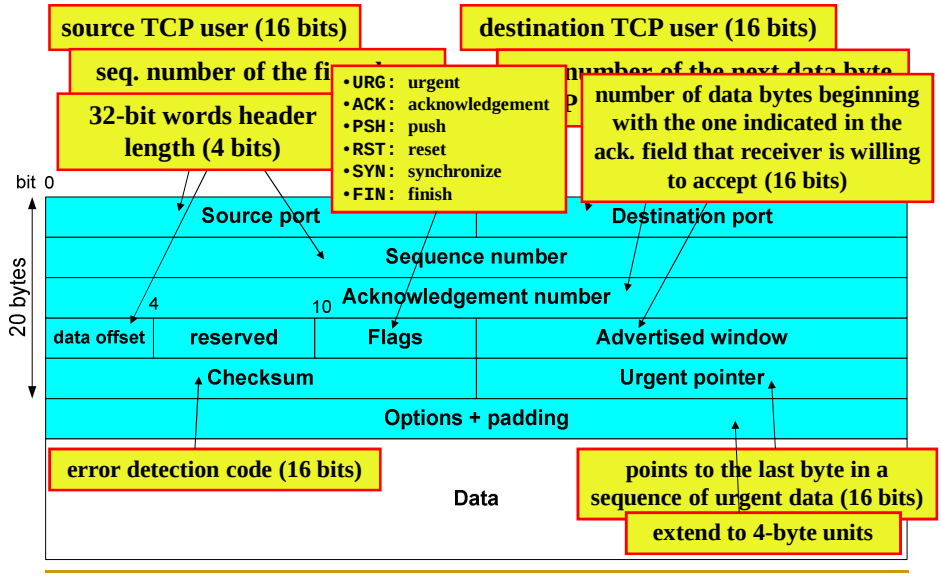
Transmission Control Protocol (TCP)

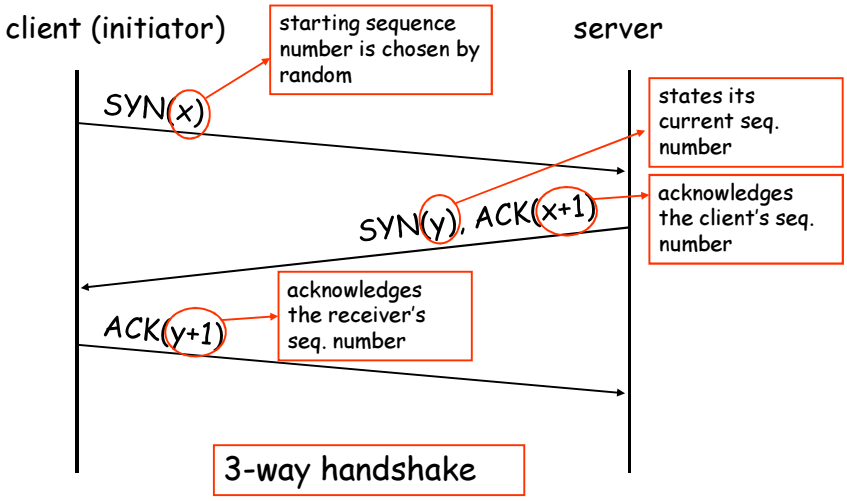
Sonkoly Balázs
sonkoly@tmit.bme.hu

2015.11.10.

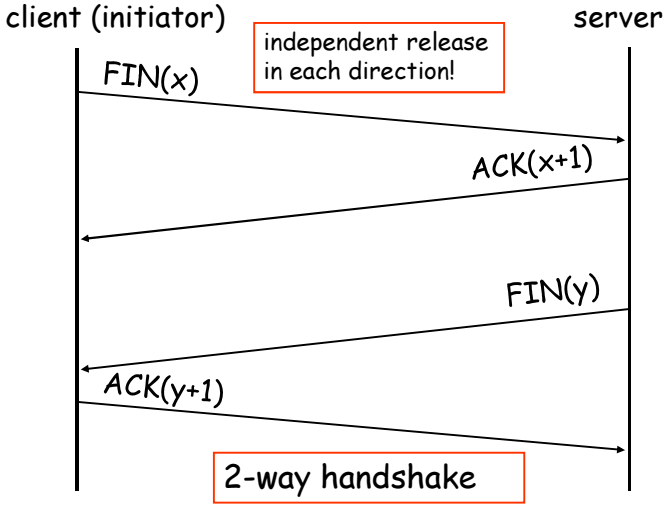
TCP header format



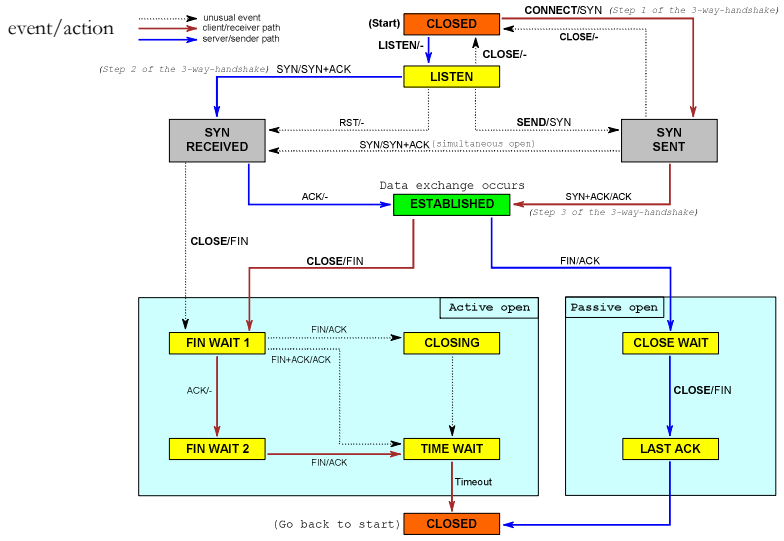
Connection setup



Connection release

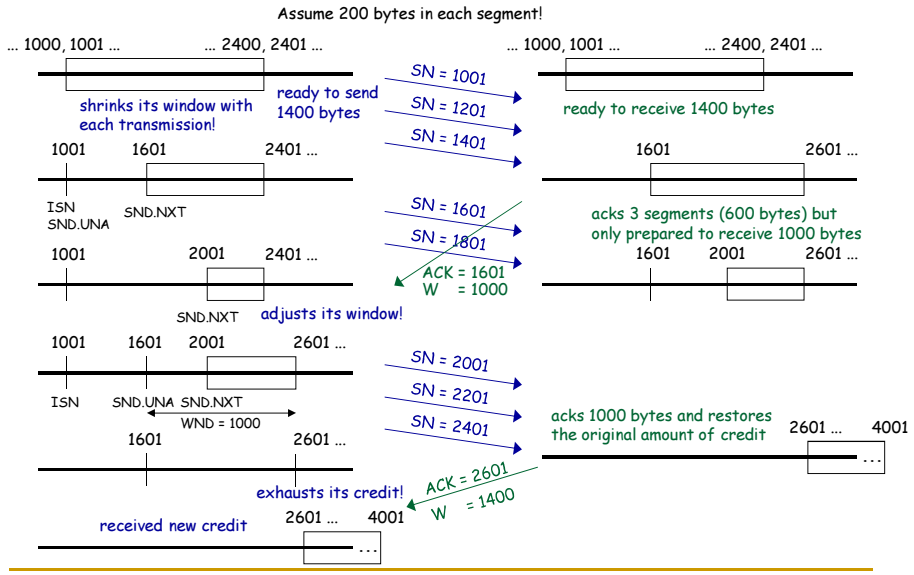


State transition diagram



Source: http://en.wikipedia.org/wiki/Image:Tcp_state_diagram.svg

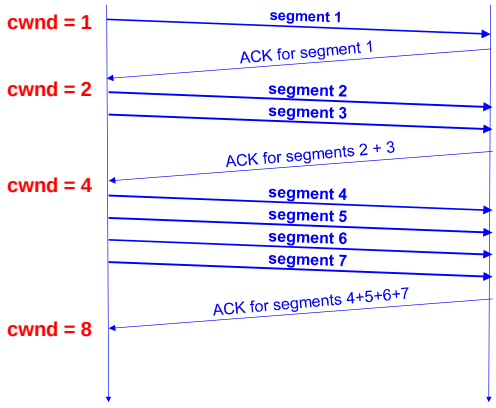
TCP flow control – example



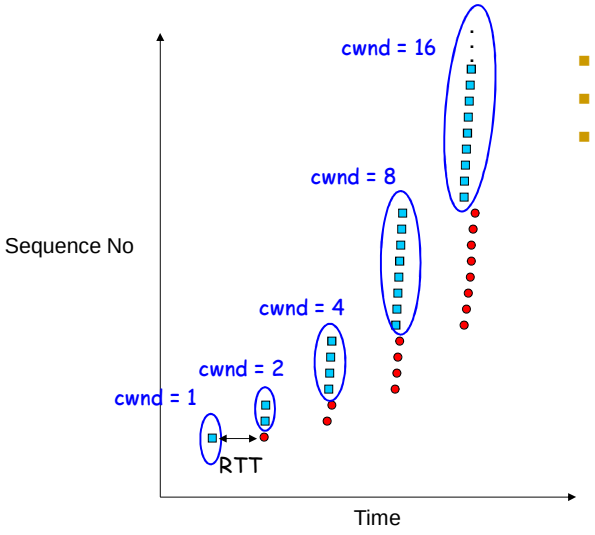
Slow start

- Determine available capacity at first
- TCP transmission is constrained
  - **`awnd = min (adwnd, cwnd)`**
    - allowed window (in segments)
    - advertised window
      - set by receiver
      - unused credit + granted in the most recent ACK
    - congestion window
      - set by sender
- Algorithm
  - set cwnd = 1
  - cwnd++ for each received ACK (~ doubled in one RTT)
  - indication of loss
    - timeout
    - receipt of duplicate ACKs
  - end of slow start
    - loss OR
    - **`cwnd`** exceeds a threshold (**`ssthresh`**)
- Properties
  - exponential growth (not very slow!)
  - but slower growth compared to burst arrival

Slow start – example



Slow start – sequence plot



- data segments
- ACKs
- cwnd doubles every round-trip time

Congestion avoidance

- Easy to drive the network in saturation
- but hard for the network to recover
- Slow start is too aggressive
- Solution: slow start + linear growth in cwnd
- Initialization
  - `cwnd = 1`
  - `ssthresh` = (e.g.) 65,535 bytes (OR arbitrarily high – RFC 2581)
- After timeout
  - `ssthresh = cwnd / 2`
  - `cwnd = 1` → slow start until `cwnd == ssthresh`
  - for `cwnd > ssthresh`
    - increase cwnd by one for each RTT (***Additive Increase***)
    - in practice:

$$\texttt{cwnd} = \texttt{cwnd} + 1 \quad \Leftarrow \text{for each RTT}$$

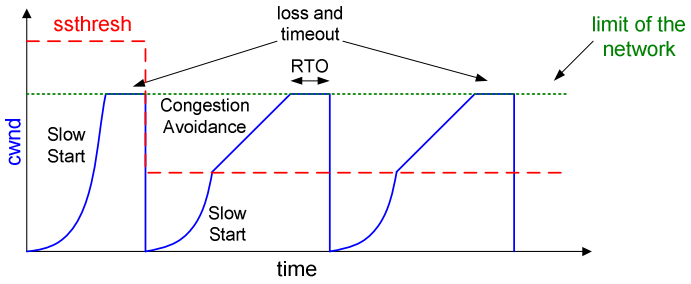
in segments:

$$\texttt{cwnd} = \texttt{cwnd} + \frac{1}{\texttt{cwnd}} \quad \Leftarrow \text{for each ACK}$$

in bytes:

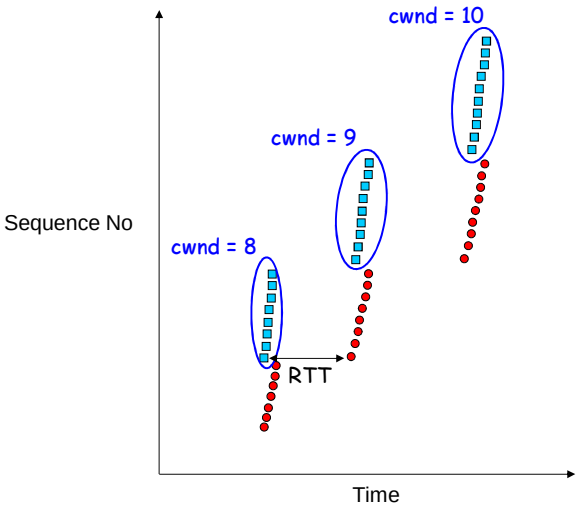
$$W = W + \frac{MSS}{\texttt{cwnd}} = W + \frac{MSS^2}{W}$$

Congestion avoidance



- Goals (RFC 1122)
  - keep cwnd around optimal size as much as possible
  - Slow start
    - increase cwnd rapidly to reach maximum safety transfer rate as fast as possible
    - max. safety: half of the rate that caused packet loss (conservative!)
  - Congestion avoidance
    - increase cwnd slowly to avoid packet losses as long as possible

Congestion avoidance – sequence plot

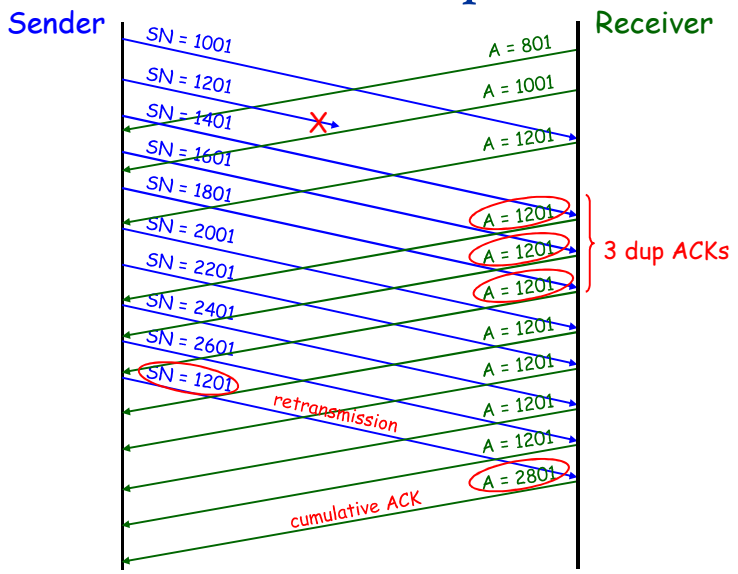


- data segments
- ACKs
- cwnd is increased by 1 for each RTT

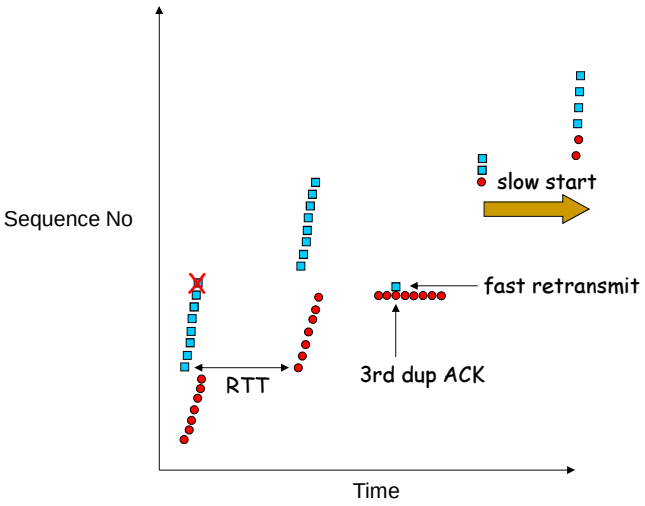
Fast retransmit

- After a segment lost TCP may be slow to retransmit
- if this is the only missing segment
  - it delays the whole flow transmission
  - receiver has to wait for the missing segment
- Solution: retransmit packet without waiting for RTO!
- receiver
  - if receives a segment out of order → ACK for the last inordered segment that was received
  - continues repeat this ACK until missing segment arrives
- source
  - when receives a duplicate ACK it means
    1. the segment following the ACKed segment was delayed
       - no action needed
    2. segment was lost
       - retransmission needed
    - test
      - wait for the next ACK
      - 3 dup ACKs → retransmit the segment
- TCP Tahoe (implemented in 4.3 BSD Tahoe, Net/1, ~1988)

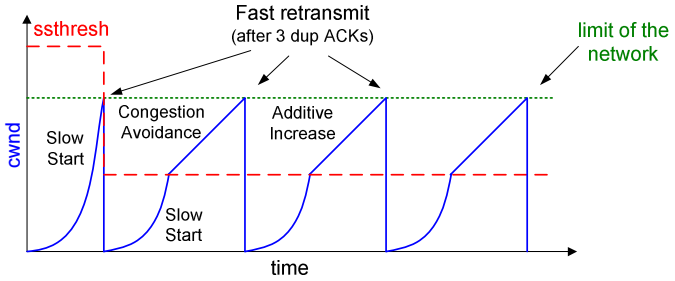
Fast retransmit – example



Fast retransmit (TCP Tahoe) – sequence plot



Fast retransmit – TCP Tahoe

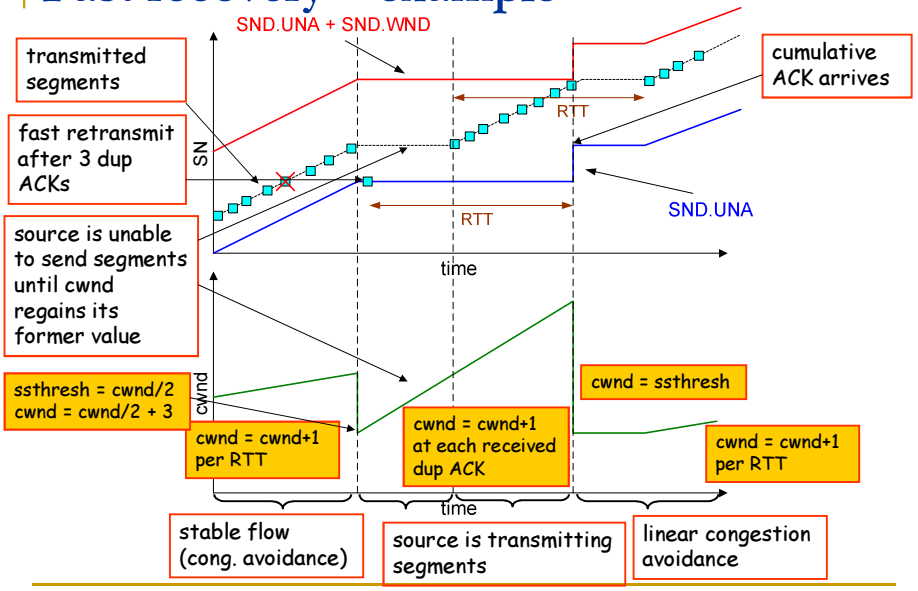


- TCP Tahoe
  - slow start and congestion avoidance phases
  - + fast retransmit
- Problem
  - after fast retransmit we know that congestion occured
  - BUT make slow start is too conservative
  - we know that consecutive packets have been received
  - Tahoe is very sensitive to packet loss (1% loss rate may cause 50-75% decrease in throughput!)
- Solution: two type of congestion
  - RTO expires → serious congestion
  - 3 dup ACKs → no serious congestion (at least 3 packets could arrive)

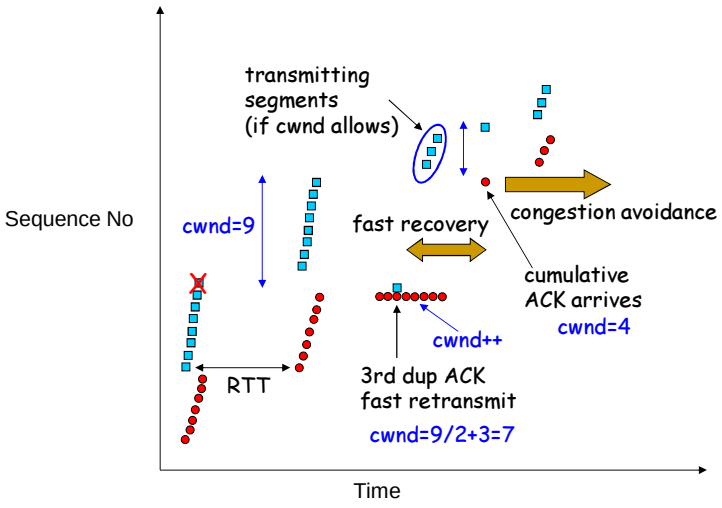
Fast recovery

- Goal: avoid slow start!
- after receiving the third dup ACK
  - `ssthresh = cwnd / 2`
  - retransmit the segment (fast retransmit)
  - `cwnd = ssthresh + 3` (***inflating*** the window)
  - if additional dup ACKs arrives
    - `cwnd = cwnd + 1` (***inflating*** the window)
    - transmit a segment if possible
  - if the next ACK arrives (for new segment)
    - `cwnd = ssthresh` (***deflating*** the window)
- Inflating the window
  - dup ACK means → one packet arrived and cached at receiver
  - one new packet can be sent

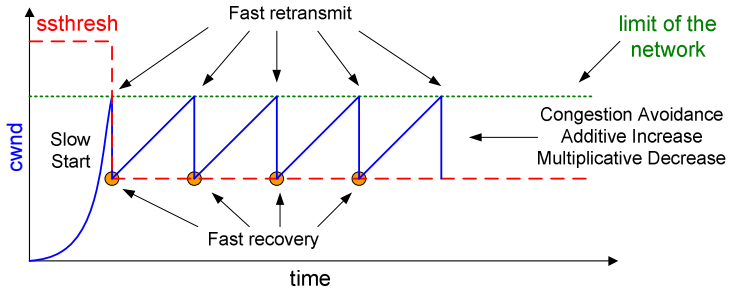
Fast recovery – example



Fast recovery (TCP Reno) – sequence plot

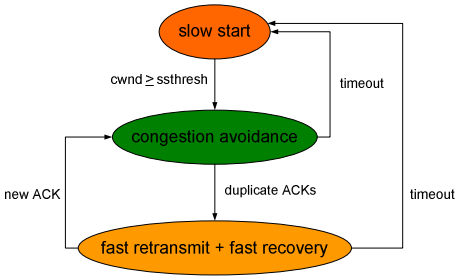


Fast recovery – TCP Reno



- TCP Reno
  - implemented in 4.3 BSD Reno, Net/2, ~1990
  - Slow start
  - Congestion avoidance: AIMD (Additive Increase Multiplicative Decrease)
  - Fast retransmit
  - Fast recovery
- Problem
  - multiple losses from a single window??

Summary of the algorithm (TCP Reno)



- Initialization
  - cwnd = 1 (segment)
  - ssthresh = 65,535 bytes
- TCP sender sends segment: effwnd
  - maxwnd = min(cwnd, adwnd)
  - effwnd = maxwnd – (lastbytesent – lastbyteacked)
- Congestion avoidance
  - cwnd = cwnd + 1 for each RTT
  - cwnd = cwnd + 1/cwnd for each ACK
  - if congestion:
    - ssthresh = max(2, min(cwnd, adwnd)/2)
- Slow start
  - cwnd = 1
  - cwnd = cwnd + 1 for each ACK
  - if cwnd > ssthresh → congestion avoidance
- Fast recovery
  - cwnd = ssthresh + 3
  - if additional dup ACKs
    - cwnd = cwnd + 1
    - transmit segment if effwnd > 0
  - if new ACK
    - cwnd = ssthresh
    - congestion avoidance