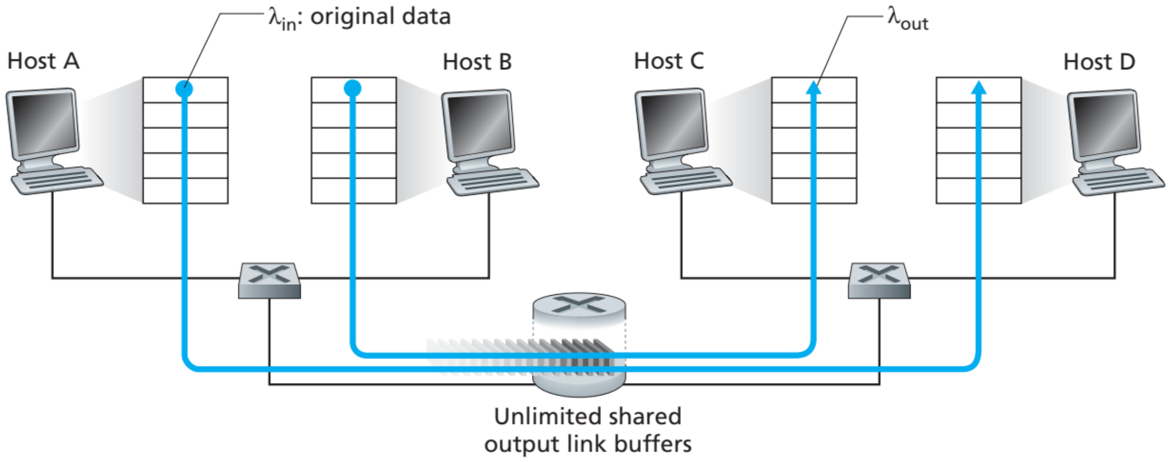
Module 3: Transport Layer (Lecture – 4)

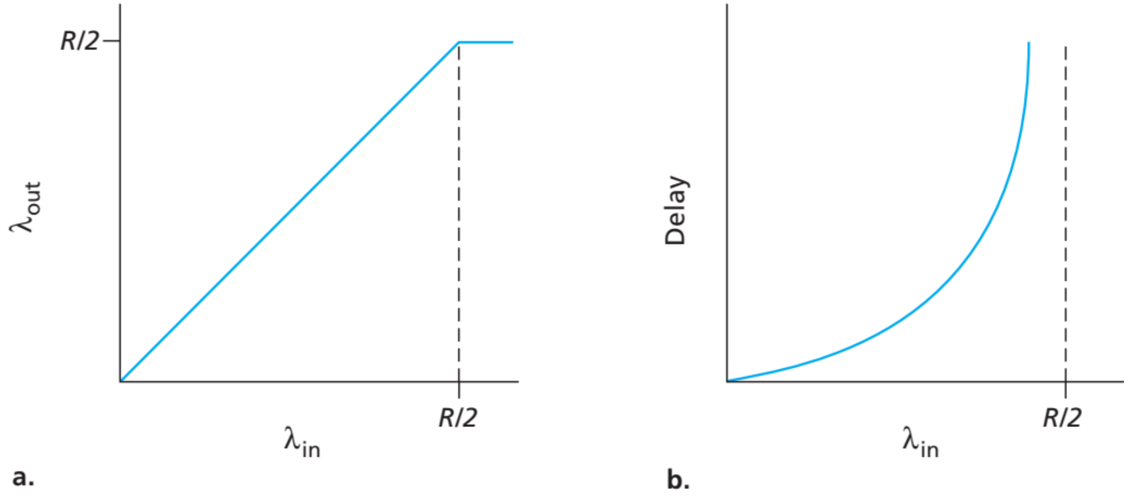
Dr. Nirnay Ghosh
Assistant Professor
Department of Computer Science & Technology
IIEST, Shibpur

Congestion Control: Causes & Costs

- Scenario 1: two senders, a router with infinite buffers
  - Cause:
    - Let the maximum shared link capacity = *R*
    - Throughput bounded by the capacity of shared link
    - Router can store the packets in its infinite buffer if the packet arrival rate exceeds the link capacity
    - For sending rate between 0 and *R/2*: packets are received at the router with increasing average delay (Fig. a)
    - For sending rate above *R/2:* the throughput is only *R/2* and number of packets queued in the router becomes unbounded - infinite average delay (Fig. b)
  - Cost :
    - Larger queuing delays are experienced as the packet arrival rate nears the link capacity



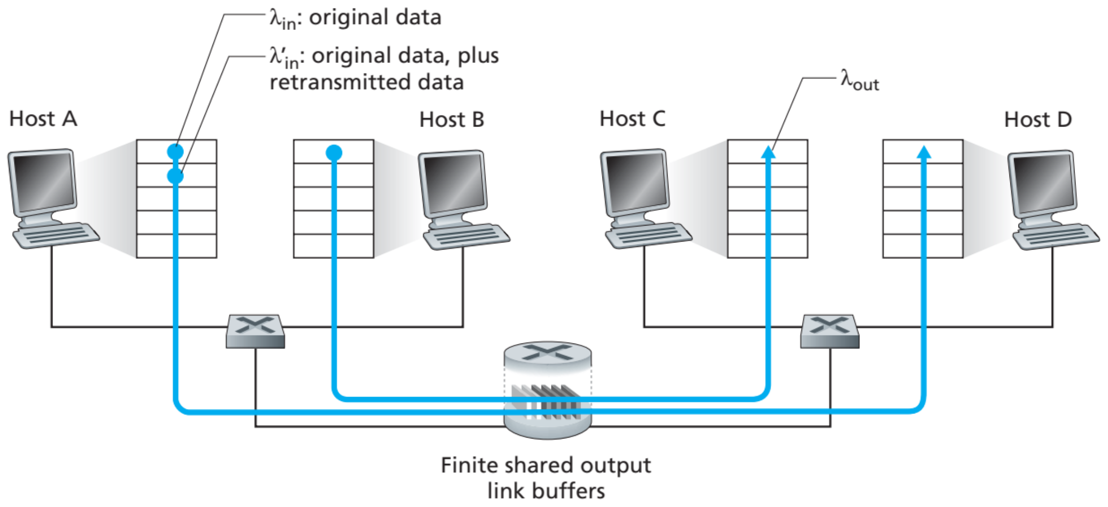
Scenario 1



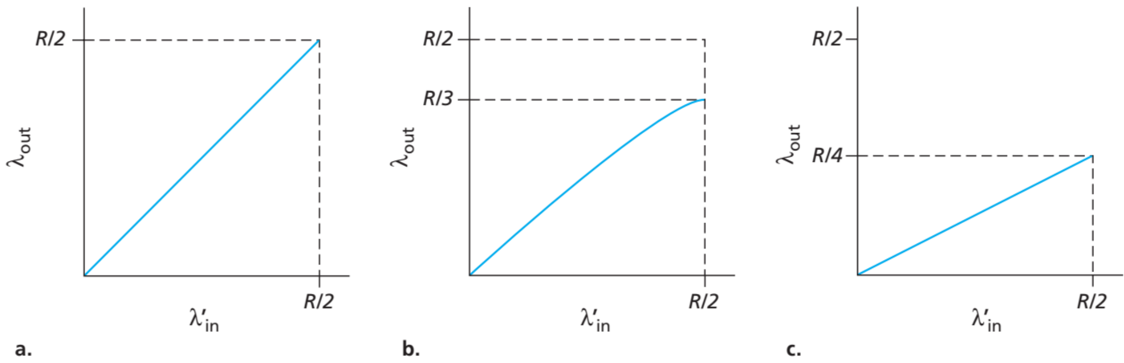
Throughput and Delay as a Function of Sending Rate

Congestion Control: Causes & Costs

- Scenario 2: two senders, a router with finite buffers
  - Cause:
    - Finite buffer – possibility of packet drop
    - Reliability of link: ensured by retransmission of packets by the sender
    - Case 1: Sender sends a packet only when a buffer in the router is free
      - Packet loss will not take place – throughput is equal to the sending rate (Fig. a)
    - Case 2: Sender retransmits only when a packet is known for certain to be lost
      - If offered load (original transmission + retransmission) exceeds *R/2*, the data (original) delivery rate would be upper-bounded by *R/3* (Fig. b)
    - Case 3: Sender retransmits the packet which is not lost but waiting in the router's buffer queue (may be due to premature timeout)
      - Both original and retransmitted packet reach the receiver
      - Receiver discards the retransmitted packet
      - Work done by the router in forwarding the retransmitted copy of the original packet was wasted
      - Only half the link capacity is used (Fig. c)



Scenario 2

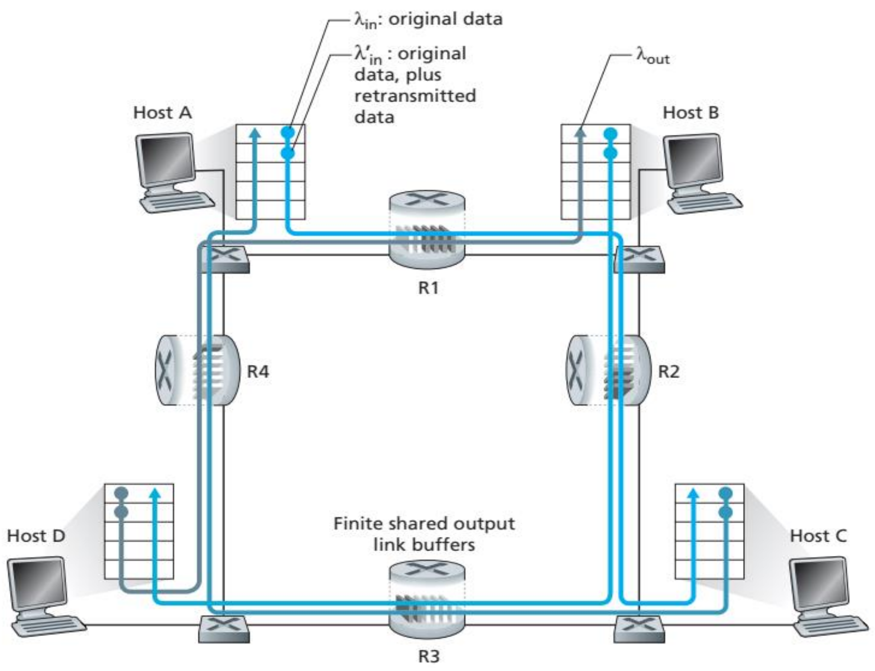


Throughput (Data Delivery) as a Function of Sending Rate

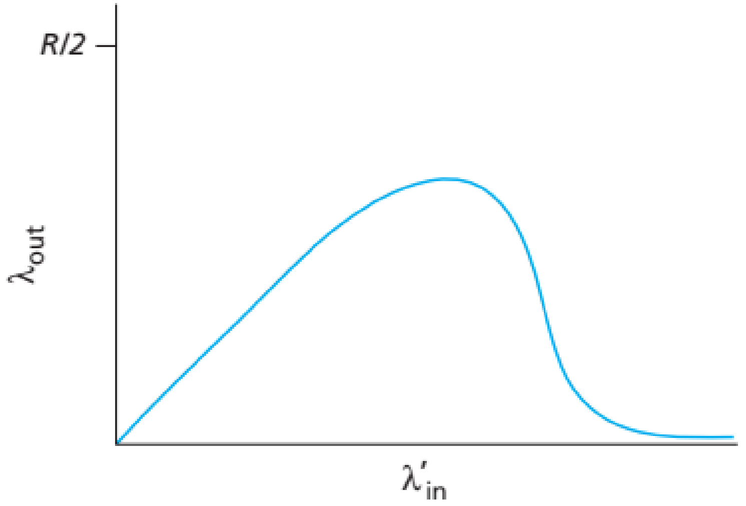
- Cost :
  - Sender must perform retransmissions in order to compensate for dropped (lost) packets due to buffer overflow
  - Router uses up its link bandwidth to forward unneeded retransmitted packet

Congestion Control: Causes & Costs

- Scenario 3: four senders, routers with finite buffers, and multi-hop paths
  - Cause:
    - Router shared between multiple connections
    - *R1* shared between connections A-C and D-B; *R2* shared between connections A-C and B-D
    - Large offered load from one connection rapidly fills up the router buffer making throughput for other connection 0 (see Fig.)
    - Router is busy in forwarding retransmitted packets of one connection and dropping original transmissions from the other connection
    - Transmission capacity is not optimally used as routers are not transmitting different packets
  - Cost:
    - If a packet is dropped along a path, the transmission capacity that was used at each upstream link to forward the packet ends up having being wasted



Scenario 3



Throughput (Data Delivery) as a Function of Sending Rate

Congestion Control: End-to-End Approach

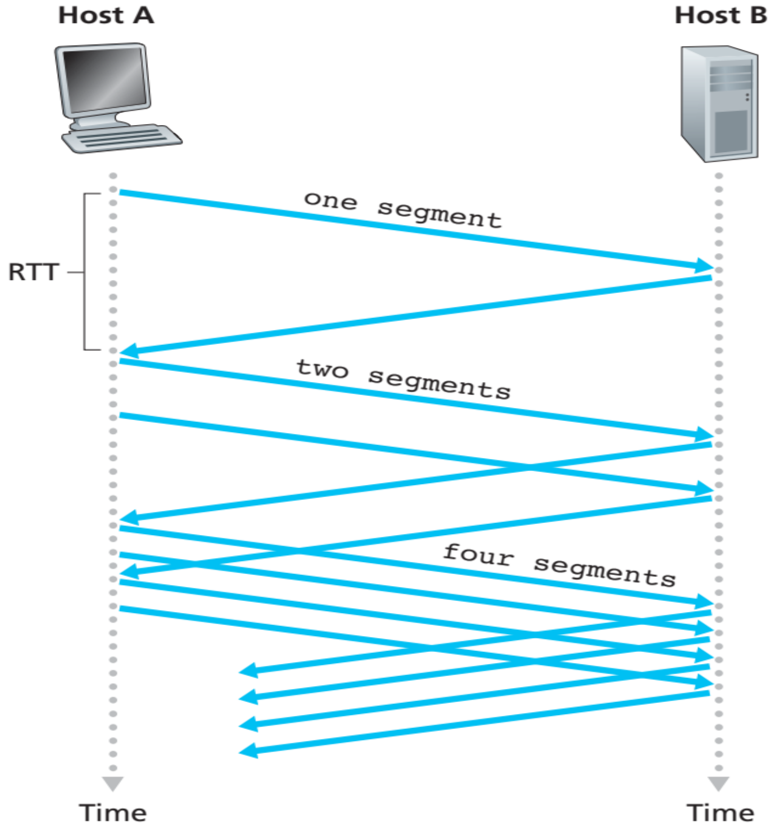
- Network layer provides no explicit support to the transport layer for congestion control
- Presence of congestion is inferred by the end systems based on observed network behavior (such as packet loss, delay)
- TCP congestion control: end-to-end approach
  - IP layer does not provide explicit feedback to the end systems regarding network congestion
- Limits the rate at which sender sends traffic into its connection as a function of the perceived network congestion
- Keeps track of an additional variable: congestion window (*cwnd*)
  - Imposes a constraint on the rate at which a TCP can send traffic into the network
- Amount of unacknowledged data at the sender may not exceed the minimum of *rwnd* and *cwnd*

```
LastByteSent – LastByteAcked ≤ min{cwnd, rwnd}
```

- Limiting the amount of unacknowledged data at the sender has the following advantages:
  - Indirectly limits the sender's send rate at *cwnd/RTT* bytes/sec
  - Adjust the value of *cwnd* to determine the rate at which it sends data into its connection
- Indication of congestion on sender-to-receiver path
  - Dropping of a datagram (containing a TCP segment) by an overloaded router
  - "Loss event" at sender: timeout or the receipt of three duplicate ACKs
- Self-clocking feature of TCP
  - Use the arrival of acknowledgements to adapt the size of the congestion window
    - ACKs arriving at slow rate – *cwnd* will be increased at a relatively slow rate
    - ACKs arriving at high rate – *cwnd* will be increased relatively quickly
- Trade-off: how do the TCP senders determine the optimal sending rates?
  - Too fast transmission: results in network congestion
  - Too slow transmission: results in under utilization of the link bandwidth

TCP Congestion Control Algorithm

- Guiding principles:
  - A lost segment implies congestion – TCP sender's rate should be decreased
  - An acknowledgement segment indicates that the network is delivering the sender's segment to the receiver – TCP sender's rate can be increased
  - Bandwidth probing: increase the transmission rate linearly to probe for the rate at which congestion onset begins
- Three major states of the TCP congestion control algorithm are:
  - Slow Start
  - Congestion Avoidance
  - Fast Recovery
- Slow Start State
  - Value of *cwnd* begins at 1 MSS and increases by 1 MSS every time a transmitted segment is first acknowledged
  - Doubling of the sending rate at every RTT
    - TCP send rates start slow but grows exponentially
  - Exponential growth stops under two conditions:
    - Condition 1: value of *cwnd* exceeds a predefined second state variable *ssthresh* (Slow Start Threshold)
    - Condition 2: If there is a loss event: timeout or receipt of three duplicate ACKs

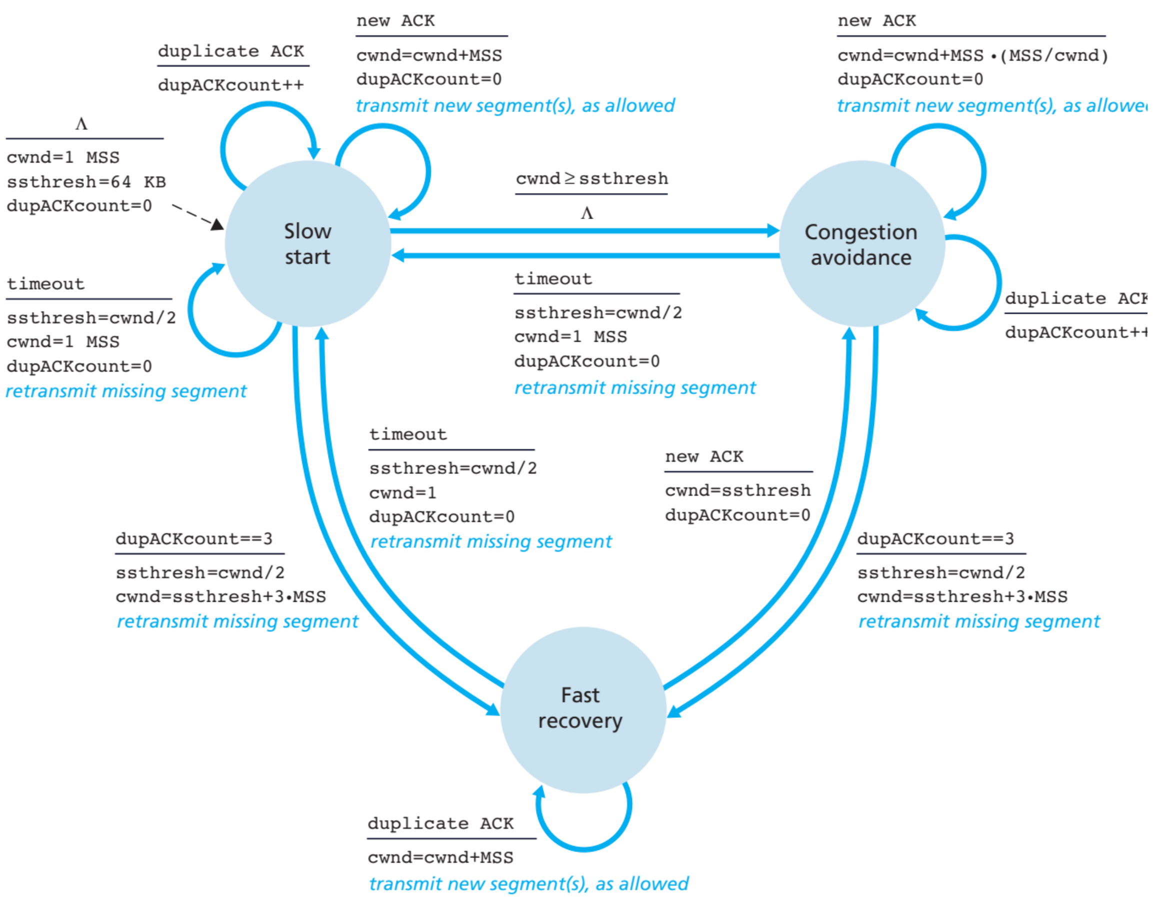


Slow Start State

- Under condition 1:
  - Value of *ssthresh* is set to *cwnd/2* – half the *cwnd* when congestion was detected
  - Slow start ends and TCP transitions into congestion avoidance state
- Under condition 2:
  - Timeout event: TCP sender sets the value of *cwnd* to 1 – starts slow start process afresh
  - Three duplicate ACKs received: TCP performs fast recovery and enters the fast recovery state

TCP Congestion Control Algorithm

- <u>Congestion Avoidance State</u>
  - The *ssthresh* is assigned *cwnd/2* – half the value observed when the congestion was last encountered
  - Conservative approach: rather than doubling the value of *cwnd* in every RTT, it starts with *cwnd = 1* and is increased by just a single MSS every RTT
  - Linear increase (1 MSS per RTT) ends when a loss event triggered by triple duplicate ACK occurs
  - Congestion status different from timeout event
    - Network continues to deliver segments unlike timeouts
  - TCP halves the value of *cwnd* and adds 3 MSSs for good measure to account for the triple duplicate ACK received
    - *ssthresh = cwnd/2*
    - *cwnd = ssthresh + 3.MSS*
  - Congestion avoidance ends and TCP transitions into fast recovery state
- <u>Fast Recovery State</u>
  - Value of *cwnd* is increased by 1 MSS for every duplicate ACK received for the missing segment

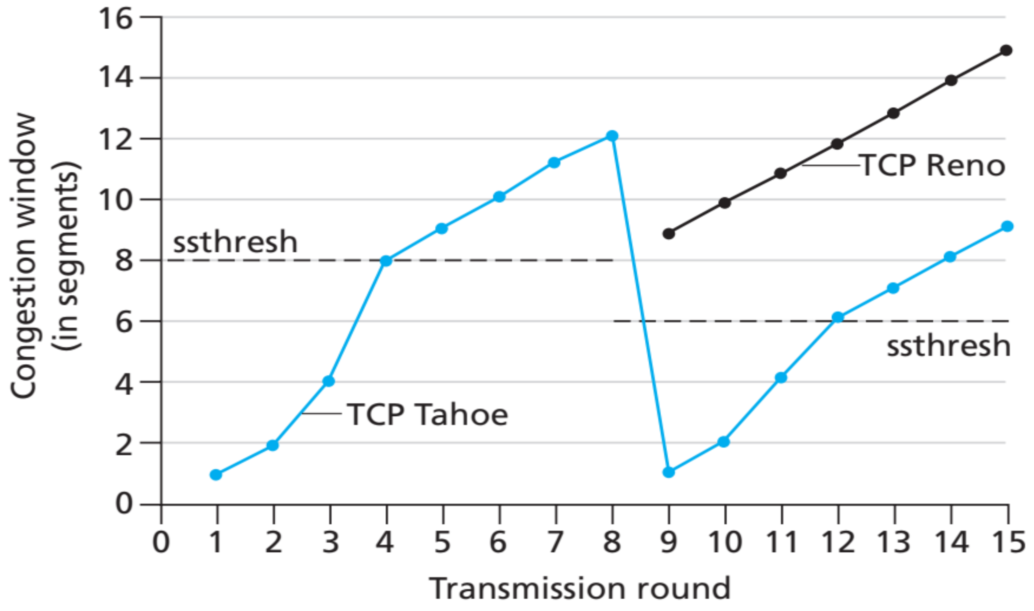


FSM Description of Congestion Control

- ACK arrives for the missing segment: TCP enters the congestion avoidance state and sets *cwnd = ssthresh*
- Timeout event occurs: TCP transitions into slow start state by setting *ssthresh = cwnd/2* and the value of *cwnd = 1* (value observed at the onset of congestion)

Evolution of TCP's Congestion Window

- Fast recovery – recommended but not a required component of TCP
- TCP Tahoe
  - Earlier version of TCP
  - Cut its congestion window to 1 MSS and enter into the slow-start phase after either a timeout-indicated or triple-duplicate-ACK-indicated loss event.
- TCP Reno
  - Newer version of TCP
  - Incorporates Fast Recovery
- Evolution of TCP's congestion window (see fig.)
  - Threshold is initially equal to 8 MSS
  - Both versions take identical actions for the first 8 transmission rounds
  - The congestion window climbs exponentially fast during slow start and hits the threshold at the fourth round of transmission.

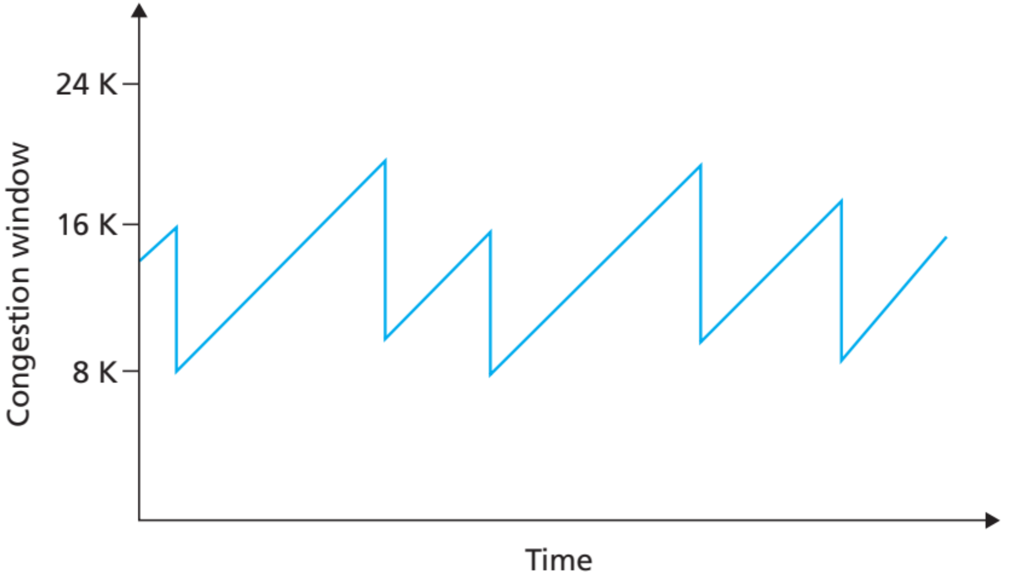


Evolution of TCP's Congestion Window (Tahoe and Reno)

- Congestion window – climbs linearly until a triple duplicate ACK event occurs just after round 8 (*cnwd = 12*MSS*)
- *ssthresh = 0.5*cnwd = 6*MSS*
- TCP Reno: *cnwd = ssthresh + 3*MSS* – grows linearly (Fast recovery phase)
- TCP Tahoe: the congestion window is set to 1 MSS and grows exponentially until it reaches the value of *ssthresh*, at which point it grows linearly.

TCP Congestion Control Algorithm: AIMD Form

- TCP congestion control displays additive-increase, multiplicative-decrease (AIMD) nature
  - Losses are indicated by triple duplicate ACKs rather than timeouts
  - *Saw tooth-like behavior*: linear (additive) increase in *cwnd* of 1 MSS per RTT ; halving (multiplicative decrease) of *cwnd* on a triple duplicate ACK; begins to increase linearly (see Fig.)



Additive Increase, Multiplicative Decrease (AIMD) Congestion Control

- TCP Throughput (Macroscopic Description)
  - Ignore initial slow start period when a connection begins (typically very short since the sender grows out of phase exponentially fast)
  - Given window size *w* bytes and the current round-trip time *RTT* secs, transmission rate = *w/RTT* bytes/sec

- TCP next probes for additional bandwidth by increasing *w* by 1 MSS per RTT until a loss event occurs
- Let *W* be the value of *w* when a loss occurs
- Assumption: *RTT* and *W* are approximately constant over the duration of connection
- TCP transmission rate ranges from *W/(2.RTT) to W/RTT*
- Average throughput of a connection = *0.75*W/RTT*