Transport Layer

Anand Baswade

anand@iitbhilai.ac.in

Chapter 3: roadmap

- Transport-layer services
- Multiplexing and demultiplexing
- Connectionless transport: UDP
- Principles of reliable data transfer
- Connection-oriented transport: TCP
- Principles of congestion control
- TCP congestion control
- Evolution of transport-layer functionality

TCP: Triggering congestion control

- Two ways to trigger a congestion notification in TCP – (1) RTO, (2) Duplicate ACK
- RTO: A sure indication of congestion, however time consuming
- Duplicate ACK: Receiver sends a duplicate ACK when it receives out of order segment
  - A loose way of indicating congestion
  - TCP arbitrarily assumes that THREE duplicate ACKs (DUPACKs) imply that a packet has been lost – triggers congestion control mechanism
  - Retransmit the lost packet and trigger congestion control

TCP congestion control: AIMD

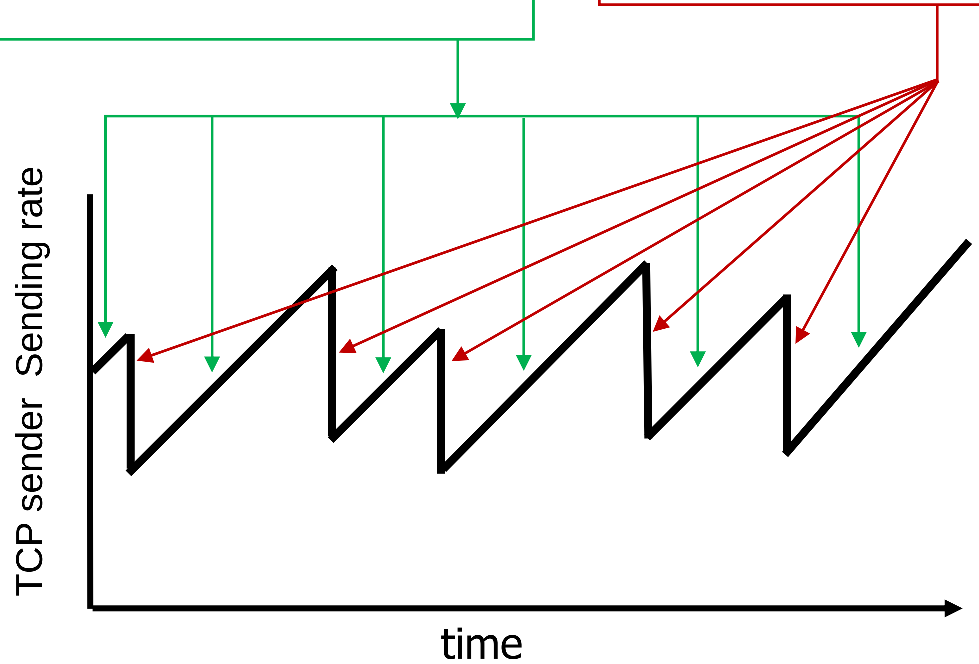
- *approach:* senders can increase sending rate until packet loss (congestion) occurs, then decrease sending rate on loss event.

Additive Increase

increase sending rate by 1 maximum segment size every RTT until loss detected

Multiplicative Decrease

cut sending rate in half at each loss event



- Chiu and Jain (1989): Let *w(t)* be the sending rate. *a (a > 0)* is the additive increase factor, and *b (0<b<1)* is the multiplicative decrease factor

$$w(t+1) = \begin{cases} w(t) + a & \text{if congestion is not detected} \\ w(t) \times b & \text{if congestion is detected} \end{cases}$$

TCP AIMD: more

Multiplicative decrease detail: sending rate is

- Cut in half on loss detected by triple duplicate ACK (TCP Reno)
- Cut to 1 MSS (maximum segment size) when loss detected by timeout (TCP Tahoe)

TCP Congestion Control

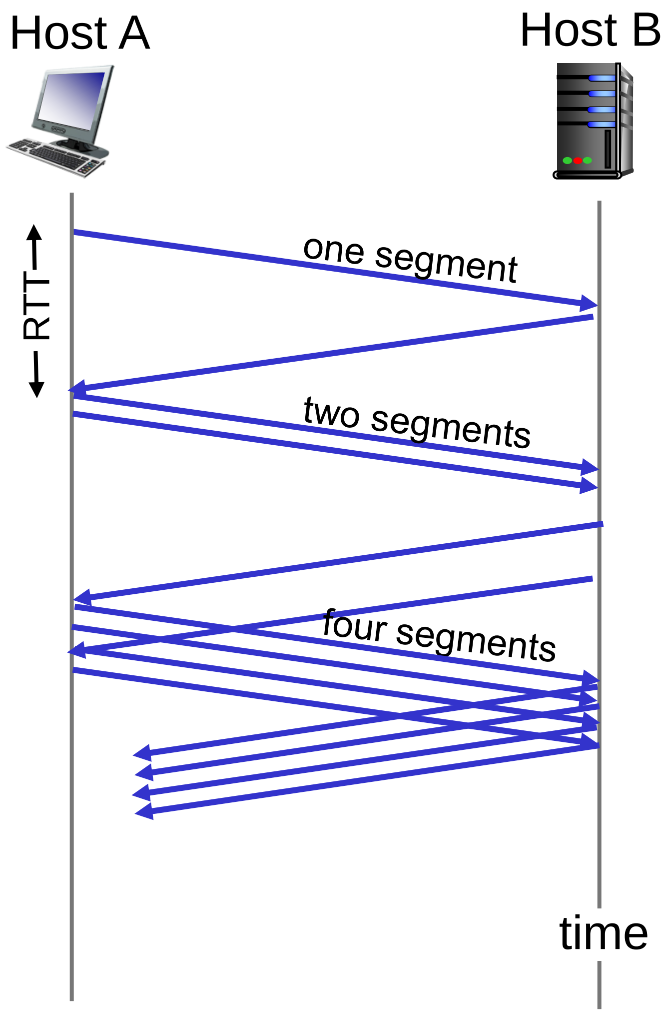
- TCP maintains a **Congestion Window (CWnd) –** number of bytes the sender may have in the network at any time

- **Sender Window (SWnd) = Min (CWnd, RWnd)**

- RWnd – Receiver advertised window size

TCP slow start

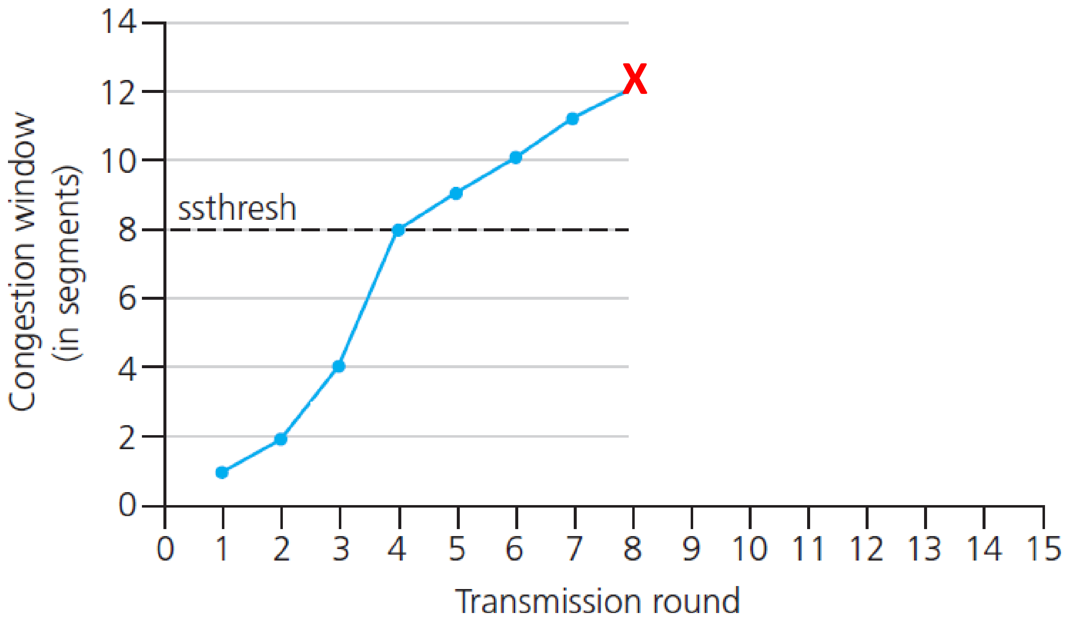
- when connection begins, increase rate exponentially until first loss event:
  - initially **cwnd** = 1 MSS
  - double **cwnd** every RTT
  - done by incrementing **cwnd** for every ACK received
- *summary:* initial rate is slow, but ramps up exponentially fast



Initially cwnd = 1 After 1 RTT, cwnd = 2^(1) = 2 2 RTT, cwnd = 2^(2) = 4 3 RTT, cwnd = 2^(3) = 8 …..

TCP: from slow start to congestion avoidance

The sender has two parameters for congestion control:
- **Congestion Window** (***cwnd***; Initial value is MSS bytes)
- **Threshhold Value** (***ssthresh***; Initial value is 65536 bytes)



Implementation:

- variable **`ssthresh`**
- on loss event, **`ssthresh`** is set to 1/2 of **`cwnd`** just before loss event

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

Slow Start Cont..

- Slow start causes exponential growth, eventually it will send too many packets into the network too quickly.

- To keep slow start under control, the sender keeps a threshold for the connection called the **slow start threshold (ssthresh).**

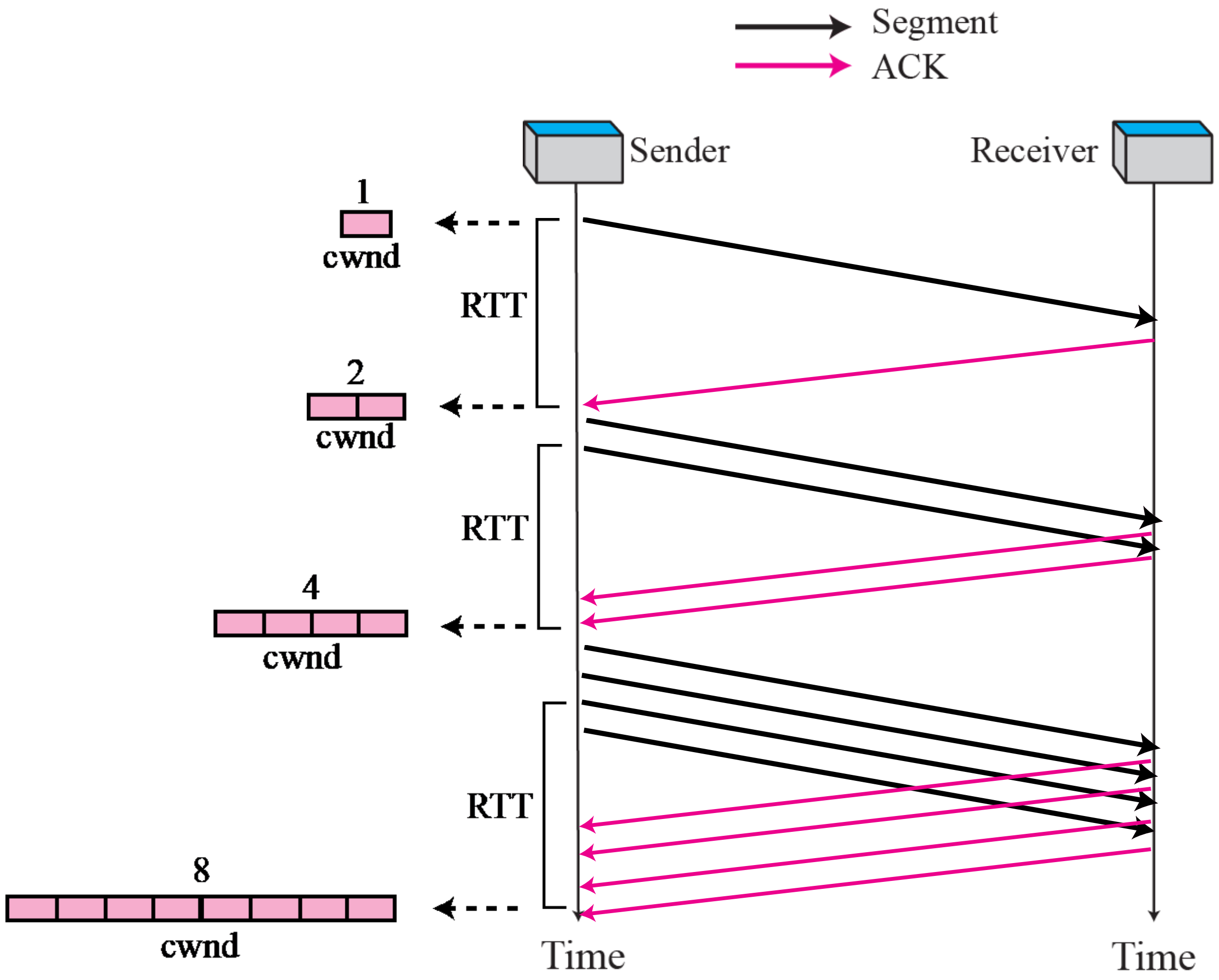
- Whenever a packet loss is detected by a timeout, the ssthresh is set to be half of the congestion window

Congestion Avoidance (Additive Increase)

- Whenever ssthresh is crossed, TCP switches from slow start to additive increase.
- Usually implemented with an partial increase for every segment that is acknowledged, rather than an increase of one segment per RTT.
- A common approximation is to increase Cwnd for additive increase as follows:

$$\boldsymbol{CWnd = Cwnd + \frac{MSS \times MSS}{CWnd}}$$

Congestion Control: Slow start, exponential increase



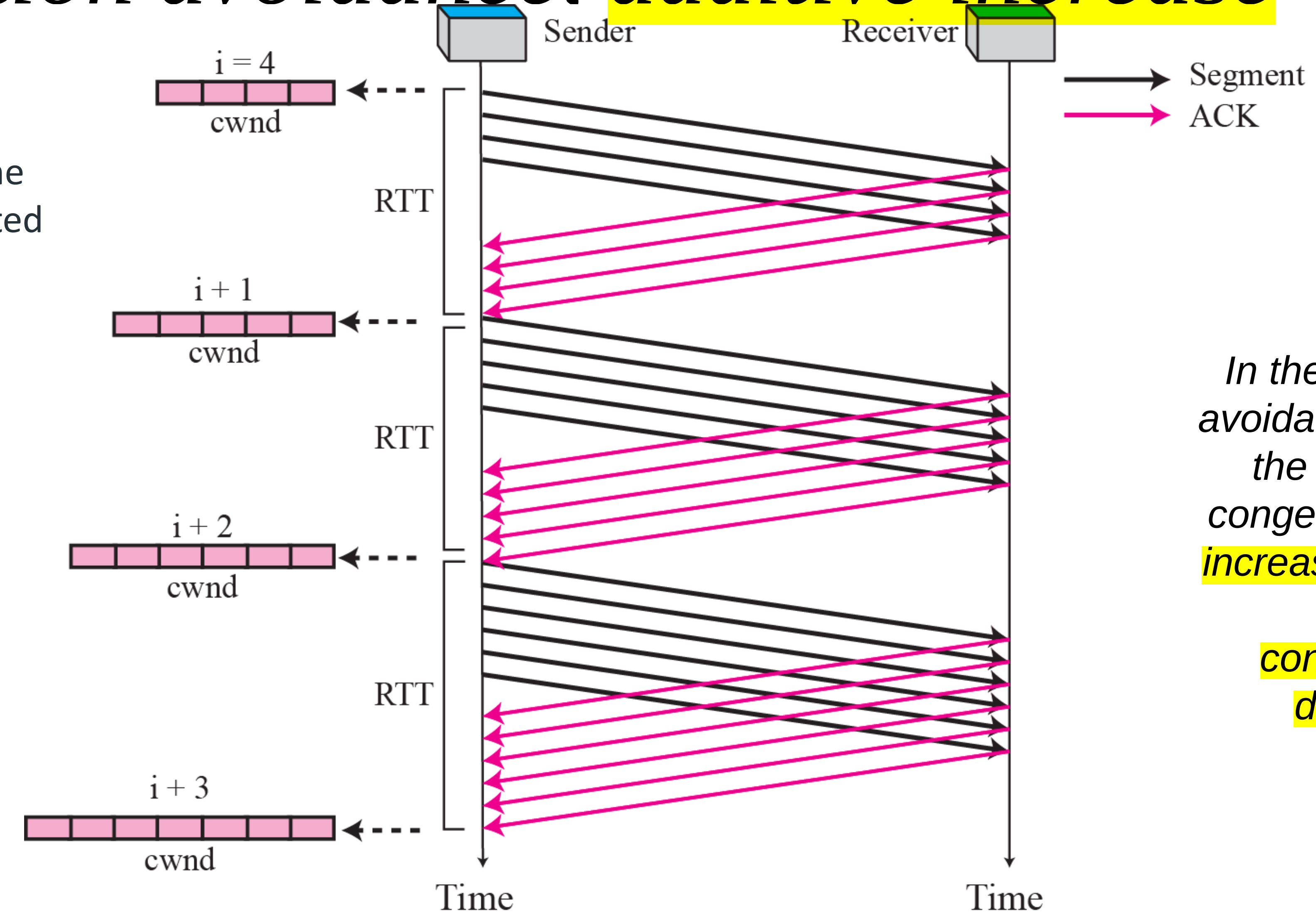
In the slow start algorithm, the size of the congestion window increases exponentially until it reaches a threshold.

TCP/IP Protocol Suite

Congestion avoidance. additive increase

->This phase starts after the threshold value also denoted as *ssthresh*.

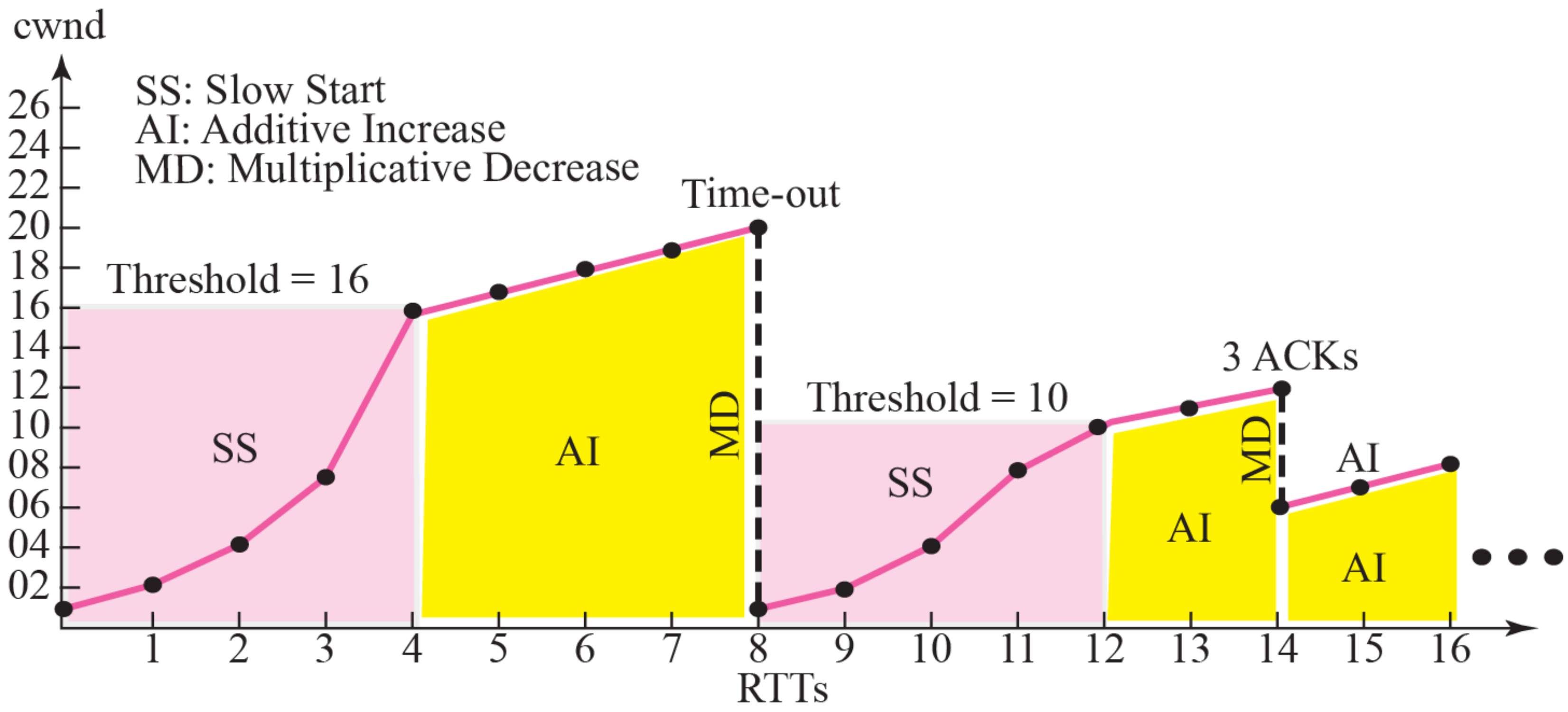
Initially cwnd = I,
After 1 RTT, cwnd = i+1
2 RTT, cwnd = i+2
3 RTT, cwnd = i+3



In the congestion avoidance algorithm the size of the congestion window increases additively until congestion is detected.

TCP/IP Protocol Suite

Congestion example

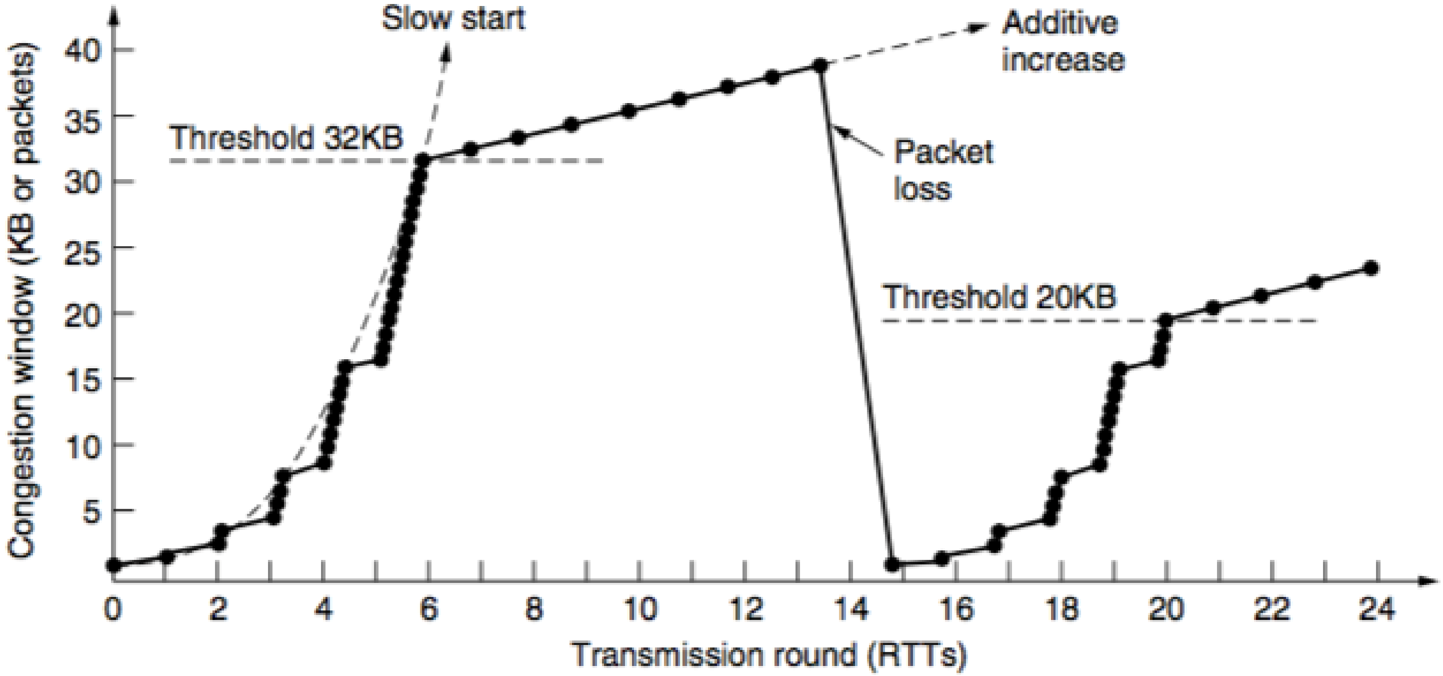


Fast Retransmission - TCP Tahoe

Use THREE DUPACK as the sign of congestion

Once 3 DUPACKs have been received,

- Retransmit the lost packet (**fast retransmission)**
- Set ssthresh as half of the current CWnd
- Set CWnd to 1 MSS



Fast Recovery – TCP Reno

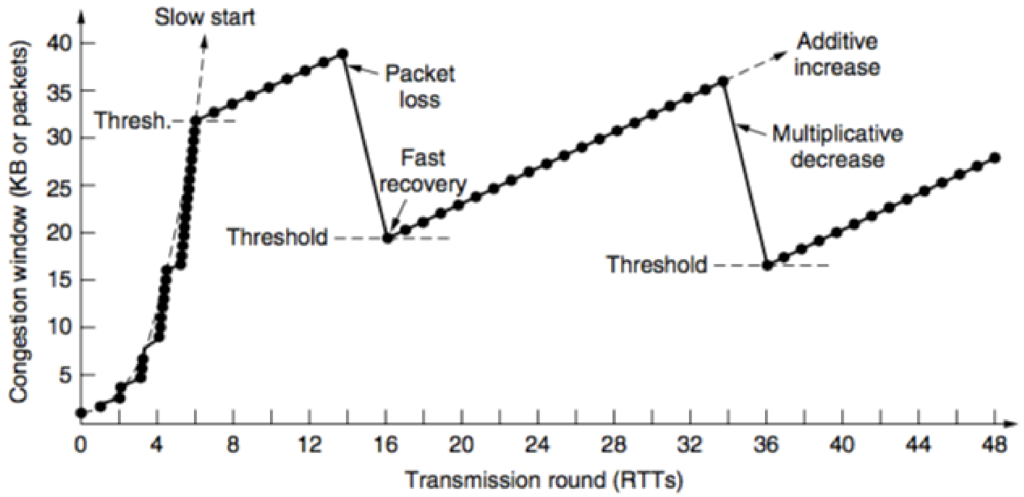
- **Once a congestion is detected through 3 DUPACKs, do TCP really need to set CWnd = 1 MSS ?**

- DUPACK means that **some segments are still flowing in the network** – a signal for temporary congestion, but not a prolonged one

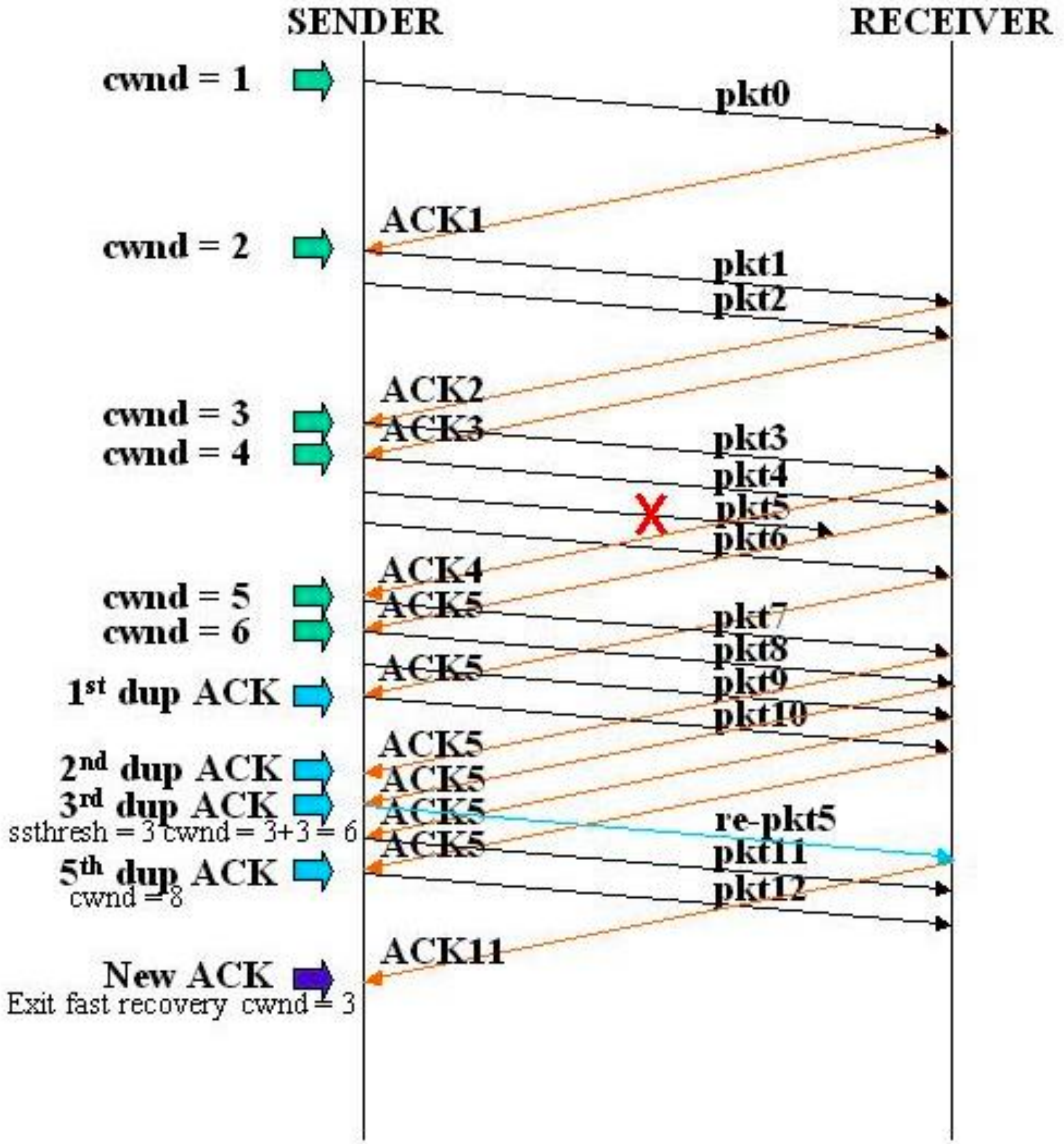
- Immediately transmit the lost segment **(fast retransmit)**, then transmit additional segments based on the DUPACKs received **(fast recovery)**

Fast Recovery – TCP Reno

- **Fast recovery:**
- set ssthresh to half of the current congestion window. Retransmit the missing segment.
- set cwnd = ssthresh + 3.
- Each time another duplicate ACK arrives, set cwnd = cwnd + 1. Then, send a new data segment if allowed by the value of cwnd.
- Once receive a new ACK (an ACK which acknowledges all intermediate segments sent between the lost packet and the receipt of the first duplicate ACK), exit fast recovery. This causes setting cwnd to ssthresh (the ssthresh in step 1). Then, continue with linear increasing due to congestion avoidance algorithm.



Example: Fast Recovery – TCP Reno



TCP Congestion Control Algorithms

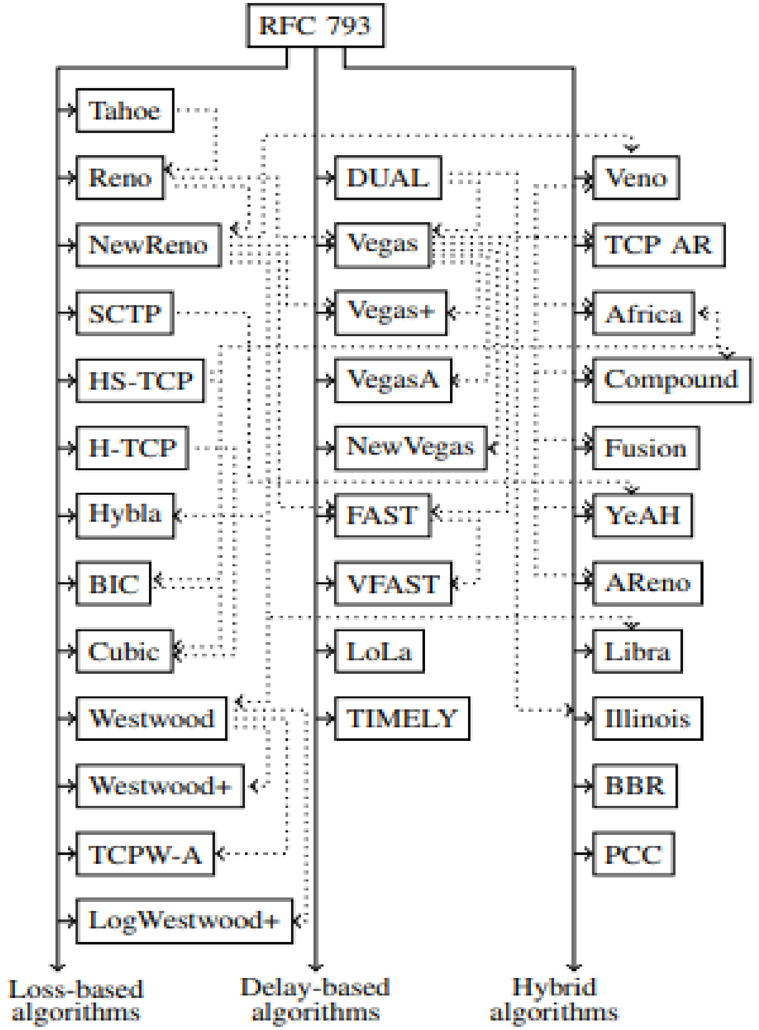


Fig. 3: Classification of different congestion control algorithms. Dotted arrows indicate that one was based on the other.

- TCP Cubic: Used by Many Linux systems

Summary: TCP congestion control