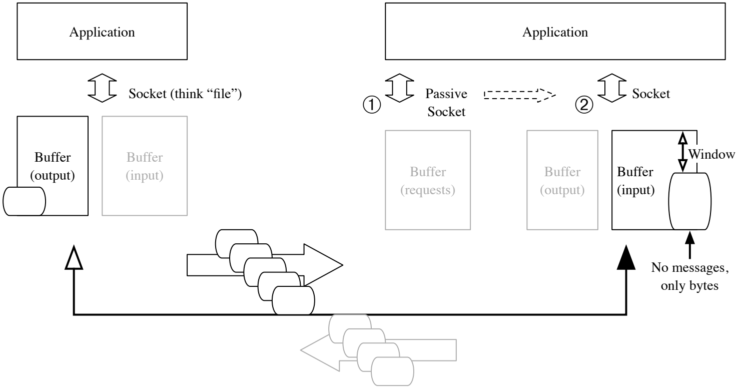
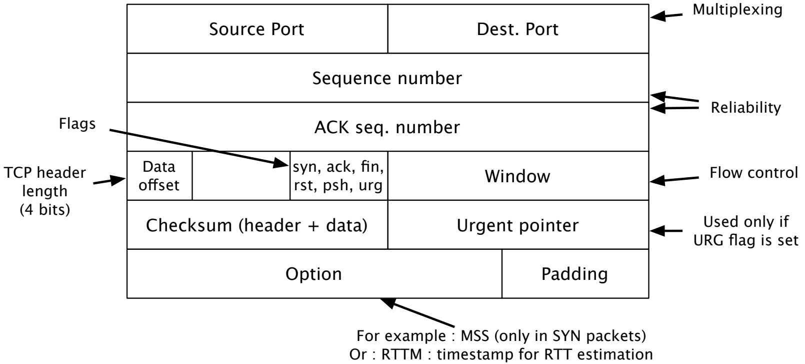
TCP
Transport Control Protocol

Martin Heusse

TCP is for transferring *files*



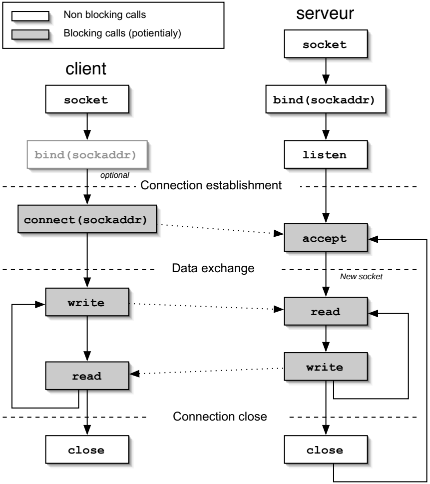
TCP is a reliable transport protocol



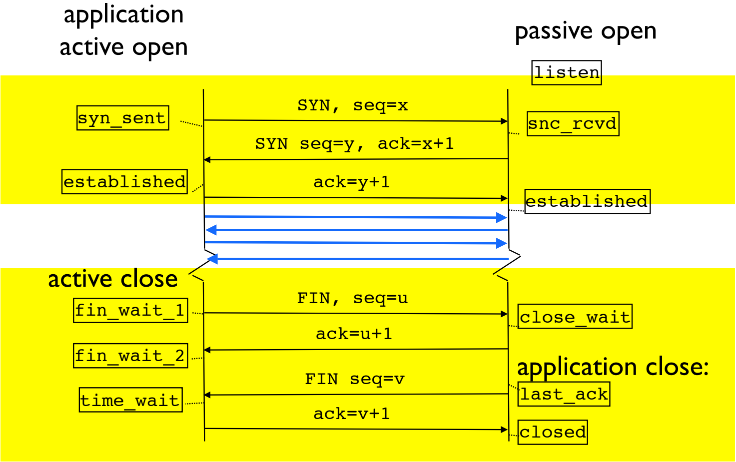
Connection establishment/tear-down

- 3 way handshake (`connect`)
  - ✓ On server side, a passive socket waits for a connection
  - ✓ SYN – SYN-ACK –ACK
  - ✓ On server side, an active socket is created
- Connection tear-down
  - ✓ Fin – ACK / Fin – ACK
  - ✓ Reading from a closed connection: first receive 0 byte, then error

Connection establishment/tear-down

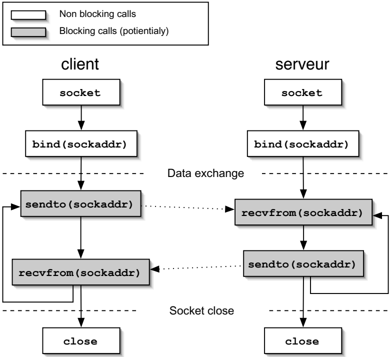


Connection establishment/tear-down (cont.)



Side note: UDP communication

- Compare with previous slides…



TCP acks

- The ACK tells the next expected seq number
- ACKs are cumulative
- Timeout triggers loss recovery
  - ✓ Or duplicate ACKS (see fast retransmit below)
  - ✓ Need to use a proper timeout value
- Delayed ACK mechanism (up to 500ms!)
  - ✓ Leave time to the application to answer
  - ✓ At least one ACK every other received segment
- **Nagle** Algorithm
  - ✓ No more than one unacknowledged short packet
  - ✓ $\Rightarrow$ More efficient network usage

…might be a problem

Can be turned on/off with the NO_DELAY socket option.

TCP Retransmission Timeout RTO

- Too short $\rightarrow$ spurious retransmissions
- Too long is not good either
- each time a new measurement is available:
  $\textit{EstimatedRTT} \leftarrow (1 - \textit{x}) \times \textit{EstimatedRTT} + \textit{x} \times \textit{measuredRTT}$
  x = 1/8
  (+ similar formula for estimating standard deviation)

$$\boxed{\textit{RTO} \leftarrow \textit{EstimatedRTT} + 4 \times \textit{EstimatedStdDev}}$$

or **200ms** (or 1s on some systems) if RTO is below this!!!

TCP sliding window

- TCP allows to send *cwnd* (congestion window) unacknowledged bytes
- *cwnd* always smaller than WIN field (buffer size at receiver)(flow control)
- *cwnd* changes over the connection life : slow start, then congestion avoidance

Silly window syndrome

- If the data consumer reads byte by byte in the reception buffer
  This (would) cause(s) sending of ACKs with WIN=1
  $\rightarrow$ small segments…
- Solutions:
  - ✓ At the receiver: advertise a non-null window only if an entire MSS would fit
  - ✓ At the sender: delay sending small packets.

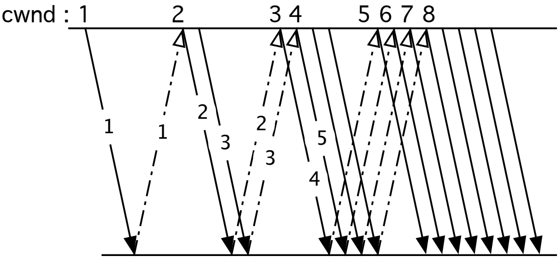
TCP congestion control

- Avoids receiver buffer saturation (WIN)
- Adapt sending rate to network capacity

  Segment losses signal that a saturation occured
- Fair resource sharing
- 2 (3) states:
  1. Slow start
  2. Congestion avoidance
  3. (Fast recovery)

slow start

- Double *cwnd* each RTT
- Increase *cwnd* by one MSS upon ACK reception



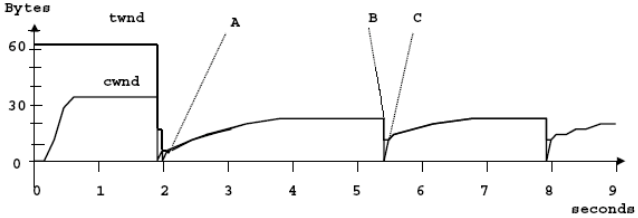
Congestion avoidance

- When *cwnd* reaches *ssthresh* (twnd on figure below), linear increase of *cwnd* in time (instead of exponential)
- The update still takes place upon reception of an ACK:

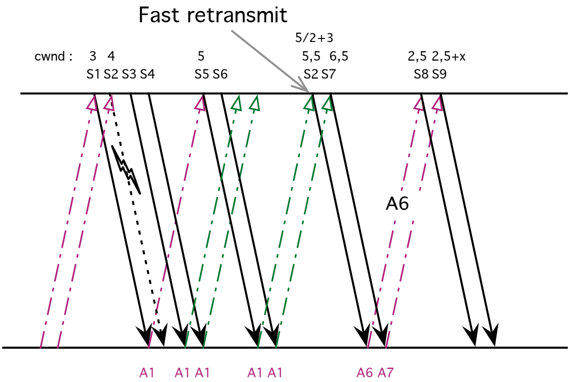
$$cwnd \leftarrow cwnd + \frac{MSS \times MSS}{cwnd}$$

The sender sends $\approx \frac{cwnd}{MSS}$ segments per RTT, so that the *cwnd* will grow by $\approx$ 1 MSS/RTT

Tahoe example: (slow start after each loss; $ssthresh = \frac{1}{2} inflight$)



TCP Reno – Fast recovery



- No slow start phase!
- The window is halved + 3 MSS to account for the received dup acks; For each new duplicate: `cwnd`+=MSS (window inflate)
- New Reno: deals better with multiple losses in same congestion period…

Why is TCP fair??

Two connections share the same buffer,

their throughputs are x1 and x2

1. Additive increase
2. Multiplicative decrease
3. Additive increase
4. Multiplicative decrease