Advanced Computer Networks

Congestion control in TCP

Prof. Andrzej Duda
duda@imag.fr

http://duda.imag.fr

Contents

- Principles
- TCP congestion control states
  - Slow Start
  - Congestion Avoidance
  - Fast Recovery
- TCP fairness

TCP and Congestion Control

- TCP is used to avoid congestion in the Internet
  - a TCP source adjusts its sending window to the congestion state of the network
  - this avoids congestion collapse and ensures some fairness
- TCP sources interpret losses as a negative feedback
  - used to reduce the sending rate
- Window-based control
  - modulate window not rate

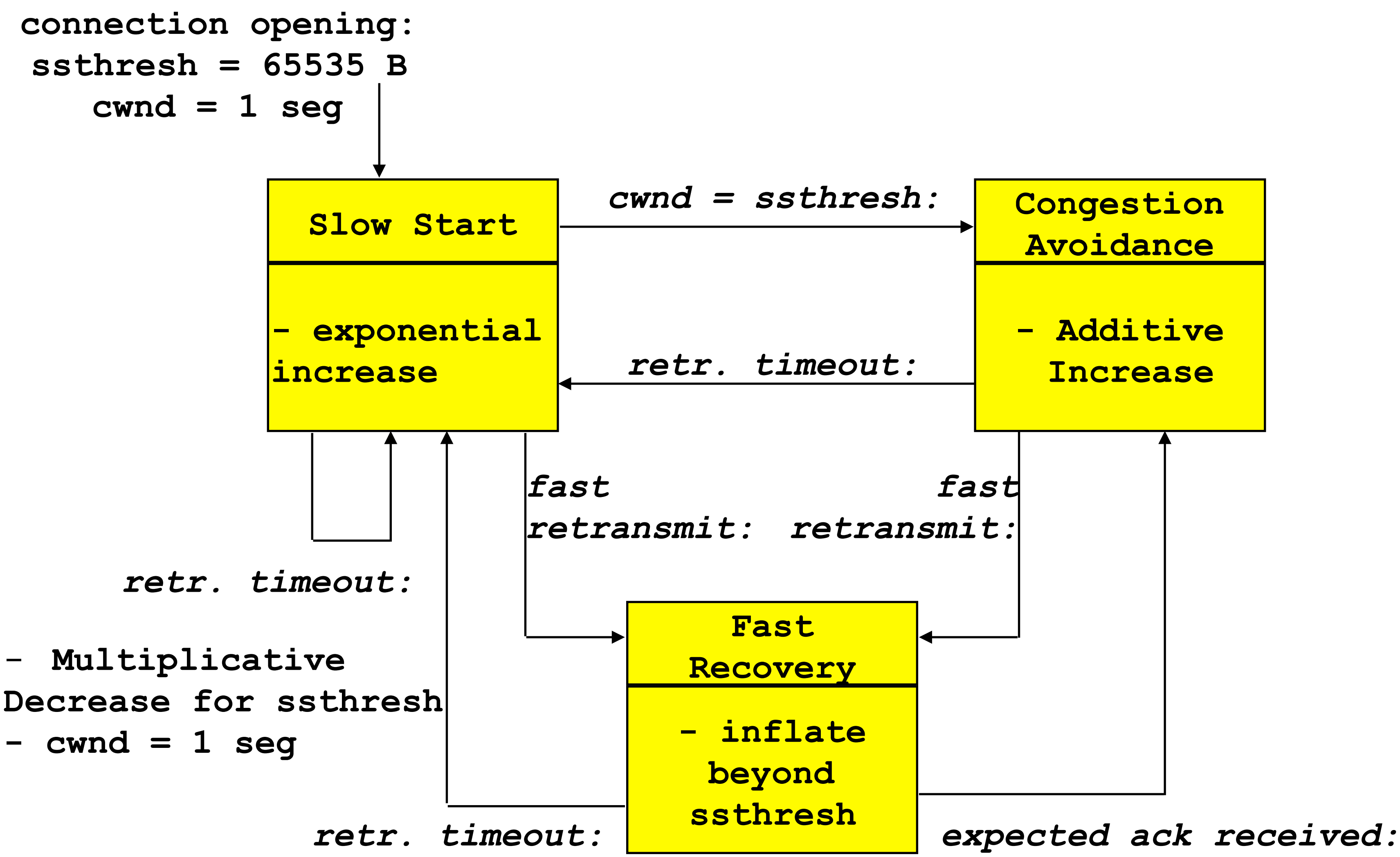
Sending window

- Sending window - number of non ACKed bytes
  - W = min (**`cwnd`**, **`OfferedWindow`**)
  - **`cwnd`**
    - congestion window - maintained by TCP source
  - **`OfferedWindow`**
    - announced by destination in TCP header
    - flow control
    - reflects free buffer space
- Same mechanism used for flow control and for congestion control

Congestion control states

- TCP connection may be in three states with respect to congestion
  - **Slow Start** (Démarrage Lent) after loss detected by retransmission timer
  - **Fast Recovery** (Récupération Rapide) after loss detected by Fast Retransmit (three duplicated ACKs)
  - **Congestion Avoidance** (Évitement de Congestion) otherwise
- Terminology
  - *ssthresh* – target window, same as *ssthresh*
  - *flightSize* - the amount of data that has been sent but not yet acknowledged, roughly *cwnd*

Congestion Control States



Slow Start

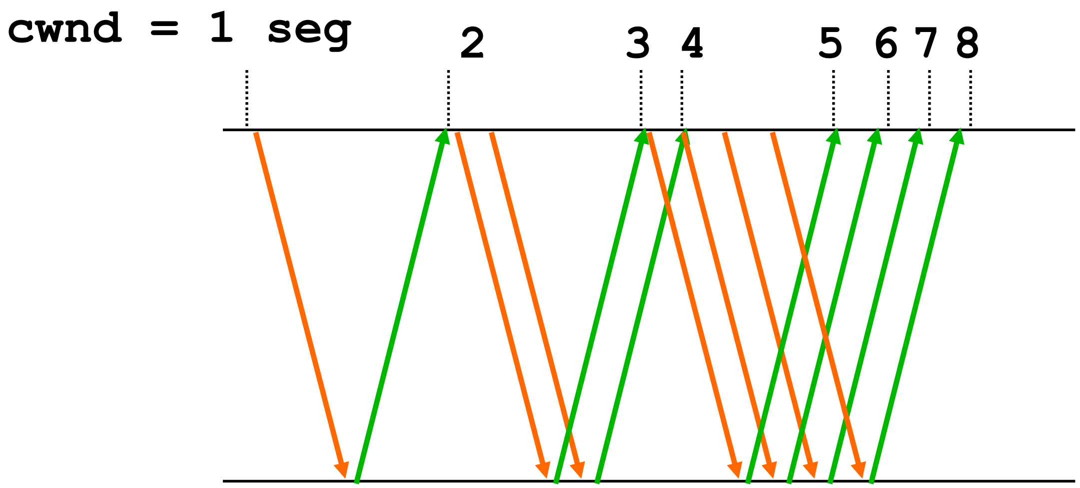
```
/ * exponential increase for cwnd */

  non dupl. ack received during slow start ->

          cwnd = cwnd + MSS (in bytes)
  if cwnd = ssthresh then transition to
  congestion avoidance
```

- Window increases rapidly up to the value of **`ssthresh`**
  Not so slow, rather exponential

Slow Start

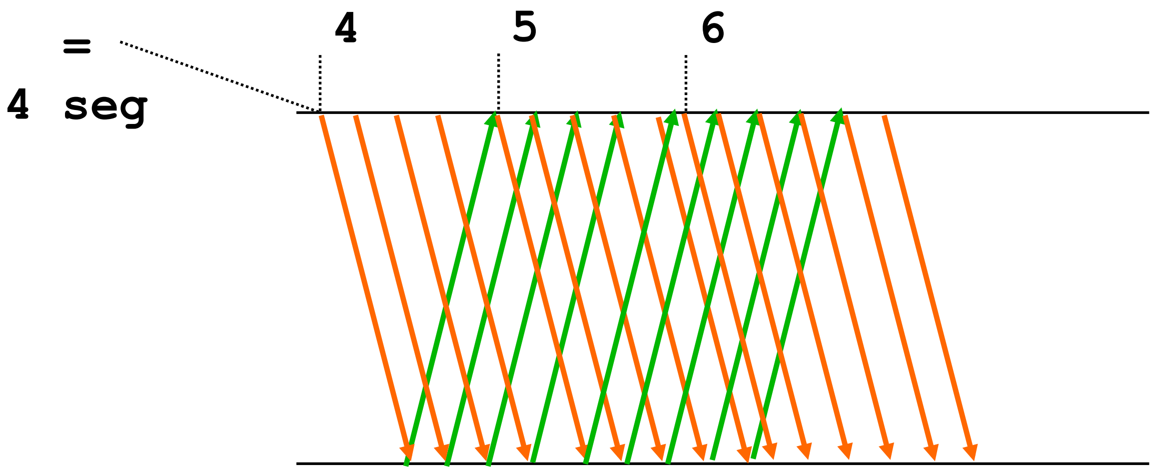


- purpose of this phase: avoid bursts of data at the beggining or after a retransmission timeout

Increase/decrease

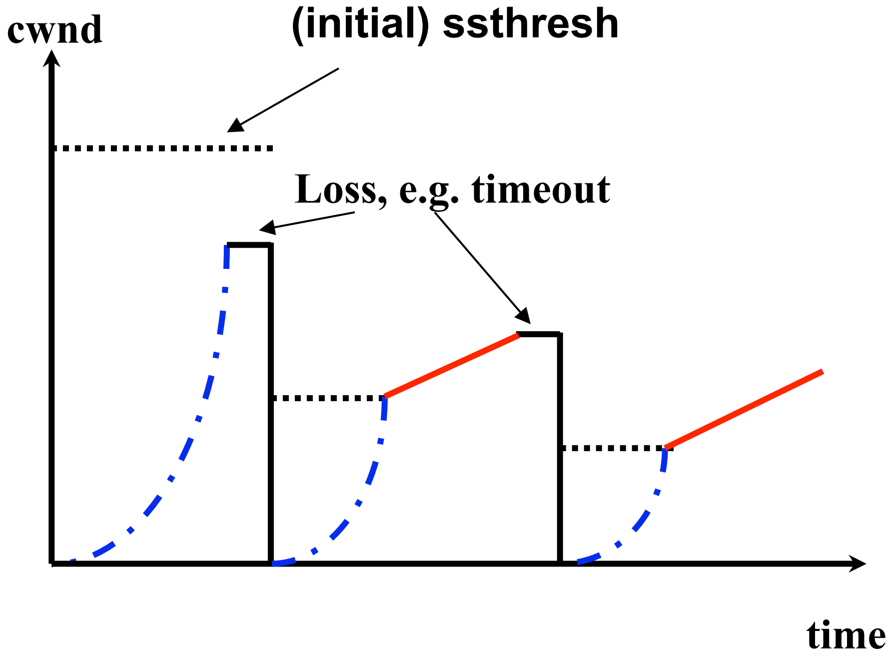
- Multiplicative decrease
  - **`ssthresh = 0.5 × flightSize`**
  - **`ssthresh = max (ssthresh, 2 × MSS)`**
  - **`cwnd = 1 MSS`**
- Additive increase
  - for each ACK
    - **`cwnd = cwnd + MSS × MSS / cwnd`**
    - **`cwnd = min (cwnd, max-size) (64KB)`**
  - **`cwnd`** is in bytes, counting in segments, this means that
    - we receive (**`cwnd/MSS`**) ACKs per RTT
    - for each ACK: **`cwnd/MSS`** $\leftarrow$ 1/W
    - for a full window: W$\leftarrow$W + 1 MSS

`cwnd` Additive Increase



- during one round trip + interval between packets: increase by 1 MSS (linear increase)

Example

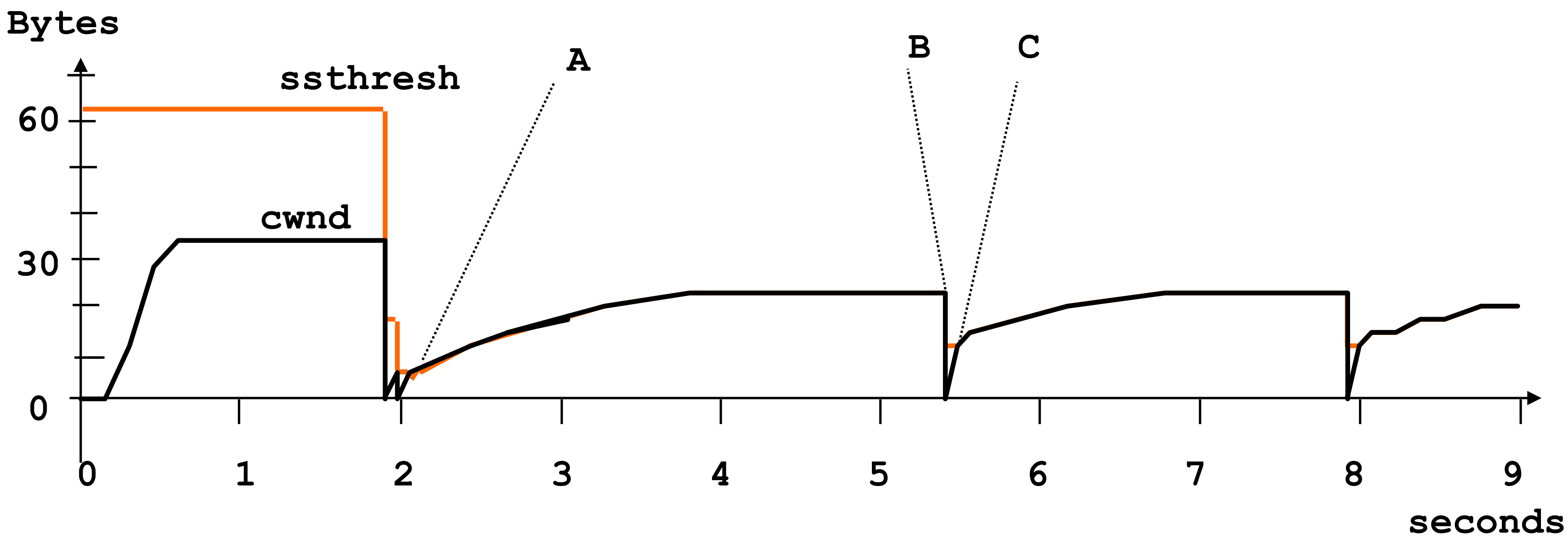


slow start – in bleu

congestion avoidance – in red

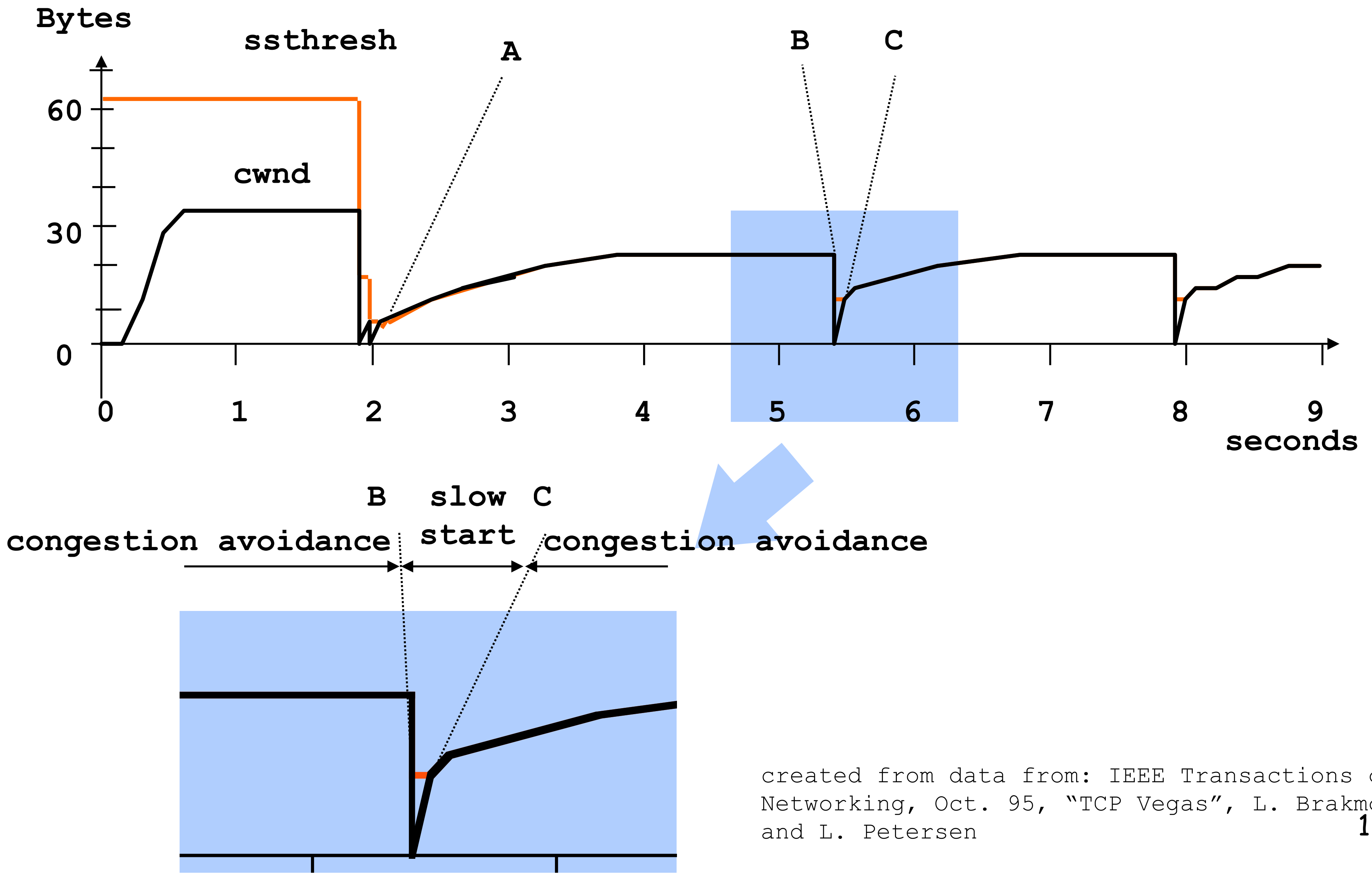
`flightSize = cwnd`

Example



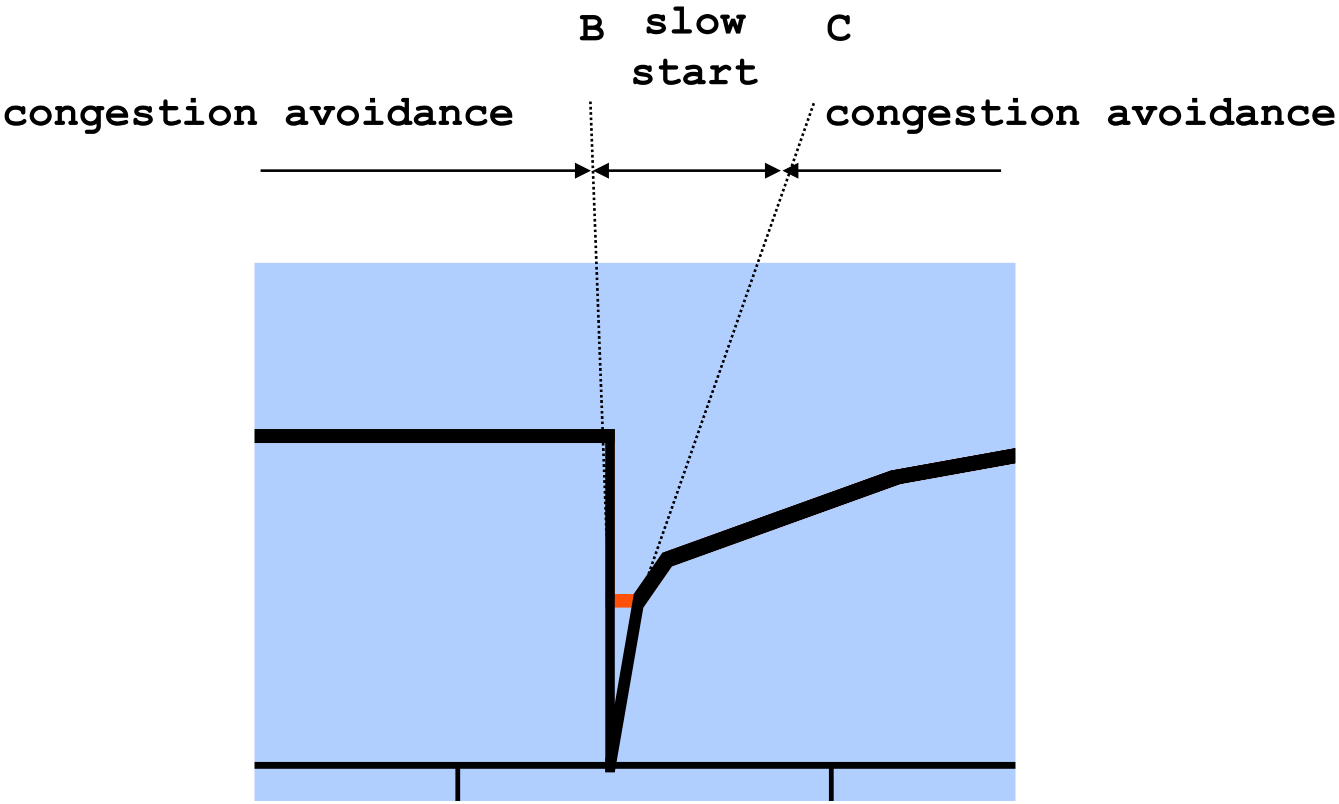
created from data from: IEEE Transactions on Networking, Oct. 95, "TCP Vegas", L. Brakmo and L. Petersen

Example



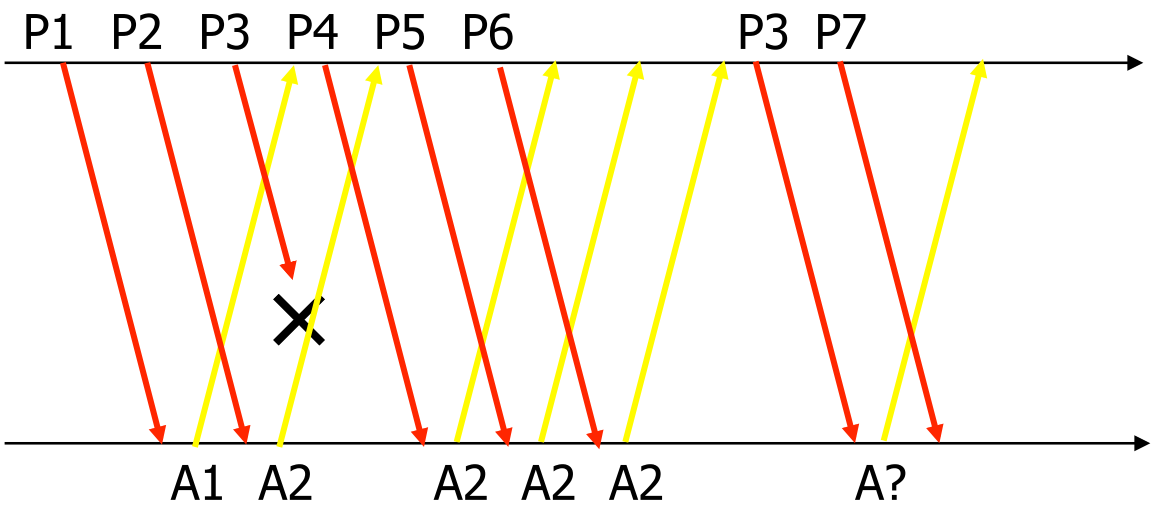
created from data from: IEEE Transactions on Networking, Oct. 95, "TCP Vegas", L. Brakmo and L. Petersen

Slow Start and Congestion Avoidance



created from data from: IEEE Transactions on Networking, Oct. 95, “TCP Vegas”, L. Brakmo and L. Petersen

Fast Retransmit



- Fast Retransmit
  - retransmit timer can be large
  - optimize retransmissions similarly to Selective Retransmit
  - if sender receives 3 duplicated ACKs, retransmit missing segment

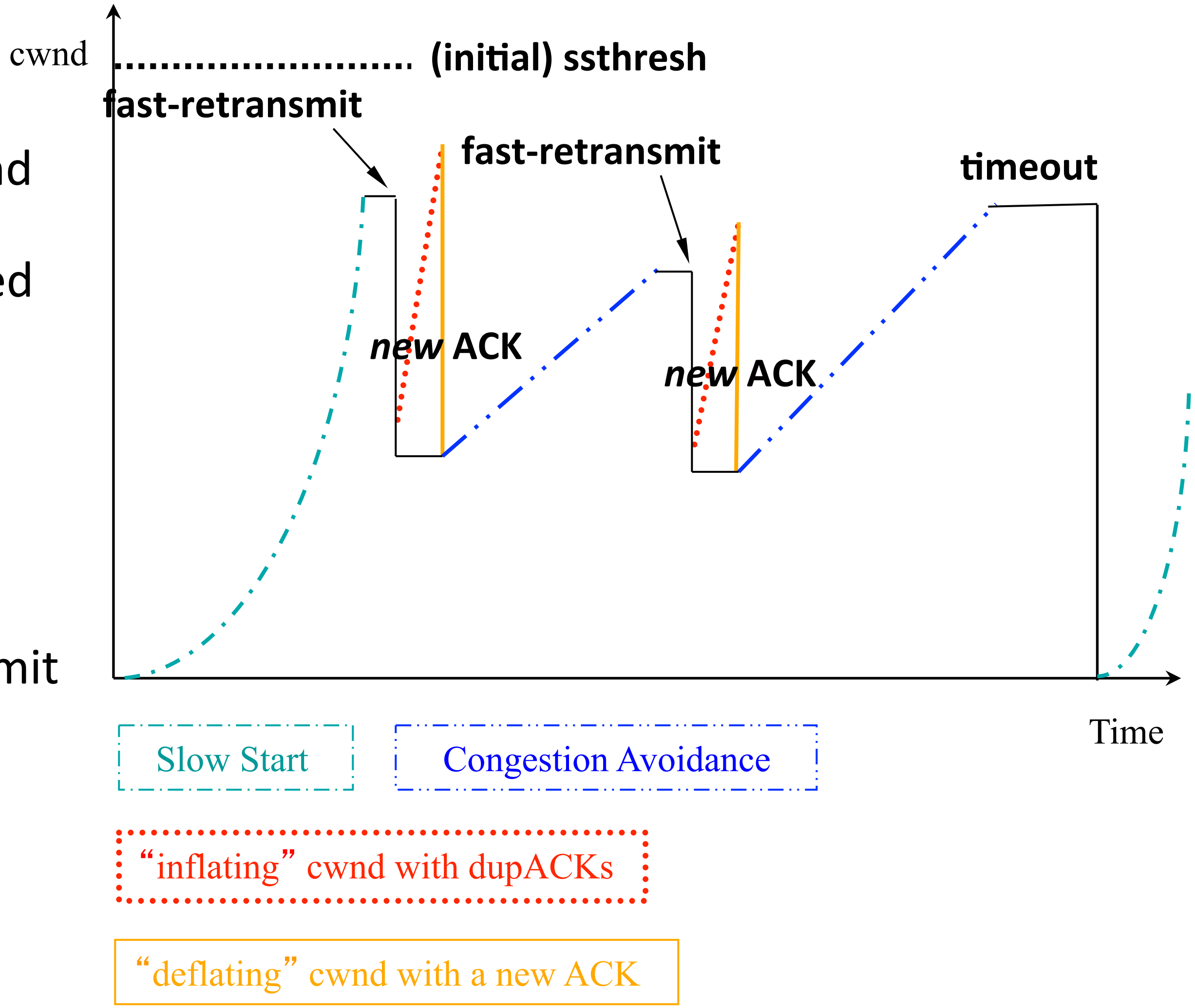
Fast Recovery

Concept:

- After fast retransmit, reduce cwnd by half, and continue sending segments at this reduced level.

Problems:

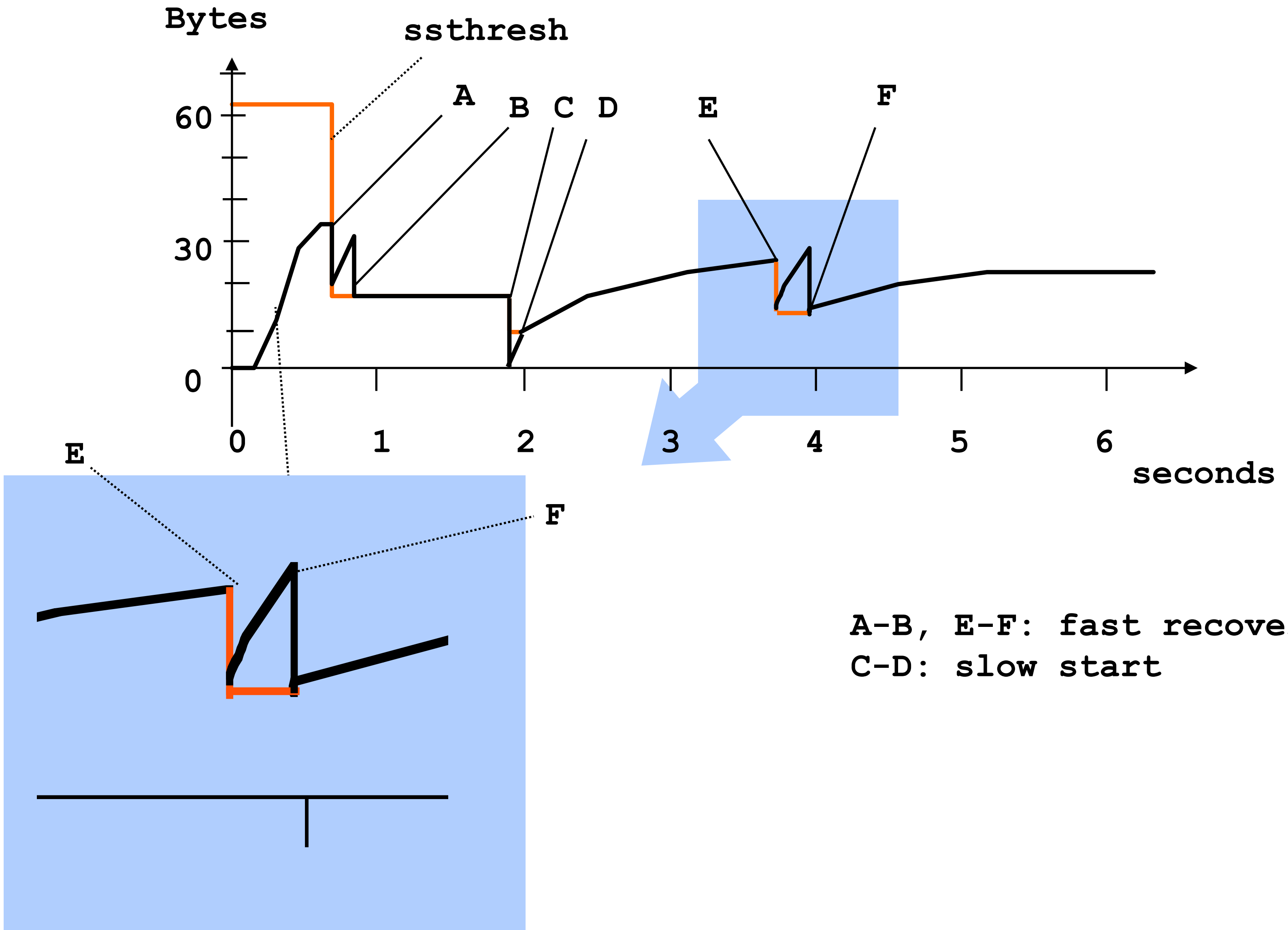
- Sender has too many outstanding segments.
- How does sender transmit packets on a dupACK? Need to use a “trick” - inflate cwnd.



Fast Recovery

- Multiplicative decrease
  - **`ssthresh = 0.5 × flightSize`**
  - **`ssthresh = max (ssthresh, 2 × MSS)`**
- Fast Recovery
  - **`cwnd = ssthresh + 3 × MSS (inflate)`**
  - **`cwnd = min (cwnd, 64K)`**
  - **`retransmit the missing segment (n)`**
- For each duplicated ACK
  - **`cwnd = cwnd + MSS (keep inflating)`**
  - **`cwnd = min (cwnd, 64K)`**
  - **`keep sending segments in the current window`**
- For partial ACK
  - **`retransmit the first unACKed segment`**
  - **`cwnd = cwnd – ACKed + MSS (deflate/inflate)`**

Fast Recovery Example



A-B, E-F: fast recovery
C-D: slow start

TCP Loss - Throughput formulae

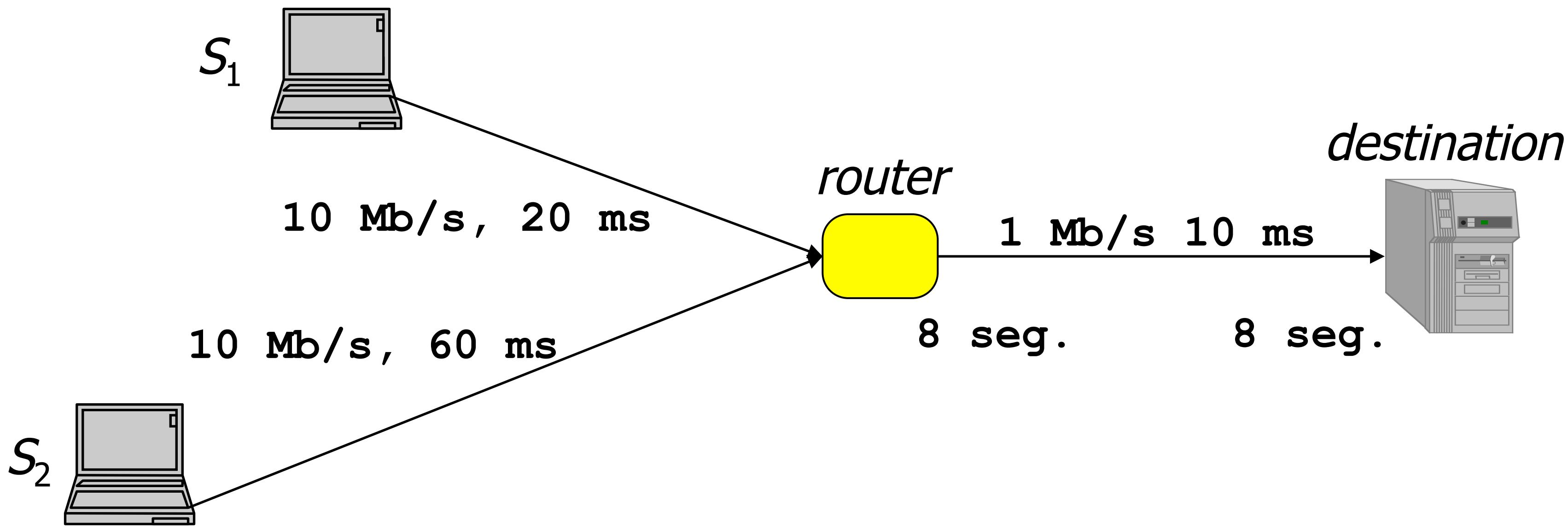
$$\theta = \frac{L}{T}\frac{C}{\sqrt{q}}$$

- TCP connection with
  - RTT *T*
  - segment size *L*
  - average packet loss ratio *q*
  - constant *C* = 1.22
- Transmission time negligible compared to RTT, losses are rare, time spent in Slow Start and Fast Recovery negligible

Fairness of the TCP

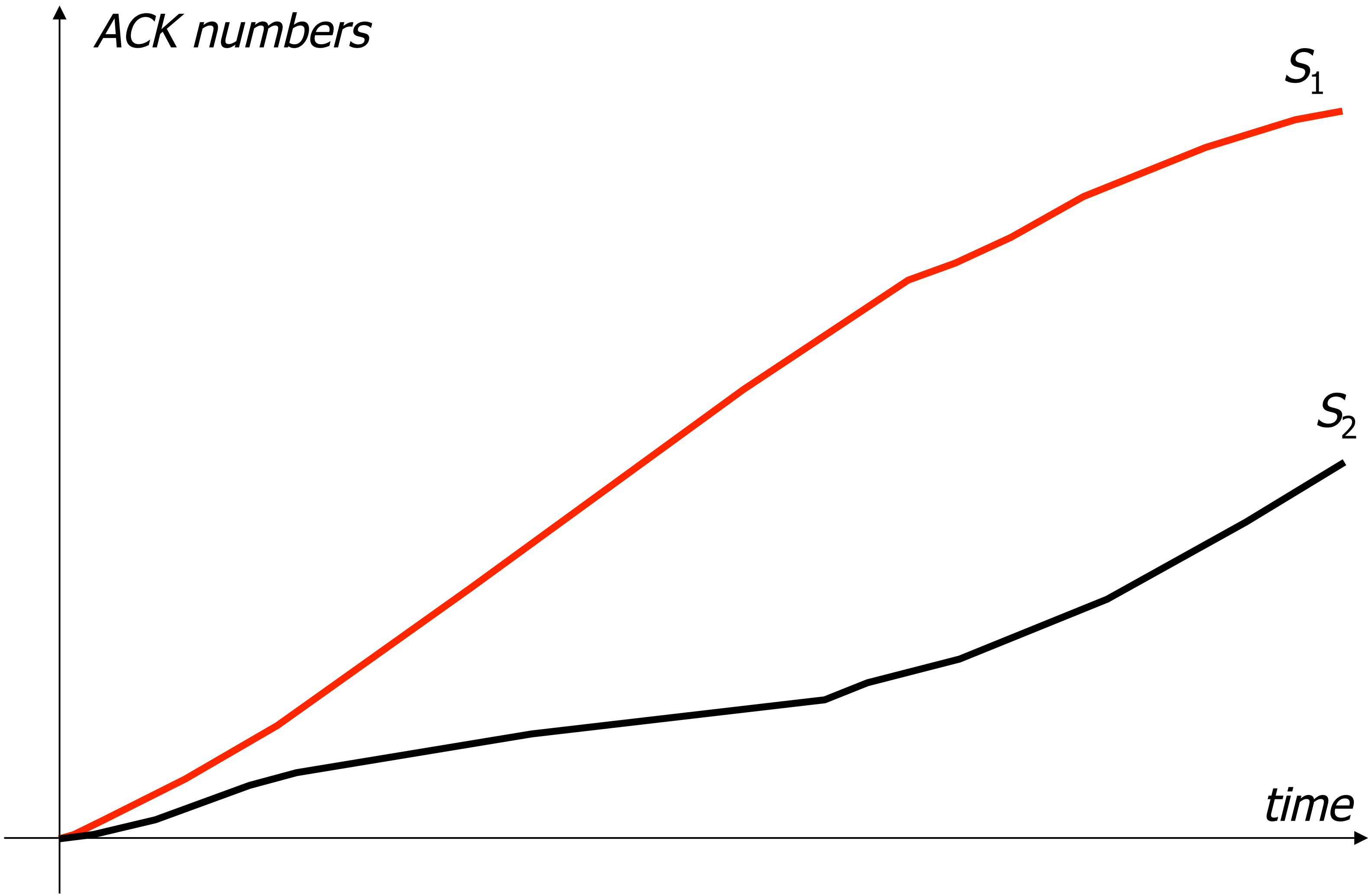
- TCP differs from the pure AI-MD principle
  - window based control, not rate based
  - increase in rate is not strictly additive - window is increased by 1/W for each ACK
- Like with **proportional fairness**, the adaptation algorithm gives less to sources using many resources
  - not the number of links, but RTT
- TCP fairness: negative bias of long round trip times

Fairness of the TCP



- Example network with two TCP sources
  - link capacity, delay
  - limited queues on the link (8 segments)
- NS simulation

Throughput in time



Facts to remember

- TCP performs congestion control in end-systems
  - sender increases its sending window until loss occurs, then decreases
    - additive increase (no loss)
    - multiplicative decrease (loss)
- TCP states
  - slow start, congestion avoidance, fast recovery
- Negative bias towards long round trip times
- UDP applications should behave like TCP with the same loss rate