CS321: Computer Networks

Congestion Control in TCP

Dr. Manas Khatua
Assistant Professor
Dept. of CSE
IIT Jodhpur
E-mail: manaskhatua@iitj.ac.in

Causes and Cost of Congestion

- **Scenario-1**: Two Senders, a Router with Infinite Buffers
- Host A and B share a link of capacity *R*

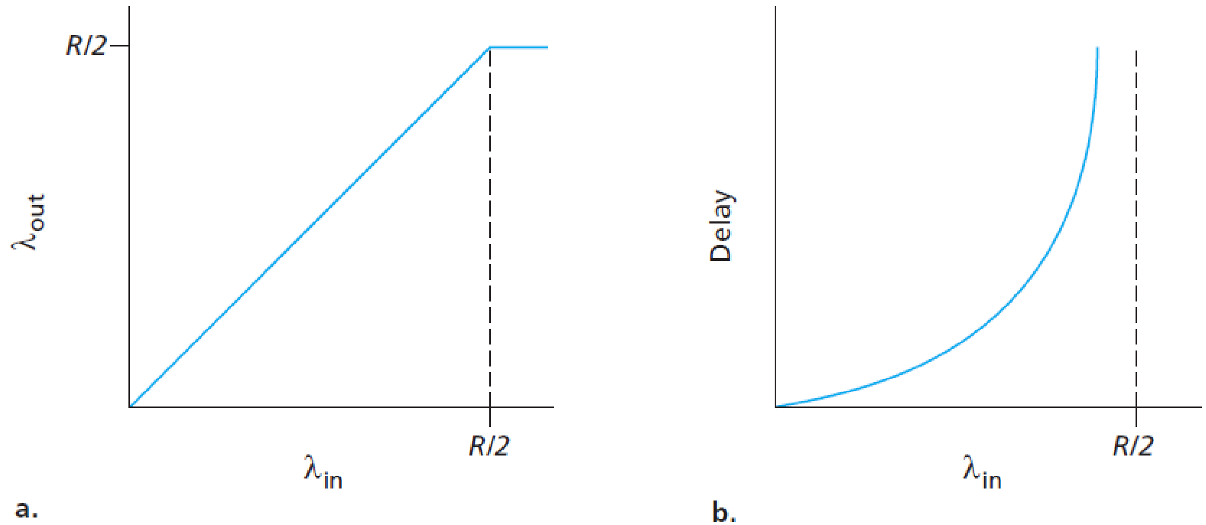


Figure 3.44 ♦ Congestion scenario 1: Throughput and delay as a function of host sending rate

- No matter how high Hosts A and B set their sending rates, they will each never see a throughput higher than *R*/2.

- When the sending rate exceeds *R*/2, the average number of queued packets in the router is unbounded, and the average delay between source and destination becomes infinite.

Cont…

- Thus, while operating at an aggregate throughput of near *R* may be ideal from a throughput standpoint, it is far from ideal from a delay standpoint.

- *Even in this (extremely) idealized scenario*
  - *one cost of a congested network—large queuing delays are experienced as the packet arrival rate nears the link capacity.*

- **Scenario-2**: Two Senders, a Router with Finite Buffers
  - Case1: Host A is able to somehow determine whether or not a buffer is free in the router and thus sends a packet only when a buffer is free.

  - Case2: the sender retransmits only when a packet is known for certain to be lost.
    - cost of a congested network— the sender must perform retransmissions in order to compensate for dropped (lost) packets due to buffer overflow

  - Case3: the sender may time out prematurely and retransmit a packet that has been delayed in the queue but not yet lost.
    - cost of a congested network—unneeded retransmissions by the sender in the face of large delays may cause a router to use its link bandwidth to forward unneeded copies of a packet.

Cont…

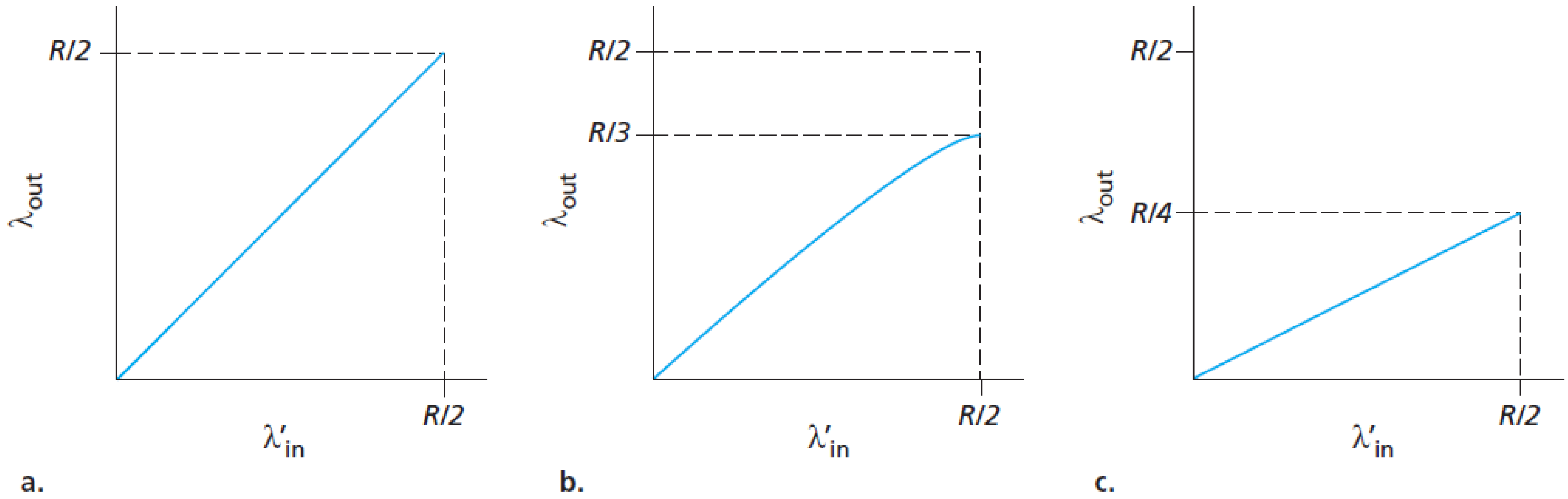
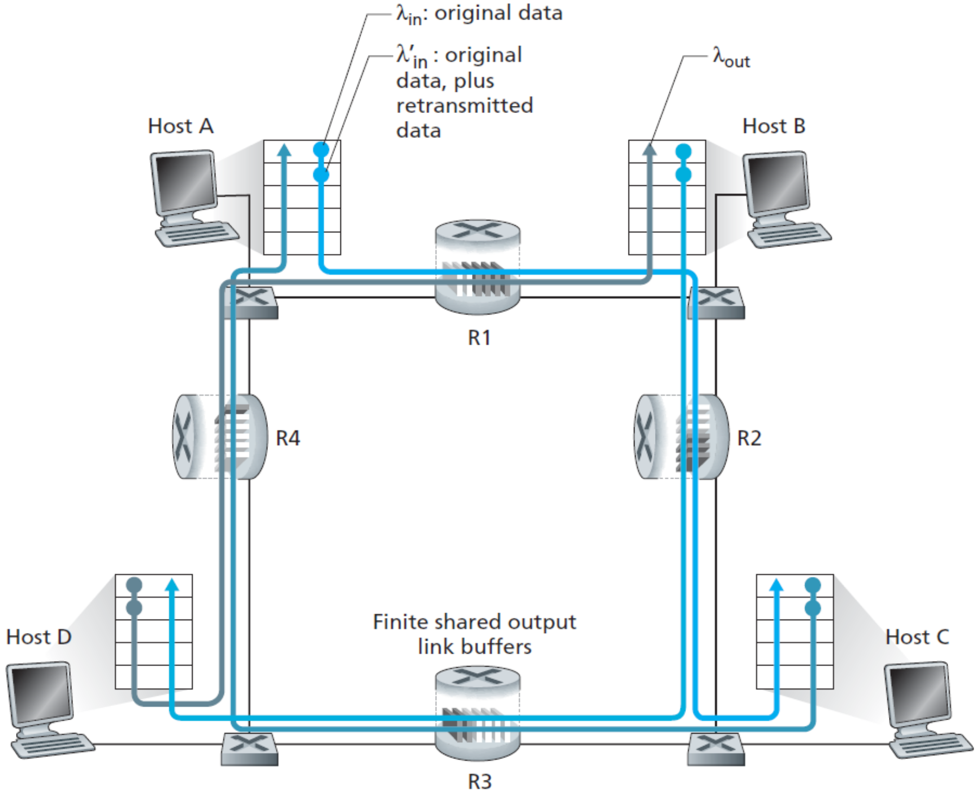
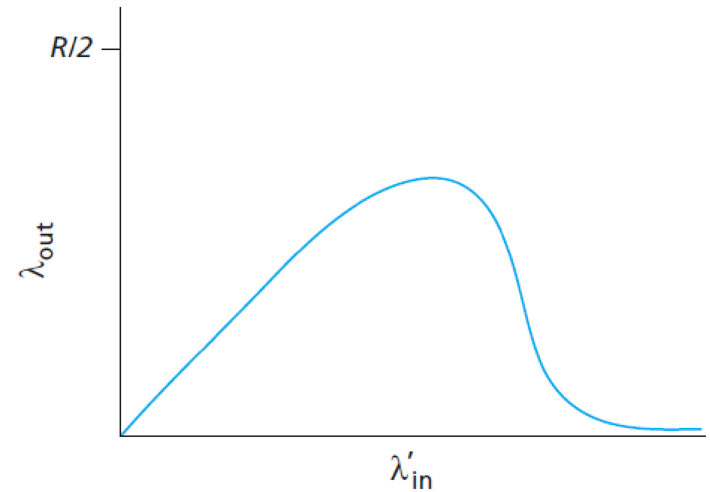


Figure 3.46 ♦ Scenario 2 performance with finite buffers

Cont…

- **Scenario 3**: Four Senders, Routers with Finite Buffers, and Multihop Paths
  - cost of congestion—when a packet is dropped along a path, the transmission capacity that was used at each of the upstream links to forward that packet to the point at which it is dropped ends up having been wasted.





Scenario 3 performance with finite buffers and multihop paths

Approaches to Congestion Control

- two broad approaches to congestion control
  - End-to-end congestion control
    - The presence of congestion in the network must be inferred by the end systems based only on observed network behavior (for example, packet loss and delay).
    - Suitable for Datagram based approach
    - TCP follows this approach
  - Network-assisted congestion control
    - Network-layer components (that is, routers) provide explicit feedback to the sender regarding the congestion state in the network.
    - Suitable for virtual-circuit based approach
    - used in ATM available bit-rate (ABR) congestion control
    - ECN based scheme for TCP/IP

    - Direct feedback: sent from a network router to the sender
    - Indirect feedback: router marks/updates a field in a packet flowing from sender to receiver to indicate congestion. Upon receipt of a marked packet, the receiver then notifies the sender of the congestion indication.

TCP Congestion Control

- The use of flow control in TCP cannot avoid congestion in intermediate routers
  - because a router may receive data from more than one sender
  - Flow control is for individual TCP sender
  - There is no congestion at the either end
  - there may be congestion in the middle.

- TCP cannot ignore the congestion in network (at the intermediate points) as it wants to provide end-to-end reliability

- TCP must use end-to-end congestion control rather than network-assisted congestion control

- Basic approach for Congestion control in TCP:
  - each sender limit the rate at which it sends traffic into its connection as a function of perceived network congestion.

Cont…

- If a TCP sender perceives less congestion on the path between itself and the destination
  - the TCP sender increases its send rate
- if the TCP sender perceives huge congestion along the path
  - the TCP sender reduces its send rate
- It should not aggressively send segments to the network
- It cannot be very conservative, either, sending a small number of segments in each time interval
- **Questions** need to answer:
  - How does a TCP sender limit the rate at which it sends traffic into its connection?
  - How does a TCP sender perceive that there is congestion on the path between itself and the destination?
  - What congestion control algorithm should the sender use to change its send rate as a function of perceived end-to-end congestion?

Answers

Answer of 1st Question:

- To control the number of segments to transmit, TCP uses another variable called Congestion Window (*cwnd*)

- Actually, the *cwnd* variable and the *rwnd* variable (used for flow control) together define the size of the send window in TCP

  – Actual send window size = minimum *(rwnd, cwnd)*

- The constraint above limits the amount of unacknowledged data at the sender and therefore indirectly limits the sender's send rate.

Answer of 2nd Question:

- TCP sender uses the occurrence of two events as signs of congestion:
  – time-out
  – receiving three duplicate ACKs

Cont…

Answer of 3rd Question:

- There exist many congestion control algorithm for adjusting the value of *cwnd* based upon end-to-end congestion

- Modified TCP with congestion control algorithms
  - Taho TCP: both signs of occurrence are treated equally
  - Reno TCP: both signs of occurrence are treated differently
  - New Reno TCP: TCP checks to see if more than one segment is lost in the current window when three duplicate ACKs arrive

- Further Issues:
  - If TCP senders collectively send too fast, they can congest the network, leading to congestion collapse
  - if TCP senders are too cautious and send too slowly, they could under utilize the bandwidth in the network;
  - there is no explicit signaling of congestion state by the network — ACKs and loss events serve as implicit signals — and that each TCP sender acts on local information asynchronously from other TCP senders

- TCP congestion-control algorithm has three major components
  - (1) slow start
  - (2) congestion avoidance
  - (3) fast recovery

Slow Start

- When a TCP connection begins
  - the value of *cwnd* is typically initialized to a small value of 1 MSS,
  - resulting in an initial sending rate of roughly MSS/RTT.

    (MSS: maximum-sized segments; default value is 536 octets)

- In the **slow-start** state, the value of *cwnd* begins at 1 MSS and increases by 1 MSS every time a transmitted segment is first acknowledged.

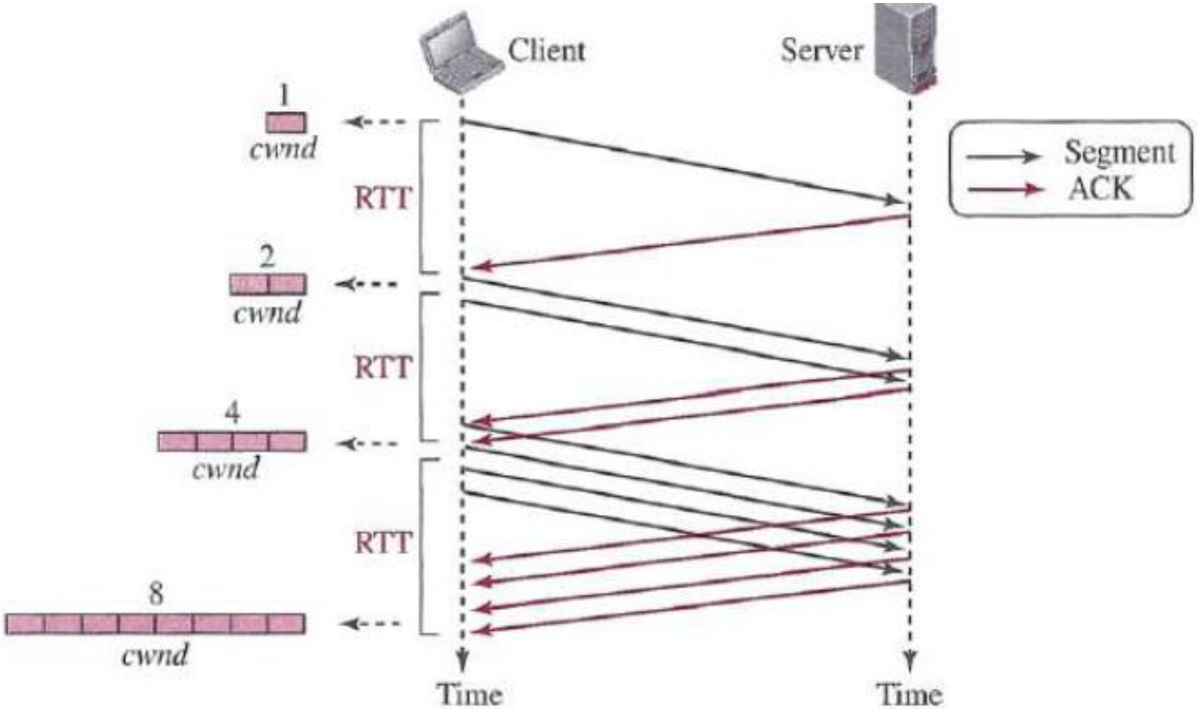
- TCP sends the first segment into the network and waits for an ACK.
- When this ACK arrives, the TCP sender increases the *cwnd* by 1 MSS and sends out 2 MSS
- These segments are then ACKed, with the sender increasing the *cwnd* by 1 MSS for each of the ACKed segments, giving a *cwnd* of 4 MSS, and so on.

- This process results in a doubling of the sending rate every RTT.

- Thus, the TCP send rate starts slow but grows exponentially during the slow start phase.

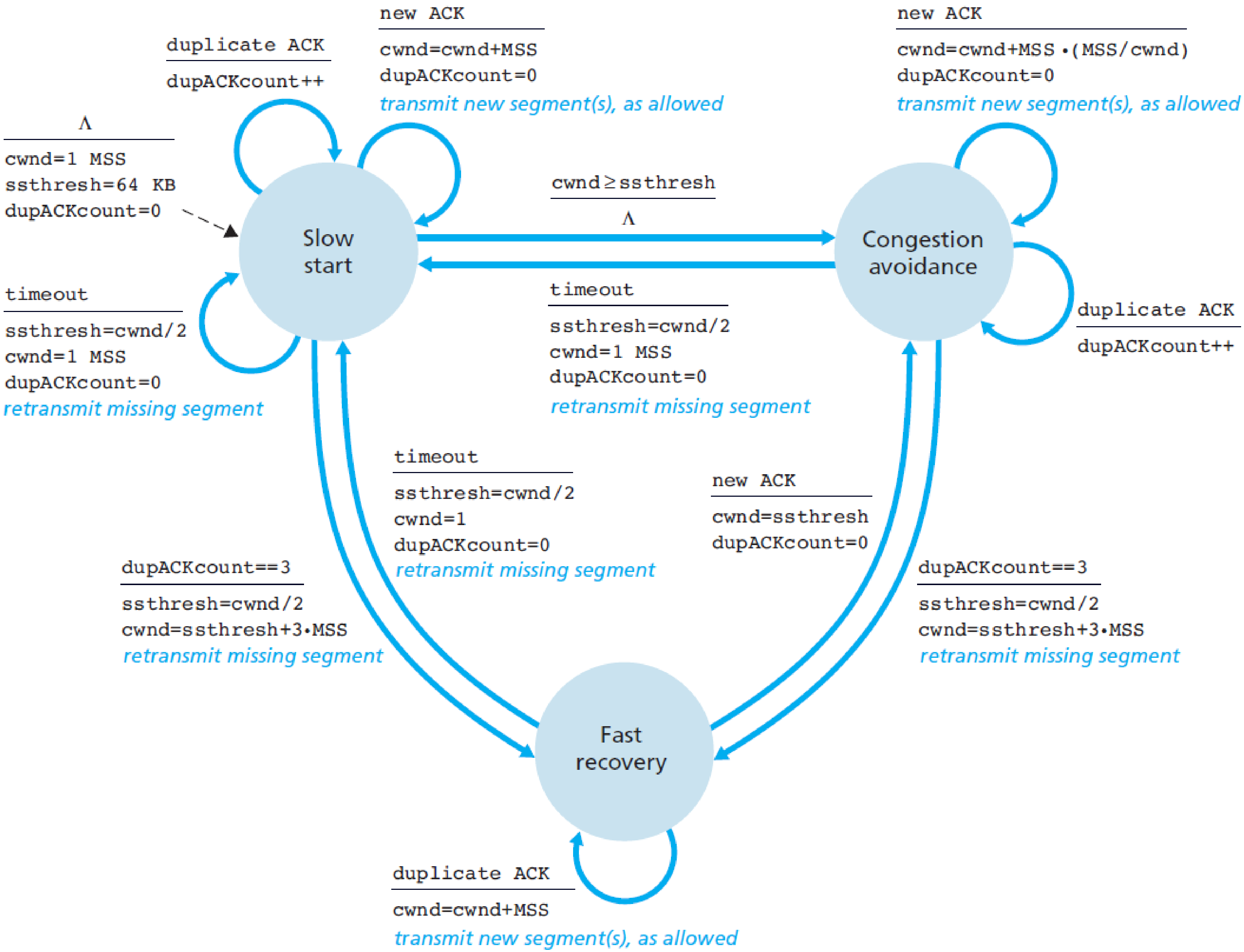
Cont…

- the size of the *cwnd* increases exponentially until it reaches a threshold
- the size of the *cwnd* is determined as follows:
  - If an ACK arrives, *cwnd* = *cwnd* + 1.



Cont…

- But when should this exponential growth end?
- **Answer:**
  - First, if there is a loss event (i.e., congestion) indicated by a timeout,
    - the TCP sender sets the value of *cwnd* to 1
    - begins the slow start process anew.
    - sets the value of *ssthresh* (slow start threshold) to *cwnd/2*.
  - Second, when the value of *cwnd* equals *ssthresh*,
    - TCP transitions into congestion avoidance state
  - Third, if three duplicate ACKs are detected,
    - TCP performs a fast retransmit and enters the fast recovery state
    - sets the value of *ssthresh* to *cwnd/2*.
    - sets the value of *cwnd* to *ssthresh + 3 MSS*.
- Slow-start strategy is slower in the case of delayed ACK.
- If two segments are ACKed cumulatively, the size of the *cwnd* increases by 1, not 2. With one ACK for every two segments, the growth is a power of 1.5, but still exponential



F
S
M

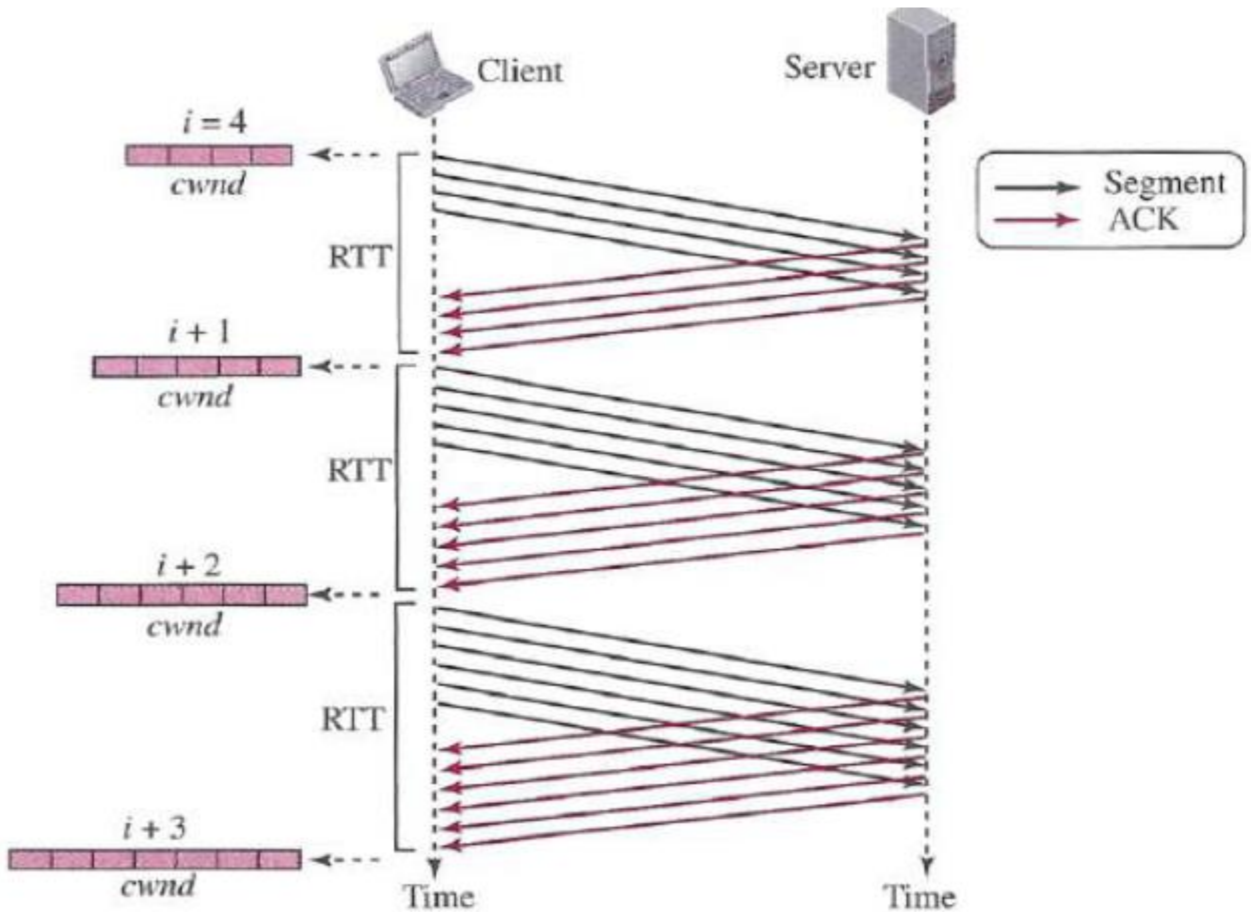
Figure 3.52 ♦ FSM description of TCP congestion control

Congestion Avoidance

- On entry to the congestion-avoidance state, the value of *cwnd* is approximately half its value when congestion was last encountered
- To avoid congestion before it happens, we must slow down the exponential growth of *cwnd*
- When the size of the *cwnd* reaches the *ssthresh* (slow-start threshold), the slow-start phase stops and the additive phase begins.
- increase the *cwnd* additively instead of exponentially.
- If three duplicate ACKs are detected, TCP performs a fast retransmit and enters the fast recovery state

Cont…

- If a new ACK arrives, cwnd = cwnd + (1/ cwnd)



Fast Recovery

- this algorithm is also an additive increase, but it starts when three duplicate ACK arrives

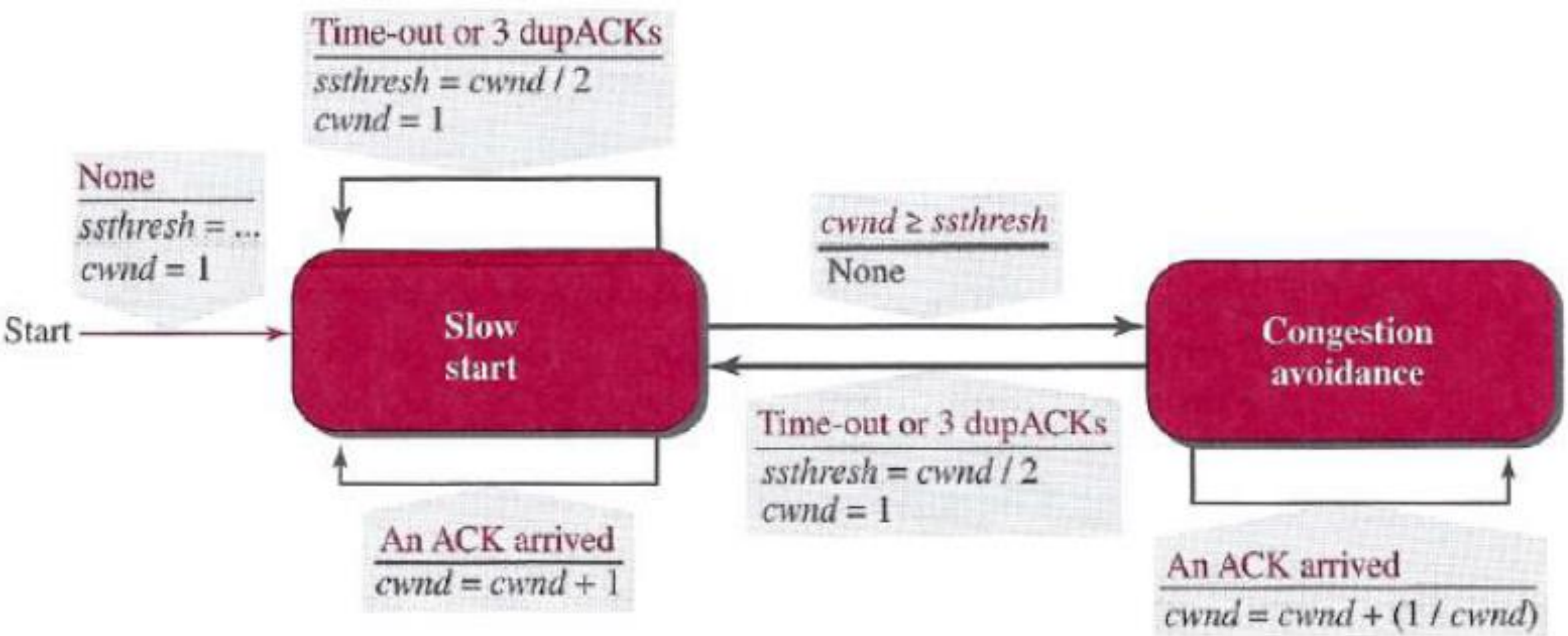
- If a duplicate ACK arrives (after the three duplicate ACK which triggers the recovery)
  - $cwnd = cwnd + (1/ cwnd)$

- If timeout occurs, TCP moves back to slow start state
- If any new ACK arrives, TCP moves back to congestion avoidance state

- This state is recommended, but not mandatory in TCP

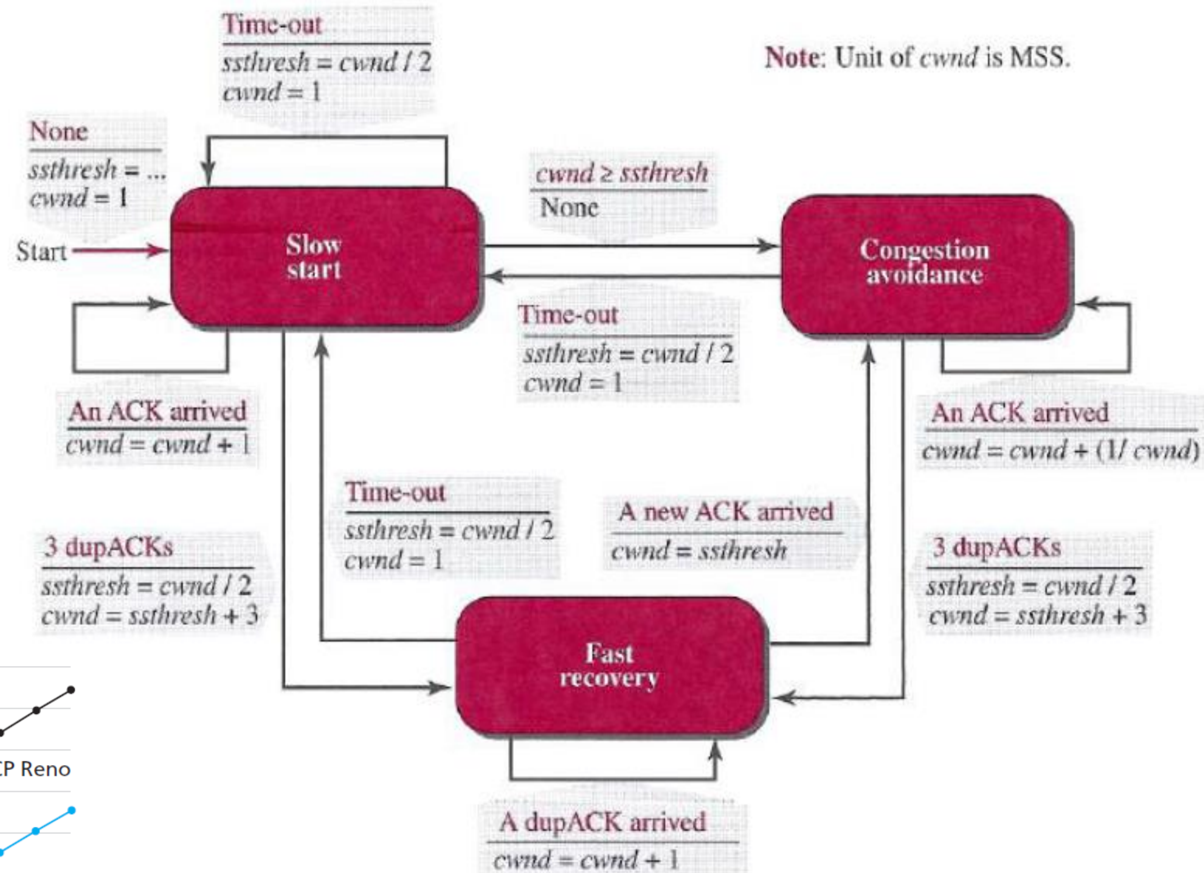
TCP Tahoe

- In TCP Tahoe
  - both signs of congestion occurrence (time-out, 3 duplicate ACK) are treated equally
  - uses only *slow start* and *congestion avoidance* states



TCP Reno

Incorporated the
Fast Recovery State



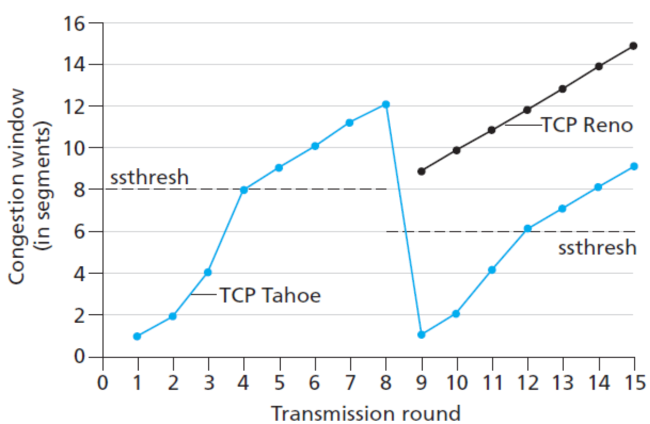


Figure 3.53 ♦ Evolution of TCP's congestion window (Tahoe and Reno)

TCP New Reno

- It differs from RENO in that it doesn't exit fast-recovery until all the data which was outstanding at the time it entered fast recovery is acknowledged.

- TCP New Reno version is most common today

- If we ignore the slow-start states at the beginning and the loss of segment is inferred by 3 duplicate ACK,
  - the TCP congestion window is $cwnd = cwnd + (1/cwnd)$ when an ACK arrives
  - $cwnd = cwnd / 2$ when congestion is detected

- It appears like Additive Increase Multiplicative Decrease (AIMD). Therefore, TCP Congestion control scheme is often referred as AIMD scheme.

TCP Vegas

- variations of the Reno algorithm
- TCP Vegas algorithm attempts to avoid congestion while maintaining good throughput
- The basic idea of Vegas is to
  - (1) detect congestion in the routers between source and destination *before* packet loss occurs, and
  - (2) lower the rate linearly when this imminent packet loss is detected.

TCP Throughput

- it's natural to consider what the average throughput of a long-lived TCP connection might be?
- we'll ignore the slow-start phases that occur after timeout events as these phases are typically very short.
- During a particular round-trip interval, the rate at which TCP sends data is a function of the congestion window (*cwnd*) and the current *RTT*
- Let, *cwnd* = *W* when a loss event occurs.
- Assuming that *RTT* and *W* are approximately constant over the duration of the connection, the TCP transmission rate ranges from 0.5 (*W* /*RTT*) to (*W* /*RTT*).
- Steady-state TCP throughput is the average throughput of a connection = 0.75*(W/RTT)

Thanks!