Computer Networks Basic Protocols

Assoc. Prof. Dr. Berk CANBERK

15 November 2017

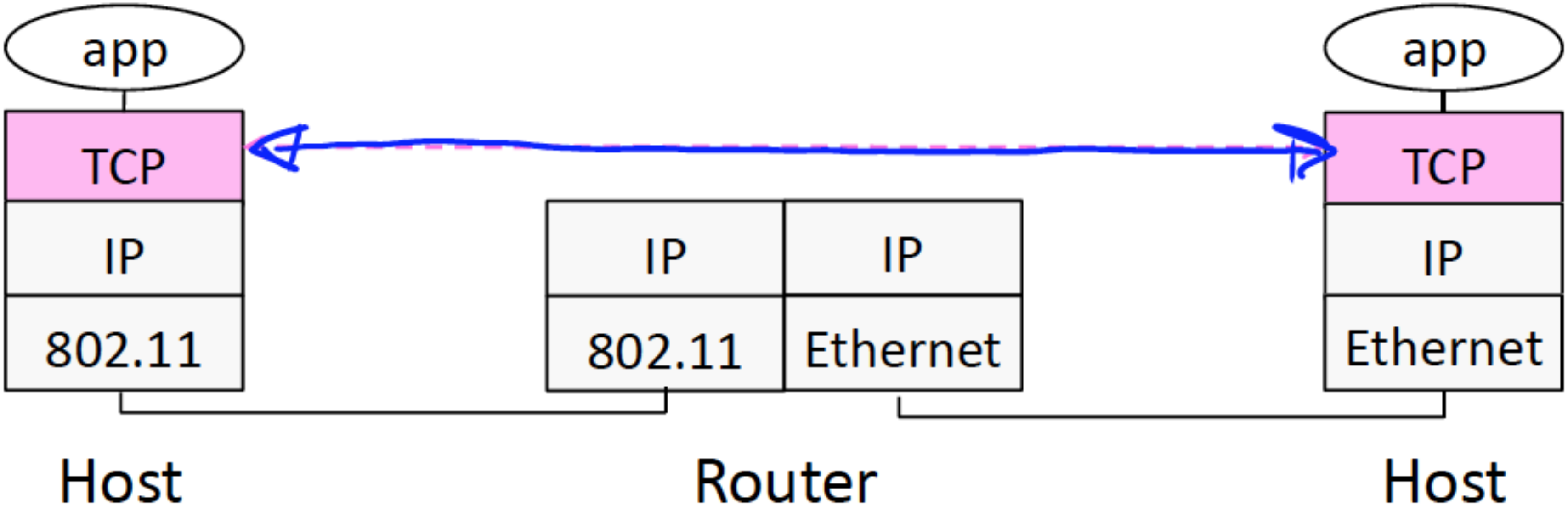
-Transport Layer-

References:

-*Data and Computer Communications*, William Stallings, Pearson-Prentice Hall, 9th Edition, 2010.

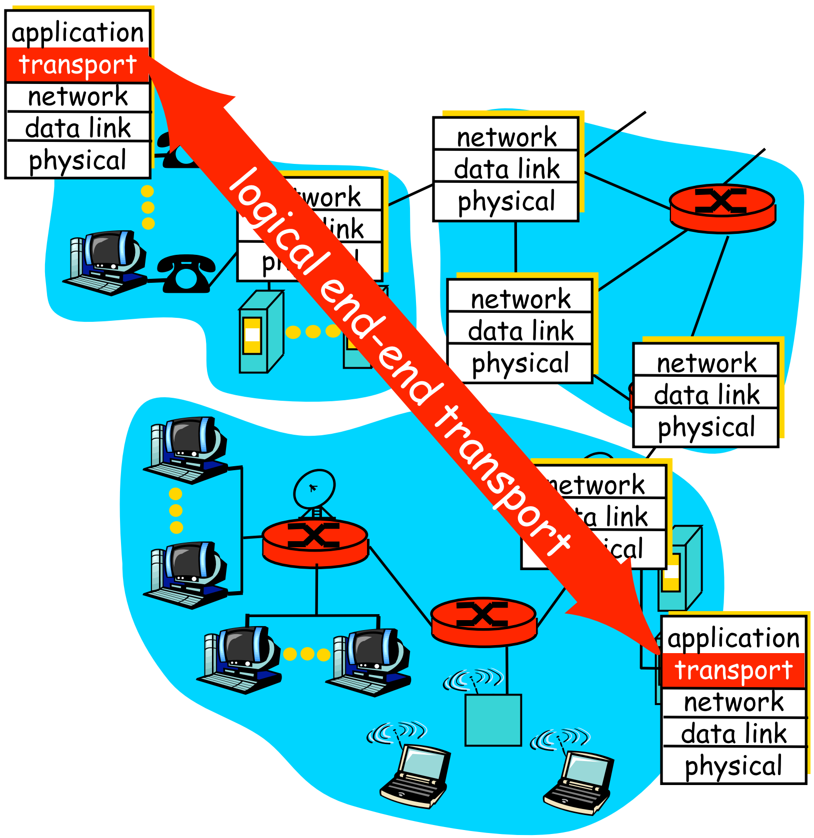
-*Computer Networking, A Top-Down Approach Featuring the Internet*, James F.Kurose, Keith W.Ross, Pearson-Addison Wesley, 6th Edition, 2012.

Transport layer provides end-to-end connectivity across the network



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
  - Internet: TCP and UDP

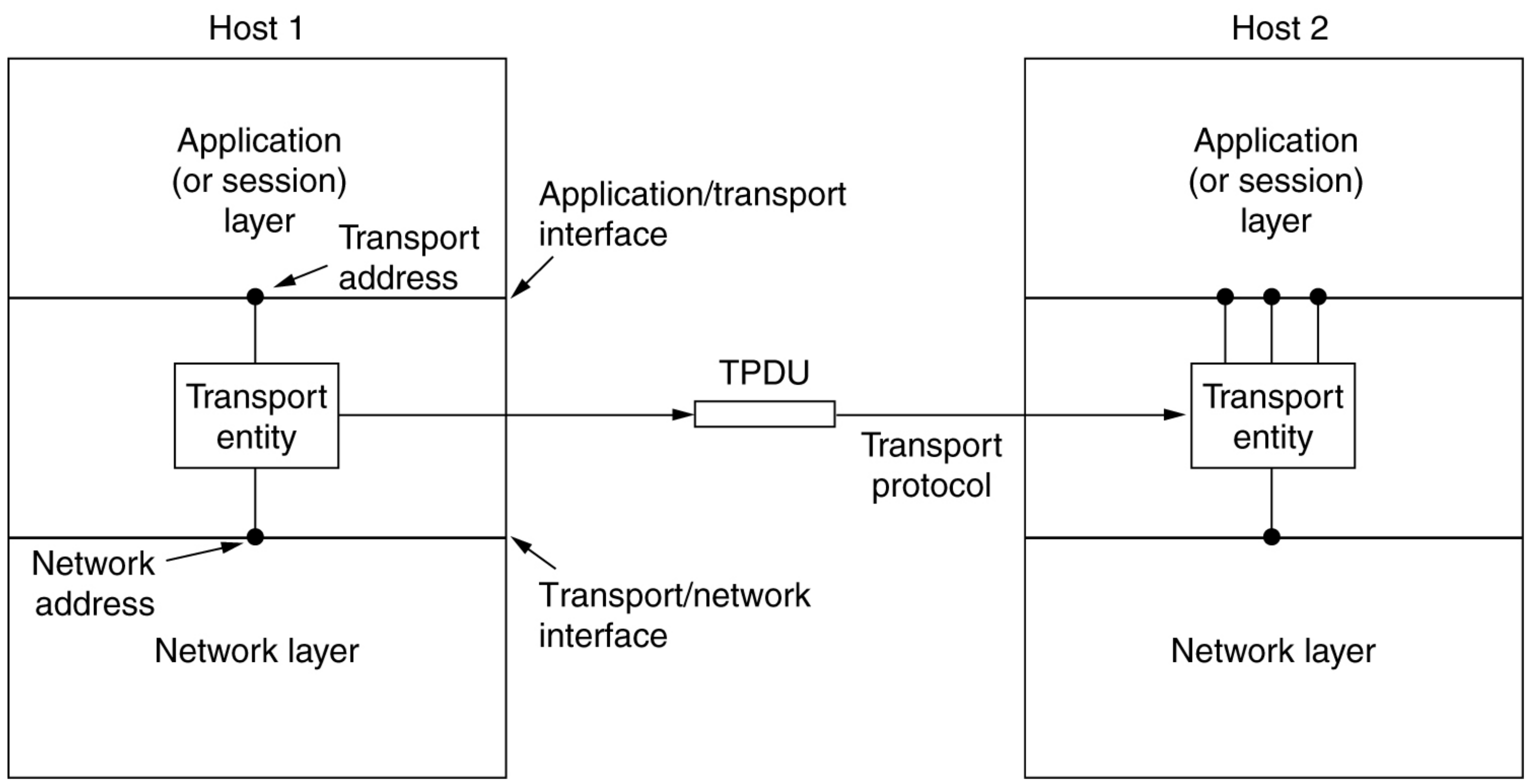


Transport vs. network layer

- *Network layer:* logical communication between hosts
  - Not reliable, no control on network layer
- *Transport layer:* logical communication between processes
  - relies on, enhances, network layer services
  - Efficient reliable service to users (processes in the application layer)
  - Addressing
  - Multiplexing
  - Flow (Congestion) Control
  - Connection establishment and termination

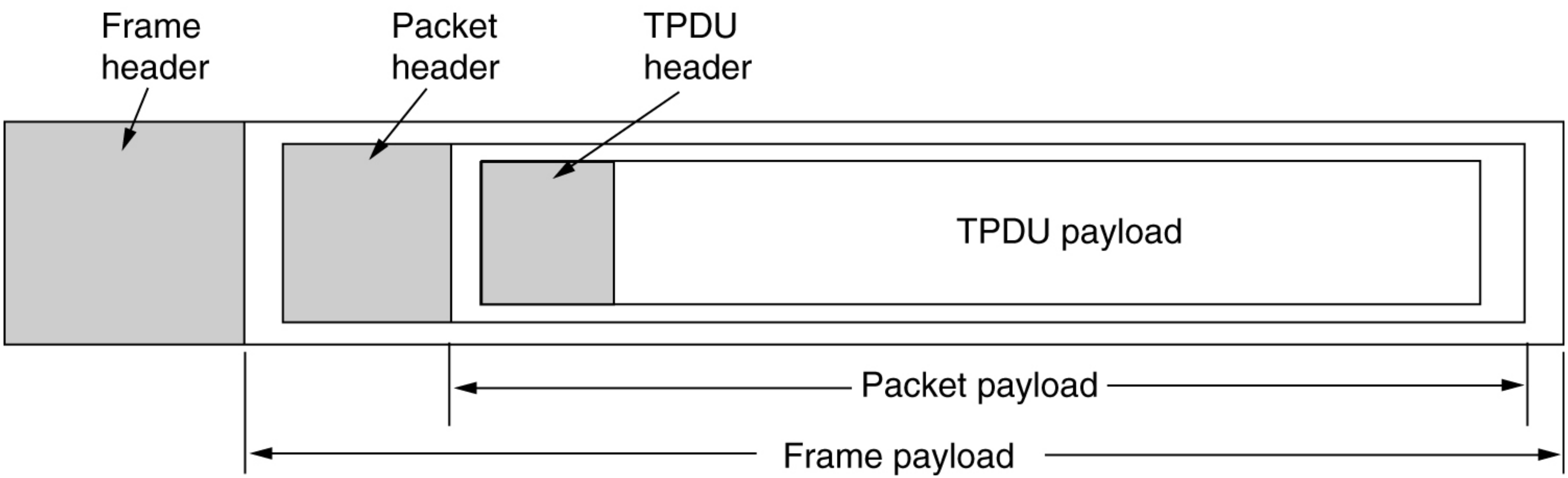
Services Provided to Upper Layers

The network, transport, and application layers.



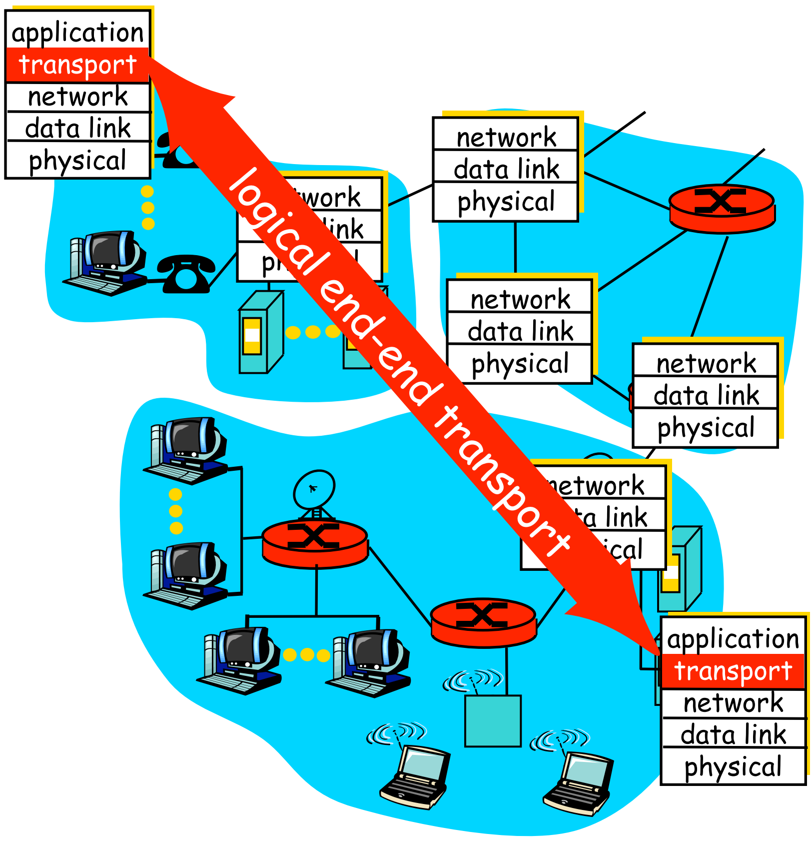
Transport Service Primitives

The nesting of TPDUs, packets, and frames.



Internet transport-layer protocols

- reliable, in-order delivery (TCP)
  - flow/congestion control
  - connection setup
- unreliable, unordered delivery: UDP
  - extension of "best-effort" IP
- services **not** available:
  - delay guarantees
  - bandwidth guarantees



Comparison of Internet Transport Protocols

TCP (Streams)	UDP (Datagrams)
Connections	Datagrams
Bytes are delivered once, reliably, and in order	Messages may be los reordered, duplicate
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

UDP: User Datagram Protocol [RFC 768]

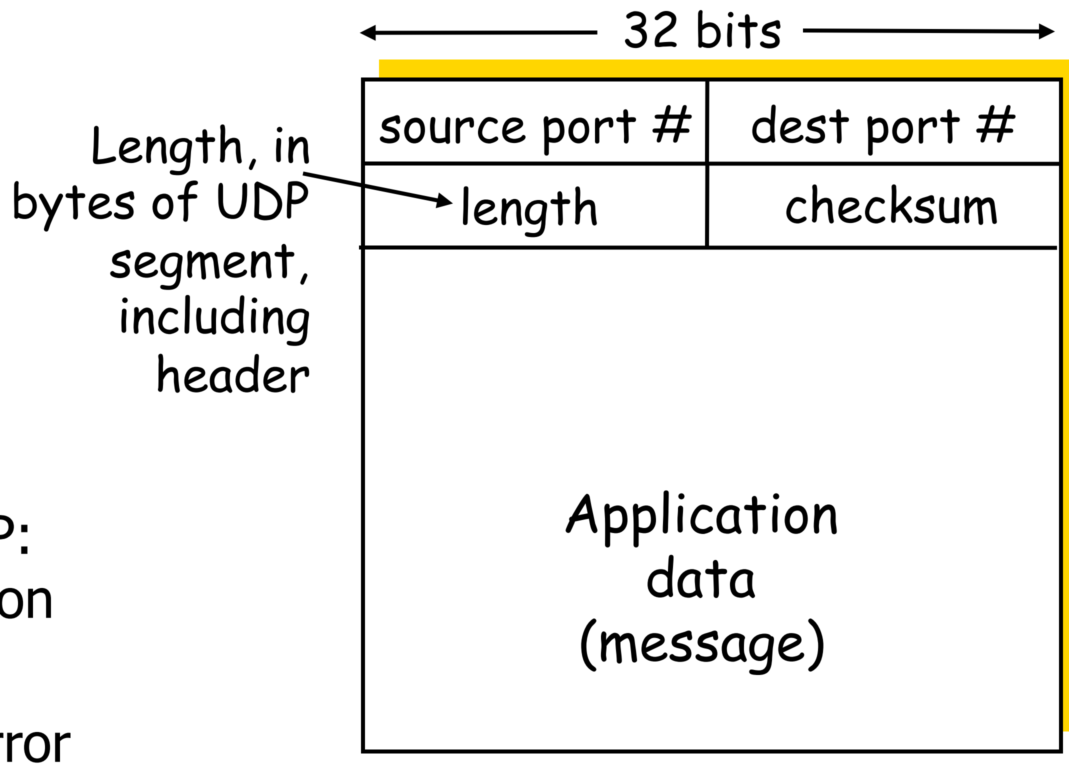
- "best effort" service, UDP segments may be:
  - lost
  - delivered out of order to app
- *connectionless:*
  - no handshaking between UDP sender, receiver
  - each UDP segment handled independently of others

Why is there a UDP?

- no connection establishment (which can add delay)
- simple: no connection state at sender, receiver
- small segment header
- no congestion control: UDP can blast away as fast as desired

UDP

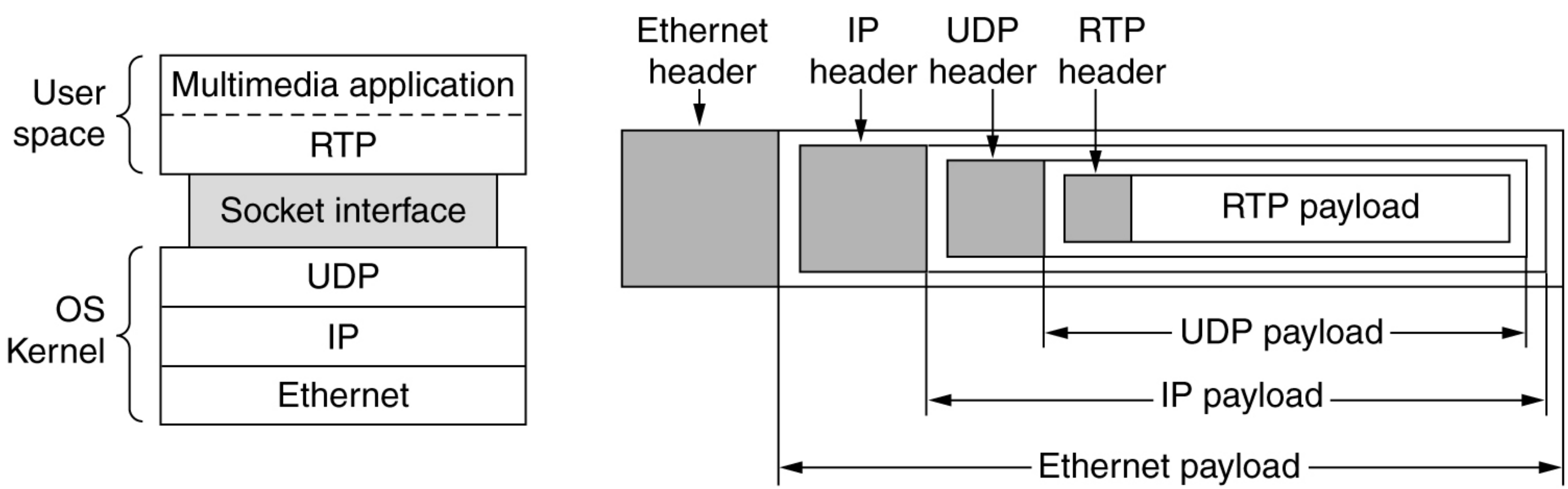
- often used for streaming multimedia apps
  - loss tolerant
  - rate sensitive
- other UDP uses
  - DNS
  - SNMP
- reliable transfer over UDP: add reliability at application layer
  - application-specific error recovery!



UDP segment format

The Real-Time Transport Protocol

(a) The position of RTP in the protocol stack.
(b) Packet nesting.



(a) (b)

Transport Control Protocol (TCP)

RFCs: 793, 1122, 1323, 2018, 2581

point-to-point:

one sender, one receiver

reliable, in-order *byte steam:*

no "message boundaries"

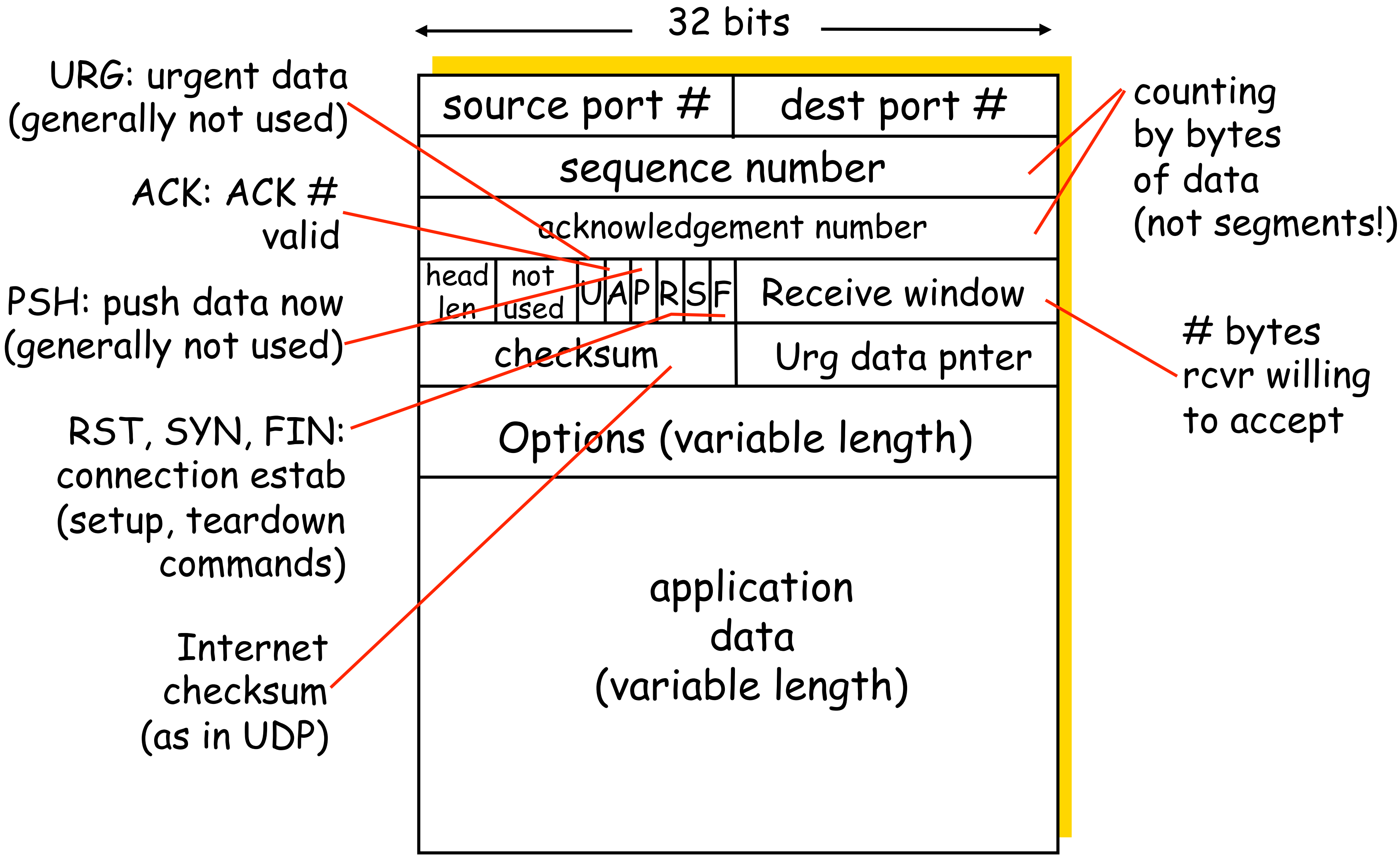
pipelined:

TCP congestion and flow control set window size

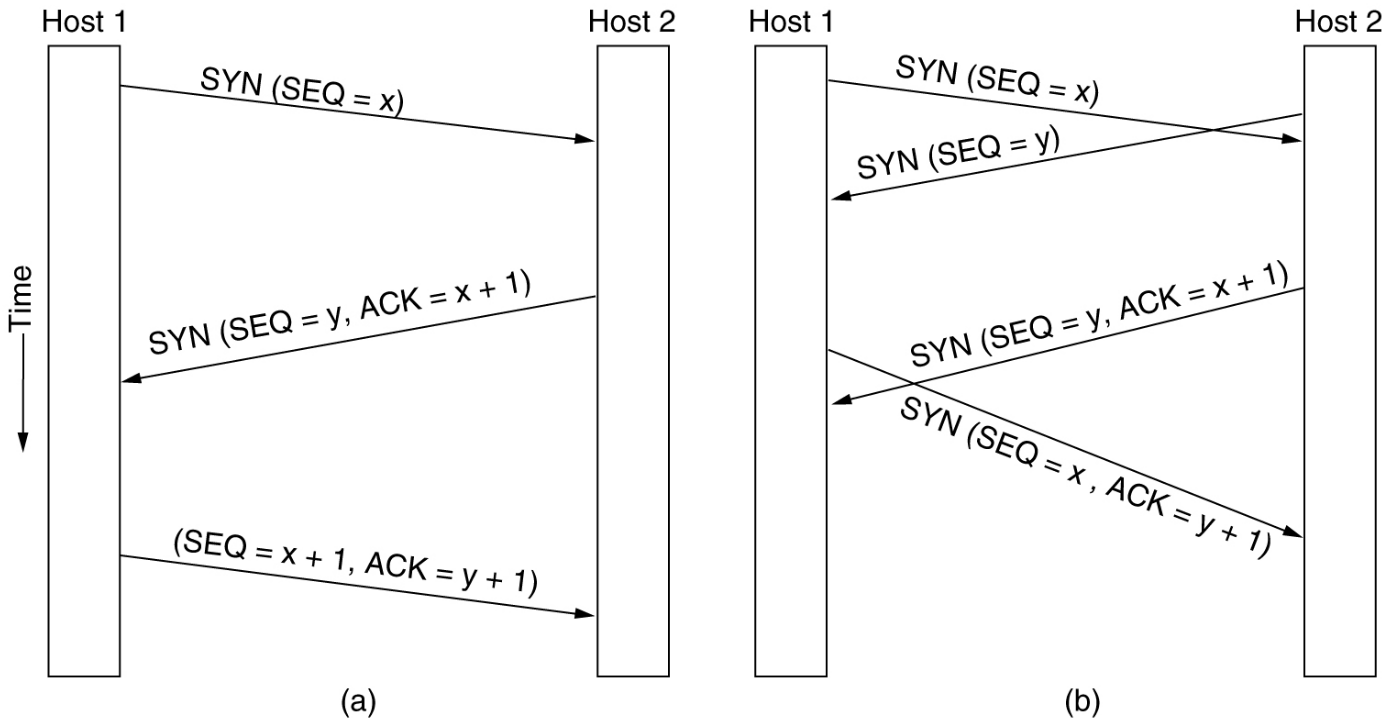
send & receive buffers

- full duplex data:
  - bi-directional data flow in same connection
  - MSS: maximum segment size
- connection-oriented:
  - handshaking (exchange of control msgs) init's sender, receiver state before data exchange
- flow controlled:
  - sender will not overwhelm receiver

TCP segment structure



TCP Connection Establishment



(a) TCP connection establishment in the normal case.
(b) Call collision.

Berkeley Sockets

The socket primitives for TCP

Primitive	Meaning
SOCKET	Create a new communication end point
BIND	Attach a local address to a socket
LISTEN	Announce willingness to accept connections; give queue size
ACCEPT	Block the caller until a connection attempt arrives
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

Only Stream

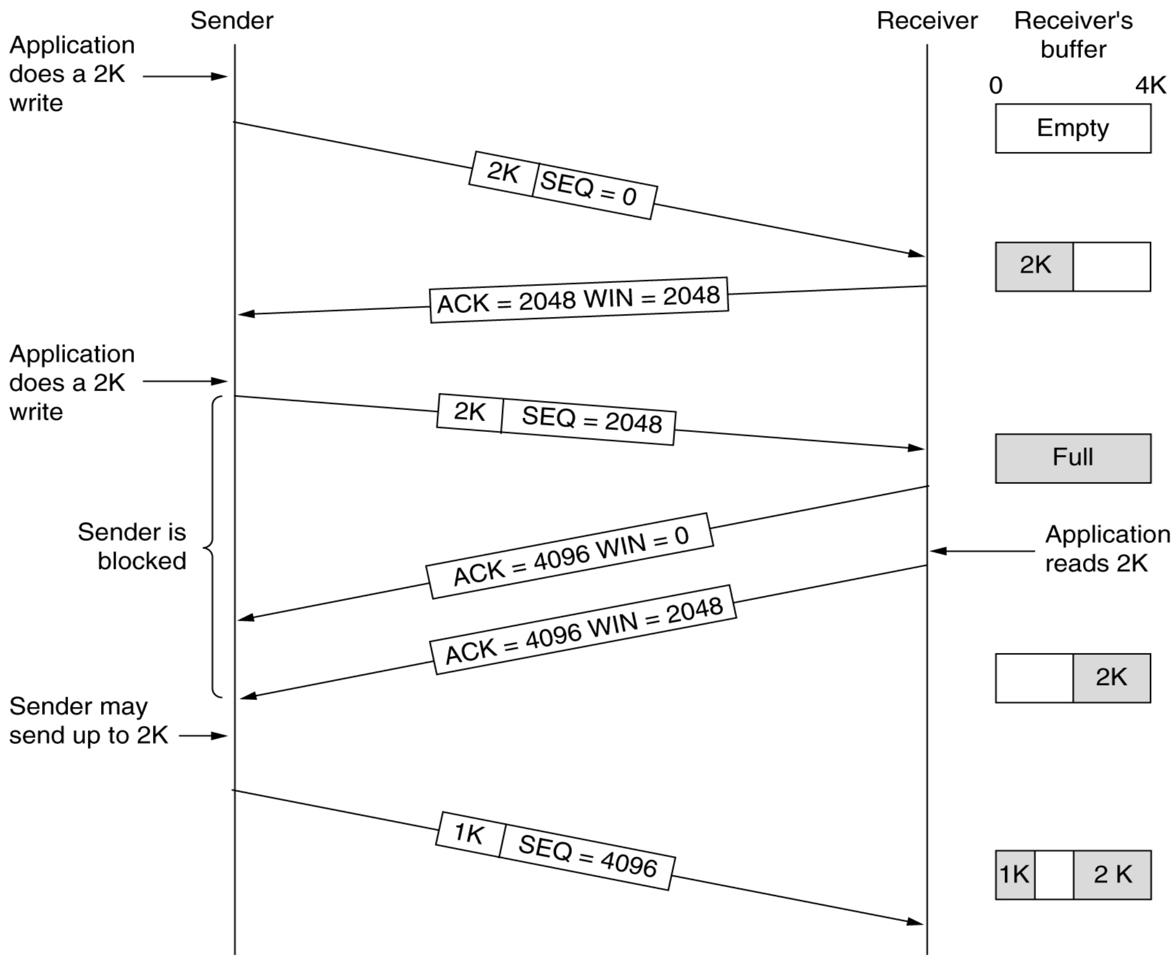
UDP

The TCP Service Model

Some assigned ports

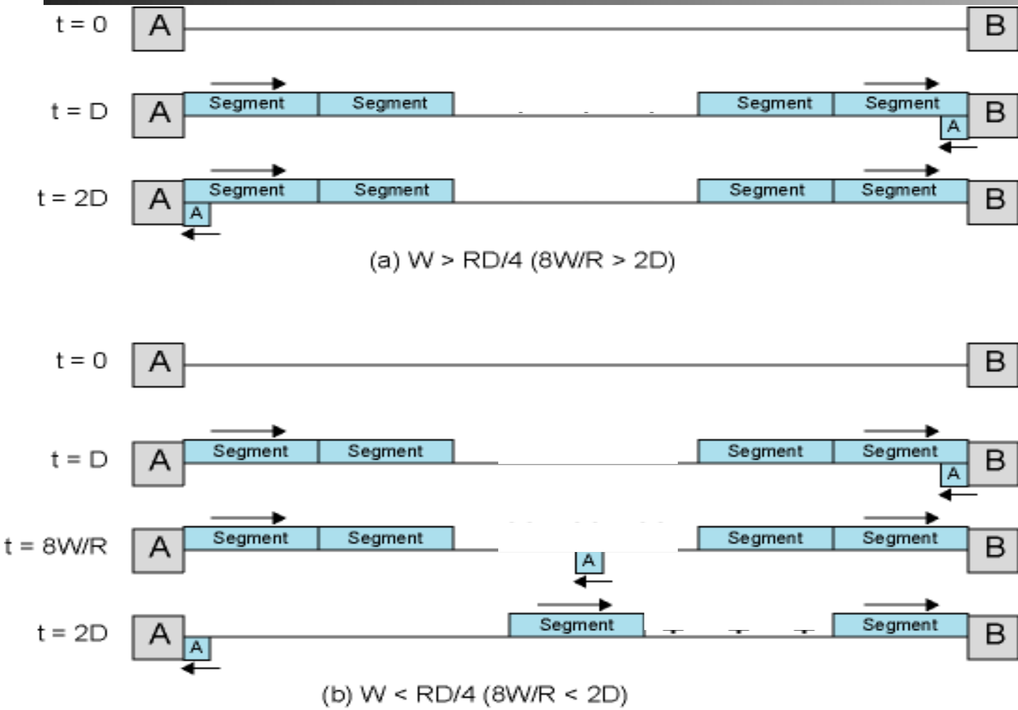
Port	Protocol	Use
21	FTP	File transfer
23	Telnet	Remote login
25	SMTP	E-mail
69	TFTP	Trivial File Transfer Protocol
79	Finger	Lookup info about a user
80	HTTP	World Wide Web
110	POP-3	Remote e-mail access
119	NNTP	USENET news

TCP Transmission Policy



Window management in TCP.

Effect of Window Size



$$S = \begin{cases} 1, & W > RD/4 \\ 4W/RD, & W < RD/4 \end{cases}$$

Normalized Throughput

- W = TCP window size (octets)
- R = Data rate (bps) at TCP source
- D = Propagation delay (seconds)
- After TCP source begins transmitting, it takes D seconds for first octet to arrive, and D seconds for acknowledgement to return
- TCP source could transmit at most 2RD bits, or RD/4 octets

TCP reliable data transfer

- TCP creates rdt service on top of IP’s unreliable service
- Pipelined segments
- Cumulative acks
- TCP uses single retransmission timer
- Retransmissions are triggered by:
  - timeout events
  - duplicate acks

Fast Retransmit

- Time-out period often relatively long:
  - long delay before resending lost packet
- Detect lost segments via duplicate ACKs
  - Sender often sends many segments back-to-back
  - If segment is lost, there will likely be many duplicate ACKs.
- If sender receives 3 ACKs for the same data, it supposes that segment after ACKed data was lost:
  - fast retransmit: resend segment before timer expires

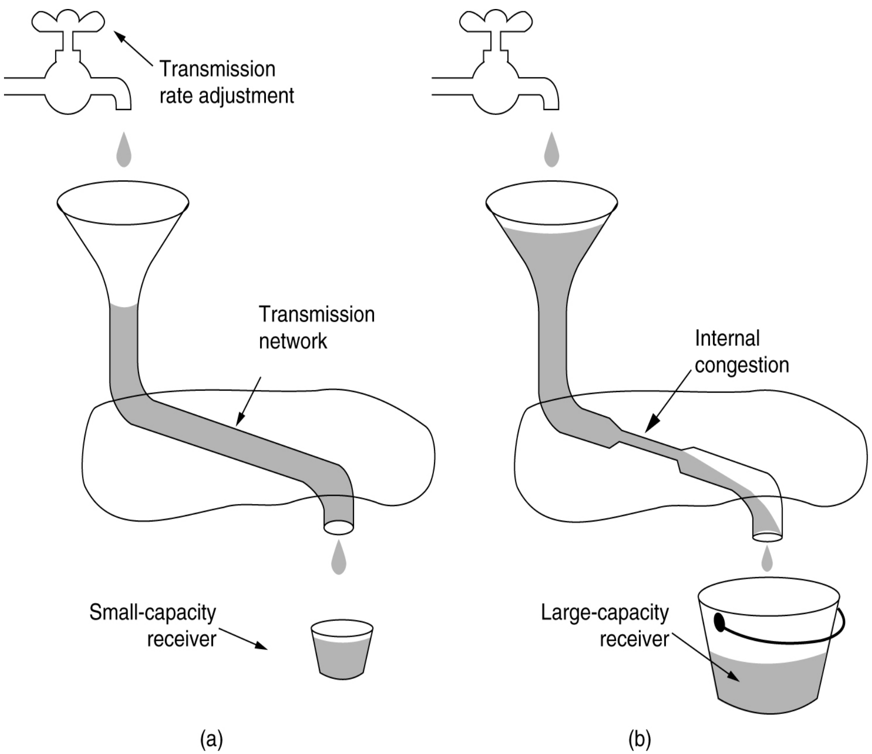
Congestion Control

(a) fast network feeding low capacity receiver

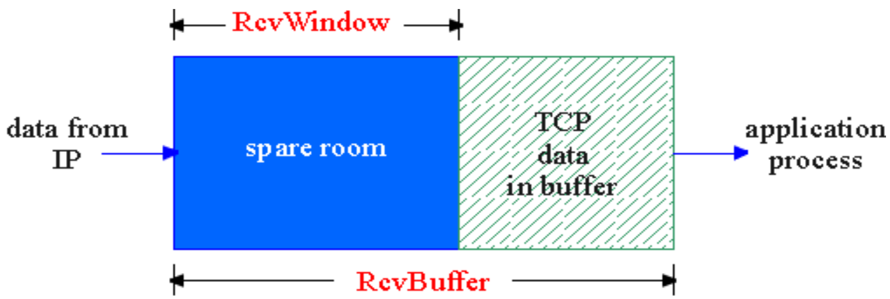
(b) Slow network feeding high-capacity receiver

Congestion:

- informally: "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)



TCP Congestion Control



(Suppose TCP receiver discards out-of-order segments)

- spare room in buffer

= RcvWindow (=rcwnd)

= RcvBuffer-[LastByteRcvd - LastByteRead]

Rcvr advertises spare room by including value of **`RcvWindow`** in segments

Sender limits unACKed data to **`RcvWindow`**

- guarantees receive buffer doesn't overflow

Send `min(rcwnd,cwnd)`

TCP Congestion Control

- end-end control (no network assistance)
- sender limits transmission:

  `LastByteSent-LastByteAcked ≤ cwnd`
- Roughly,

$$\text{rate} = \frac{\text{cwnd}}{\text{RTT}} \text{ Bytes/sec}$$

- **`cwnd`** is dynamic, function of perceived network congestion

<u>How does sender perceive congestion?</u>

loss event = timeout *or* 3 duplicate acks

TCP sender reduces rate (**`cwnd`**) after loss event

<u>three mechanisms:</u>

AIMD

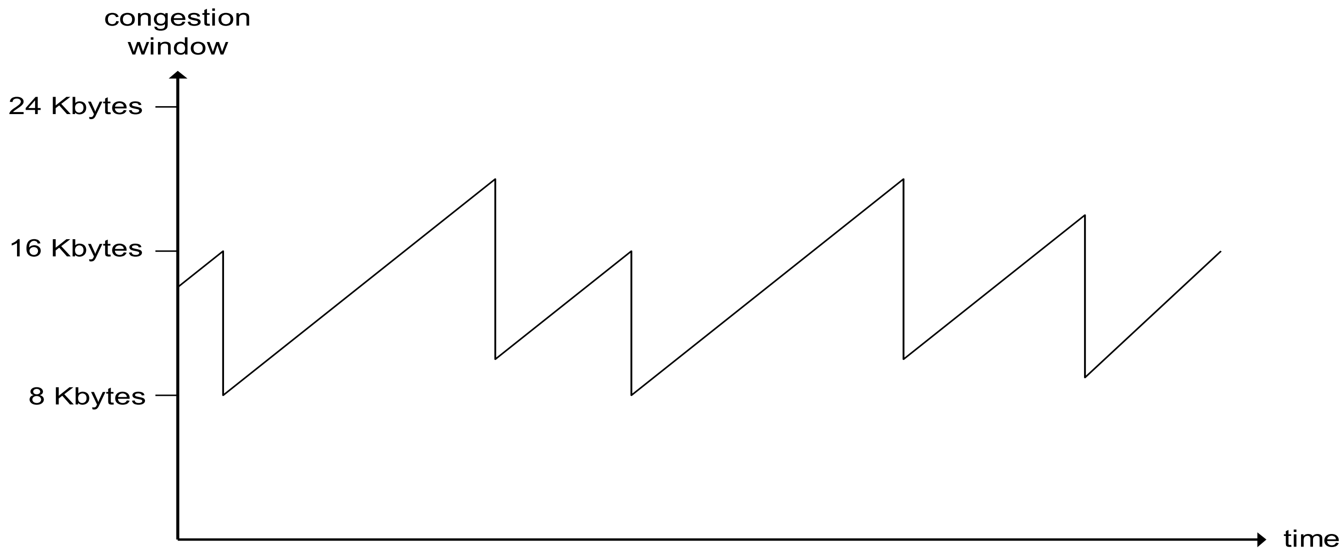
Slow Start

Congestion Avoidance

TCP AIMD

additive increase: increase **`cwnd`** by 1 segment size every RTT in the absence of loss events: *probing*

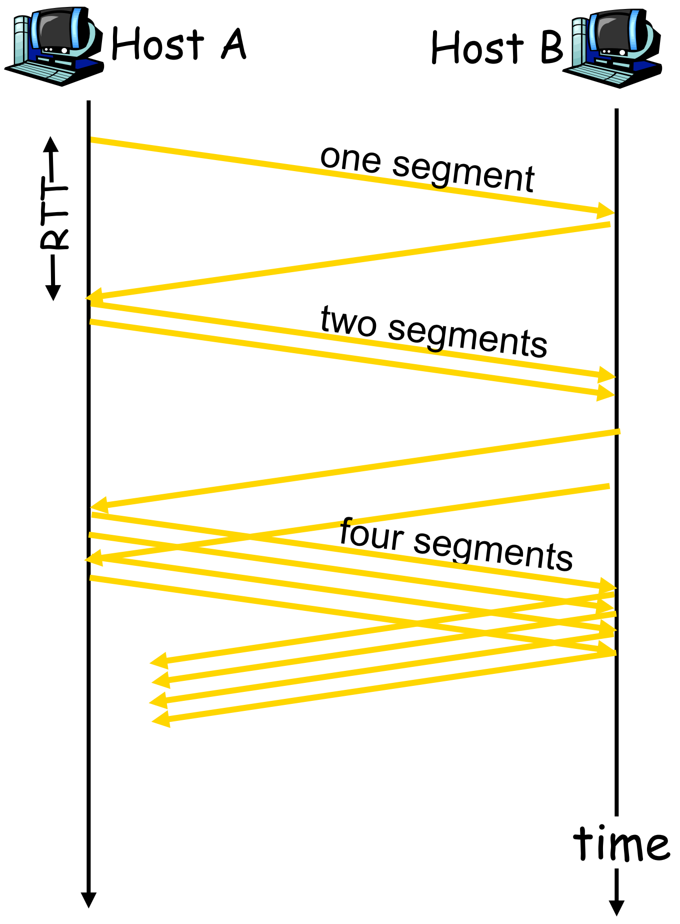
multiplicative decrease: cut **`cwnd`** in half after loss event



Long-lived TCP connection

TCP Slow Start

- When connection begins, **`cwnd`** = 1 MSS
  - Example: MSS = 500 bytes & RTT = 200 msec
  - initial rate = 20 kbps
- Available bandwidth may be >> MSS/RTT
  - desirable to quickly ramp up to respectable rate
- When connection begins, increase rate exponentially fast until first loss event
  - double **`cwnd`** every RTT
  - done by incrementing **`cwnd`** for every ACK received
- Summary: initial rate is slow but ramps up exponentially fast



Refinement

- After 3 dup ACKs:
  - `cwnd` is cut in half
  - window then grows linearly
- But after timeout event:
  - `cwnd` instead set to 1 MSS;
  - window then grows exponentially
  - to a threshold, then grows linearly

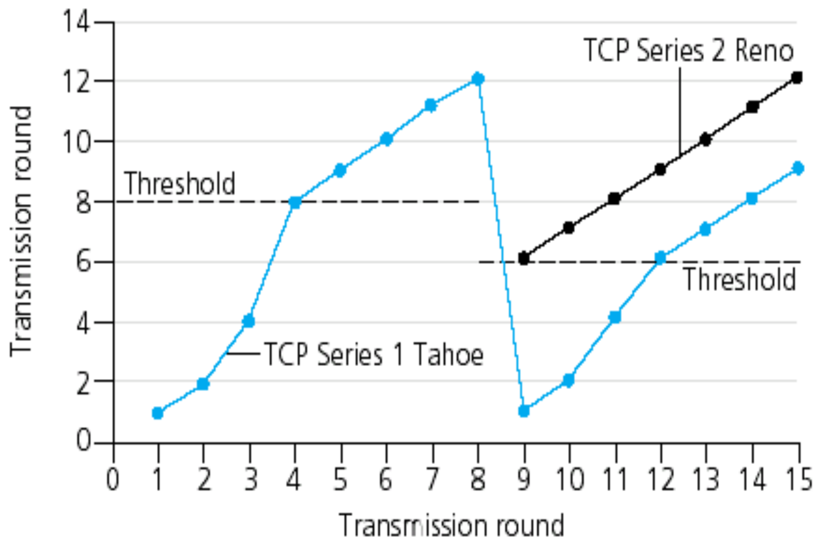
Philosophy:

- 3 dup ACKs indicates network capable of delivering some segments
- timeout before 3 dup ACKs is "more alarming"

Refinement (more)

Q: When should the exponential increase switch to linear?

A: When **`cwnd`** gets to 1/2 of its value before timeout.



<u>Implementation:</u>

Variable Threshold

At loss event, Threshold is set to 1/2 of cwnd just before loss event

Summary: TCP Congestion Control

- When `cwnd` is below Threshold

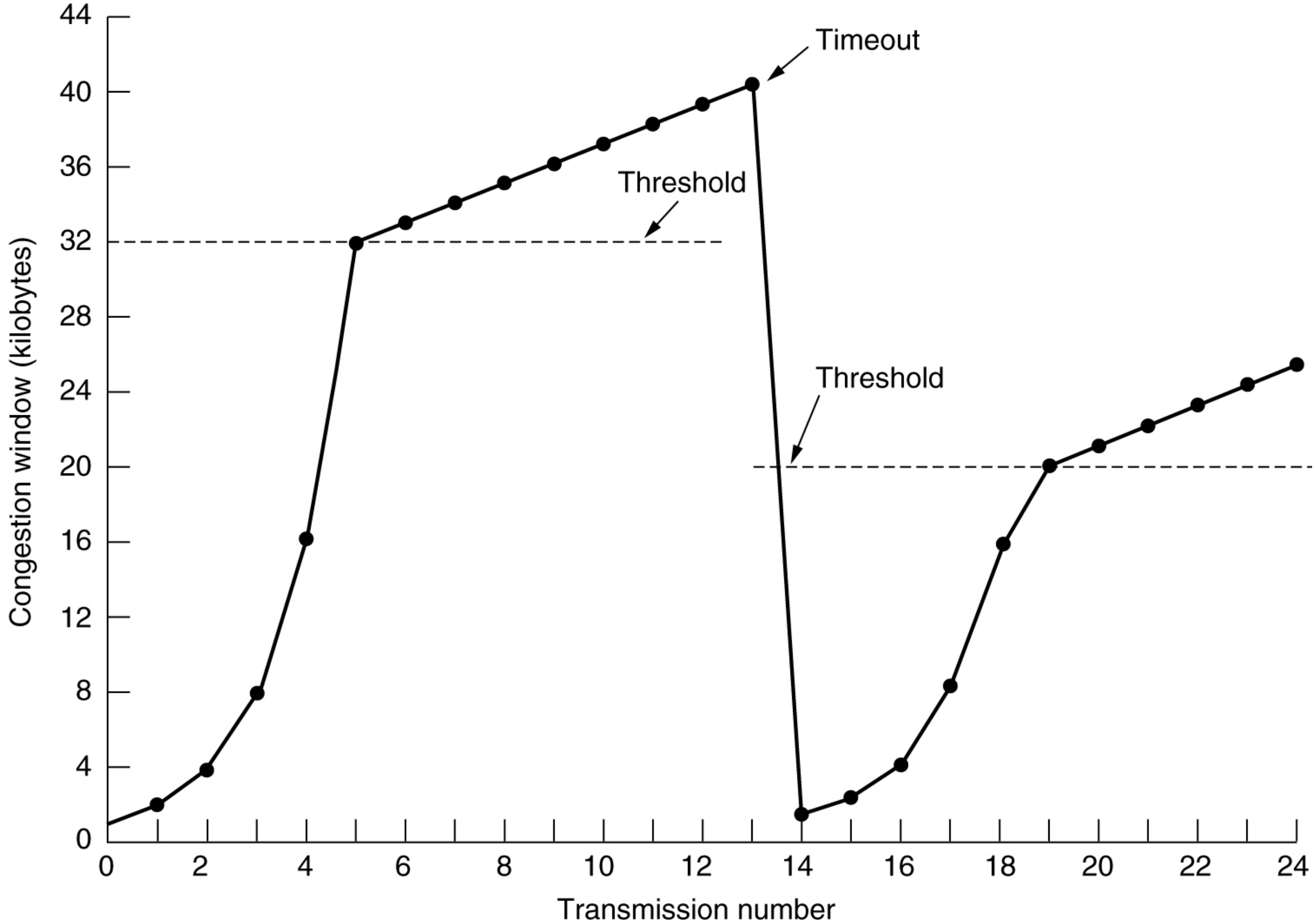
  → sender in slow-start phase, `cwnd` grows exponentially
- When `cwnd` is above Threshold

  → sender is in congestion-avoidance phase, `cwnd` grows linearly
- When a triple duplicate ACK occurs

  → `cwnd=cwnd/2`
- When timeout occurs

  → threshold set to `cwnd/2` and `cwnd` is set to 1 MSS.

TCP Congestion Control



An example of the Internet congestion algorithm.

TCP Round Trip Time and Timeout

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

How to estimate RTT?

- **`SampleRTT`**: measured time from segment transmission until ACK receipt
  - ignore retransmissions
- **`SampleRTT`** will vary, want estimated RTT "smoother"
  - average several recent measurements, not just current **`SampleRTT`**

TCP Round Trip Time and Timeout

- RTT: the best current estimate of the round-trip time to the destination
- TCP measures how long an ACK took (M)
- Then updates RTT according to the formula

$$RTT = \alpha RTT + (1-\alpha)M$$

α: a smoothing factor that determines how much weight is given to the old value (Typically $\alpha = 7/8$)

- For timeout period TCP uses βRTT, but the trick is choosing β.
- In the initial implementations, β was always 2, but experience showed that a constant value was inflexible

TCP Round Trip Time and Timeout

- Jacobson (1988) proposed $\beta \sim$ std. dev. of ACK arrival time pdf
  - Keep track of another smoothed variable, D, the deviation.
  - Whenever an ACK, the difference between the expected and observed values, | RTT - M |, is computed.
  - A smoothed value of this is maintained in D by the formula

$$D = \alpha D + (1-\alpha) \mid RTT - M \mid$$

- Here, α may or may not be the same value used to smooth RTT (typically ¾). Most TCP implementations use this

$$\text{Timeout} = \text{RTT} + 4\text{D}$$

- When a segment times out and is sent again, it is unclear whether the ACK is for the 1st transmission or later one
- Do not update RTT on any segments that have been retransmitted.
- Timeout is doubled on each failure until the segments get through the first time.
  - This fix is called Karn's algorithm. Most TCP implementations use it.

TCP Fairness

Fairness goal: if K TCP sessions share same bottleneck link of bandwidth R, each should have average rate of R/K