DIPARTIMENTO DI INFORMATICA
Via Salaria 113 - 00198, Roma

SAPIENZA UNIVERSITÀ DI ROMA

Chapter 3 Transport Layer

Reti di Elaboratori

Corso di Laurea in Informatica

Università degli Studi di Roma "La Sapienza"

Canale A-L

Prof.ssa Chiara Petrioli

Parte di queste slide sono state prese dal materiale associato al libro *Computer Networking: A Top Down Approach* , 5th edition.

Thanks also to Antonio Capone, Politecnico di Milano, Giuseppe Bianchi and Francesco LoPresti, Un. di Roma Tor Vergata

TCP: controllo di congestione

- Il TCP ha dei meccanismi di controllo della congestione
  - il flusso dei dati in ingresso in rete è anche regolato dalla situazione di traffico in rete
  - se il traffico in rete porta a situazioni di congestione il TCP riduce velocemente il traffico in ingresso
  - in rete non vi è nessun meccanismo per notificare esplicitamente le situazioni di congestione
  - il TCP cerca di scoprire i problemi di congestione sulla base degli eventi di perdita dei pacchetti

TCP Congestion Control

- end-end control (no network assistance)
- sender limits transmission:

  `LastByteSent-LastByteAcked`
  ≤ `CongWin`

- Roughly,

$$\boxed{\text{rate} = \frac{\text{CongWin}}{\text{RTT}} \text{ Bytes/sec}}$$

- **`CongWin`** is dynamic, function of perceived network congestion

How does sender perceive congestion?

- loss event = timeout *or* 3 duplicate acks
- TCP sender reduces rate (**`CongWin`**) after loss event

three mechanisms:

  - AIMD
  - slow start
  - conservative after timeout events

Starting a TCP transmission

- A new offered flow may suddenly overload network nodes
  - receiver window is used to avoid recv buffer overflow
  - But it may be a large value (16-64 KB)
- Idea: slow start
  - Start with small value of cwnd
  - And increase it as soon as packets get through
    - Arrival of ACKs = no packet losts = no congestion
- Initial cwnd size:
  - Just 1 MSS!
  - Recent (1998) proposals for more aggressive starts (up to 4 MSS) have been found to be dangerous

Detecting congestion and restarting

- Segment gets lost
  - Detected via RTO expiration
  - Indirectly notifies that one of the network nodes along the path has lost segment
    - Because of full queue
- Restart from cwnd=1 (slow start)
- But introduce a supplementary control: slow start threshold
    - sstresh = max(min(cwnd,window)/2,2MSS)
  - The idea is that we now KNOW that there is congestion in the network, and we need to increase our rate in a more careful manner…
  - Ssthresh defines the "congestion avoidance" region

Congestion avoidance

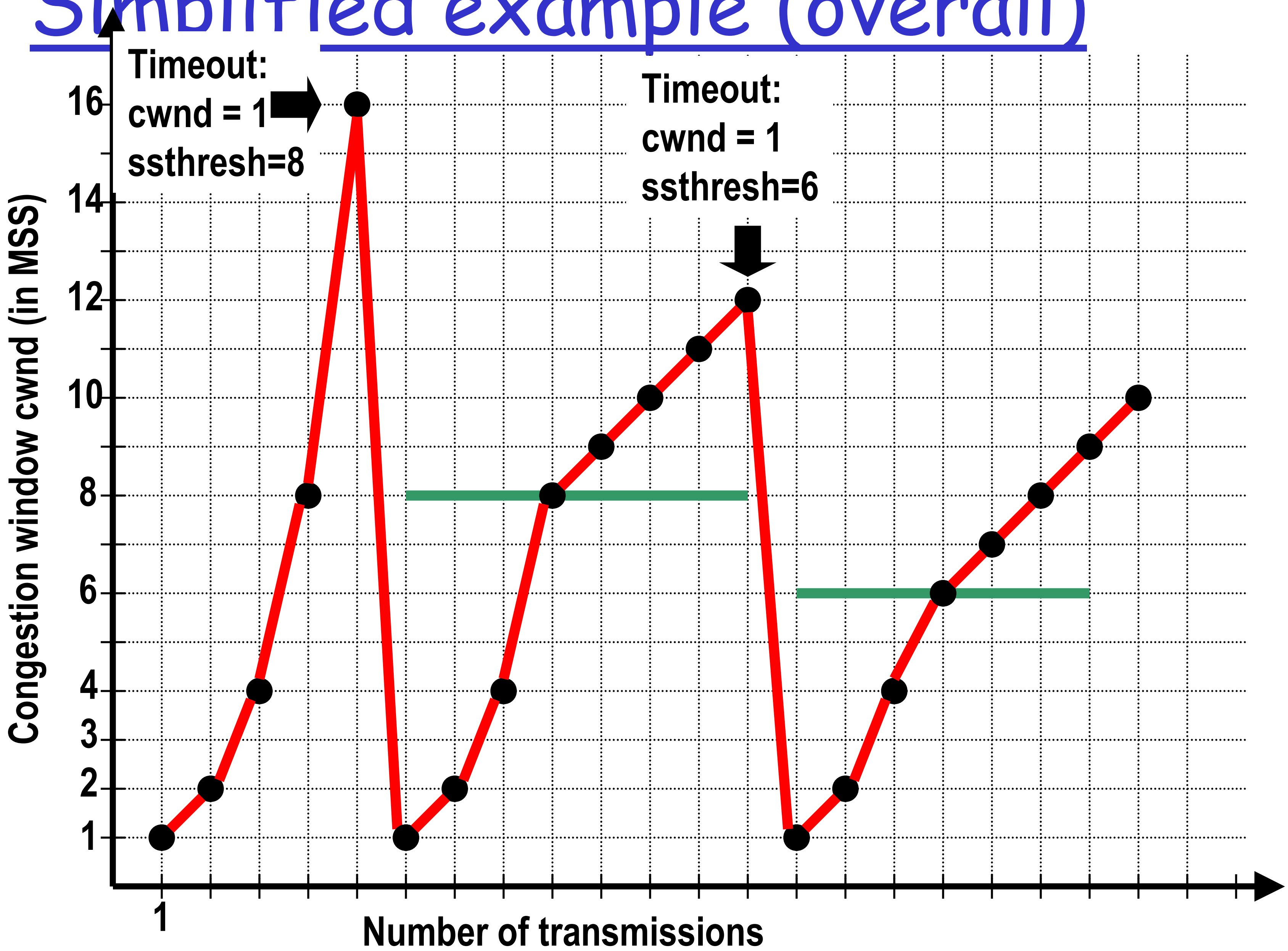
- If cwnd < ssthresh
  - Slow start region: Increase rate exponentially
- If cwnd >= ssthresh
  - Congestion avoidance region : Increase rate linearly
  - At rate 1 MSS per RTT

    Corrisponde ad un segmento per finestra

    - Practical implementation:
      cwnd += MSS*MSS/cwnd
    - Good approximation for 1 MSS per RTT
    - Alternative (exact) implementations: count!!
- Which initial ssthresh?
  – ssthresh initially set to 65535: unreachable!

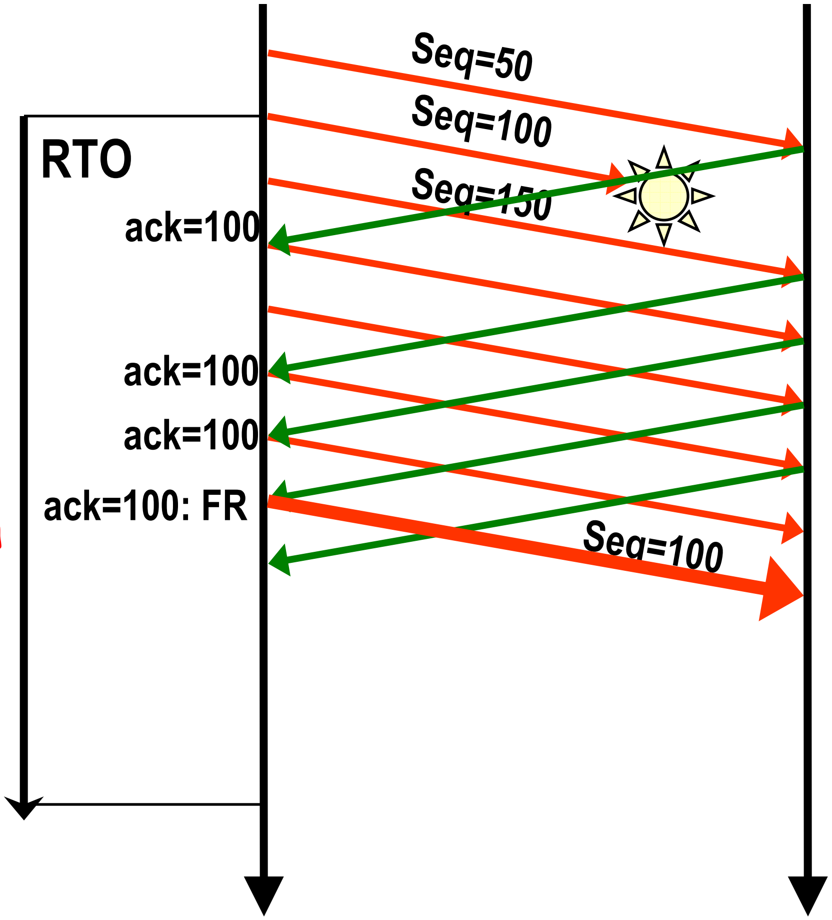
In essence, congestion avoidance is flow control imposed by sender while advertised window is flow control imposed by receiver

Simplified example (overall)

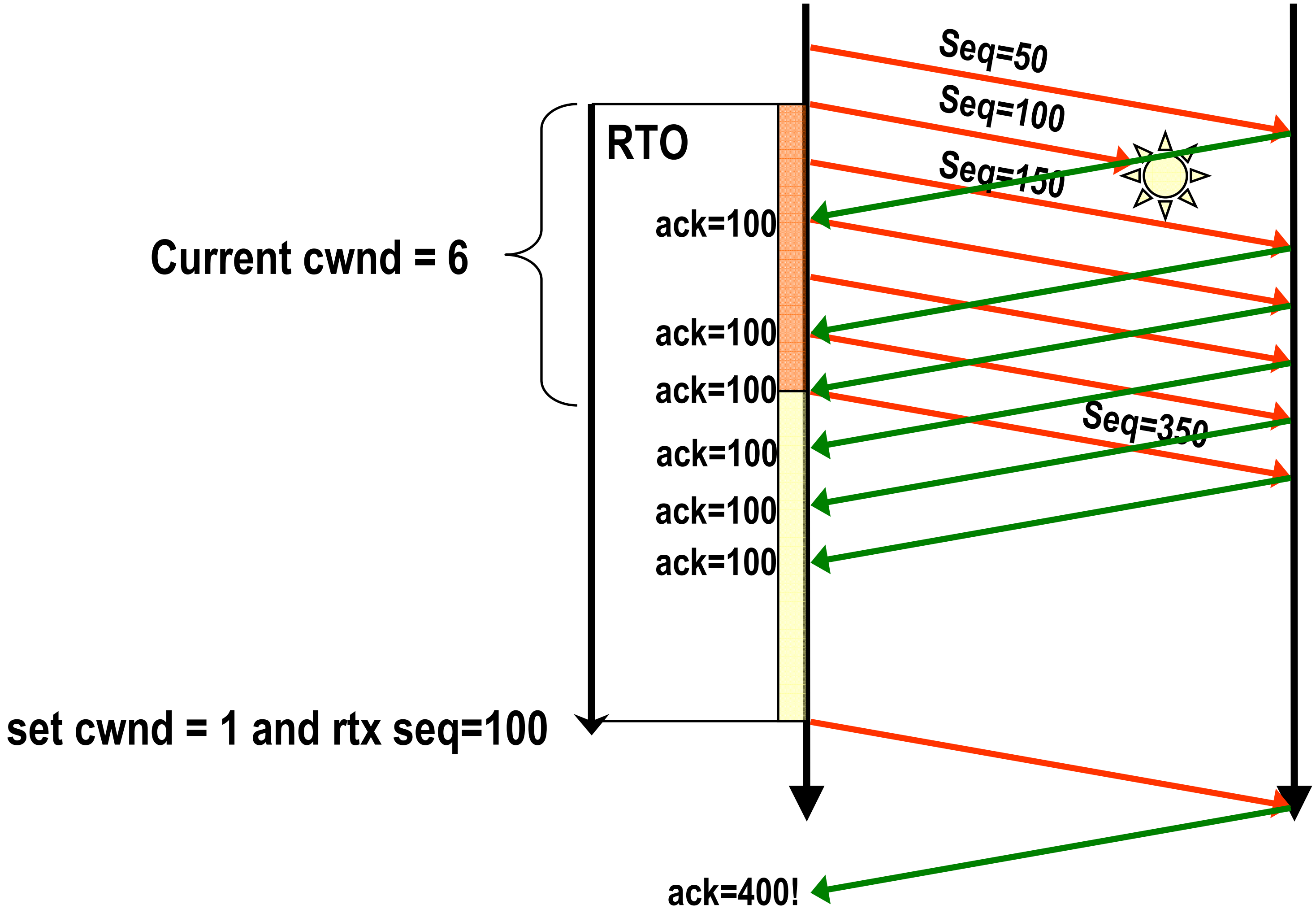


The Fast Retransmit Algorithm

- Idea: use duplicate ACKs!
  - Receiver responds with an ACK every time it receives an out-of-order segment
  - ACK value = last correctly received segment
- FAST RETRANSMIT algorithm:
  - if 3 duplicate acks are received for the same segment, assume that the next segment has been lost. Retransmit it right away.
  - Helps if single packet lost. Not very effective with multiple losses
- And then? A congestion control issue…



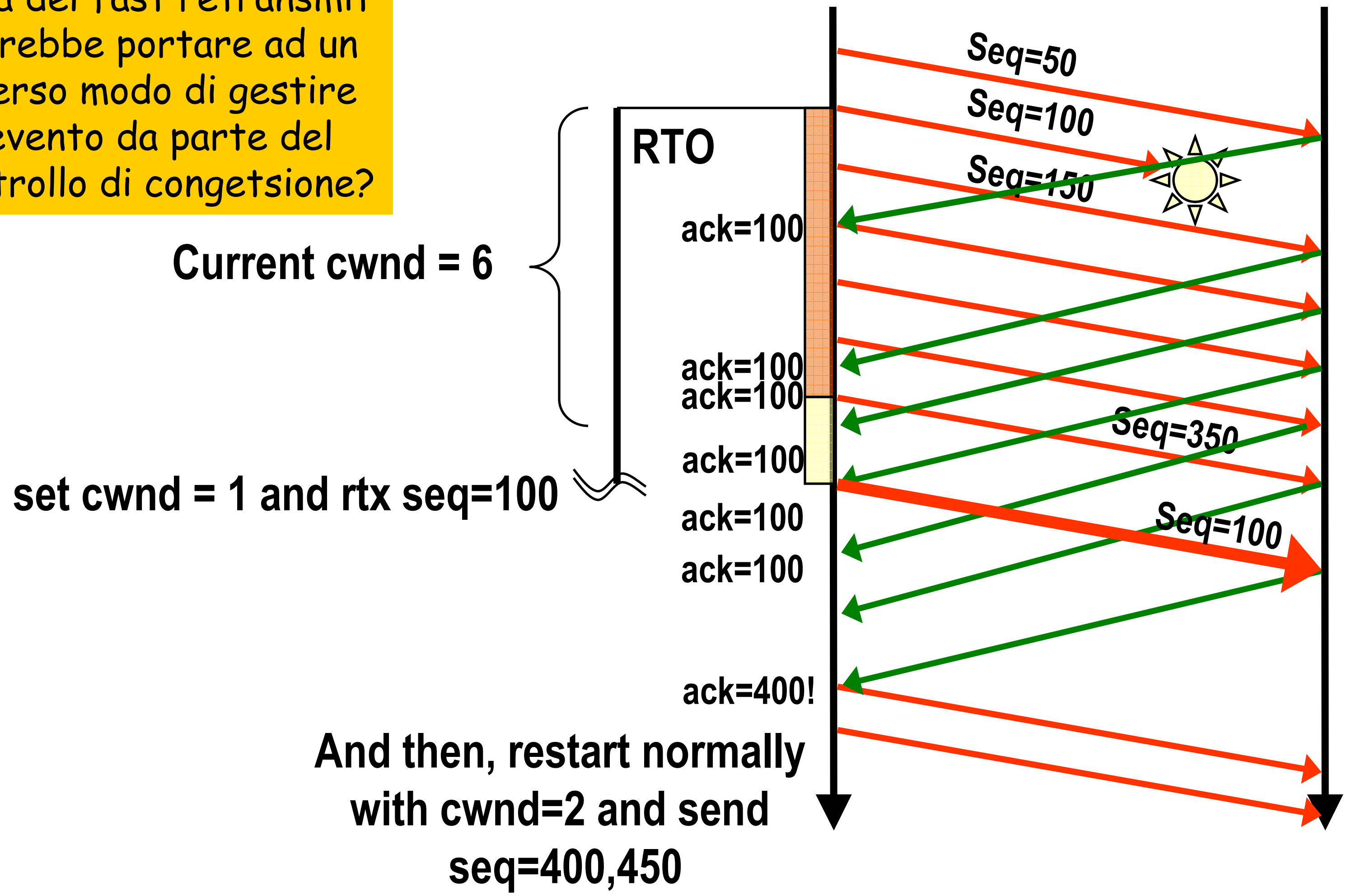
What happens AFTER RTO?
(without fast retransmit)



And then, restart normally with cwnd=2 and send seq=400,450

TCP RENO
(with fast retransmit)

Idea del fast retransmit
Dovrebbe portare ad un
Diverso modo di gestire
L'evento da parte del
Controllo di congetsione?



Same as before, but shorter time to recover packet loss!

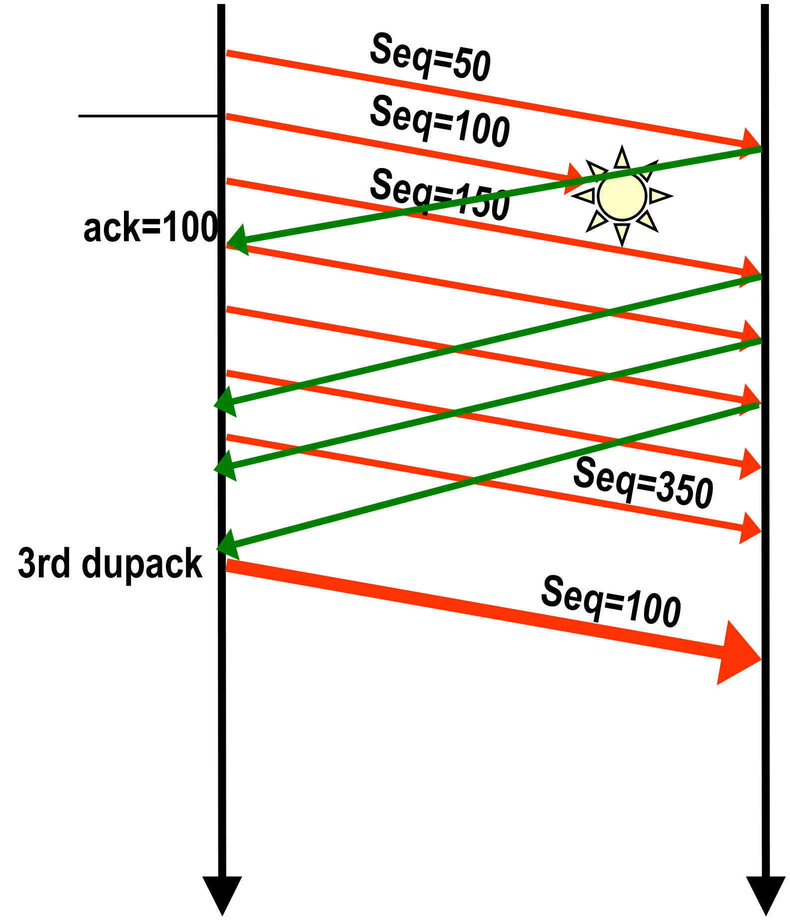
Motivations for fast recovery

FAST RECOVERY:

- ⇨ The phase following fast retransmit (3 duplicate acks received)

⇨ TAHOE approach: slow start, to protect network after congestion

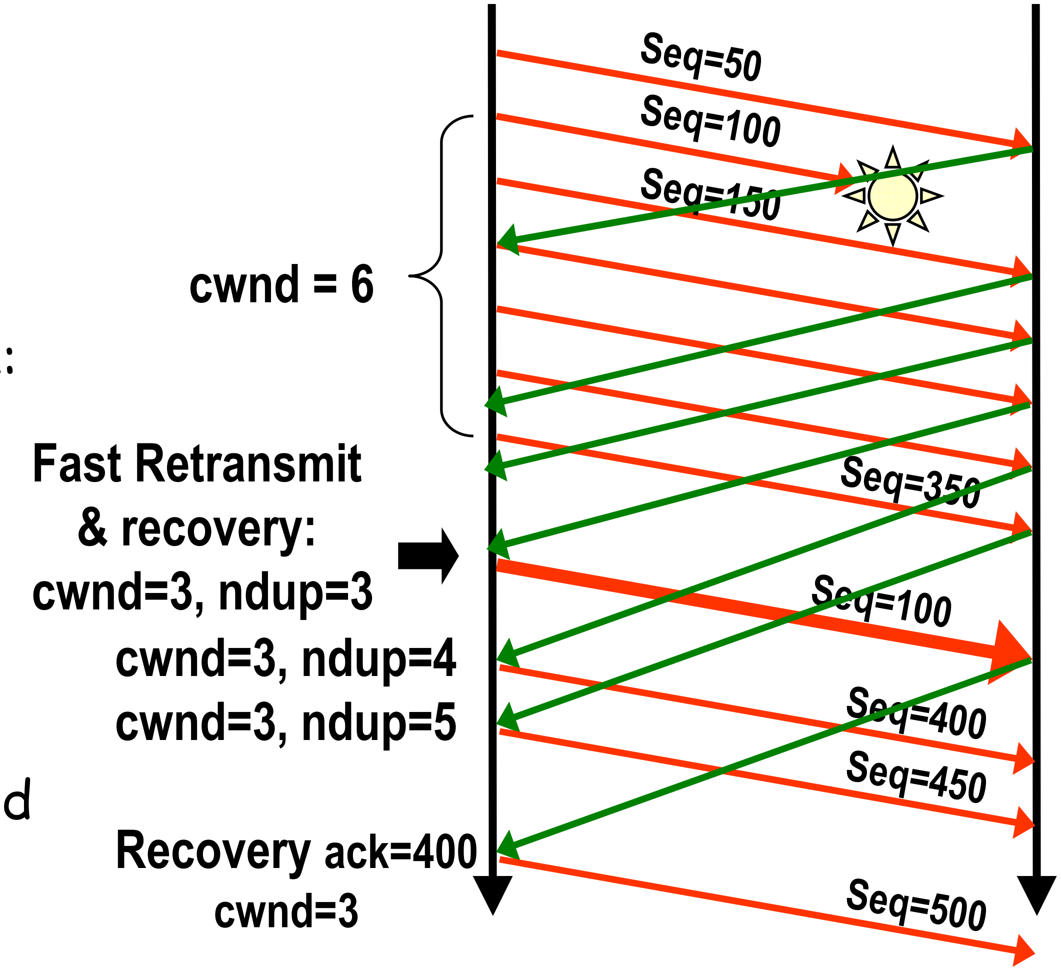
⇨ However, since subsequent acks have been received, no hard congestion situation should be present in the network: slow start is a too conservative restart!



Fast recovery rules

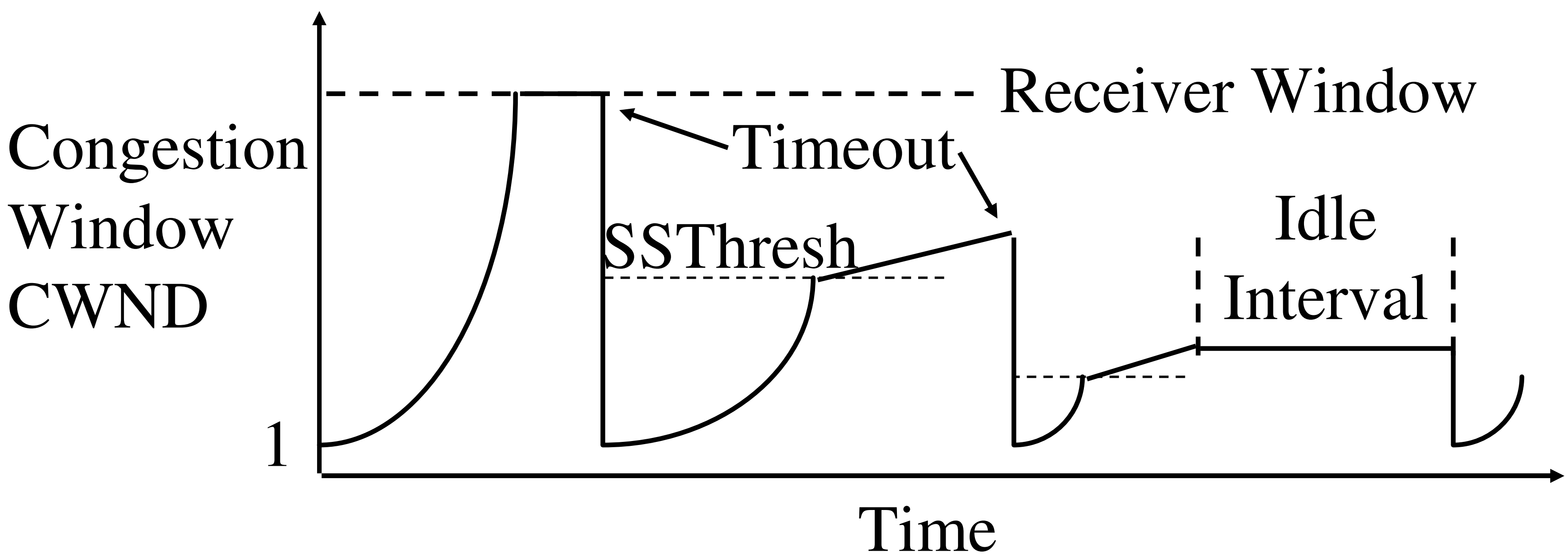
FAST RECOVERY RULES:

- ⇨ Retransmit lost segment
- ⇨ **Set cwnd = cwnd/2**
- ⇨ **Restart with congestion avoidance (linear)**
- ⇨ start fast recovery phase:
    - ⇨ Set counter for duplicate packets ndup=3
    - ⇨ Use "inflated" window: w = cwnd+ndup
    - ⇨ Upon new dup_acks, increase ndup, not cwnd (and send new data)
    - ⇨ Upon recovery ack, "deflate" window setting ndup=0



Idle periods

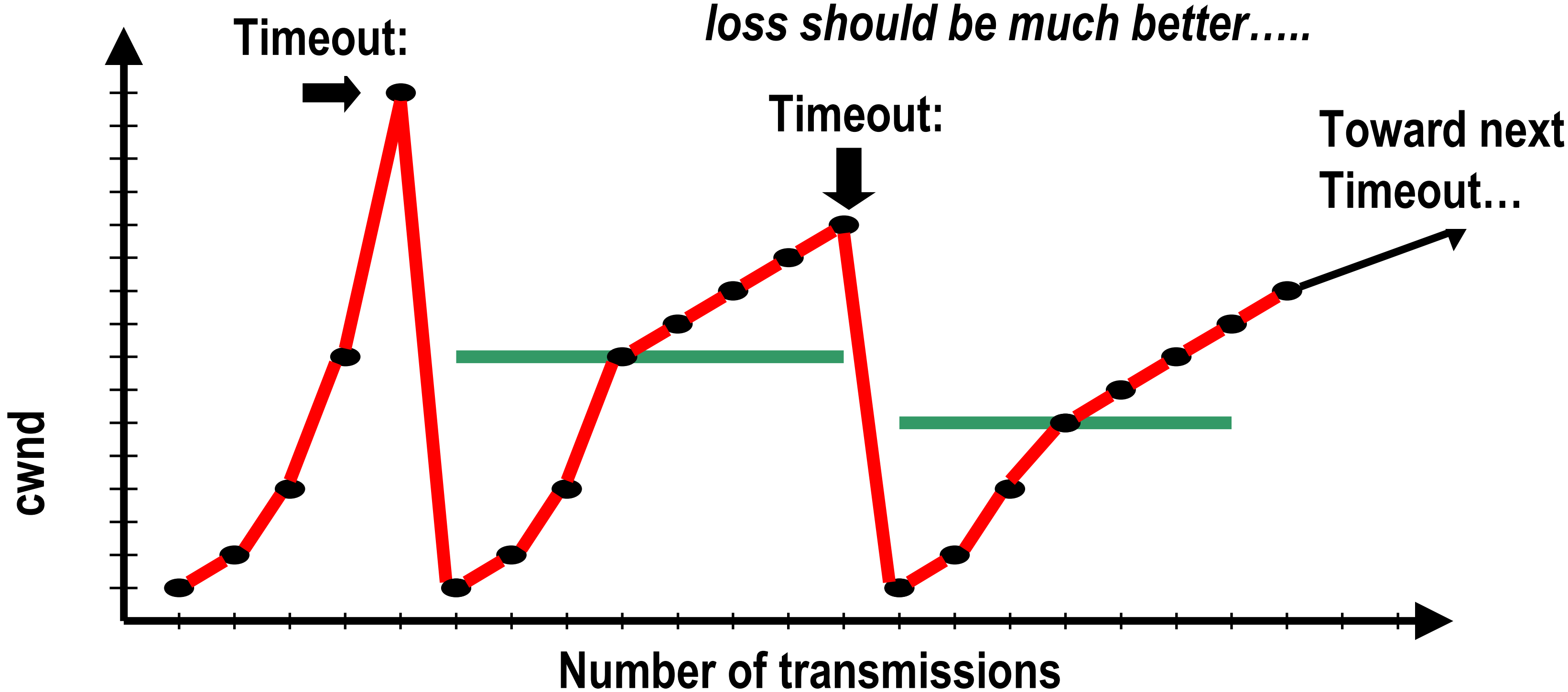
- After a long idle period (exceeding one RTO), reset the congestion window to one.



Further TCP issues

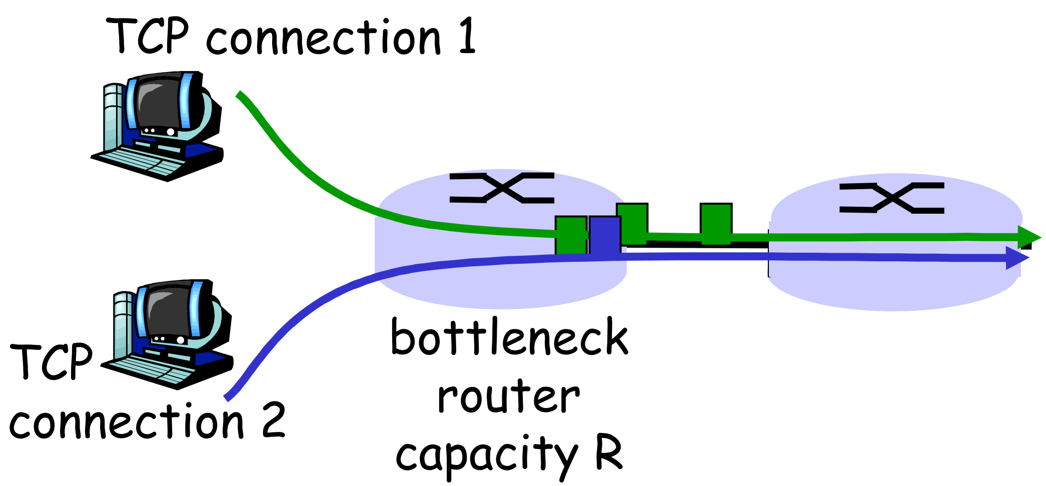
Timeout = packet loss occurrence in an internal network router
TCP (both Tahoe & Reno) does not AVOID packet loss
Simply REACTS to packet loss

CONCLUSION: a TCP able to AVOID packet loss should be much better…..



TCP Fairness

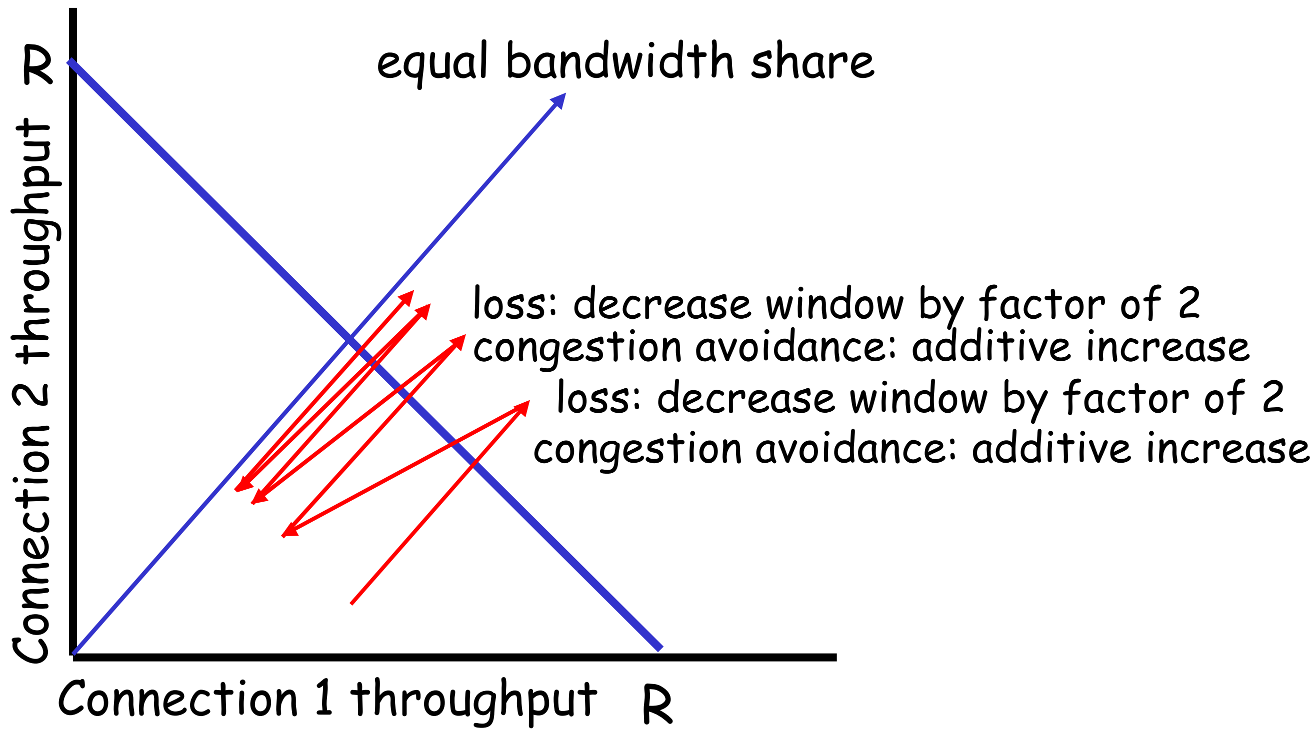
Fairness goal: if K TCP sessions share same bottleneck link of bandwidth R, each should have average rate of R/K



Why is TCP fair?

Two competing sessions:

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

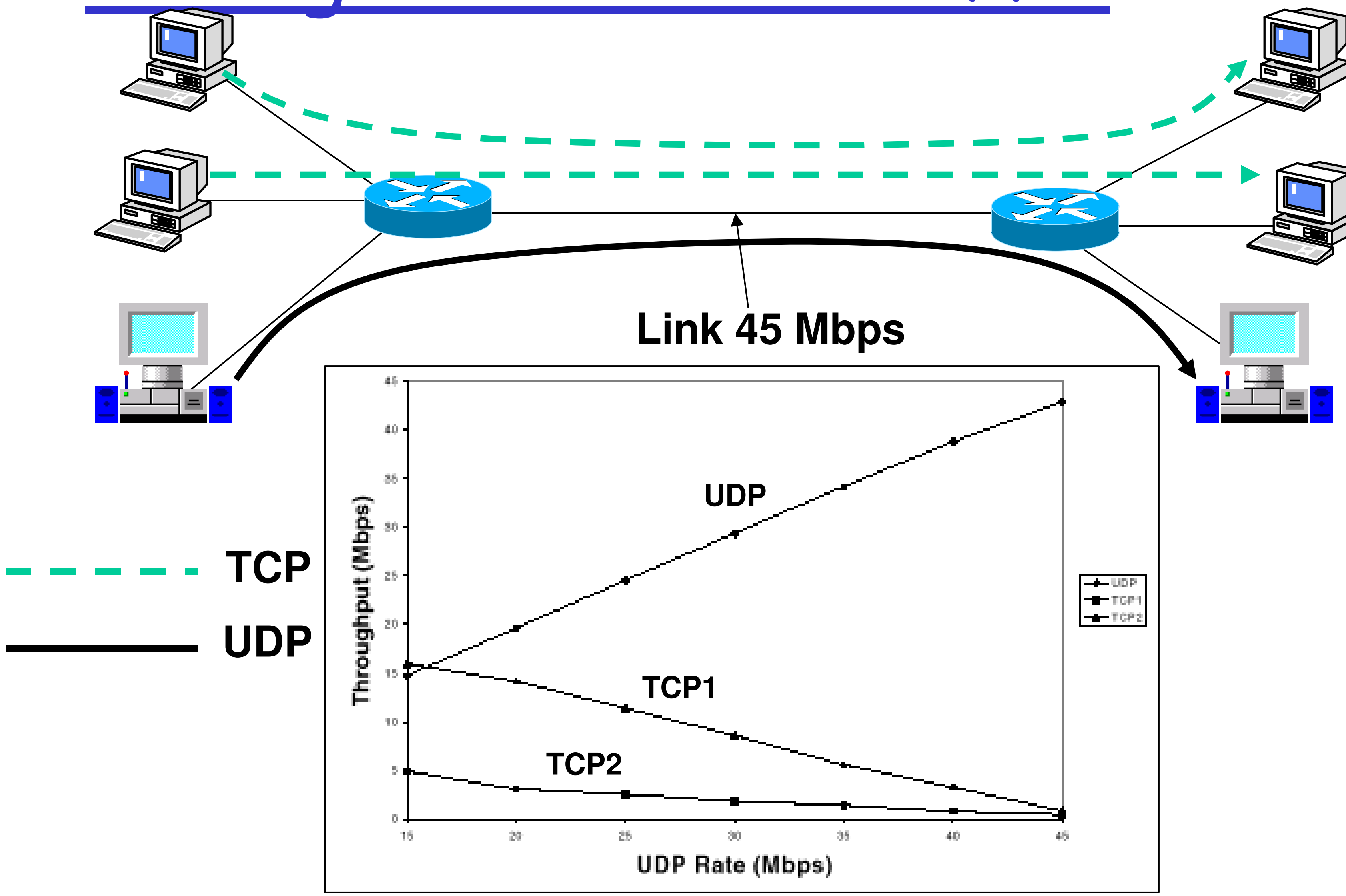


Fairness with UDP traffic

- A serious problem for TCP
  - in heavy network load, TCP reduces transmission rate. Non congestion-controlled traffic does not.
  - Result: in link overload, TCP throughput vanishes!

This is why we still live in a World Wide Wait time (Webcams are destroying TCP traffic)

Mixing TCP & UDP traffic



Fairness (more)

Fairness and UDP

- Multimedia apps often do not use TCP
  - do not want rate throttled by congestion control
- Instead use UDP:
  - pump audio/video at constant rate, tolerate packet loss
- Research area: TCP friendly

Fairness and parallel TCP connections

- nothing prevents app from opening parallel connections between 2 hosts.
- Web browsers do this
- Example: link of rate R supporting 9 cnctions;
  - new app asks for 1 TCP, gets rate R/10
  - new app asks for 11 TCPs, gets R/2 !