Transport Services and Protocols (2)

Andy Carpenter

(Andy.Carpenter@manchester.ac.uk)

Transmission Control Protocol (TCP)

P&D: 5.2

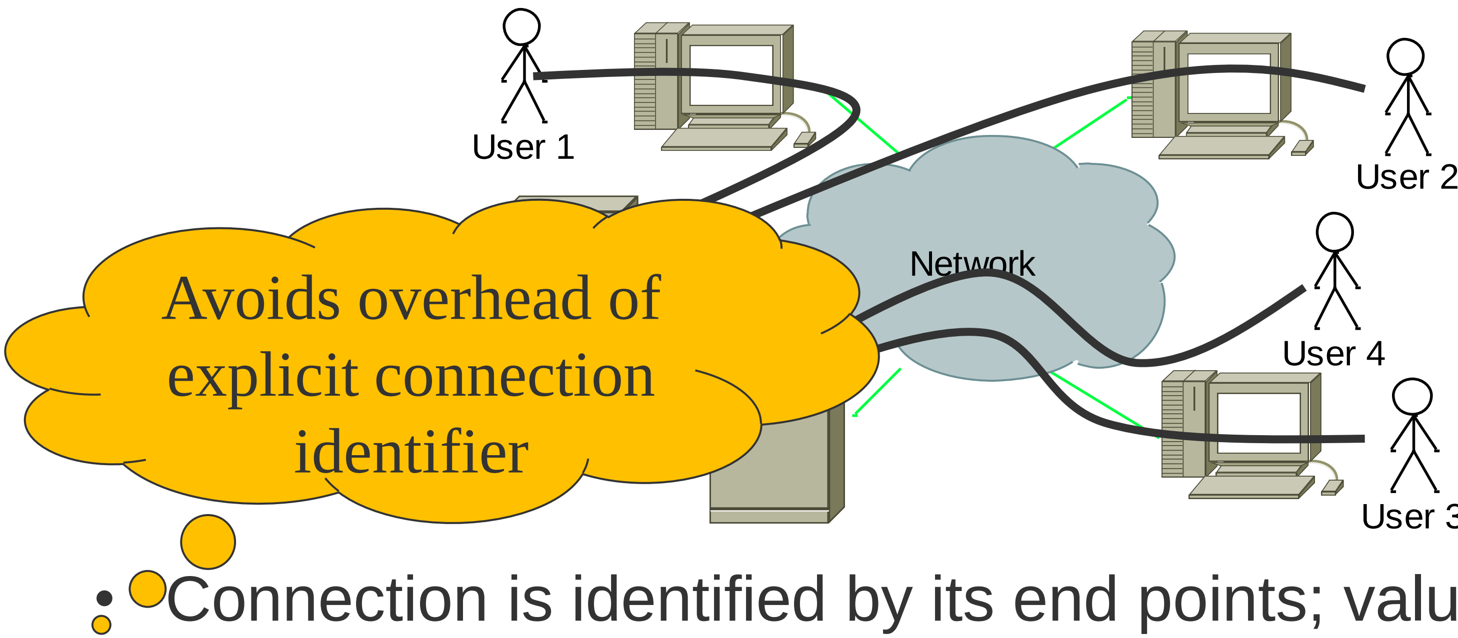
- Provides robust process-to-process delivery service
- Creates a virtual connection between applications
- Issues:
  – identifying connections, establishing connections
  – implementing service model, port mechanism
- Implements finely tuned congestion-control mechanism
- Used when application wishes to avoid:
  – complexities of network, error recovery
- Typically used for user applications; e.g.:
  – email (SMTP), web surfing (HTTP)

TCP: Service Model

- Substantially modifies service model of IP; provides:
  - connection-oriented delivery between applications
  - reliable byte stream
    - all bytes sent are delivered in-order
  - unstructured stream; no break into messages
  - full-duplex connection (two streams)
  - buffered; data not immediately sent and delivered
  - each stream is independently flow-controlled

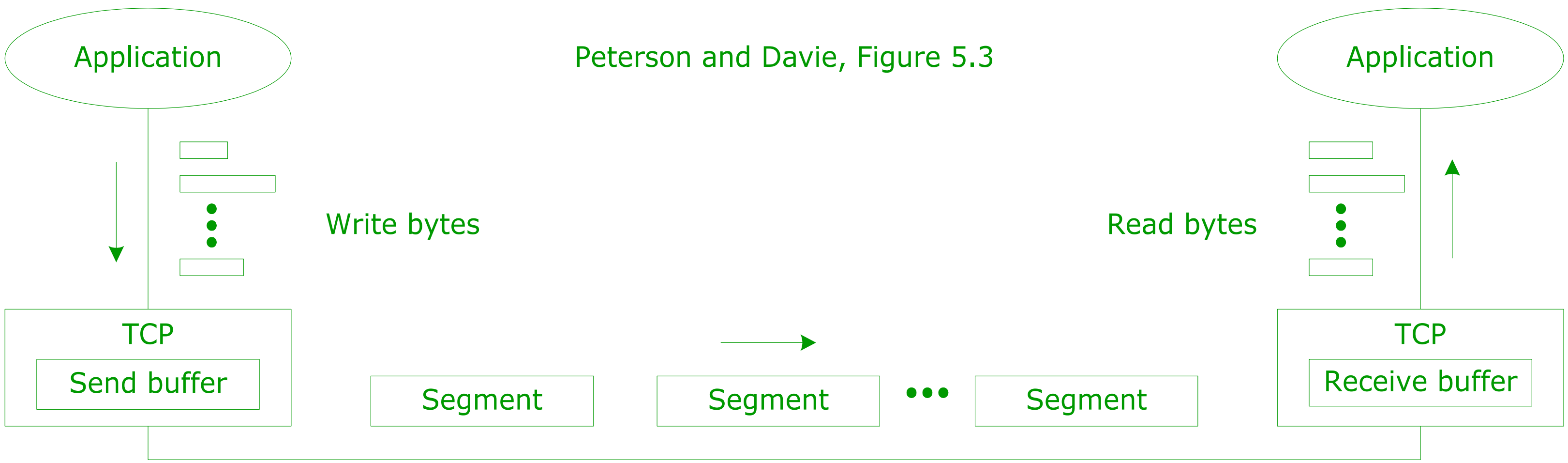
TCP: Identifying Connections

- TCP forms a virtual connection between applications
- Service end of connection may not be unique; e.g.:



- Connection is identified by its end points; value of:
  - source IP address, source port number
  - destination IP address, destination port number

TCP: Segmentation



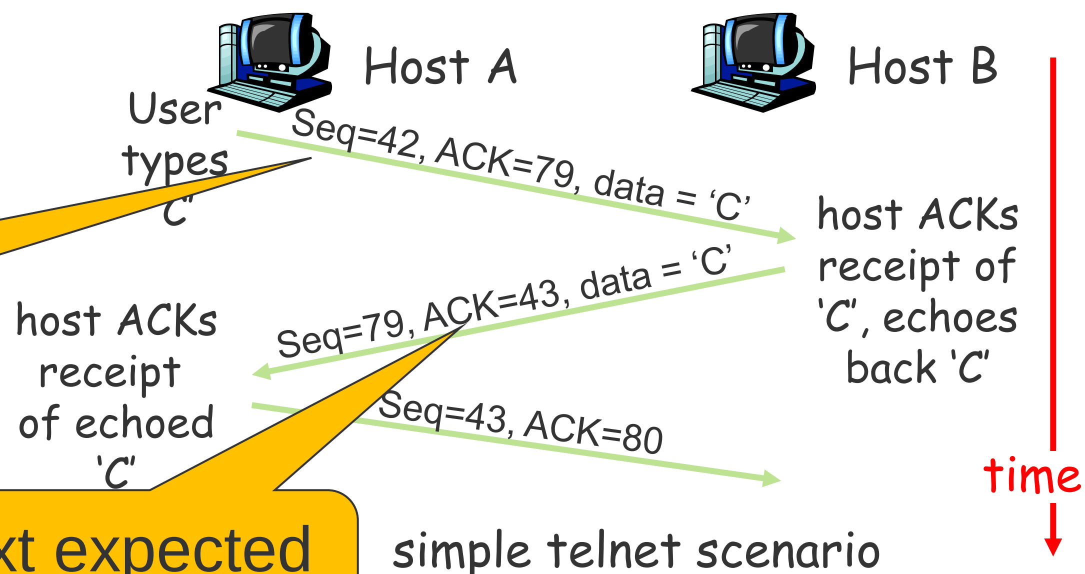
- Application writes bytes to connection as needs
- TCP collects (buffers) bytes before sending
- Sends set of bytes as a TCP segment in IP datagram
- Receiving TCP also buffers
- Application reads bytes from connection as required

TCP: Reliability

- TCP uses
  - sliding window, Go-Back-N (cumulative ACKs)
  - sequence numbers for bytes not segments
  - single retransmission timer
- Retransmissions are triggered by:
  - timeout events
  - duplicate acks

Why?

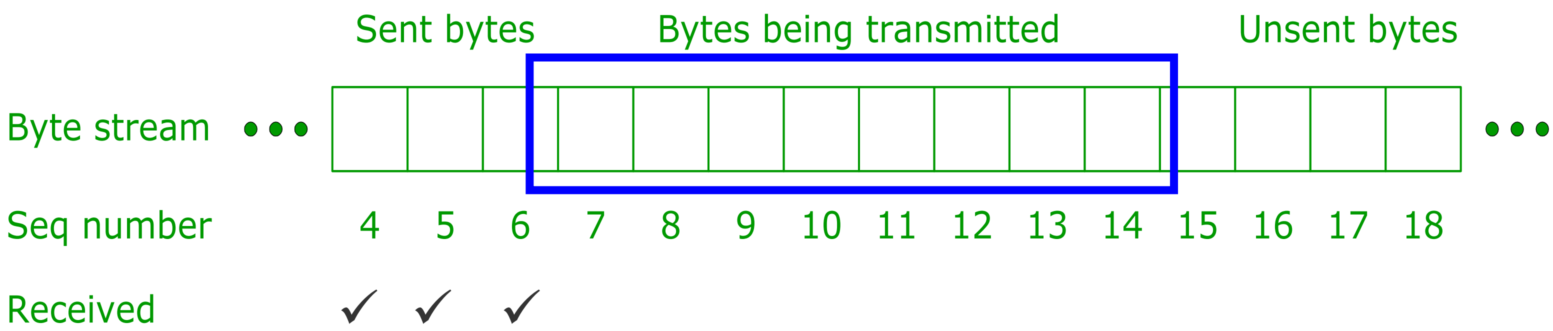
Number of first byte in data



Next expected number

TCP: Reliability

- Example, window 8, last ACK 7



TCP: Reliability

- Individual bytes have a 32-bit sequence number
- Send sequence number of first byte in segment
- Sequence numbers of other bytes implicitly transmitted
- Sequence numbers in each direction are independent
- Segment numbers must be unique; avoid wraparound
- TCP uses size of identifier and probable data rate
  - at 1MB/sec, about 1 hour to wraparound

TCP: Value of Retransmit Timeout

- If value too small, unnecessarily retransmit segments
- If value too large, get excessive delays before retransmit
- Appropriate value related to RTT, but that depends on:
  - pair of hosts involved, time, congestion in network
- For each connection:
  - use adaptable algorithm to determine RTT
- Basic algorithm sets timeout to twice estimated RTT
- Karn/Partridge algorithm reduces miscalculations
- Jacobson/Karels algorithm
  - copes with significant variance in real RTT

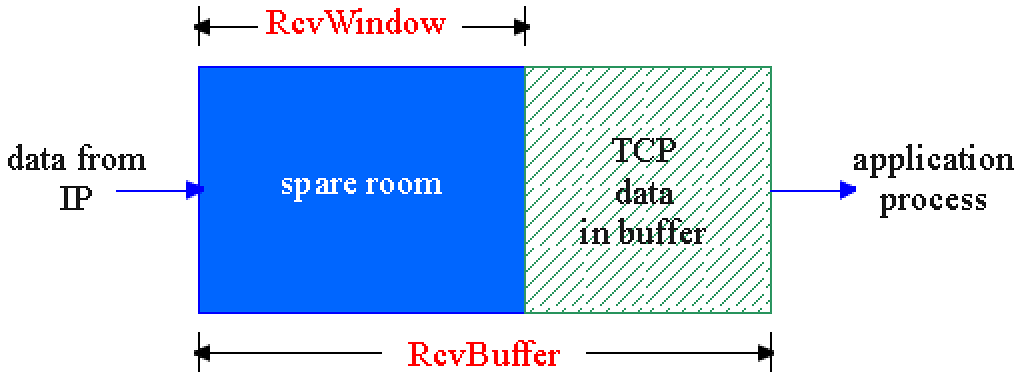
TCP: Estimation of Connection RTT

P&D: 5.2.6

- Estimate RTT as average of measured RTTs:
- When re-sending, to which transmit does ACK relate?
- Karn/Partridge algorithm:
  - only measures RTT for non-retransmitted segments
  - doubles timeout value for each retransmission
- Jacobson/Karels Algorithm:
  - if RTT variation small, average is good approx.
  - if RTT variation large, average is poor approx.
  - algorithm takes account of variation

TCP: Connection Flow Control

- Receiving buffer has finite capacity; must not overrun



Flow control

Sender will not overflow receiver's buffer by transmitting too much, too fast

Speed-matching service

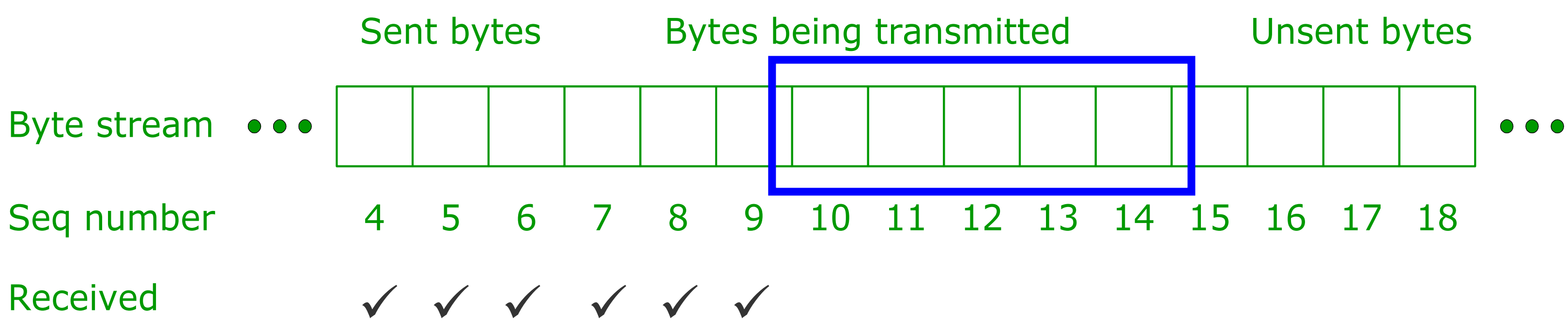
Matching the send rate to the receiving applications drain rate

Flow control allows receiver to influence transmitter

P&D: 5.2.4

TCP: Connection Flow Control

- Uses variant of sliding windows algorithm
  - – window size can be changed



- Can get deadlock

TCP: Segment Format

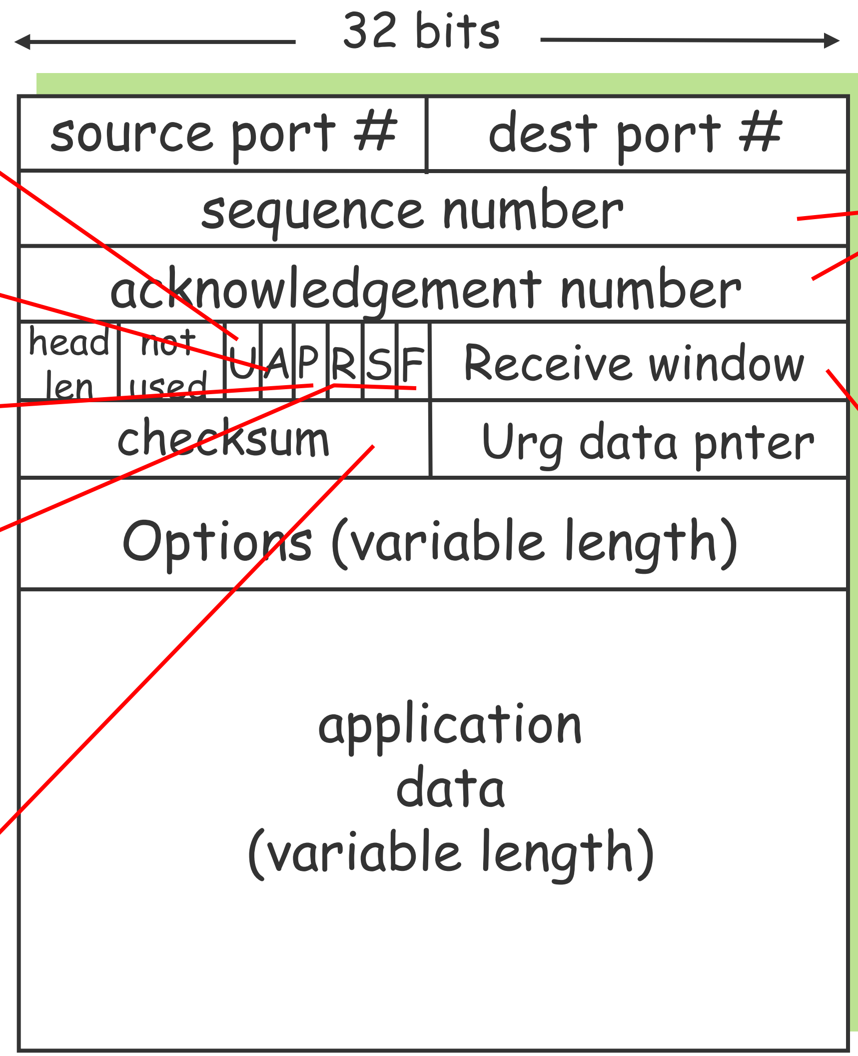
URG: urgent data (generally not used)

ACK: ACK # valid

PSH: push data now (generally not used)

RST, SYN, FIN: connection establishment (setup, teardown commands)

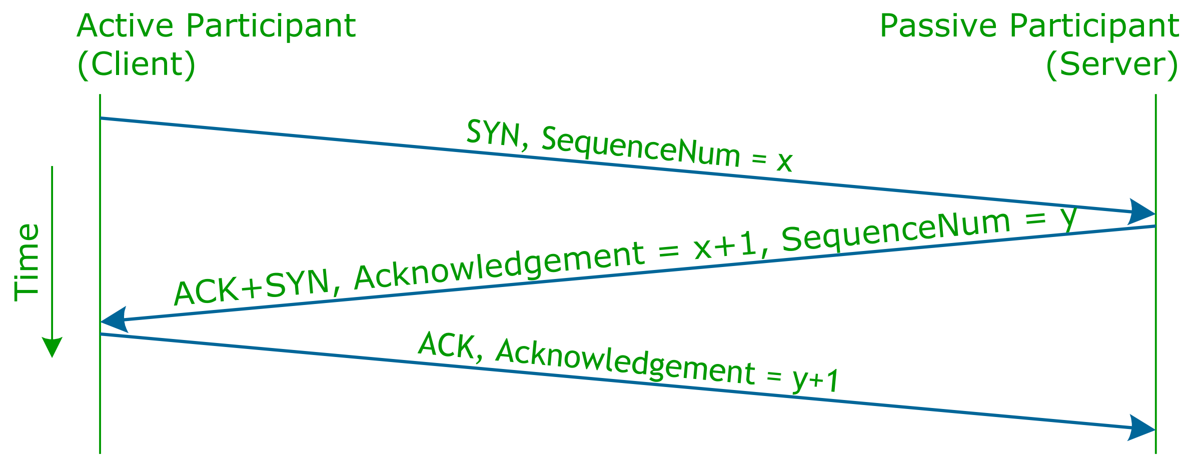
Internet checksum (as in UDP)



counting by bytes of data (not segments!)

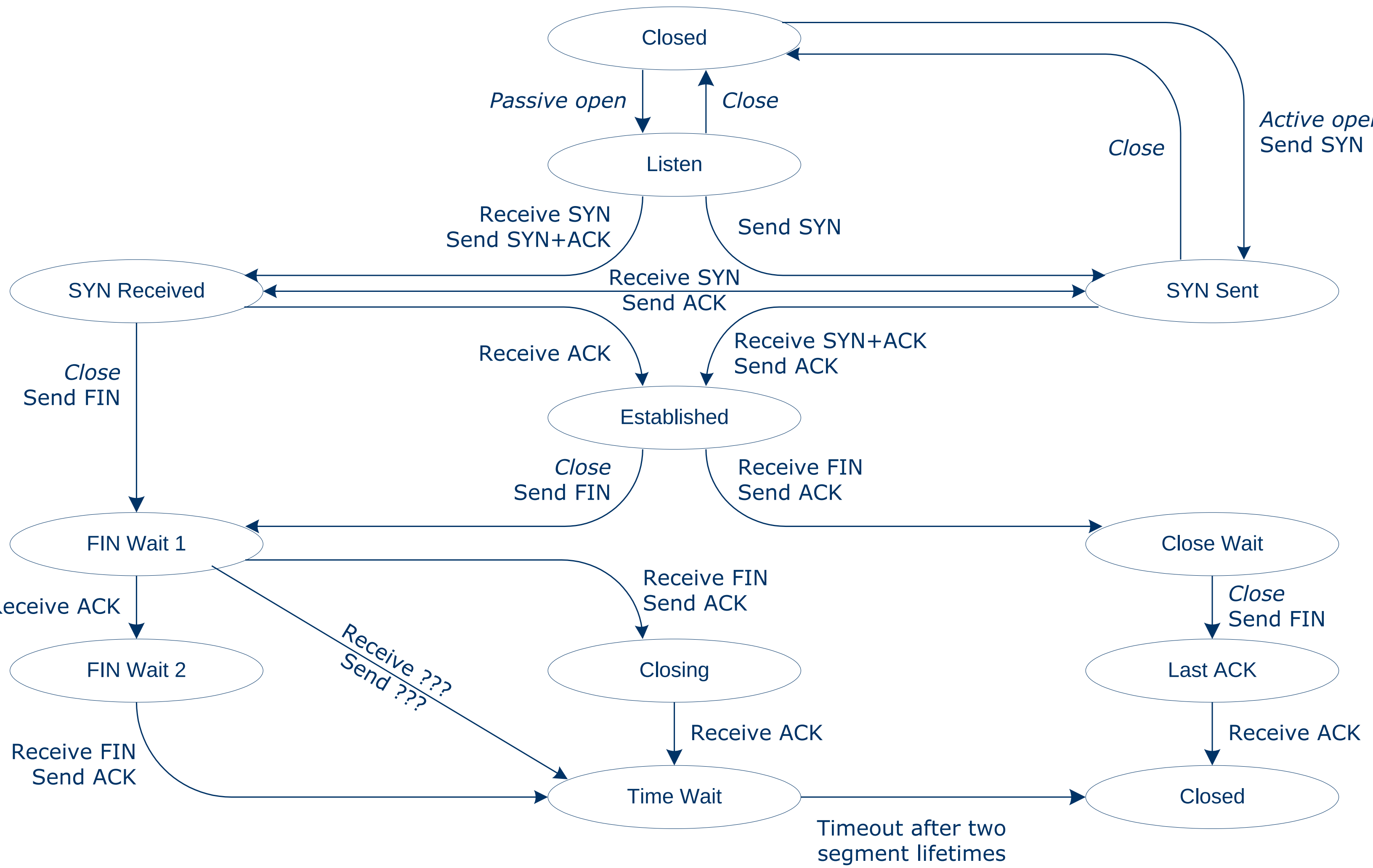
bytes receiver willing to accept

TCP: Establishing Connections



- This exchanges information needed to exchange data
  - initial sequence numbers (ISN) for each end

TCP: State Transition Diagram



TCP: Keeping the Pipe Full

- Capacity of physical connection (100ms RTT)

T1 (1.5Mbps)	18KB	Ethernet (10Mbps)	122KB
T3 (45Mbps)	549KB	FDDI (100Mbps)	1.2MB
STS-3 (155Mbps)	1.8MB	STS-12 (622Mbps)	7.4MB
STS-24 (1.2Gbps)	14.8MB		

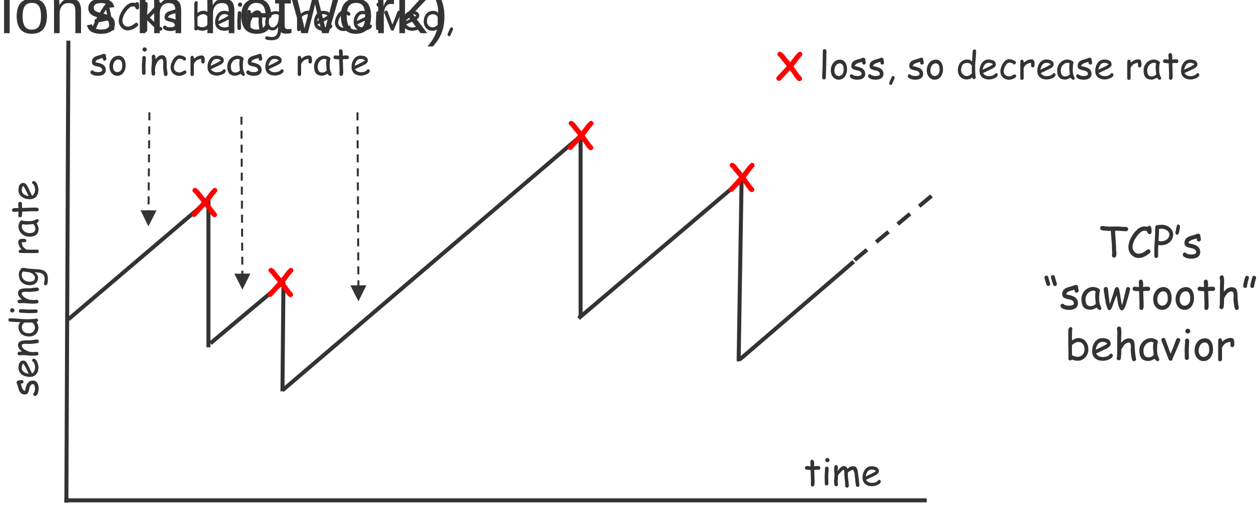
- Maximum window of single TCP connection: 2^{16} = 64KB
- Maximising use of cables relies on shared use
- Approaching point where:
  - capacity on cable exists but no acknowledgement
- Proposed extension allows:
  - multiplier factor for sequence number and window size

TCP congestion control:

- *goal:* TCP sender should transmit as fast as possible, but without congesting network
  - Q: how to find rate *just* below congestion level
- decentralized: each TCP sender sets its own rate, based on *implicit* feedback:
  - *ACK:* segment received (a good thing!), network not congested, so increase sending rate
  - *lost segment:* assume loss due to congested network, so decrease sending rate

TCP congestion control: bandwidth probing

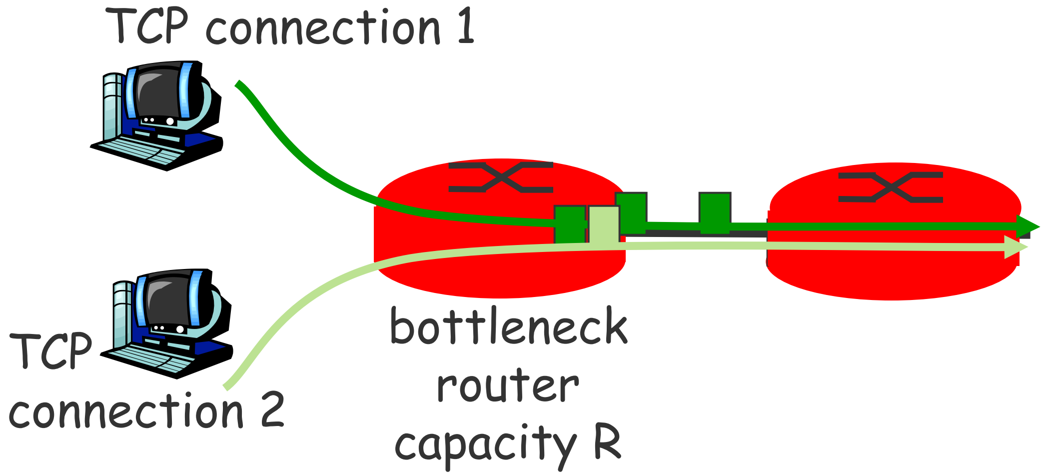
- "probing for bandwidth": increase transmission rate on receipt of ACK, until eventually loss occurs, then decrease transmission rate
  - continue to increase on ACK, decrease on loss (since available bandwidth is changing, depending on other connections in network)



- Q: how fast to increase/decrease?
  - details to follow

TCP Fairness

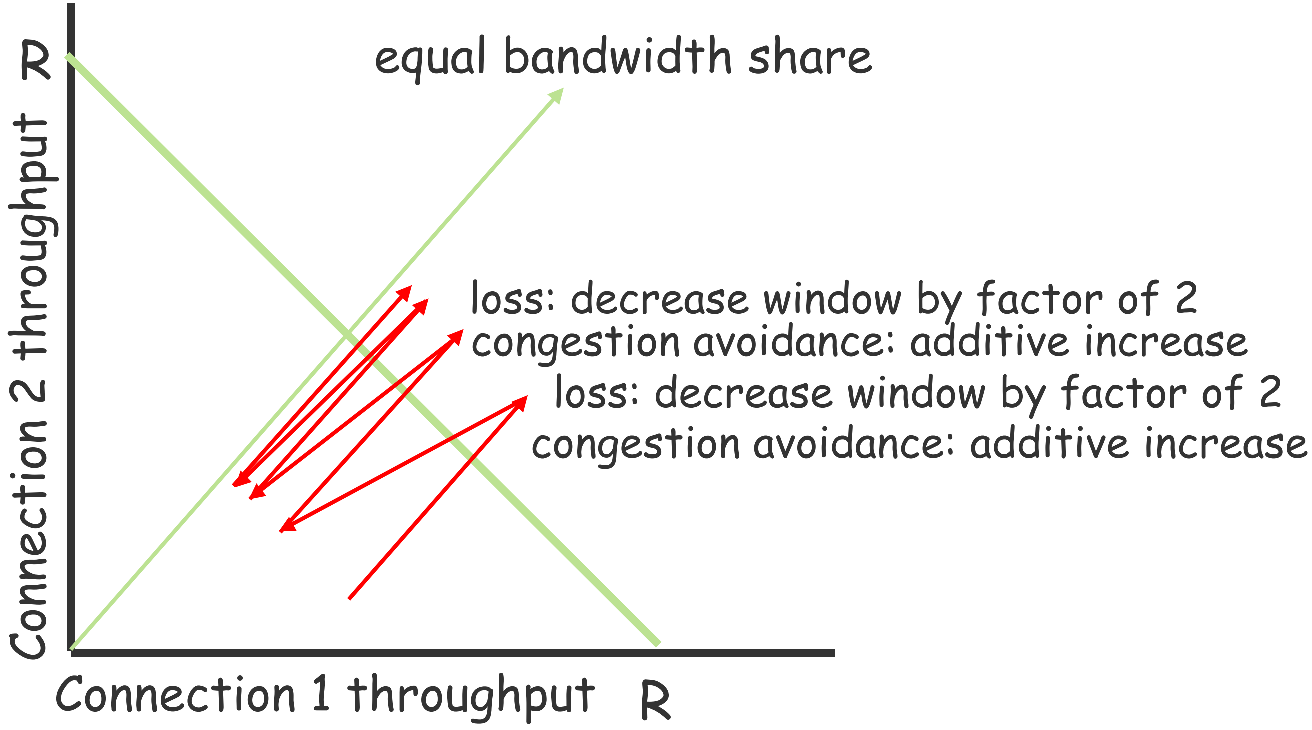
fairness goal: if K TCP sessions share same bottleneck link of bandwidth R, each should have average rate of R/K



Why is TCP fair?

Two competing sessions:

- Additive increase gives slope of 1, as throughout increases
- multiplicative decrease decreases throughput proportionally



Transport Summary

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

Next:

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