

# Advanced Computer Networks

## Congestion control in TCP

Prof. Andrzej Duda  
duda@imag.fr

<http://duda.imag.fr>

1

## Contents

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

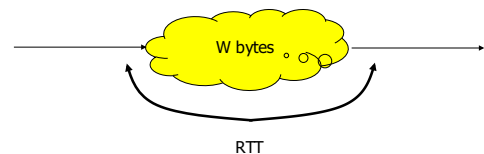
2

## 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
- UDP sources are a problem for the Internet
  - use for long lived sessions (ex: RealAudio) is a threat: congestion collapse
  - UDP sources should imitate TCP : "TCP friendly"

3

## Sending window



- W - the number of non ACKed bytes
  - throughput =  $W/RTT$  (Little's formulae)
- If congestion
  - RTT increases, automatic reduction of the source rate
  - additional control: decrease W

4

## Sending window

- Sending window - number of non ACKed bytes
  - $W = \min(\text{cwnd}, \text{OfferedWindow})$
  - **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

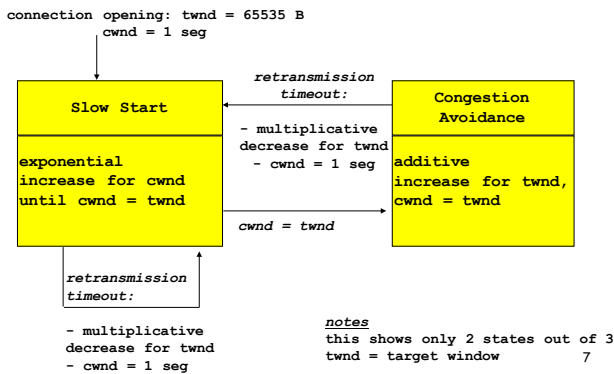
5

## 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

6

# Slow Start and Congestion Avoidance



# Slow Start

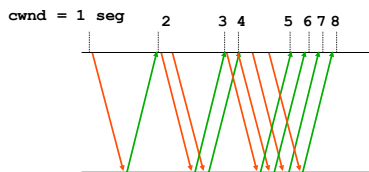
`/* exponential increase for cwnd */`

```

non dupl. ack received during slow start ->
    cwnd = cwnd + MSS (in bytes)
if cwnd = twnd then transition to
congestion avoidance
    
```

- Window increases rapidly up to the stored value of  $twnd$  (this stored value is also called  $ssthresh$  in the literature)
- Not so slow, rather exponential

# Slow Start

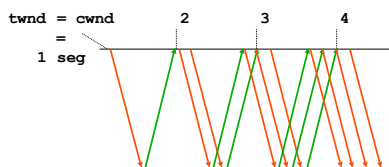


- purpose of this phase: avoid bursts of data at the beginning or after a retransmission timeout

# Increase/decrease

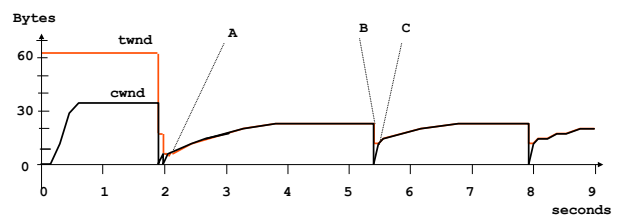
- Multiplicative decrease
  - $twnd = 0.5 \times cwnd$
  - $twnd = \max(twnd, 2 \times MSS)$
- Additive increase
  - for each ACK
    - $twnd = twnd + MSS \times MSS / twnd$
    - $twnd = \min(twnd, \text{max-size})$  (64KB)
  - $twnd$  is in bytes, counting in segments this means that
    - for each ACK:  $twnd/MSS \leftarrow 1/W$
    - for a full window:  $W \leftarrow W + 1$

# twnd Additive Increase

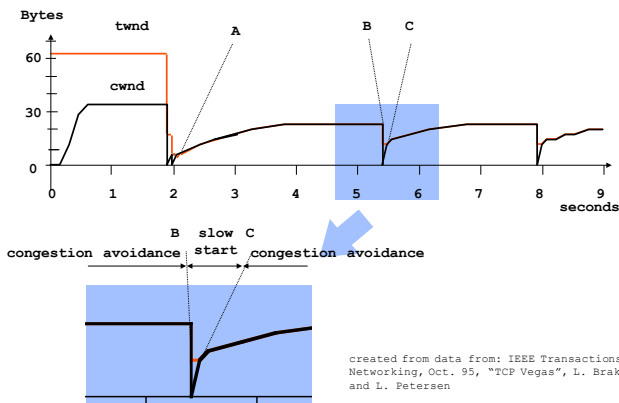


- during one round trip + interval between packets: increase by 1 MSS (linear increase)

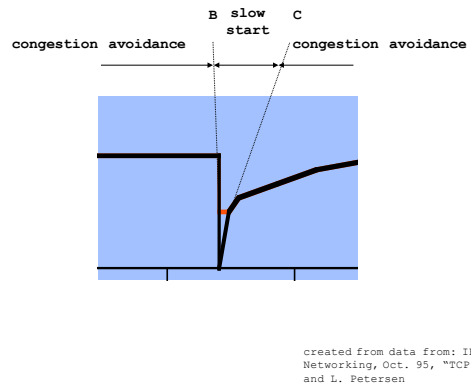
# Example



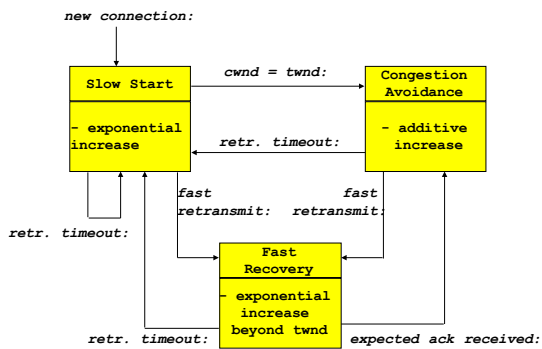
## Example



## Slow Start and Congestion Avoidance

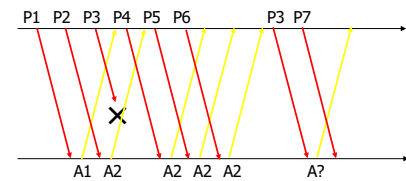


## Congestion Control States



15

## Fast Retransmit



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

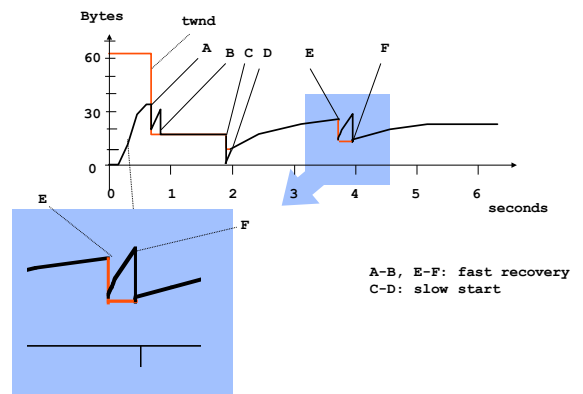
16

## Fast Recovery

- Multiplicative decrease
  - $twnd = 0.5 \times cwnd$
  - $twnd = \max(twnd, 2 \times MSS)$
- Fast Recovery
  - $cwnd = twnd + 3 \times MSS$  (exp. increase)
  - $cwnd = \min(cwnd, 64K)$
  - retransmission of the missing segment (n)
- For each duplicated ACK
  - $cwnd = cwnd + MSS$  (exp. increase)
  - $cwnd = \min(cwnd, 64K)$
  - keep sending segments in the current window

17

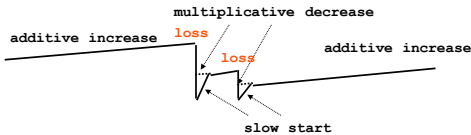
## Fast Recovery Example



18

## TCP Congestion Control

- TCP performs congestion control in end-systems
- Principle
  - sender increases its sending window until loss occurs, then decreases
- Target window
  - additive increase (no loss)
  - multiplicative decrease (loss)



19

## TCP Congestion Control

- 3 phases
  - **slow start**
    - starts with 1, exponential increase up to  $twnd$
  - **congestion avoidance**
    - additive increase until loss or max window
  - **fast recovery**
    - fast retransmission of one segment
- Slow start entered at setup or after retransmission timeout
- Fast recovery entered at fast retransmit
- Congestion avoidance entered when  $cwnd = twnd$

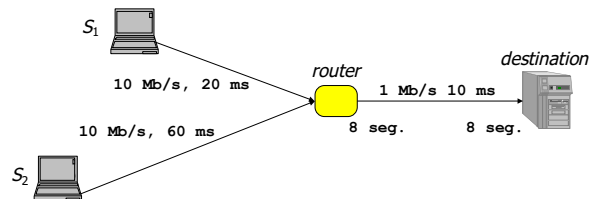
20

## 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
- Adaptation algorithm of TCP results in a negative bias against long round trip times
- Like with proportional fairness, the adaptation algorithm gives less to sources using many resources
  - not the number of links, but RTT

21

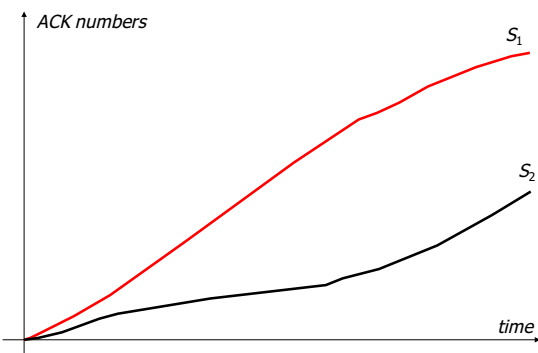
## Fairness of the TCP



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

22

## Throughput in time



23

## TCP Friendly Applications

- All TCP/IP applications that generate long lived flows should mimic the behavior of a TCP source
  - RTP/UDP flow sending video/audio data
- Adaptive algorithm
  - application determines the sending rate
  - feedback - amount of lost packets, loss ratio
  - sending rate = rate of a TCP flow experiencing the same loss ratio

24

## 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

25

## 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

26