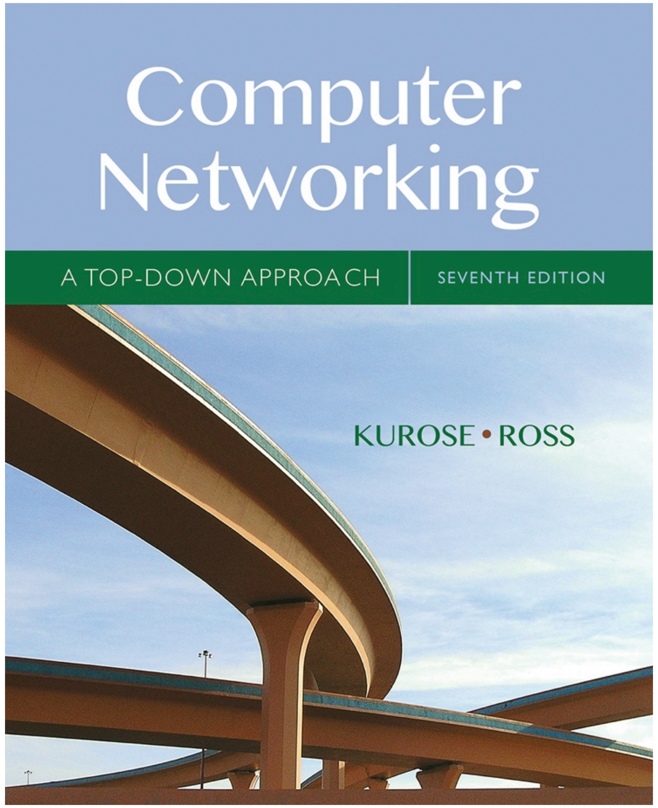
شبکه های کامپیوتری ۲

جلسه ۱۰ فصل ۳

Congestion Control

دانشگاه صنعتی اصفهان
دانشکده مهندسی برق و کامپیوتر

Chapter 3
Transport Layer



Computer Networking: A Top Down Approach

7th edition

Jim Kurose, Keith Ross

Pearson/Addison Wesley
April 2016

A note on the use of these Powerpoint slides:

We're making these slides freely available to all (faculty, students, readers). They're in PowerPoint form so you see the animations; and can add, modify, and delete slides (including this one) and slide content to suit your needs. They obviously represent a *lot* of work on our part. In return for use, we only ask the following:

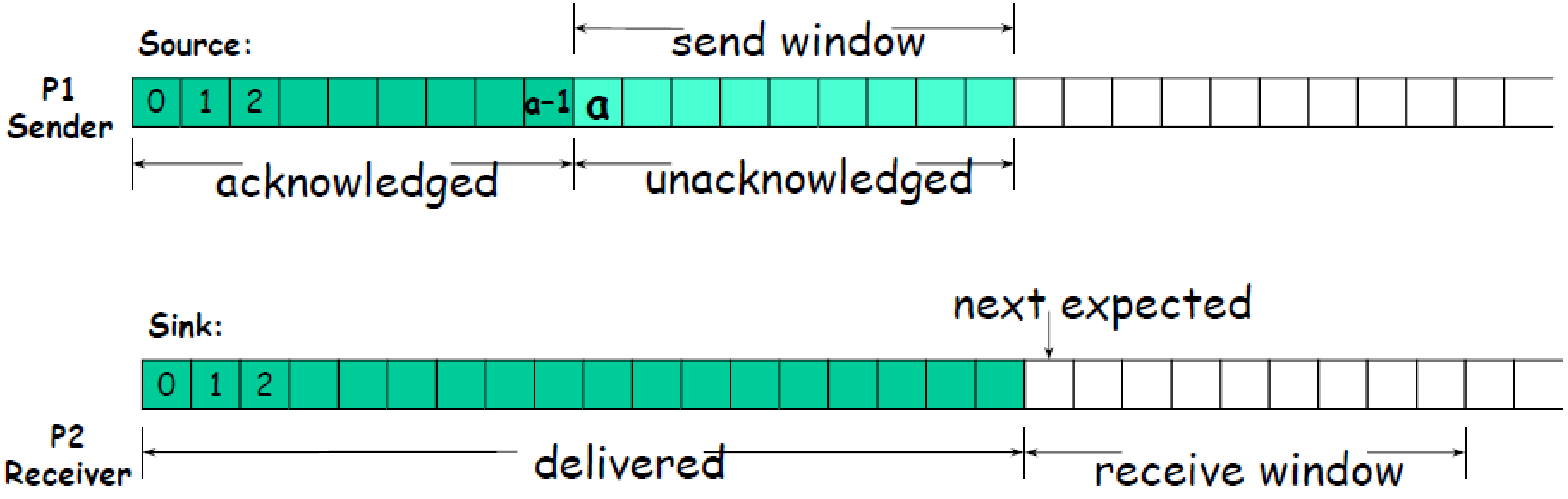
- If you use these slides (e.g., in a class) that you mention their source (after all, we'd like people to use our book!)
- If you post any slides on a www site, that you note that they are adapted from (or perhaps identical to) our slides, and note our copyright of this material.

Thanks and enjoy! JFK/KWR

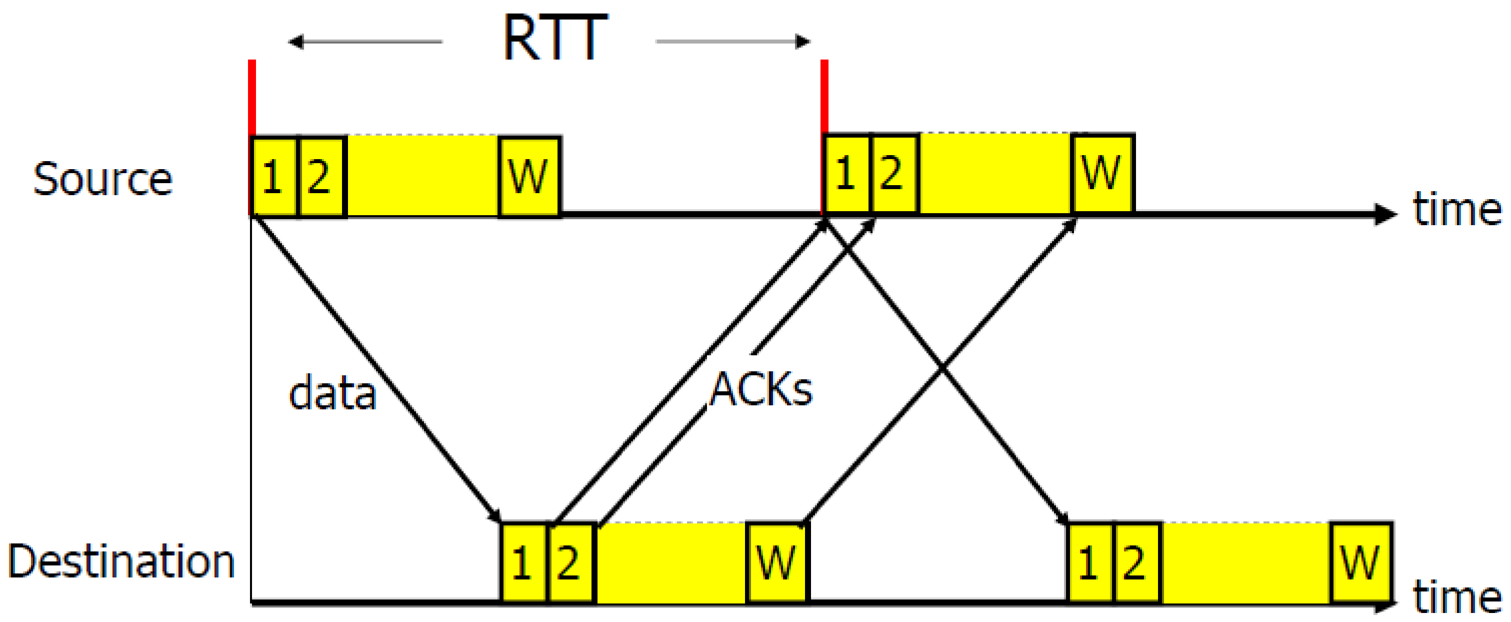
Chapter 3 outline

Sliding Window protocol

- Functions provided
  - reliable delivery (error and loss control)
  - in-order delivery
  - flow and congestion control
    - by varying send window size



Window Size Controls Sending Rate



- ~ W packets per RTT when no loss

Throughput

Max. throughput = W / RTT bytes/sec

- This is an upper bound
- Actual throughput is smaller
  - Average number in the send buffer is less than W
  - Retransmissions
- The throughput of a host's TCP send buffer is the host's send rate into the network (including original transmissions and retransmissions)

TCP Send Window Size

- TCP flow control
  - Avoid overloading receiver
  - Receiver calculates flow control window size (**rwnd**) based on the available receiver buffer space
  - Receiver sends flow control window size to sender in TCP segment heather
  - Sender keeps Send Window size less than most recently received **rwnd** value
- TCP Congestion Control
  - Avoid overloading network
  - Sender estimates network congestion from "loss indications"
  - Sender calculates congestion window size (**cwnd**)
  - Sender keeps Send Window size less than a maximum **cwnd** value
- **Sender sets W = min (cwnd, rwnd)**

TCP Congestion Control

- end-to-end control (no network assistance)
- Sender limits transmission

LastByteSent-LastByteAcked ≤ **cwnd**

Throughput ≤ **cwnd**/RTT   bytes/sec

Note: For now consider **rwnd** to be very large such that the send window size is always set equal to **cwnd**

TCP Congestion Control

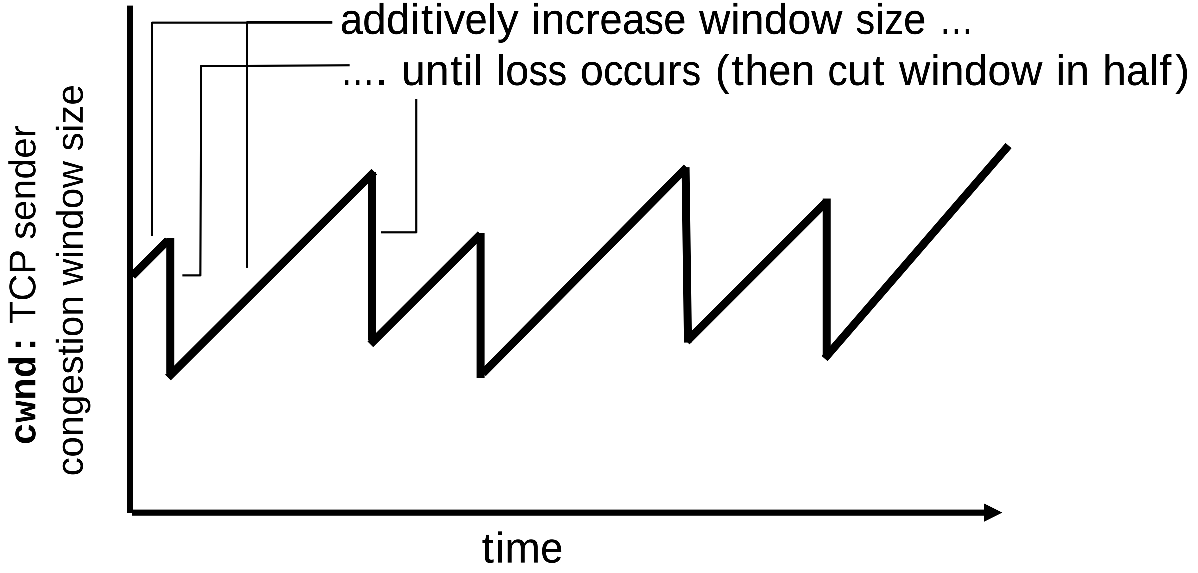
- How does sender estimate network congestion?
  - Packet loss is considered as an indication of network congestion
    - Time Out
    - Duplicate Acks
  - TCP sender reduces cwnd after a loss event

- How does sender determine **cwnd** size?
  - Sender adjusts existing cwnd according to the loss events
    - AIMD (Additive Increase Multiplicative Decrease)

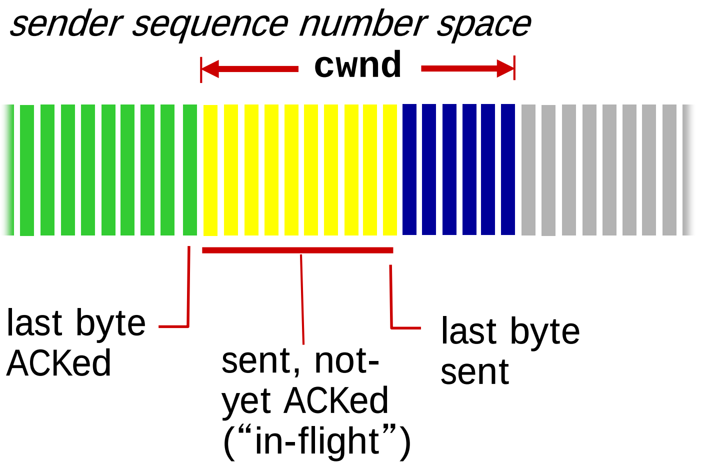
Additive Increase Multiplicative Decrease

- *approach:* sender increases transmission rate (window size), probing for usable bandwidth, until loss occurs
  - *additive increase:* increase **`cwnd`** by 1 MSS every RTT until loss detected
  - *multiplicative decrease*: cut **`cwnd`** in half after loss

AIMD saw tooth behavior: probing for bandwidth



TCP Congestion Control: details



- sender limits transmission:

$$\texttt{LastByteSent-LastByteAcked} \leq \texttt{cwnd}$$

- **cwnd** is dynamic, function of perceived network congestion

TCP sending rate:

- *roughly:* send cwnd bytes, wait RTT for ACKS, then send more bytes

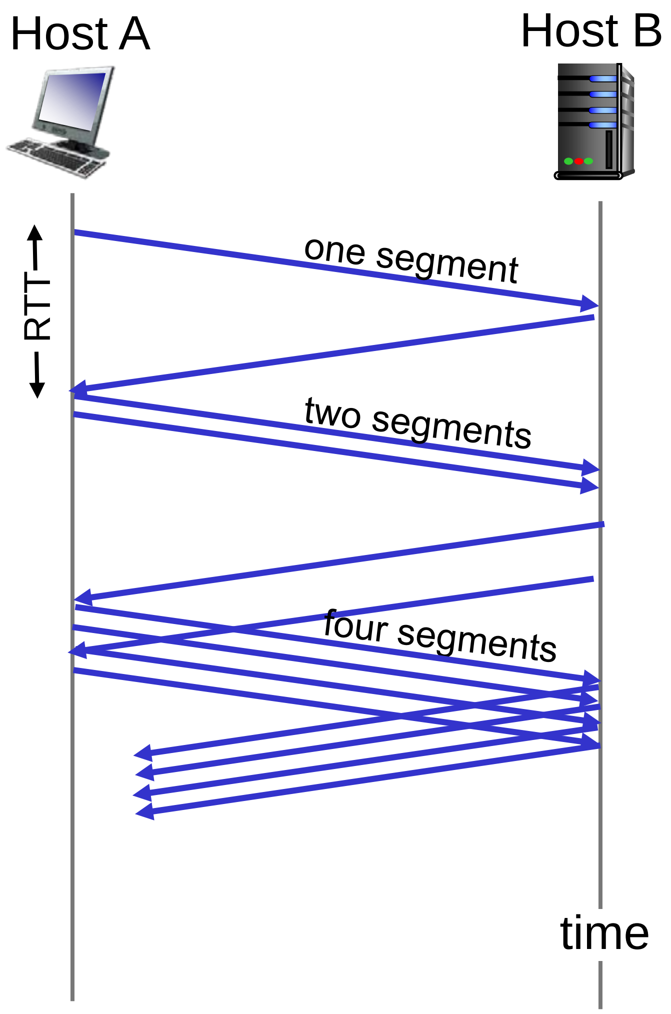
$$\text{rate} \approx \frac{\text{cwnd}}{\text{RTT}} \text{ bytes/sec}$$

At the beginning?

- TCP Slow Start:
  - Probing for usable bandwidth
  - When connection begins, **cwnd** = 1 MSS
    - Example: MSS = 500 bytes & RTT = 200 msec
    - Initial rate = 2500 bytes/sec = 20 kbps
  - Available bandwidth may be >> MSS/RTT
    - Desirable to quickly ramp up to a higher rate

TCP Slow Start

- when connection begins, increase rate exponentially until first loss event:
  - initially **cwnd** = 1 MSS
  - double **cwnd** every RTT
    - done by incrementing **cwnd** for every ACK received
- *summary:* initial rate is slow but ramps up exponentially fast



TCP: detecting, reacting to loss

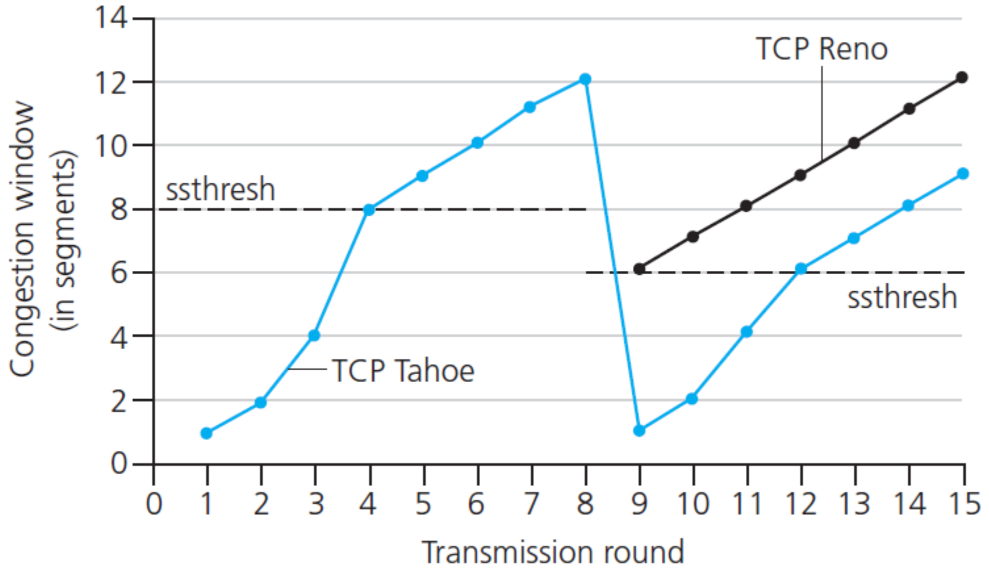
- loss indicated by timeout:
  - **cwnd** set to 1 MSS;
  - window then grows exponentially (as in slow start) to threshold, then grows linearly
- loss indicated by 3 duplicate ACKs: TCP RENO
  - dup ACKs indicate network capable of delivering some segments
  - **cwnd** is cut in half window then grows linearly
- TCP Tahoe always sets **cwnd** to 1 (timeout or 3 duplicate acks)

TCP: switching from slow start to CA

Q: when should the exponential increase switch to linear?

A: when **cwnd** gets to 1/2 of its value before timeout.

This is called slow start threshold (ssthreshold)



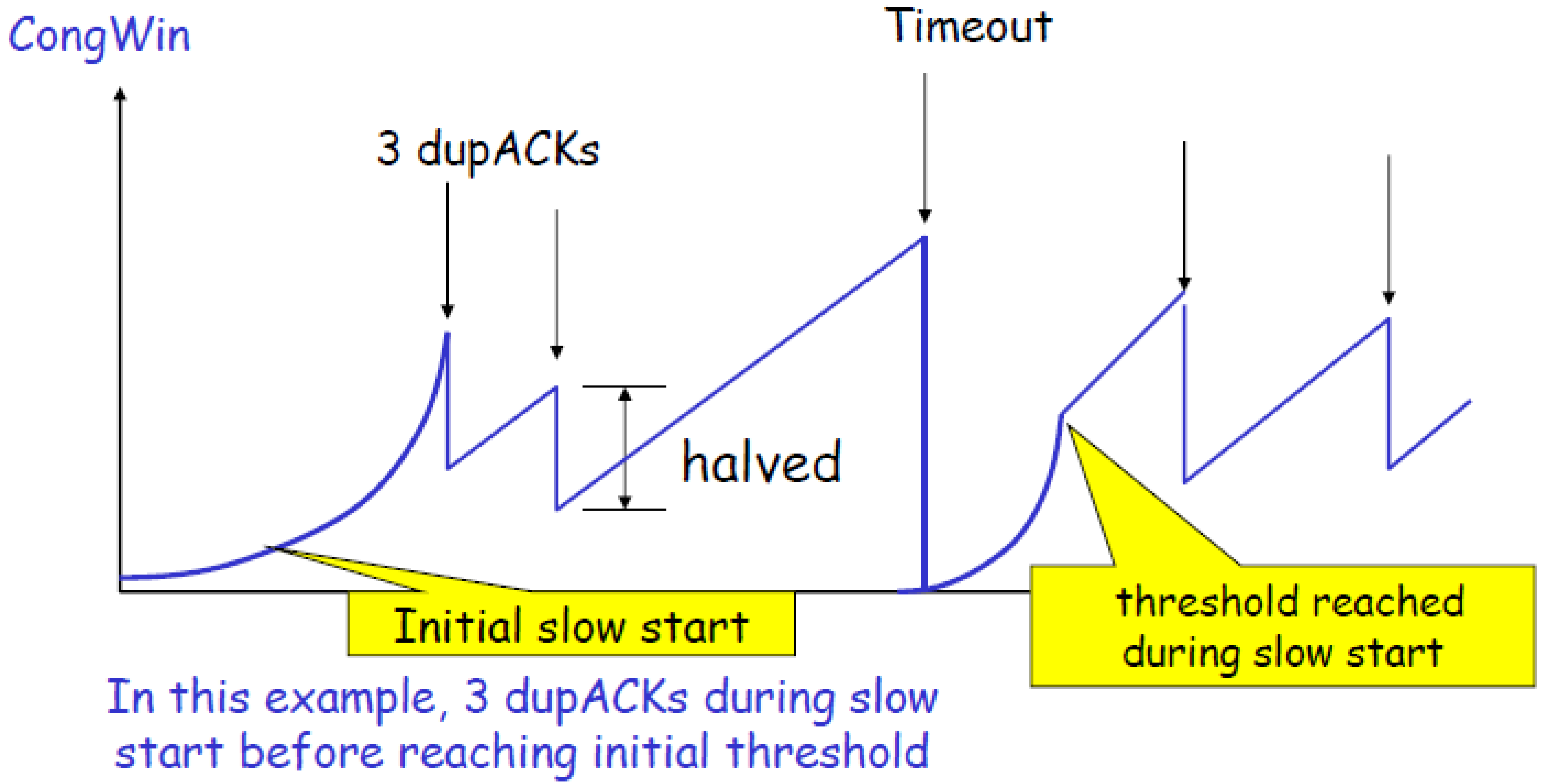
Note: For simplicity, CongWin is in number of segments in the above graph.

* Check out the online interactive exercises for more examples: http://gaia.cs.umass.edu/kurose_ross/interactive/

Threshold

- for initial slow start, threshold is set to a large value
  - assume no threshold until the first loss event
- at a loss event, threshold is set to 1/2 of **cwnd** just before loss event
- subsequently, threshold is variable

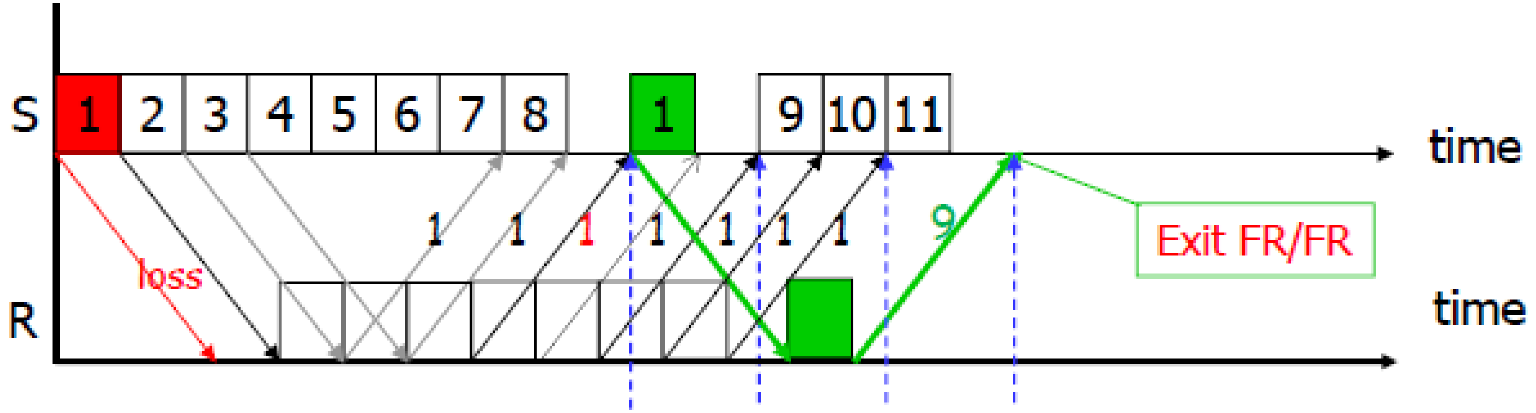
TCP Reno (example scenario)



Summary (TCP Reno)

- When cwnd is below Threshold, sender in slow-start phase, window grows exponentially (until loss event or exceeding threshold).
- When cwnd is above Threshold, sender is in congestion-avoidance phase, window grows linearly.
- When timeout occurs, Threshold set to cwnd/2 and cwnd is set to 1 MSS.
- When a triple duplicate ACK occurs, Threshold set to cwnd/2 and cwnd set to Threshold (also fast retransmit happens).

Fast Recovery entry and exit



cwnd	8	7	9	11	4 ← deflate cwnd
ssthresh		4	4	4	4

- Above scenario: Packet 1 is lost, packets 2, 3, and 4 are received; 3 dupACKs with seq. no. 1 returned
- Fast retransmit
  - Retransmit packet 1 upon 3 dupACKs
- Fast recovery (in steps)
  - Inflate cwnd with #dupACKs such that new packets 9, 10, and 11 can be sent while repairing loss

Summary: TCP Congestion Control