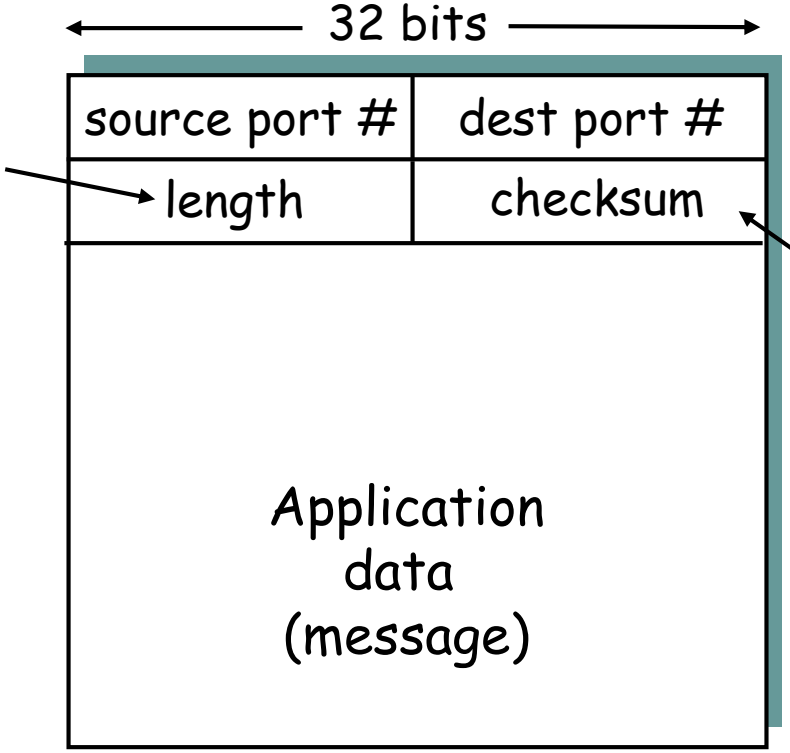
- end-to-end:
  - No support for multicasting or broadcasting.
- connection-oriented:
  - handshaking initiates sender, receiver state before data exchange
- reliable byte stream:
  - no “message boundaries”
  - hybrid of GBN and SR
- full duplex data:
  - bi-directional data flow in same connection
  - ACK piggybacked on data in reverse direction
- flow & congestion control:
  - dynamic send & receive buffers
  - TCP congestion and flow control set window size

Chapter 3 outline



- 3.1 Transport-layer services
- 3.2 Multiplexing and demultiplexing
- 3.3 Connectionless transport: UDP
- 3.4 Principles of reliable data transfer
- 3.5 Connection-oriented transport: TCP
  - segment structure
  - connection management
  - reliable data transfer
  - flow control
  - Timer management
- 3.6 Principles of congestion control
- 3.7 TCP congestion control

Recap: UDP segment format

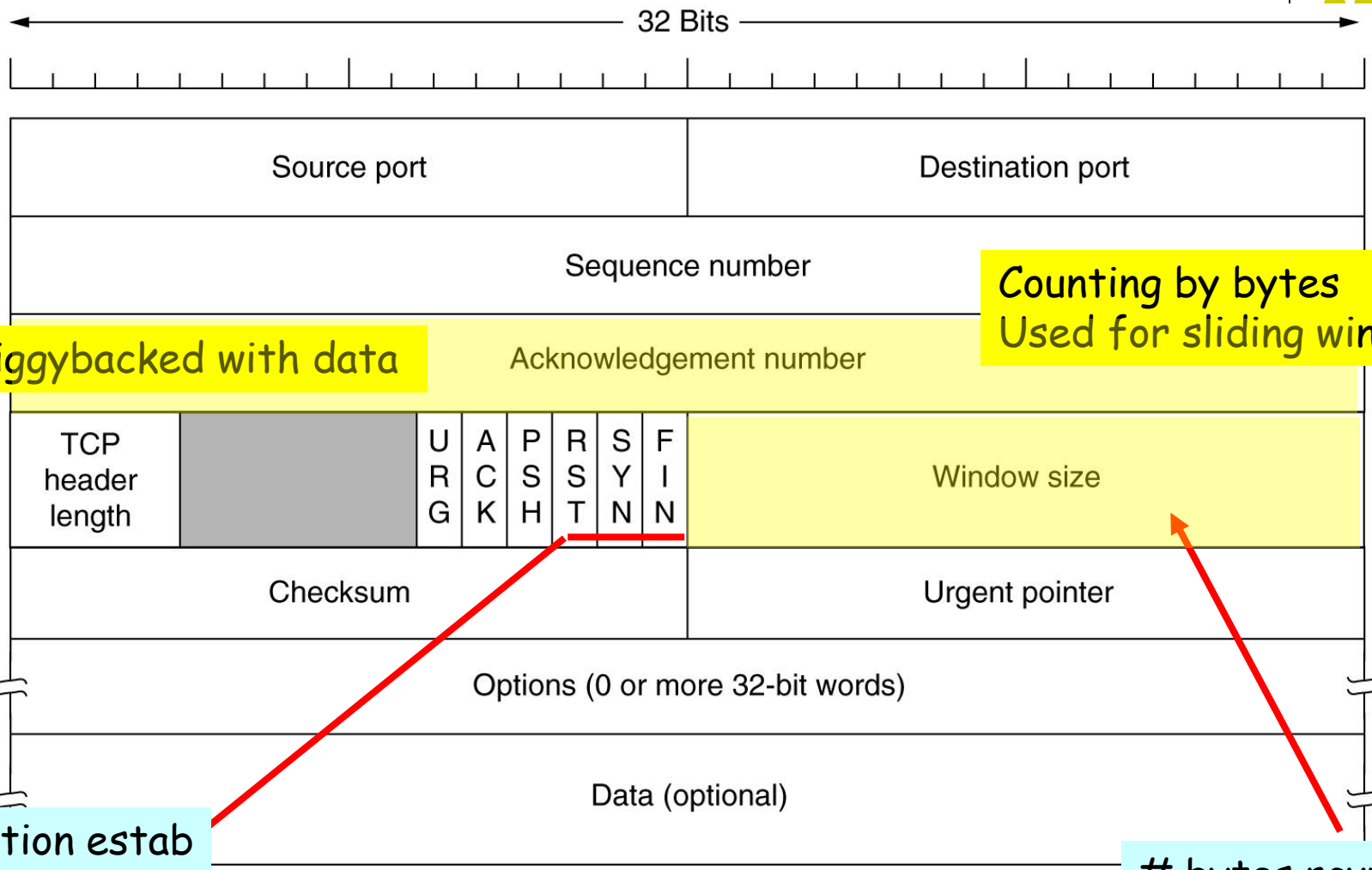


Length in bytes of UDP segment, including header

detect error, discard segment when error detected

Why "UDP has better control over what data is sent and when" ?

The TCP Header



ACK piggybacked with data

Counting by bytes
Used for sliding window

connection estab (setup, teardown commands)

bytes rcvr willing to accept

TCP Header.

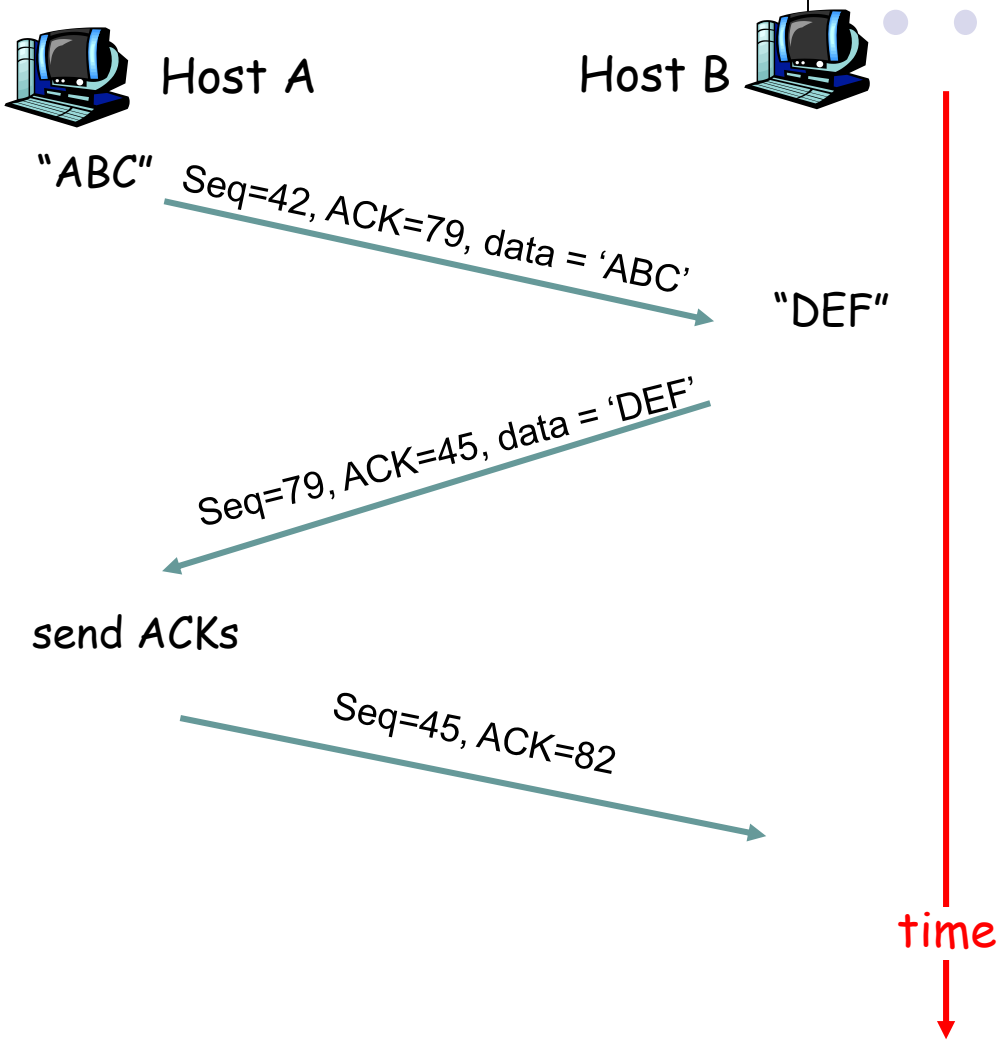
TCP seq. #'s and ACKs

Seq. #'s:

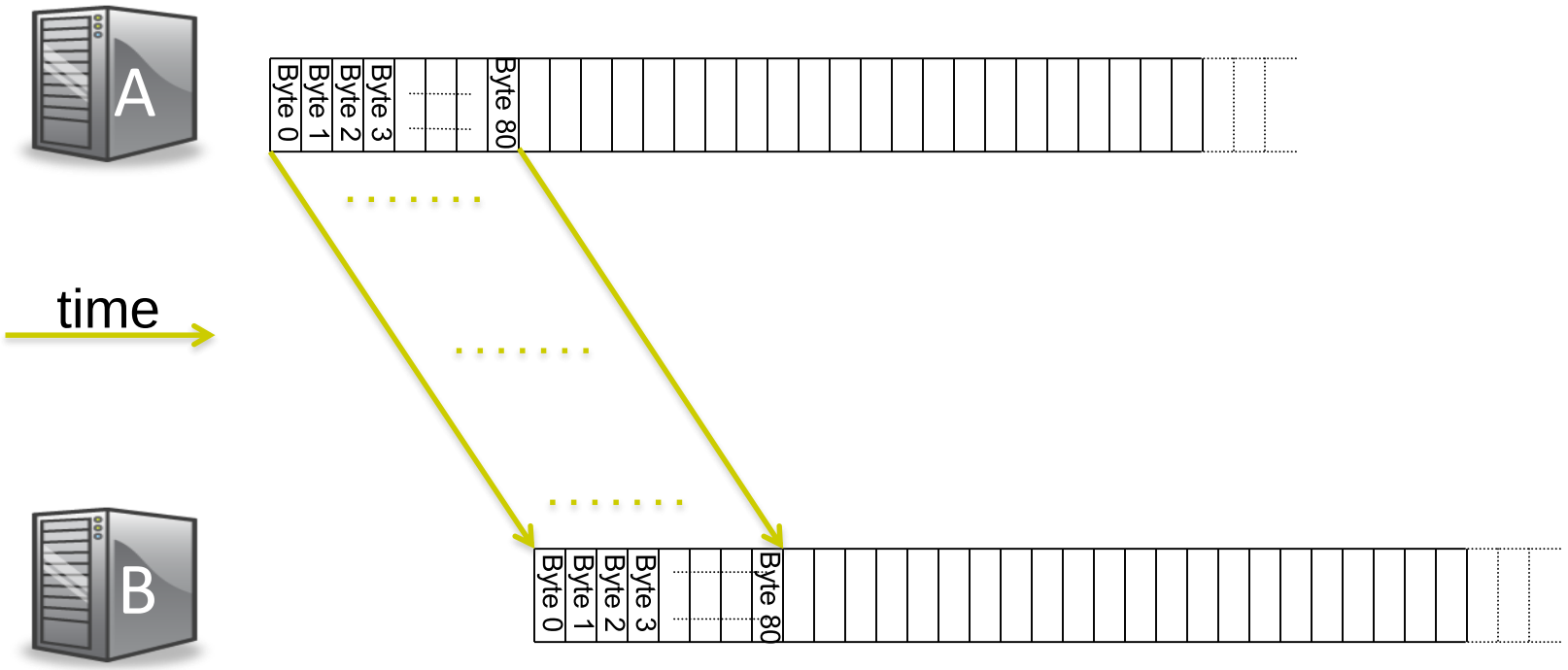
- byte stream "number" of first byte in segment's data

ACKs:

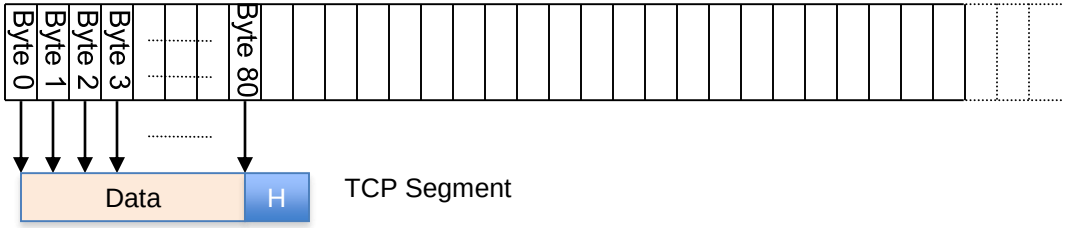
- seq # of next byte expected
- cumulative and piggyback ACK
- (optional)selective ACKs
  - ACK up to 100 and 200~299

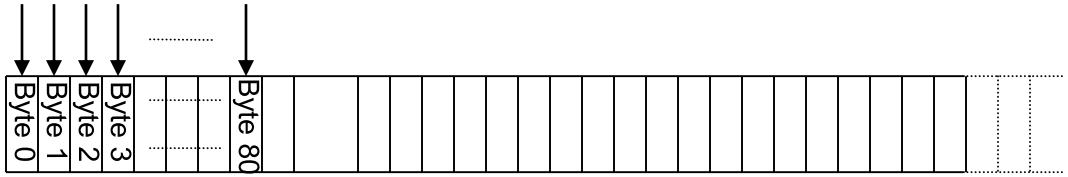


TCP “stream of bytes” service



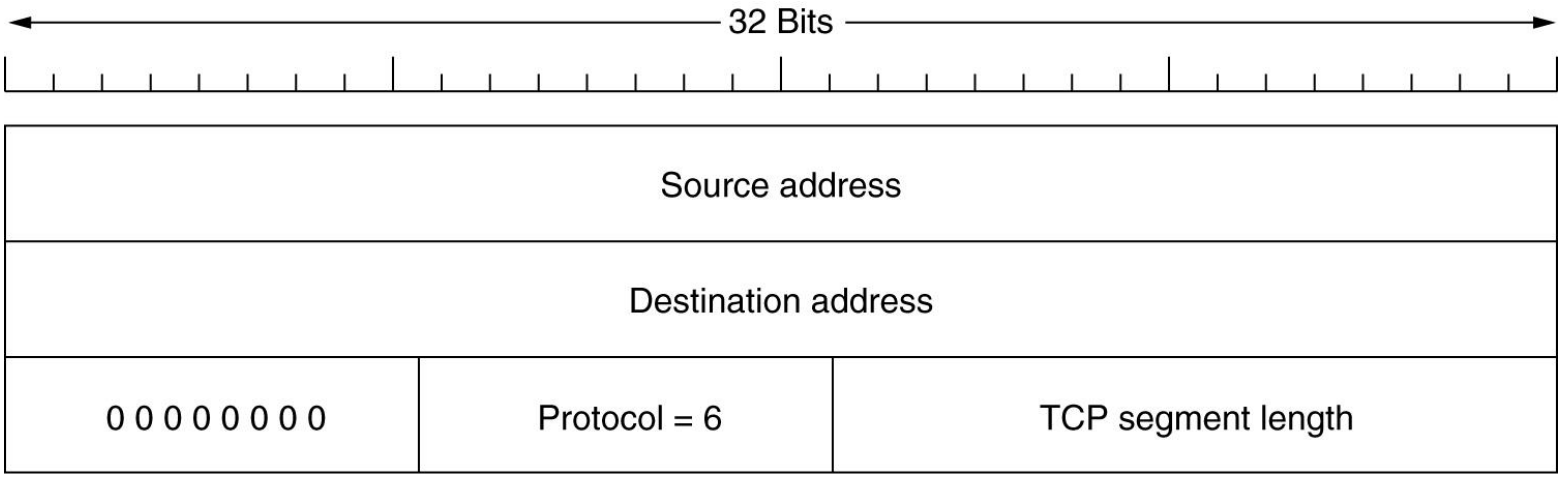
Bytes stream sent as TCP segments





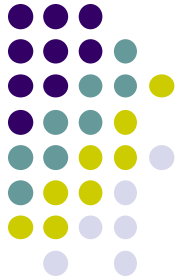
The TCP Checksum

- TCP checksum checks the header, the data, and a pseudo header.
- The pseudo-header helps detect mis-delivered packets.
- It also violates the protocol hierarchy since the IP addresses in it belong to the IP layer, not to the TCP layer.



The pseudoheader included in the TCP checksum.

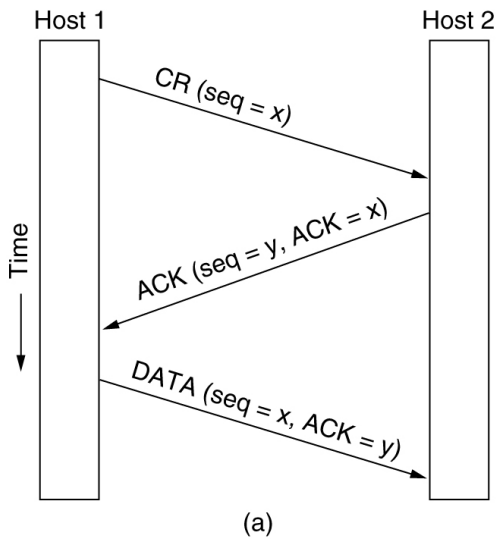
Chapter 3 outline

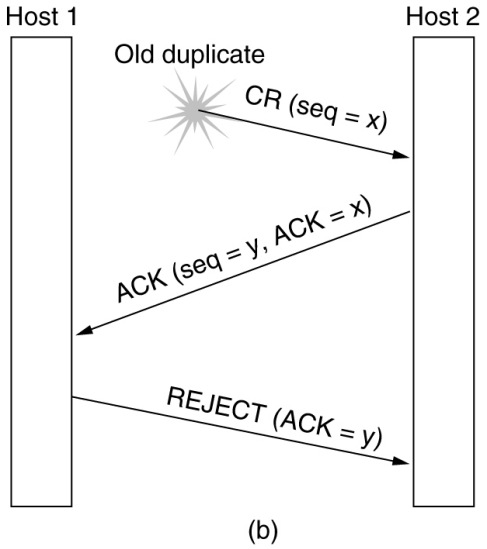


- 3.1 Transport-layer services
- 3.2 Multiplexing and demultiplexing
- 3.3 Connectionless transport: UDP
- 3.4 Principles of reliable data transfer
- 3.5 Connection-oriented transport: TCP
  - segment structure
  - connection management
  - reliable data transfer
  - flow control
  - Timer management
- 3.6 Principles of congestion control
- 3.7 TCP congestion control

Reliable TCP Connection Establishment

- One-way connection may result in duplicate connection.
- Uses three-way handshake protocol :
  - SYN/CR
  - SYN/CR+ACK
  - ACK [+data]

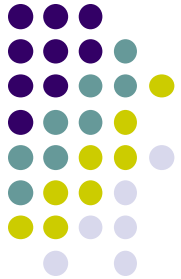




(a) normal case.

(b) delayed duplicate CR.

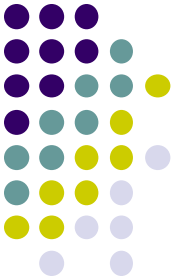
Graceful TCP Connection Release



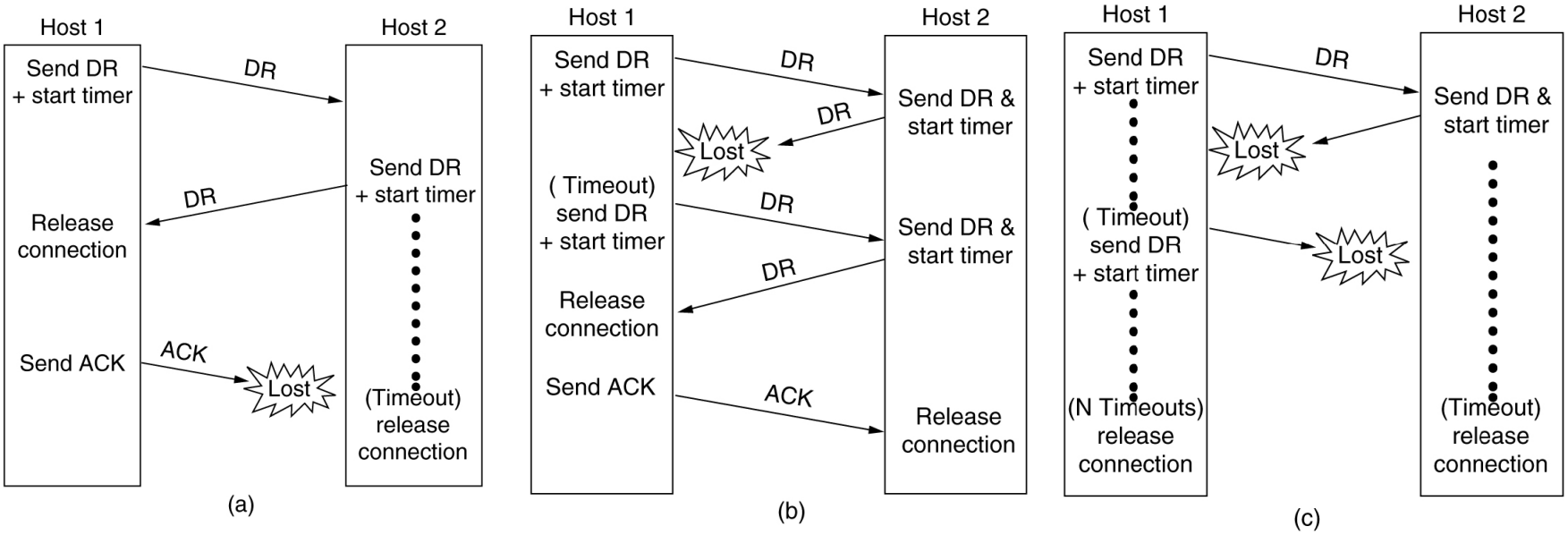
- Asymmetric release (one party just closes down the connection) may result in loss of data.
- Three-way handshake:
  - FIN/DR
  - FIN/DR+ACK
  - ACK
- Data may continue to flow indefinitely in one direction when the other direction is shut down.

Demo

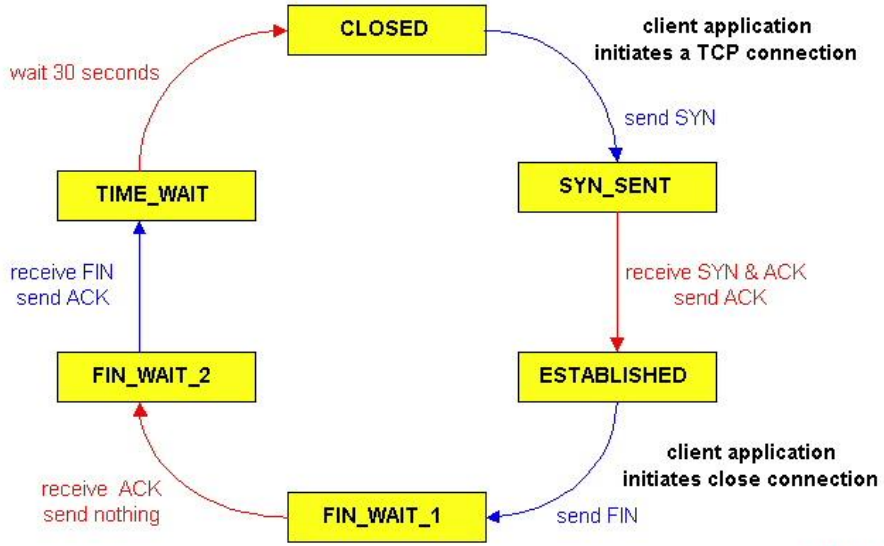
Graceful TCP Connection Release
– Keep-alive timer



- To avoid half-open connection, keep-alive timers are used by the second FIN sender.

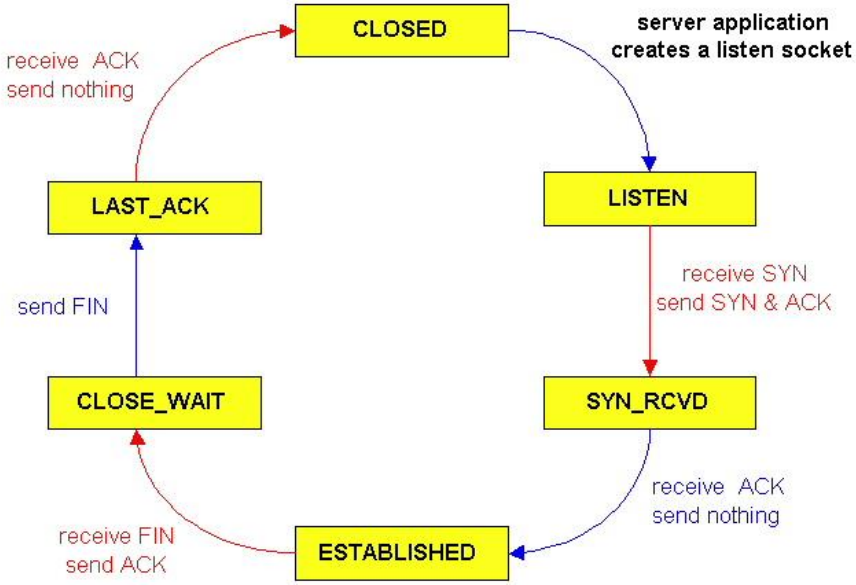


TCP Client/Server Lifecycle

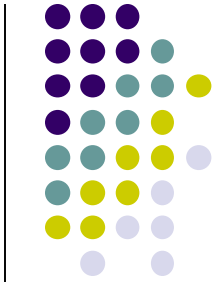


TCP client lifecycle

TCP server lifecycle

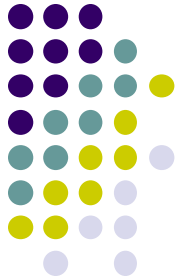


Chapter 3 outline



- 3.1 Transport-layer services
- 3.2 Multiplexing and demultiplexing
- 3.3 Connectionless transport: UDP
- 3.4 Principles of reliable data transfer
- 3.5 Connection-oriented transport: TCP
  - segment structure
  - connection management
  - reliable data transfer
  - flow control
  - Timer management
- 3.6 Principles of congestion control
- 3.7 TCP congestion control

Pipelining Protocols



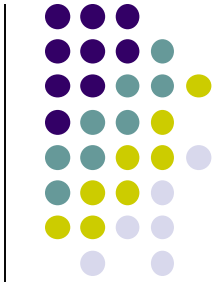
Go-back-N:

- Sender can have up to N unacked packets
- Receiver only buffer one pkt.
- Rcvr only sends cumulative acks
  - Only ack the last pkt received OK in order.
- Sender has only one timer
  - If timer expires, retransmit all unacked packets

Selective Repeat:

- Sender can have up to N unacked packets
- Receiver can buffer up to N out-of-order pkt.
- Rcvr acks individual packets
- Sender maintains timer for each unacked packet
  - When timer expires, retransmit only unack packet

TCP reliable data transfer

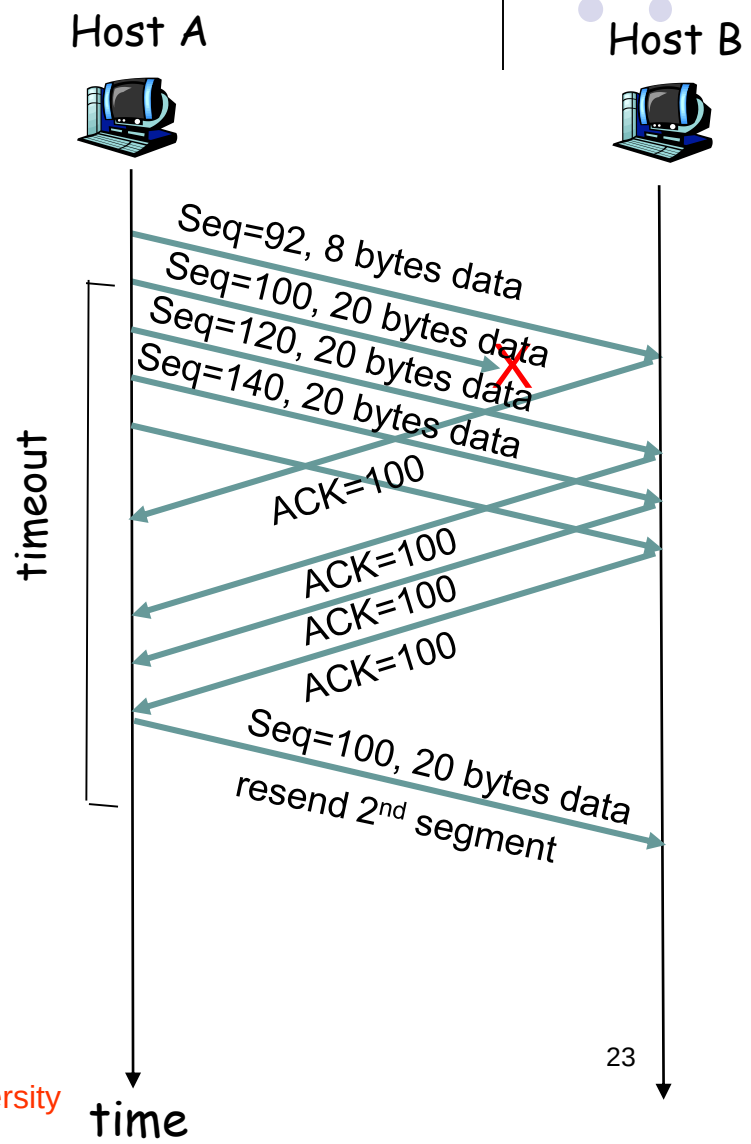


Hybrid of GBN and SR.

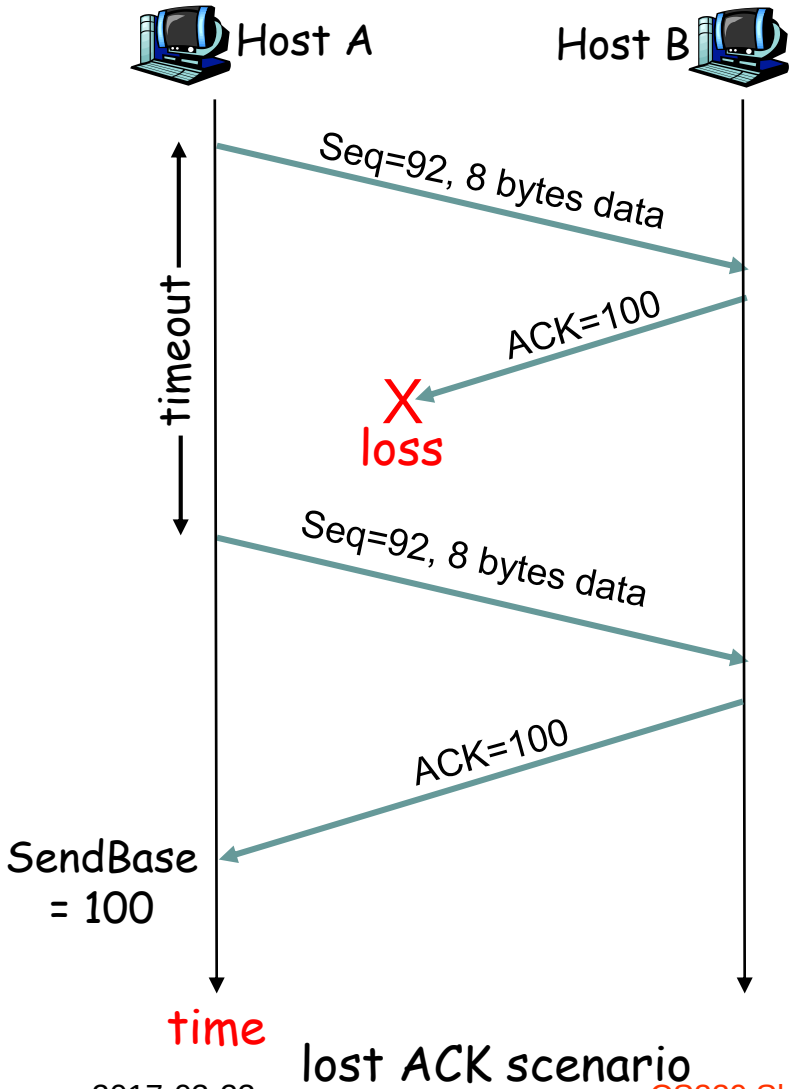
- pipelined segments
- cumulative and piggyback acks
- single retransmission timer
- retransmissions :
  - are triggered by
    - timeout events
    - 3 duplicate acks: fast retransmit
  - only retransmit one segment

Fast Retransmit

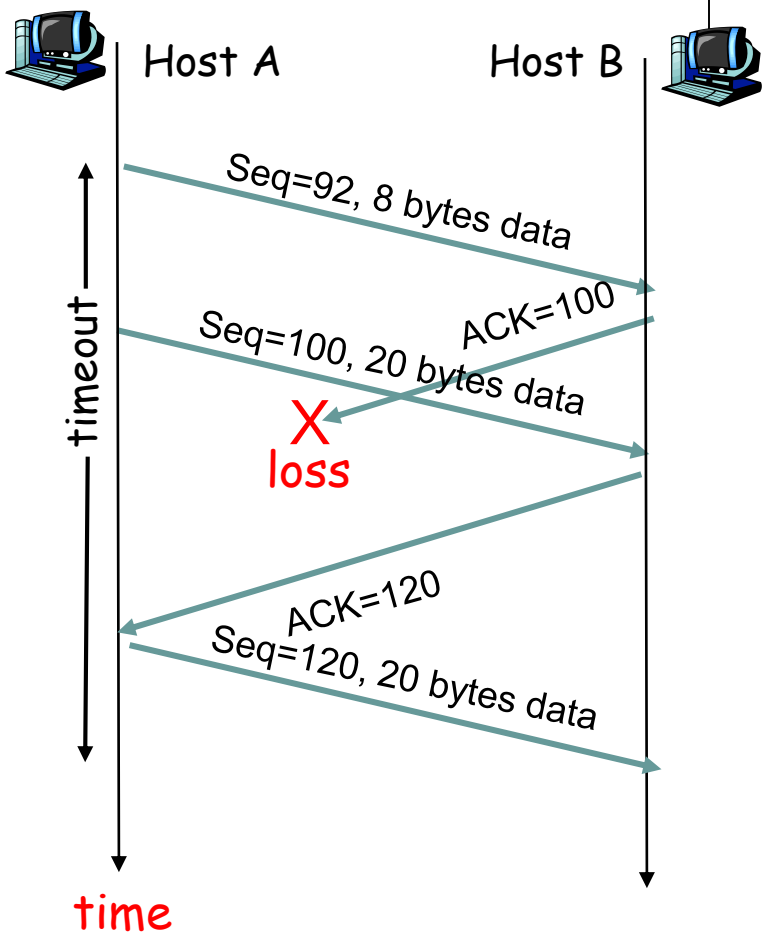
- Time-out period often relatively long:
  - long delay before resending lost packet
- Detect lost segments via duplicate ACKs.
  - If segment is lost, there will likely be many duplicate ACKs.
- fast retransmit: If sender receives 3 duplicate ACKs, it will resend segment before timer expires



TCP: retransmission scenarios

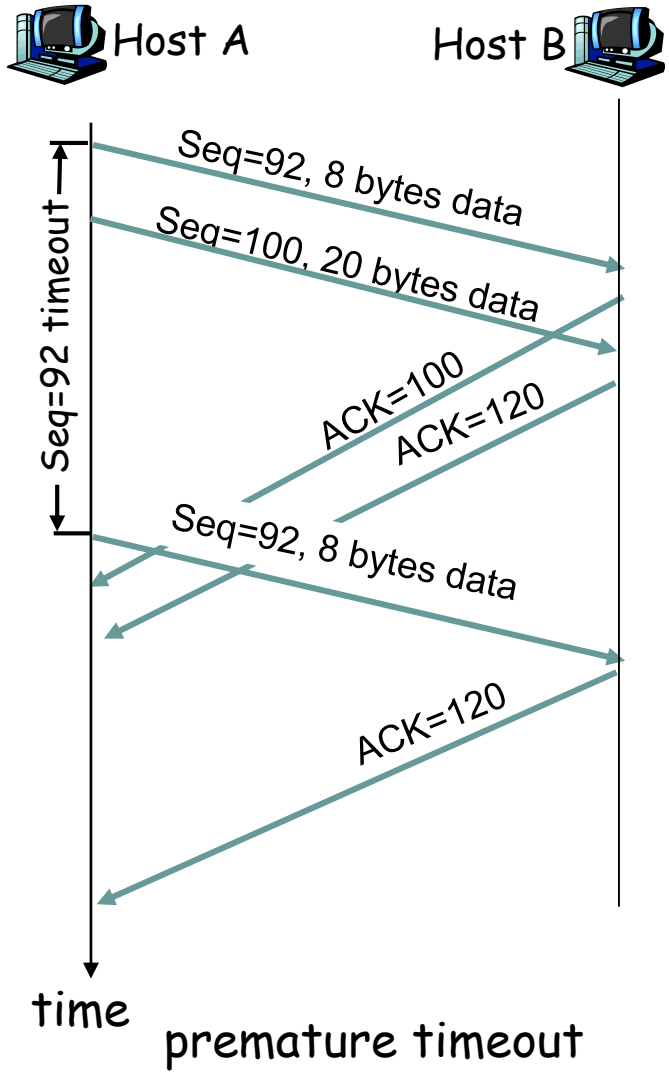


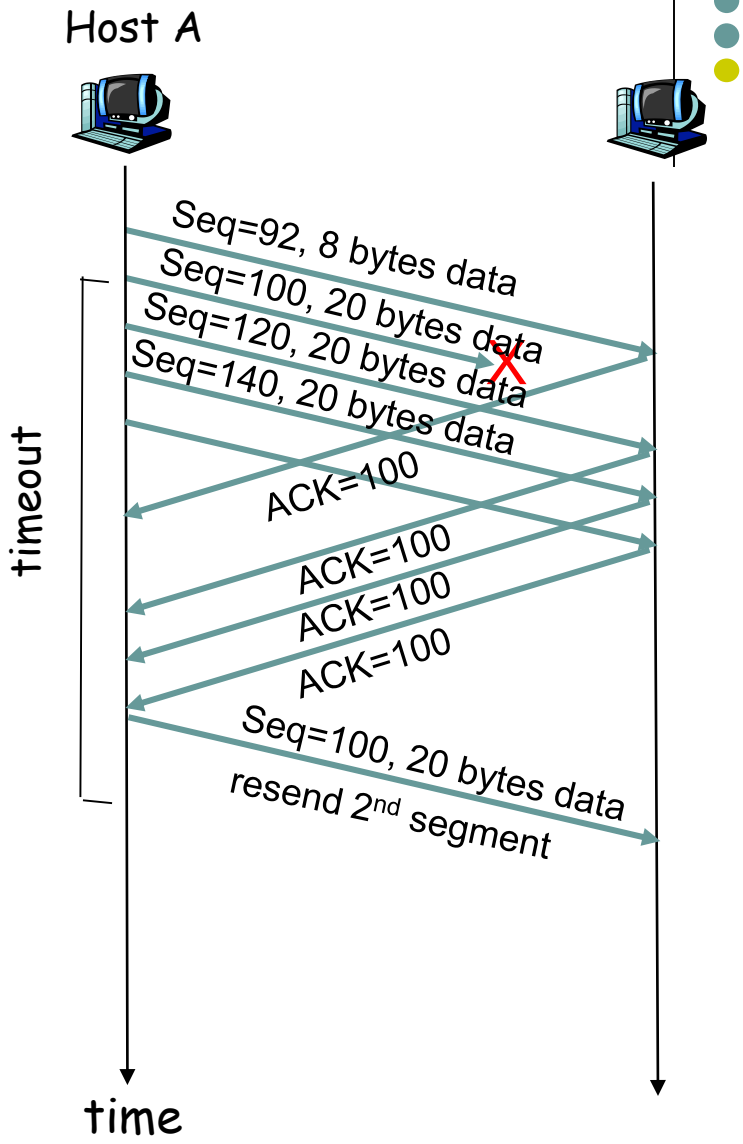
lost ACK scenario



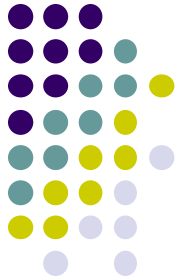
Cumulative ACK scenario

TCP retransmission scenarios (more)





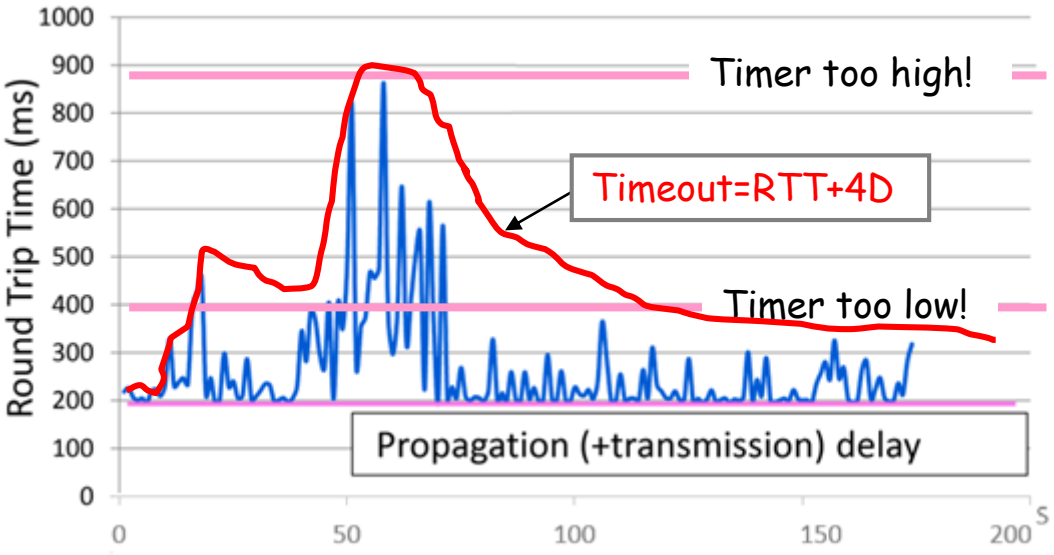
Chapter 3 outline



- 3.1 Transport-layer services
- 3.2 Multiplexing and demultiplexing
- 3.3 Connectionless transport: UDP
- 3.4 Principles of reliable data transfer
- 3.5 Connection-oriented transport: TCP
  - segment structure
  - connection management
  - reliable data transfer
  - timer management
  - flow control
- 3.6 Principles of congestion control
- 3.7 TCP congestion control

TCP Timer Management

- **Main issue**: How do we determine the best timeout value for retransmitting segments in the face of a large standard deviation of round-trip delays?
- **Solution**: uses dynamic algorithm that constantly adjusts the timeout interval, based on continuous measurements of network performance.

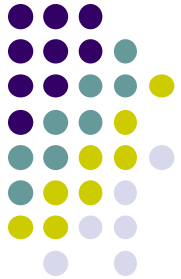


RTT	best current estimate of round-trip delay
D	estimate of deviation of round-trip delays
M	measured round-trip delay

$$RTT = \alpha RTT + (1-\alpha)M$$
$$D = \alpha D + (1-\alpha)|RTT - M|$$
$$timeout = RTT + 4 \cdot D$$

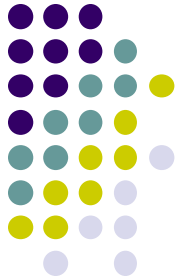
a is a smoothing factor that determines how much weight is given to the old value.

Chapter 3 outline



- 3.1 Transport-layer services
- 3.2 Multiplexing and demultiplexing
- 3.3 Connectionless transport: UDP
- 3.4 Principles of reliable data transfer
- 3.5 Connection-oriented transport: TCP
  - segment structure
  - connection management
  - reliable data transfer
  - timer management
  - flow control
- 3.6 Principles of congestion control
- 3.7 TCP congestion control

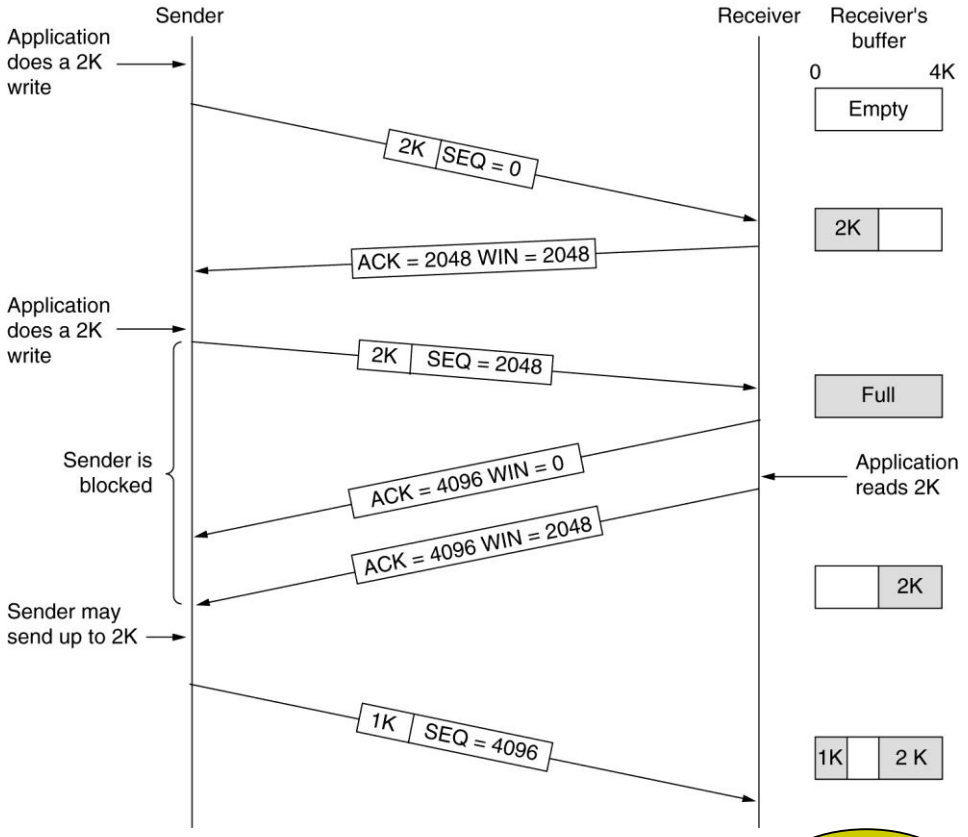
TCP Flow control: Dynamic Buffer Allocation



- The receiver maintains a single buffer pool shared by all connections:
  - infeasible to allocate a fixed buffers per connection
- Dynamic Buffer Allocation:.
  - Rcvr advertises spare room by including value of RcvWindow in segments
  - Sender limits unACKed data to RcvWindow guarantees receive buffer doesn't overflow
  - Decouple sending window from acknowledgement.

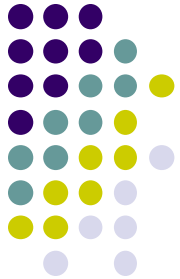
TCP Flow control: Window Size Announcement

- announcing window size: the maximum number of bytes that may be sent and received.
- zero window size: sender stop sending. Two exceptions:
  - urgent data
  - 1-byte request for reannounce the window size.



Demo

Chapter 3 outline

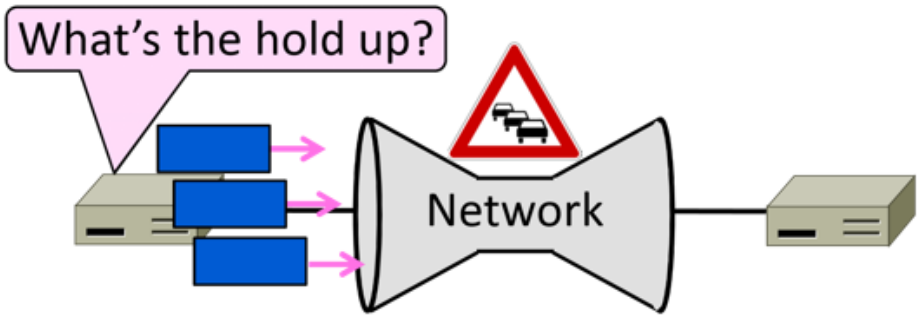


- 3.1 Transport-layer services
- 3.2 Multiplexing and demultiplexing
- 3.3 Connectionless transport: UDP
- 3.4 Principles of reliable data transfer
- 3.5 Connection-oriented transport: TCP
  - segment structure
  - connection management
  - reliable data transfer
  - flow control
  - Timer management
- 3.6 Principles of congestion control
- 3.7 TCP congestion control

Congestion

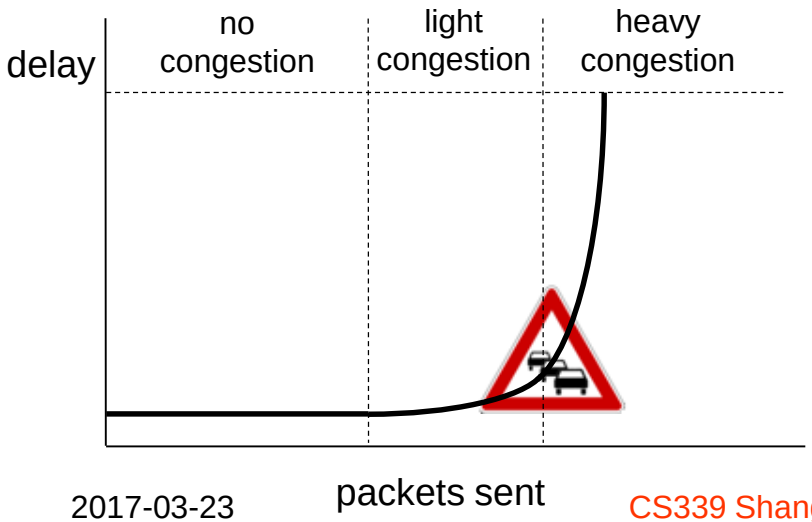
- "traffic jam" in the network: too many sources sending too much data too fast for ***network*** to handle.
- different from flow control!
- manifestations:
  - lost packets (buffer overflow at routers)
  - long delays (queueing in router buffers)
- a top-10 problem!

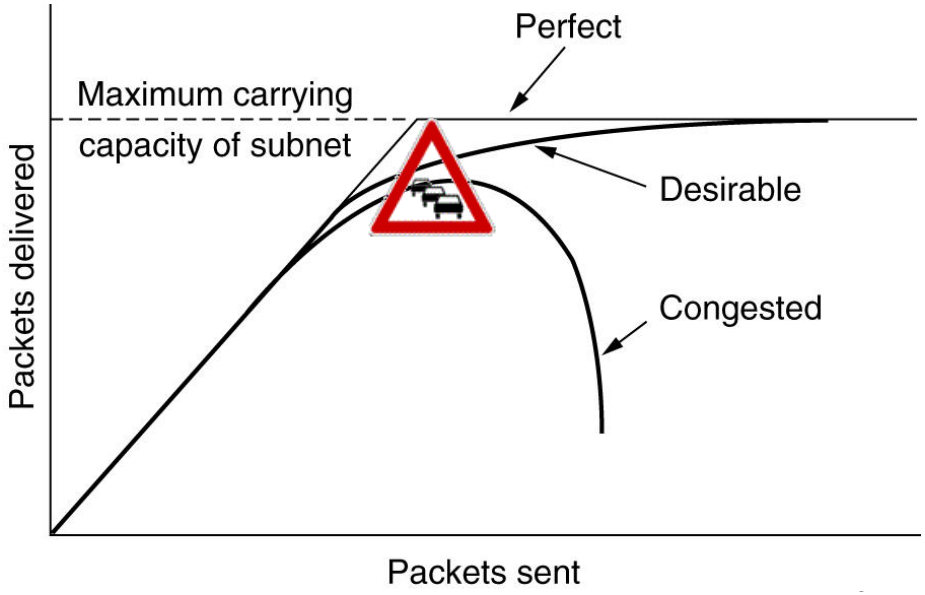
less important now



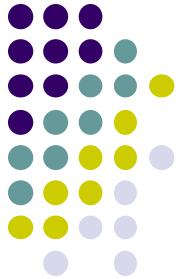
Congestion Performance

- During congestion, performance degrades
- Congestion can be caused by:
  - data in burst (app and transport layer)
  - lack of capacity/bandwidth (physical layer)
  - insufficient memory of routers (network layer)
  - slow processors of routers (network layer)



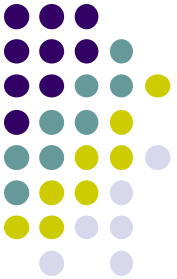


Congestion Control



- It's hard because it involves numerous senders and routers.
- Open loop: attempt to solve the problem by good design, to make sure it does not occur in the first place.
- Closed loop: are based on the concept of a feedback loop. It always has 3 parts:
  - Monitor the system to detect when and where congestion occurs.
  - Pass information to where action can be taken.
    - explicit feedback versus implicit feedback
  - Adjust system operation to correct the problem.

Congestion Prevention Policies in Open Loop Systems



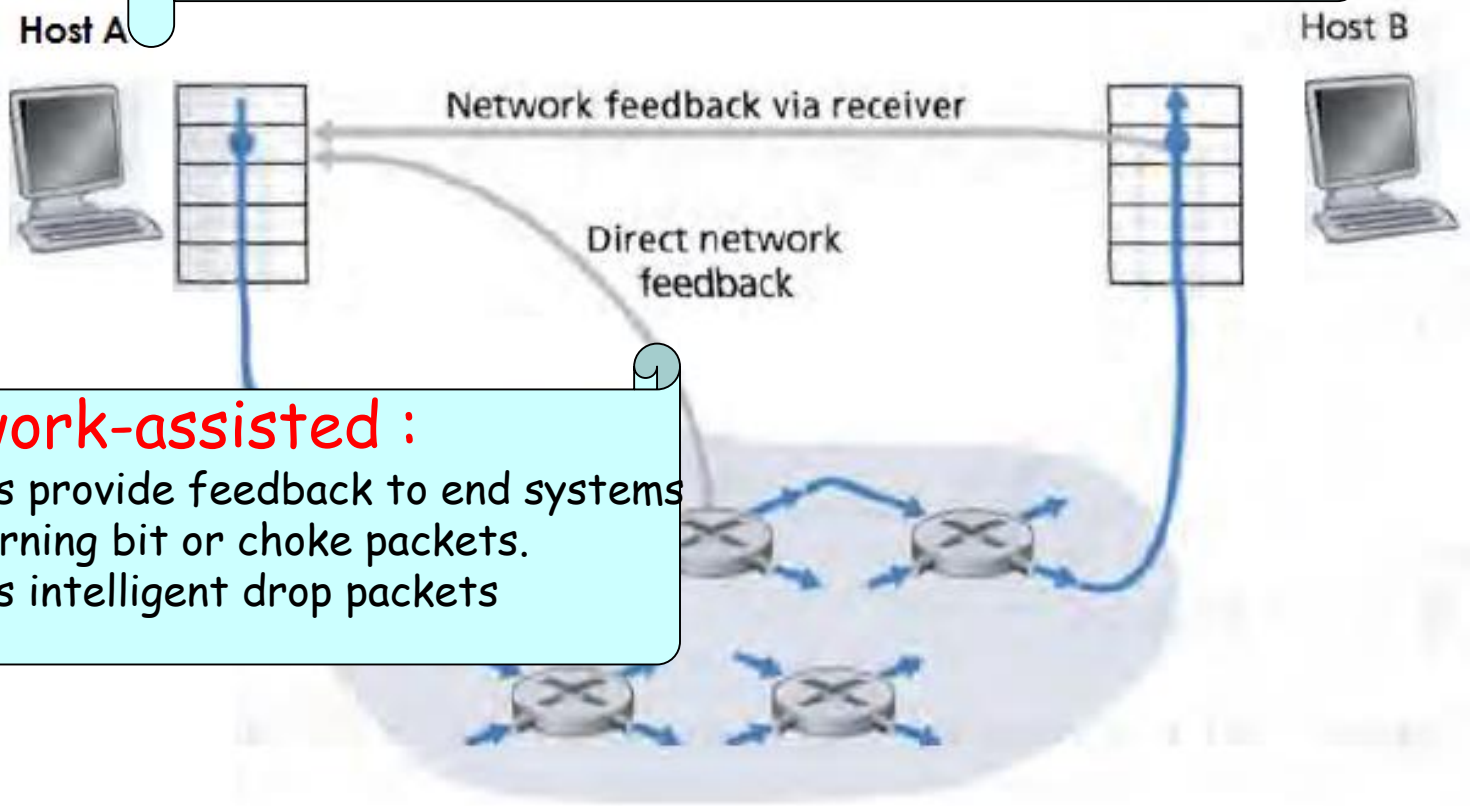
- To achieve congestion control, select appropriate policies at various levels: data link, network, and transport layer.
- Policies that affect congestion:

Layer	Policies
Transport	• Retransmission policy • Out-of-order caching policy • Acknowledgement policy • Flow control policy • Timeout determination
Network	• Virtual circuits versus datagram inside the subnet • Packet queueing and service policy • Packet discard policy • Routing algorithm • Packet lifetime management
Data link	• Retransmission policy • Out-of-order caching policy • Acknowledgement policy • Flow control policy

Approaches towards congestion control

End-end:

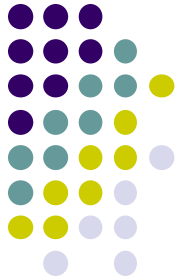
- no explicit feedback from network
- congestion inferred from end-system observed loss, delay
- approach taken by TCP



Network-assisted :

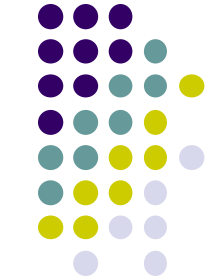
- routers provide feedback to end systems with warning bit or choke packets.
- routers intelligent drop packets
- ……

Congestion Avoidance



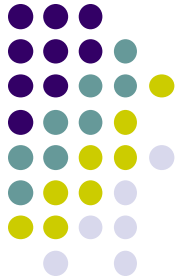
- strategy
  - predict when congestion is about to happen
  - reduce rate early
- two approaches
  - Host centric: TCP congestion control
  - Router-centric: warning bit, choke packet, load shedding

TCP Congestion Feedback Signals



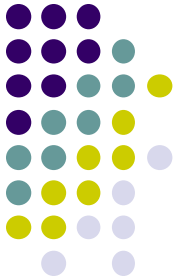
Signal	Example Protocol	Pros/cons
Packet loss	ClassicTCP, CubicTCP (Linux)	Hard to get wrong Hear about congestion late
Packet delay	Compound TCP (Windows)	Hear about congestion early Need to infer congestion
Router indication	TCPs with Explicit Congestion Notification	Hear about congestion early Require router support

Chapter 3 outline



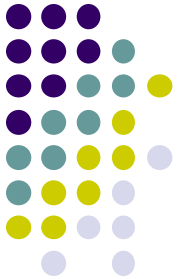
- 3.1 Transport-layer services
- 3.2 Multiplexing and demultiplexing
- 3.3 Connectionless transport: UDP
- 3.4 Principles of reliable data transfer
- 3.5 Connection-oriented transport: TCP
  - segment structure
  - connection management
  - reliable data transfer
  - flow control
  - Timer management
- 3.6 Principles of congestion control
- 3.7 TCP congestion control

TCP Congestion Control



- Early TCP used a fixed size sliding window (e.g., 8 packets)
- But something strange happened as the ARPANET grew
  - Links stayed busy but goodput fell by orders of magnitude! (congestion collapse)
- TCP Tahoe/Reno implements AIMD to limit the sending rate.

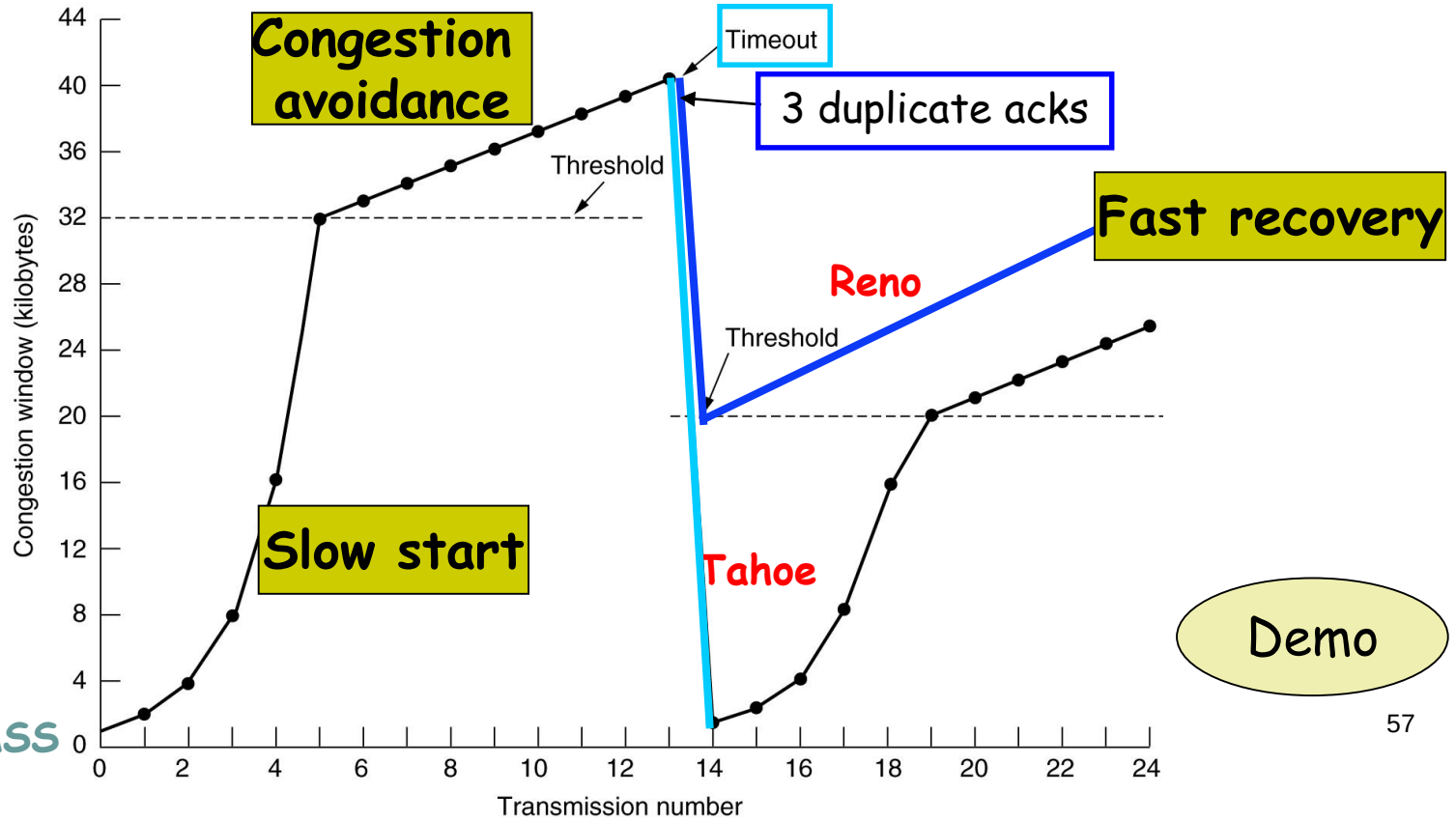
TCP Congestion Control con.



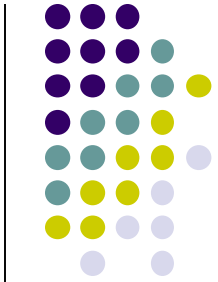
- Sender uses packet loss as the network congestion signal
  - TCP assume that lost packets are caused by congestion, not by links.
- TCP use a Congestion Window (CongWin) next to the window granted by the receiver. The actual window size is the minimum of the two.
- Self-clocking: The CongWin size should be adjusted frequently, to match the networking carrying capacity

TCP Congestion Algorithm

- Slow start: CongWin grows exponentially every ACK.
- Congestion avoidance: CongWin grows linearly every ACK.
- Rapid back-off: initiate slow start all over again.
- (optional) Fast recovery: grow linearly after 3 duplicate Acks



TCP Congestion Algorithm con.



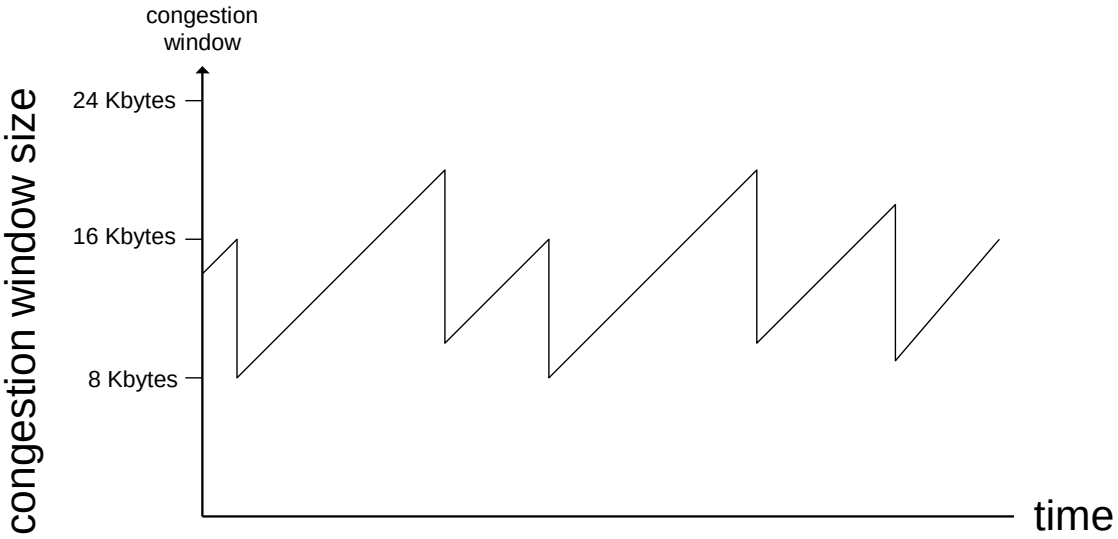
- When **`CongWin`** is below **`Threshold`**, sender in slow-start phase, window grows exponentially.
- When **`CongWin`** is above **`Threshold`**, sender is in congestion-avoidance phase, window grows linearly.
- When a triple duplicate ACK occurs, **`Threshold`** and **`CongWin`** set to **`CongWin/2`**.
- When timeout occurs, **`Threshold`** set to **`CongWin/2`** and **`CongWin`** is set to 1 MSS.

TCP AIMD

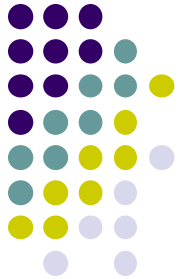
Additive Increase, Multiplicative Decrease

- *additive increase:* increase **CongWin** by 1 MSS every RTT until loss detected
- *multiplicative decrease:* cut **CongWin** in half after loss

Saw tooth behavior: probing for bandwidth



TCP throughput



- Roughly

$$\text{throughput} = \frac{\text{CongWin}}{\text{RTT}} \text{ Bytes/sec}$$

- AIMD ignoring slow start

$$\text{Average throughput} = 0.75 \frac{[\text{CongWin}]_{\text{loss}}}{\text{RTT}} \text{ Bytes/sec}$$

- Throughput in terms of loss rate:

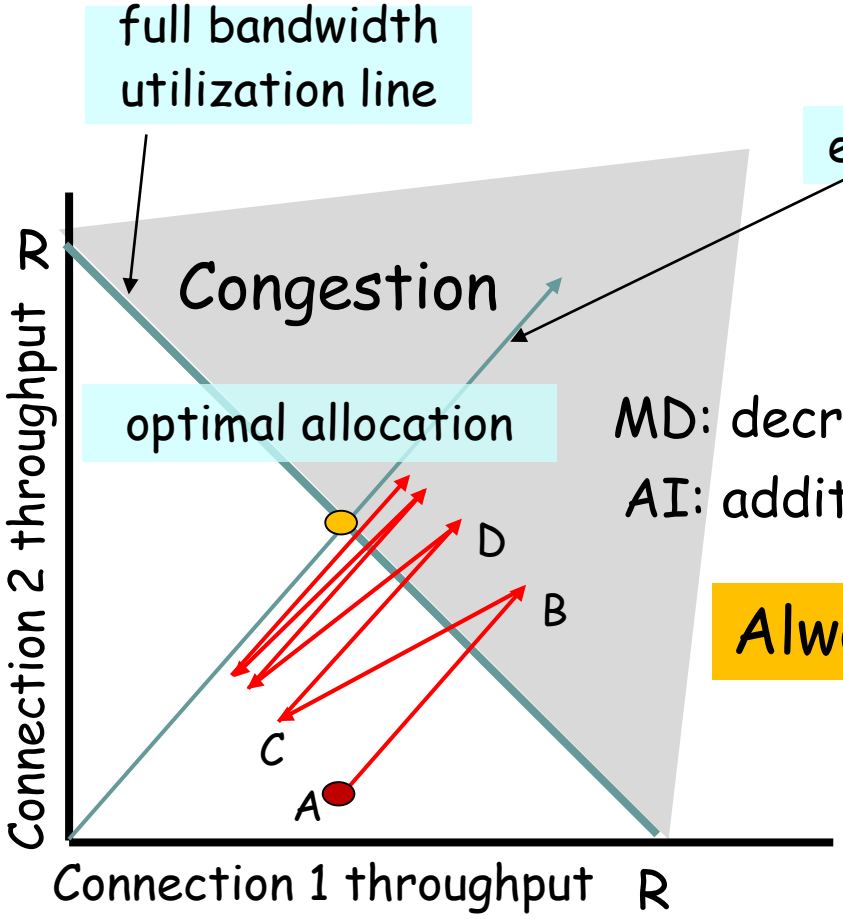
$$\text{Average throughput} = \frac{1.22 \cdot MSS}{RTT\sqrt{L}}$$

Example

1. MSS = 500 bytes
RTT = 200 msec
then
initial rate = 20 kbps

2. MSS=1500 bytes
RTT= 100ms
throughput =10 Gbps
then
$L = 2 \cdot 10^{-10}$

AIMD Game: We want a good bandwidth allocation to be fair and efficient

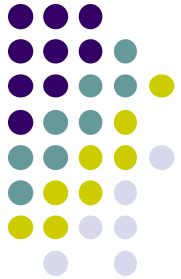


MD: decrease window by factor of 2 when loss

AI: additive increase until packet loss

Always converge to good allocation!

TCP Fairness



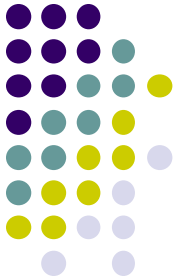
Fairness and TCP

- if K TCP sessions share same bottleneck link of capacity R, each would have average rate of R/K
- TCP provides per-connection, not per-application, fairness
- Sessions with smaller RTT enjoy higher throughput

Fairness and UDP

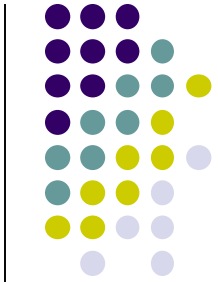
- UDP crowds out TCP
  - pump data whenever available,
- Multimedia apps use UDP
  - do not want rate throttled by congestion control
  - pump audio/video at constant rate, tolerate packet loss
- Research area: TCP friendly

The TCP Service Model



Property	Behavior
Connection oriented ***Stream of bytes***	Reliable byte delivery service.
Reliable delivery	1. Acknowledgments indicate correct delivery. 2. Checksums detect corrupted data. 3. Sequence numbers detect missing data. 4. Flow-control prevents overrunning receiver.
In-sequence	Data delivered to application in sequence transmitted.
Congestion Control	Controls network congestion.

Chapter 3: Summary

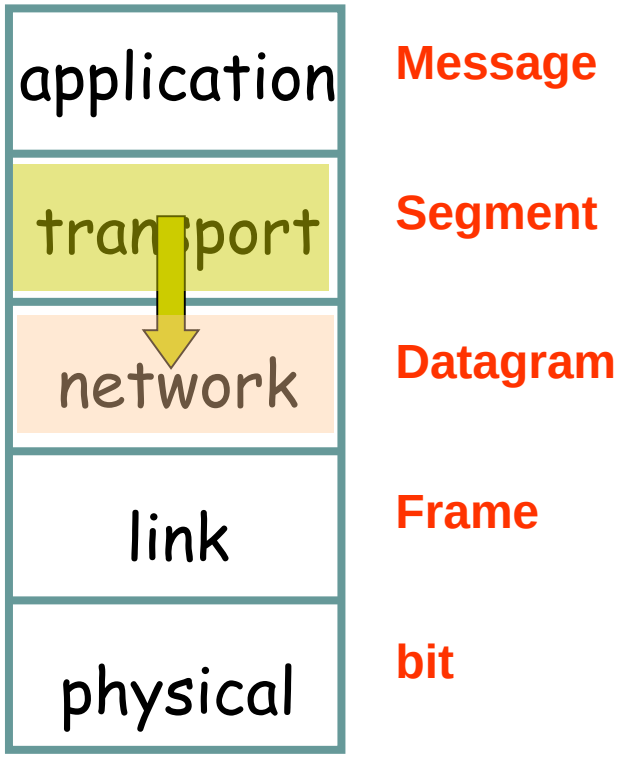


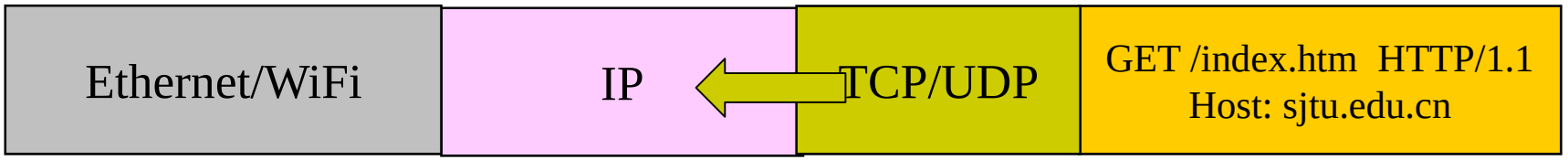
- principles behind transport layer services:
  - multiplexing, demultiplexing
  - reliable data transfer
  - flow control
  - congestion control
- implementation in the Internet
  - UDP
  - TCP

Next:

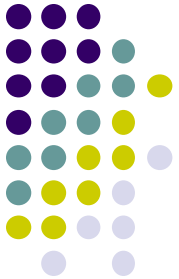
- leaving the network “edge” (application, transport layers)
- into the network “core”

Chapter 3: summary





Chapter 3 assignment



Download

- from ftp://public.sjtu.edu.cn/CompuNet
  or
- from Moodle
  http://moodle.speit.sjtu.edu.cn/