Transport Layer

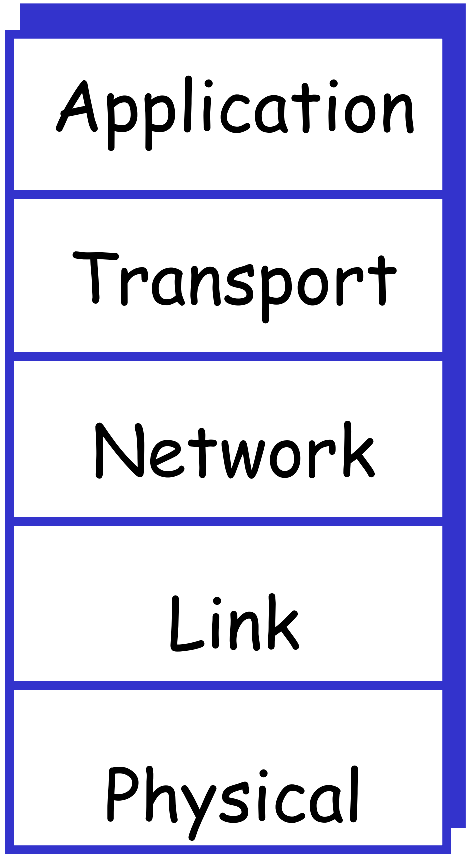
ELEC1200

- Principles behind transport layer services
- Multiplexing and demultiplexing
- UDP
- TCP Reliable Data Transfer
- TCP Congestion Control
- TCP Fairness

* *The slides are adapted from ppt slides (in substantially unaltered form) available from "Computer Networking: A Top-Down Approach," 4th edition, by Jim Kurose and Keith Ross, Addison-Wesley, July 2007. Part of the materials are also adapted from ELEC315 and MIT 6.02 course notes.*

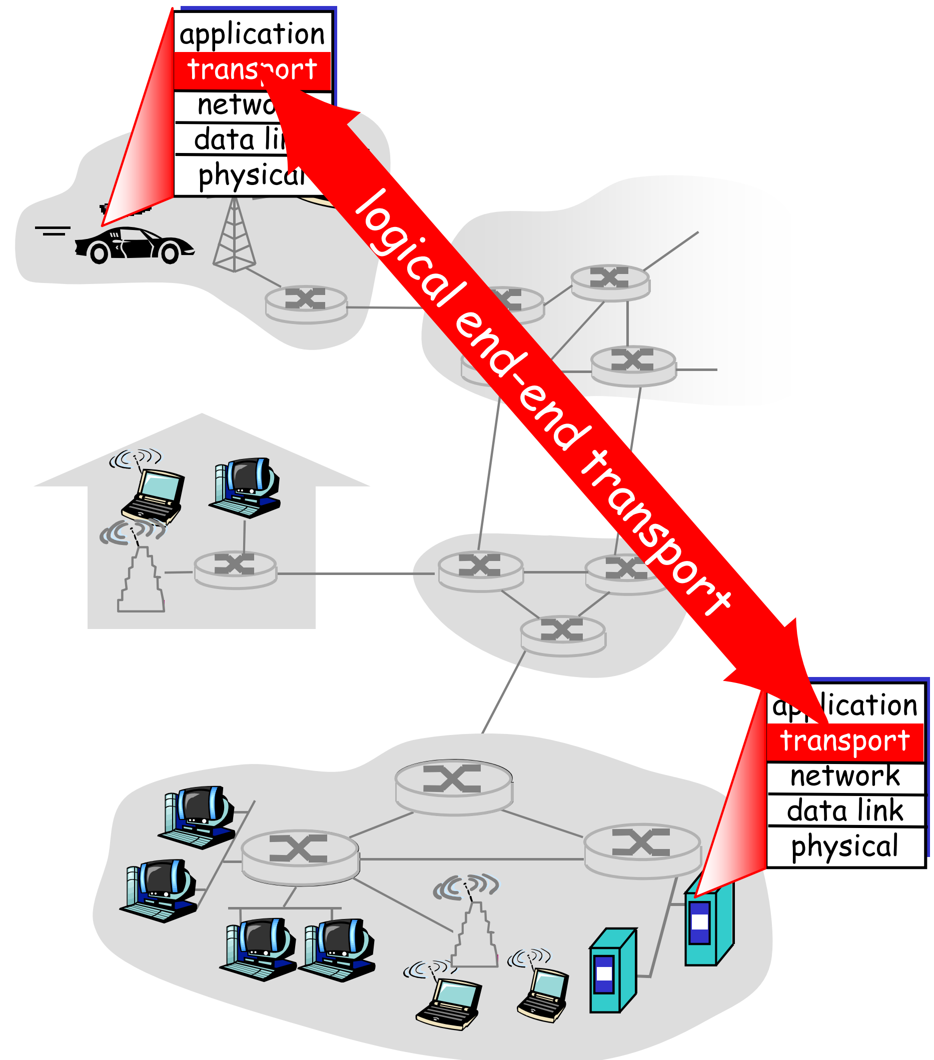
Internet protocol stack

- application: supporting network applications
  - HTTP, SMTP, FTP, DNS
- transport: process-process data transfer
  - TCP, UDP
- network: routing of datagrams from source to destination
  - IP, routing protocols
- link: data transfer between neighboring network elements
  - 802.11, Ethernet
- physical: bits "on the wire"



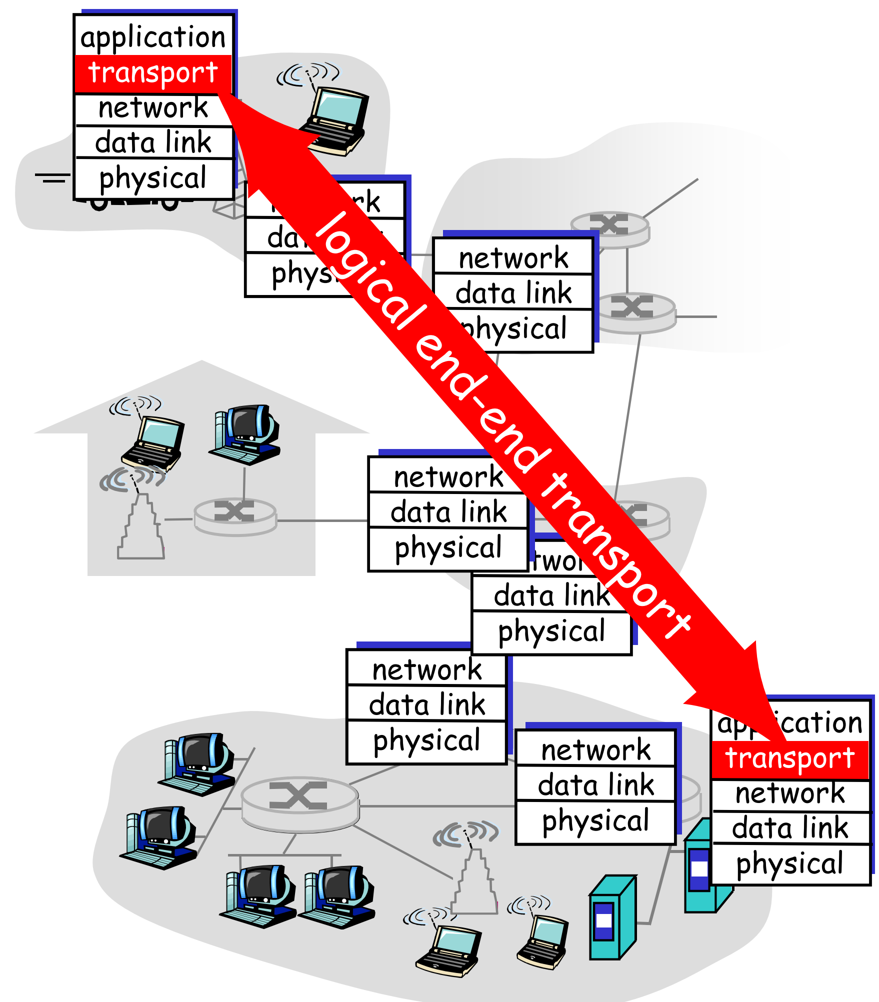
Transport services and protocols

- 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



Internet transport-layer protocols

- reliable, in-order delivery (TCP)
  - congestion control
  - flow control
  - connection setup
- unreliable, unordered delivery: UDP
  - no-frills extension of "best-effort" IP
- services not available:
  - delay guarantees
  - bandwidth guarantees



Multiplexing/demultiplexing

Demultiplexing at rcv host:

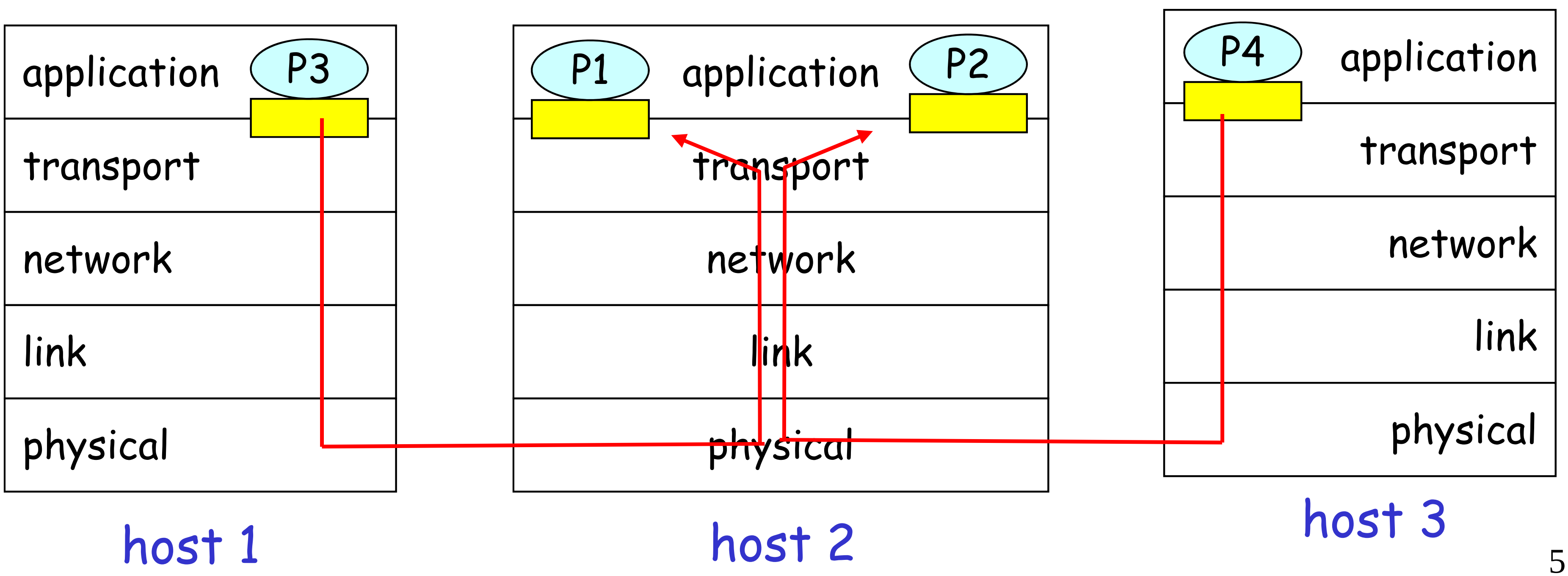
delivering received segments to correct socket

Multiplexing at send host:

gathering data from multiple sockets, enveloping data with header (later used for demultiplexing)

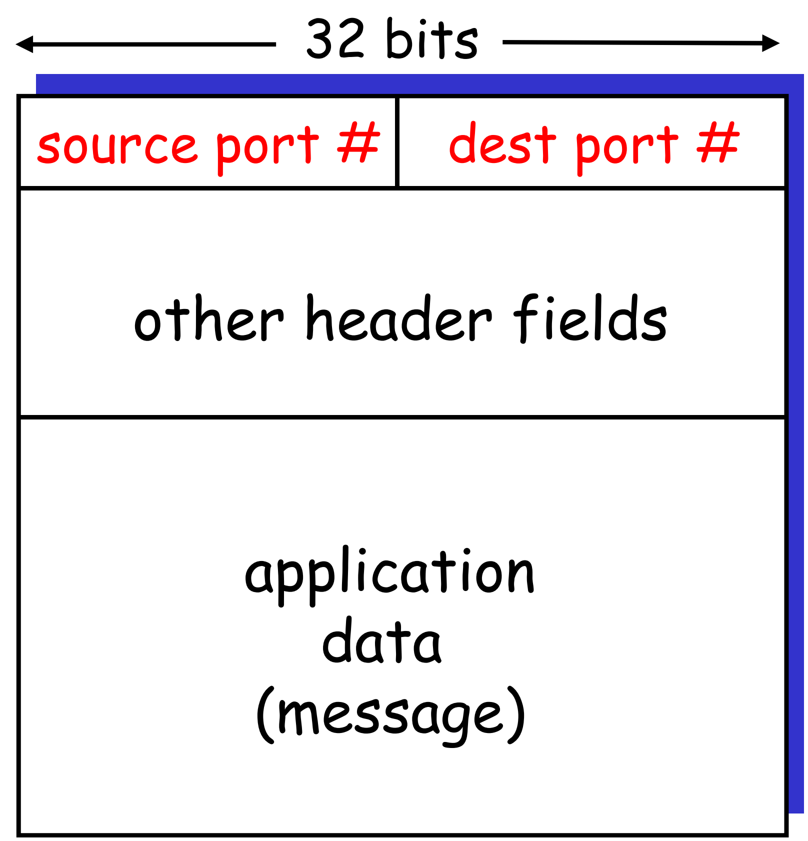
[yellow box] = socket [oval] = process

Socket: a door between application process and end-end-transport protocol (UDP or TCP)



How demultiplexing works

- host receives IP datagrams
  - each datagram has source IP address, destination IP address
  - each datagram carries 1 transport-layer segment
  - each segment has source, destination port number
- host uses IP addresses & port numbers to direct segment to appropriate socket



TCP/UDP segment format

UDP: User Datagram Protocol [RFC 768]

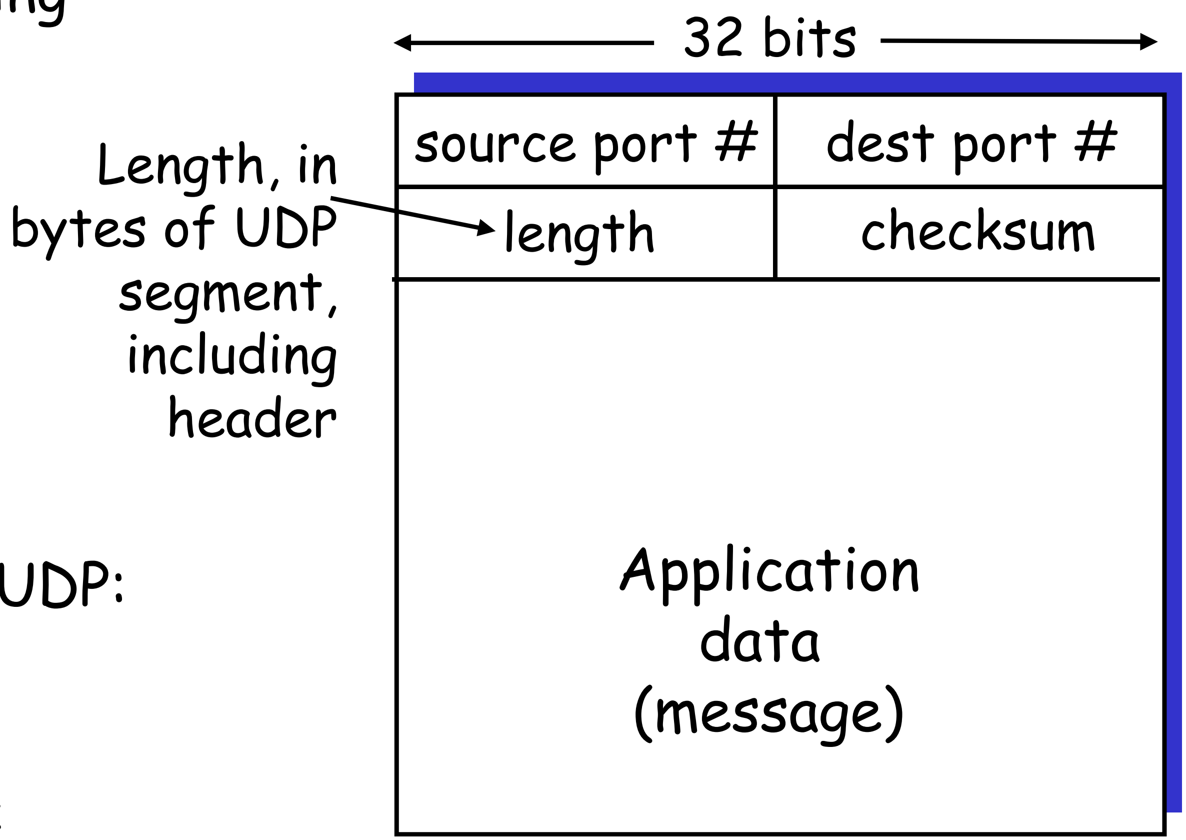
- "no frills," "bare bones" Internet transport protocol
- "best effort" service, UDP segments may be:
  - lost
  - delivered out of order to app
- *connectionless:*
  - no handshaking between UDP sender, receiver
  - each UDP segment handled independently of others

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: more

- 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

UDP checksum

Goal: detect "errors" (e.g., flipped bits) in transmitted segment

Sender:

- treat segment contents as sequence of 16-bit integers
- checksum: addition (1's complement sum) of segment contents
- sender puts checksum value into UDP checksum field

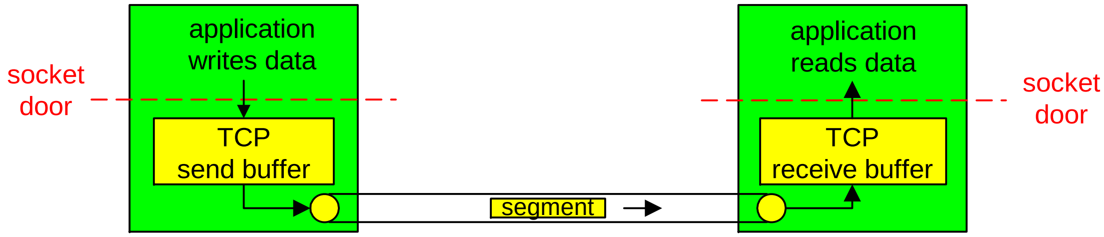
Receiver:

- compute checksum of received segment
- check if computed checksum equals checksum field value:
  - NO - error detected
  - YES - no error detected. *But maybe errors nonetheless?* More later ….

TCP: Overview

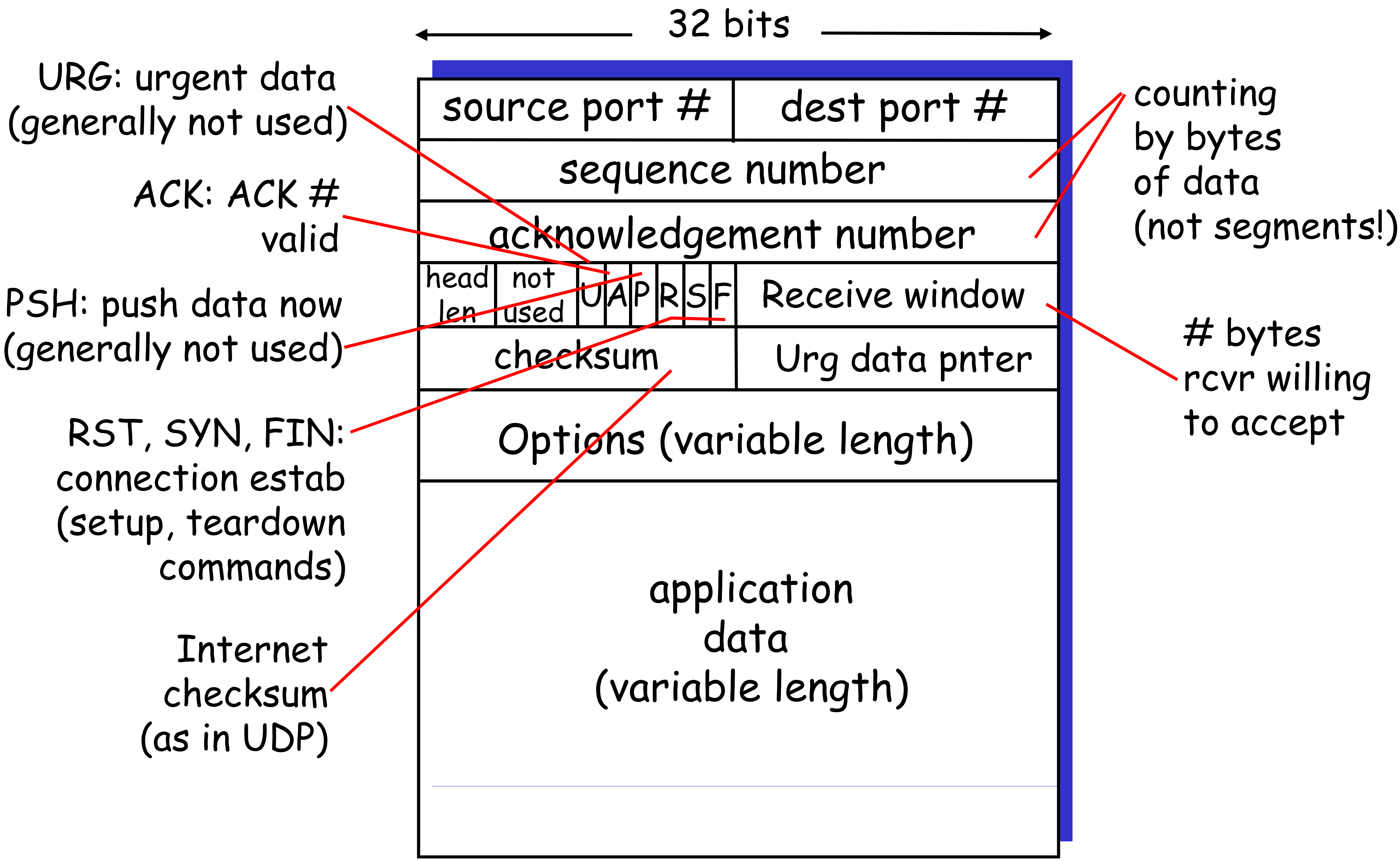
RFCs: 793, 1122, 1323, 2018, 2581

- point-to-point:
  - one sender, one receiver
- reliable, in-order *byte stream:*
  - no "message boundaries"
- pipelined:
  - TCP congestion and flow control set window size
- *send & receive buffers*



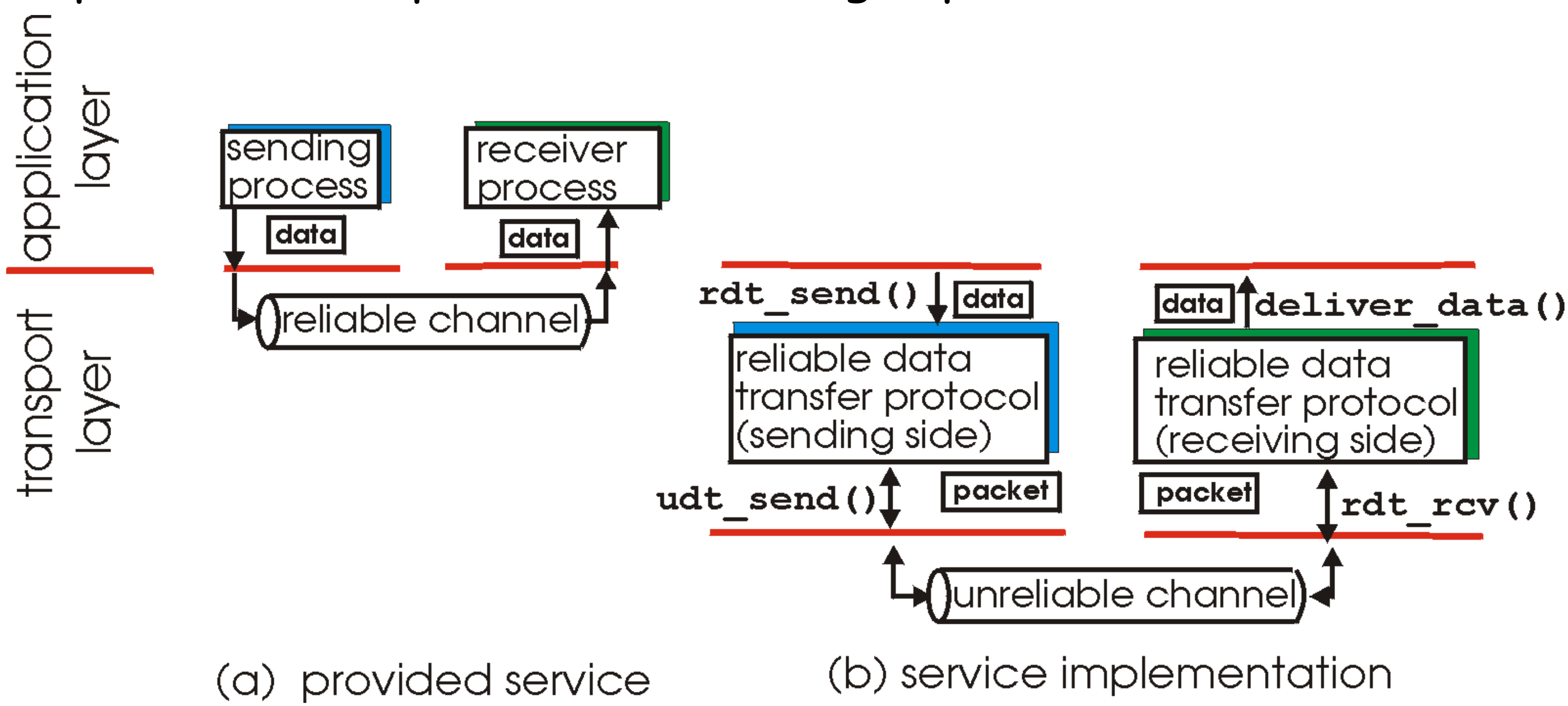
- full duplex data:
  - bi-directional data flow in same connection
  - MSS: maximum segment size
- connection-oriented:
  - handshaking (exchange of control msgs) init's sender, receiver state before data exchange
- flow controlled:
  - sender will not overwhelm receiver

TCP segment structure



Principles of Reliable data transfer

- important in app., transport, link layers
- top-10 list of important networking topics!

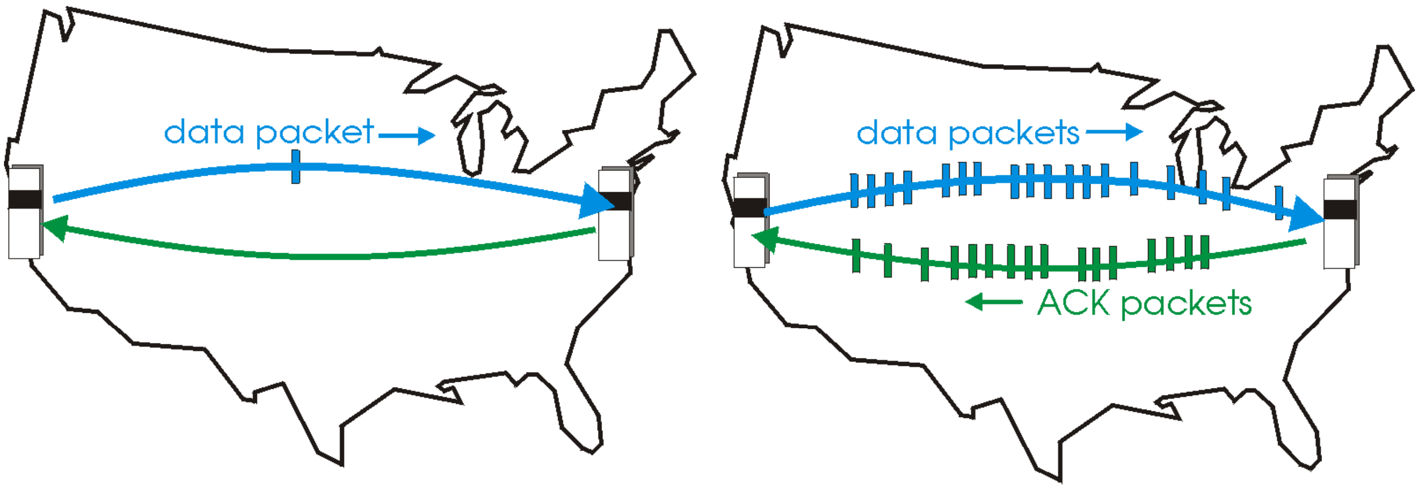


- characteristics of unreliable channel will determine complexity of reliable data transfer protocol (rdt)

Stop-and-wait & Pipelined protocols

Pipelining: sender allows multiple, “in-flight”, yet-to-be-acknowledged pkts

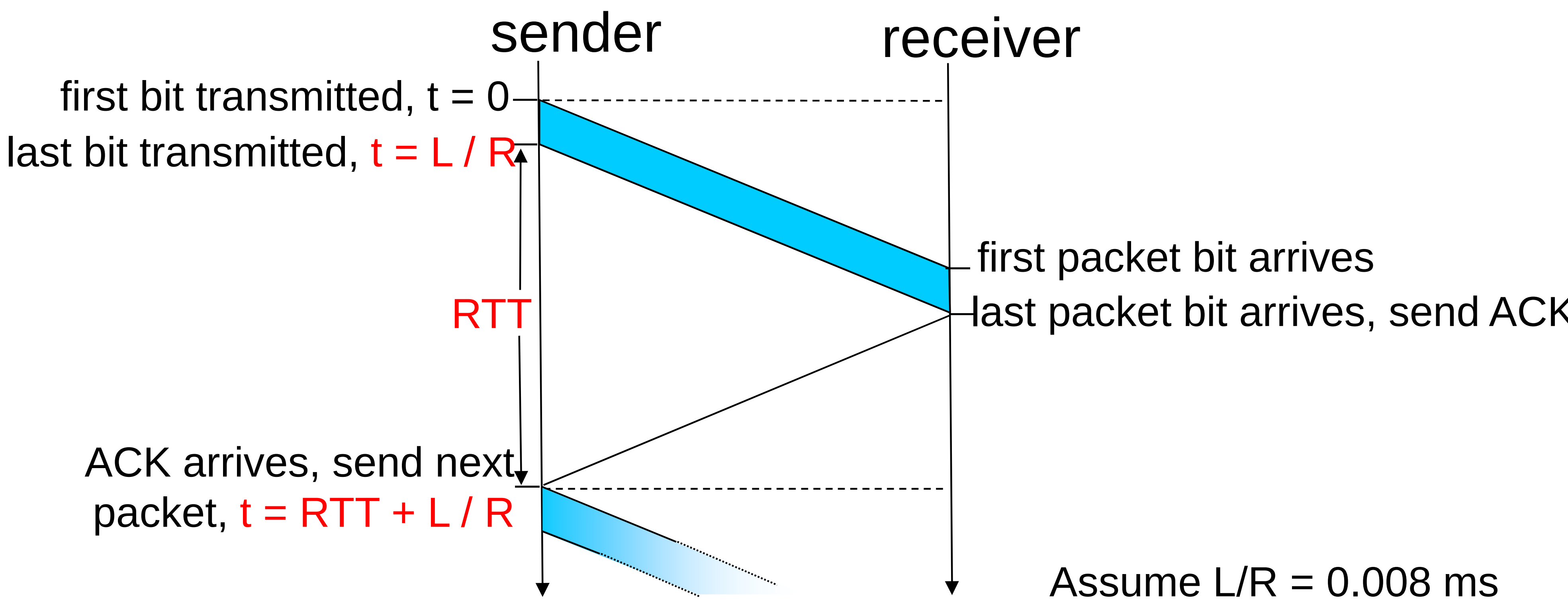
- range of sequence numbers must be increased
- buffering at sender and/or receiver



(a) a stop-and-wait protocol in operation (b) a pipelined protocol in operation

- Two generic forms of pipelined protocols: *go-Back-N, selective repeat*

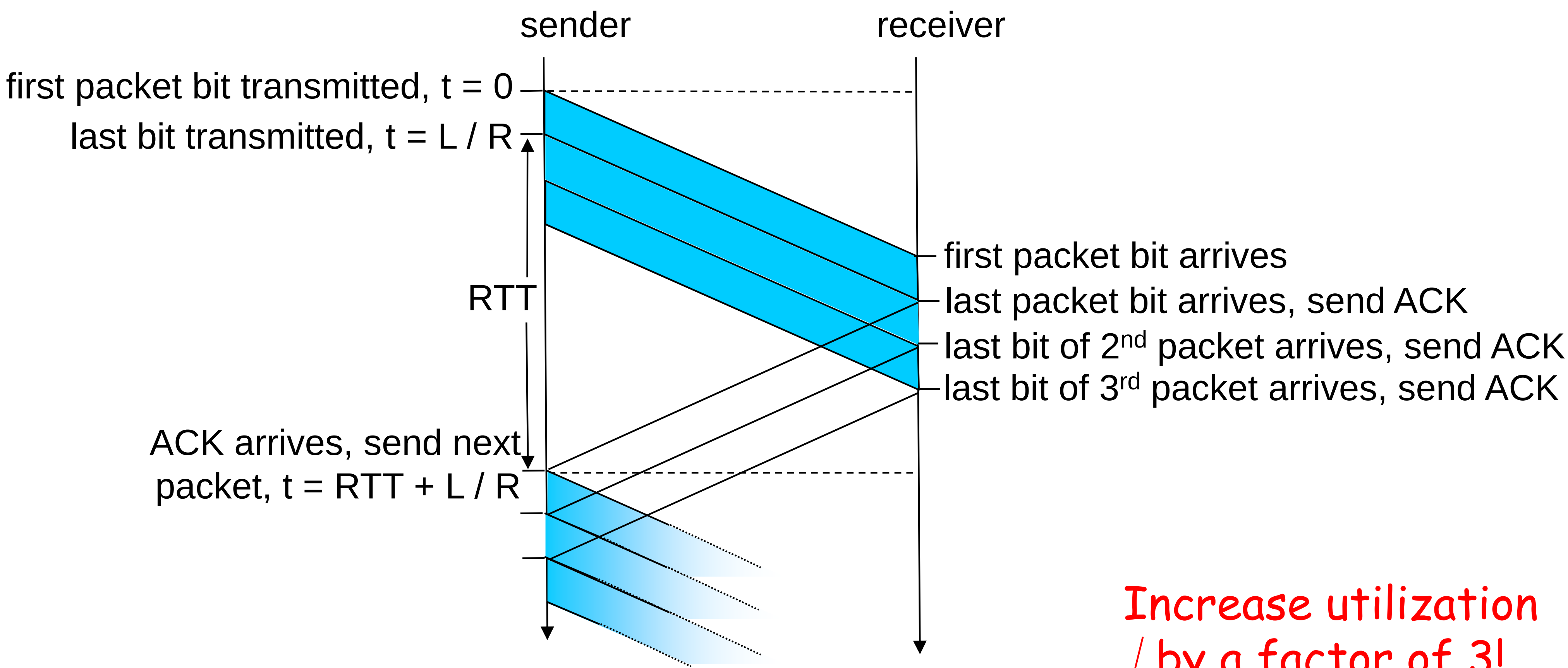
Stop-and-wait protocol



Assume L/R = 0.008 ms
and RTT = 30 ms

$$U_{sender} = \frac{L / R}{RTT + L / R} = \frac{.008}{30.008} = 0.00027$$

Pipelining: increased utilization



Increase utilization by a factor of 3!

$$U_{sender} = \frac{3 * L / R}{RTT + L / R} = \frac{.024}{30.008} = 0.0008$$

Principles of Congestion Control

Congestion:

- informally: "too many sources sending too much data too fast for *network* to handle"
- manifestations:
  - lost packets (buffer overflow at routers)
  - long delays (queueing in router buffers)
- a top-10 problem!

Approaches towards congestion control

Two broad approaches towards congestion control:

End-end congestion control:

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

Network-assisted congestion control:

- routers provide feedback to end systems
  - single bit indicating congestion (SNA, DECbit, TCP/IP ECN, ATM)
  - explicit rate sender should send at

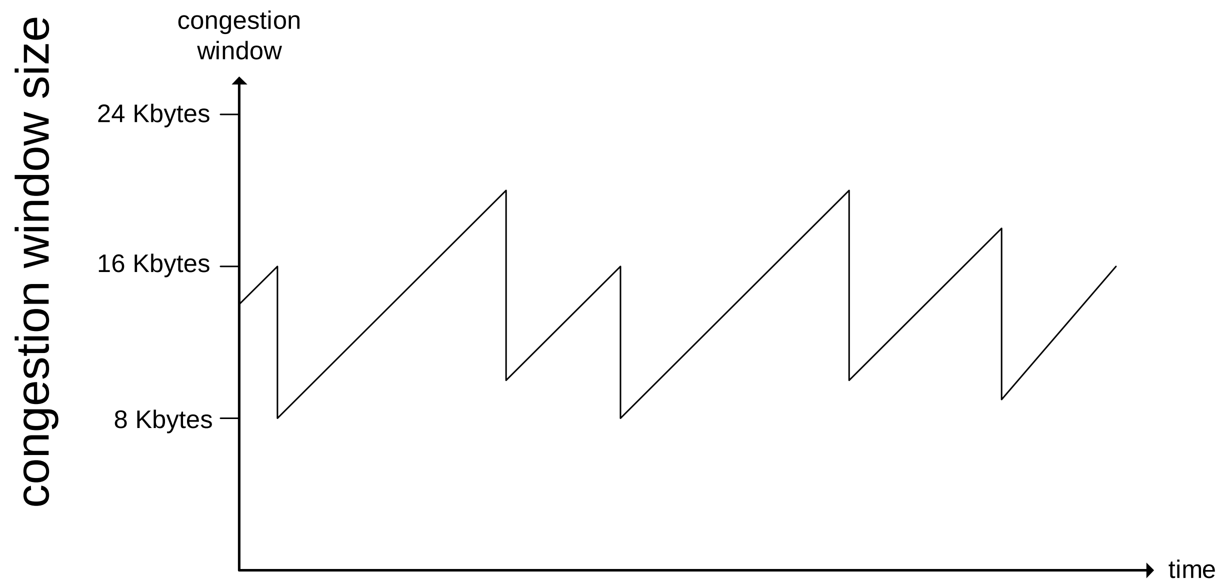
TCP congestion control: additive increase, multiplicative decrease

- *Approach:* increase transmission rate (window size), probing for usable bandwidth, until loss occurs

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

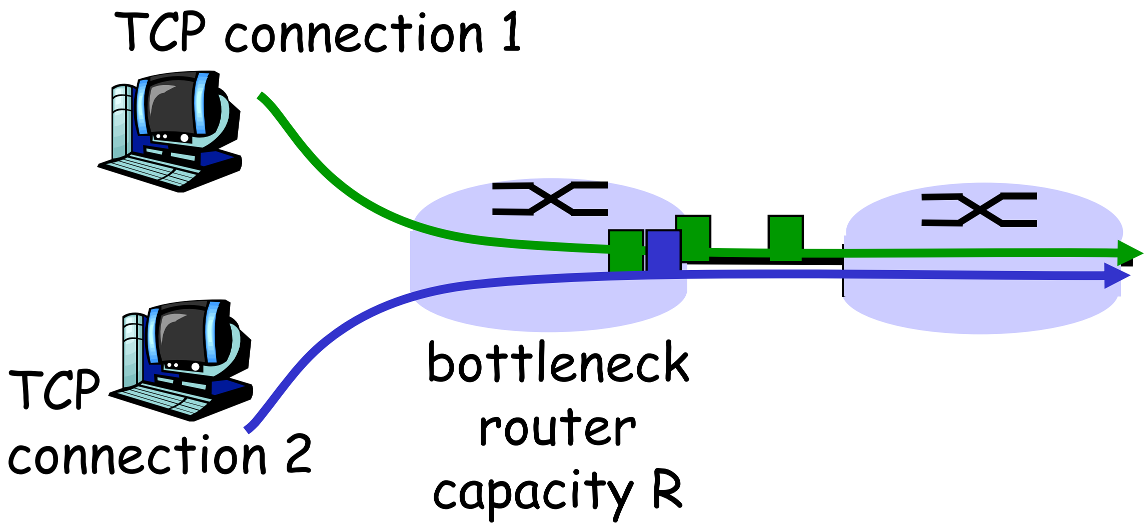
  - *additive increase:* increase **CongWin** by 1 MSS every RTT until loss detected
  - *multiplicative decrease*: cut **CongWin** in half after loss

Saw tooth behavior: probing for bandwidth



TCP Fairness

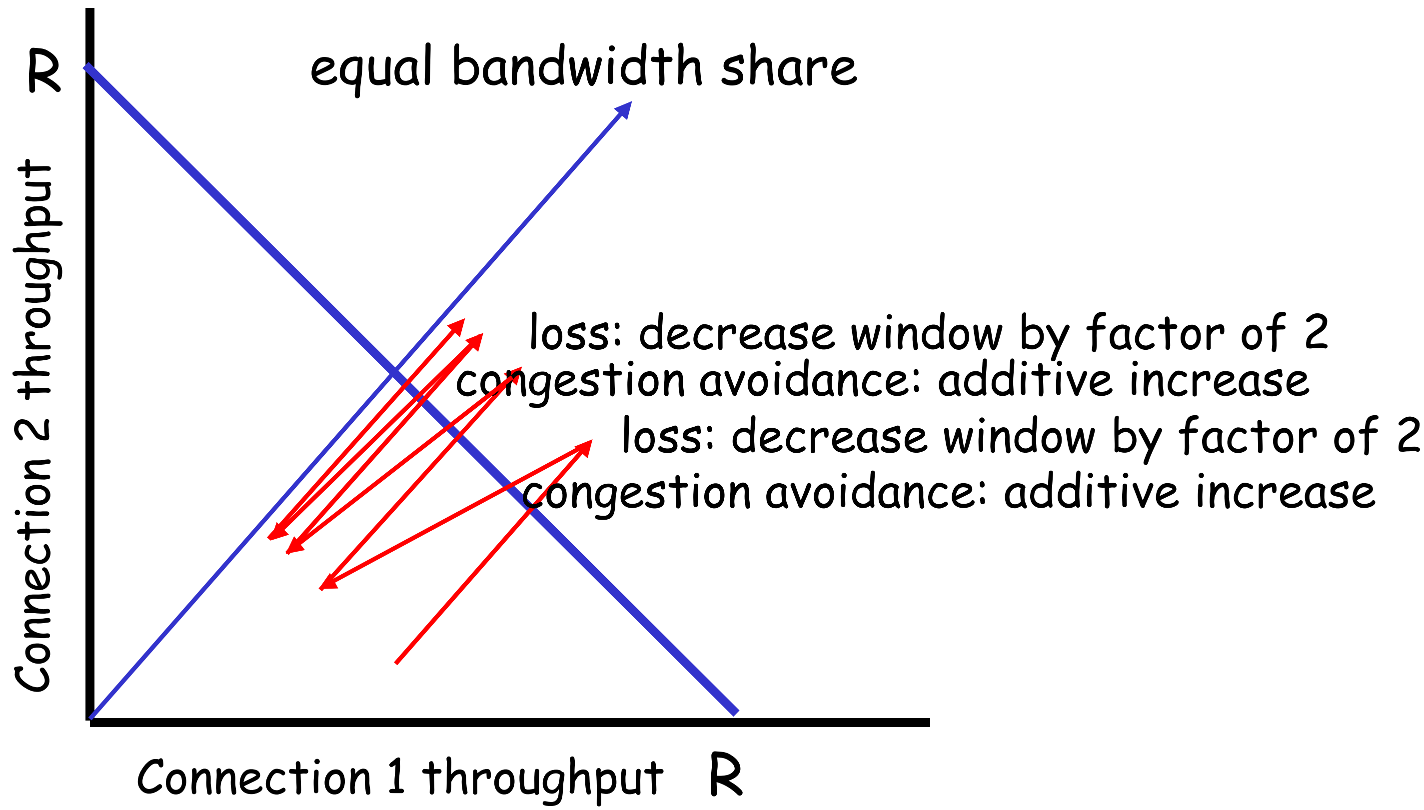
Fairness goal: if K TCP sessions share same bottleneck link of bandwidth R, each should have average rate of R/K



Why is TCP fair?

Two competing TCP connections:

- Additive increase gives slope of 1, as throughout (rate) increases
- Multiplicative decrease decreases throughput proportionally



Fairness (more)

Fairness and UDP

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

Fairness and parallel TCP connections

- nothing prevents app from opening parallel connections between 2 hosts.
- Web browsers do this
- Example: link of rate R supporting 9 connections;
  - new app asks for 1 TCP, gets rate R/10
  - new app asks for 11 TCPs, gets R/2 !

Summary

- Principles behind transport layer services:
  - multiplexing, demultiplexing
  - reliable data transfer
  - flow control
  - congestion control
- Internet provides two transport protocols
  - UDP
  - TCP