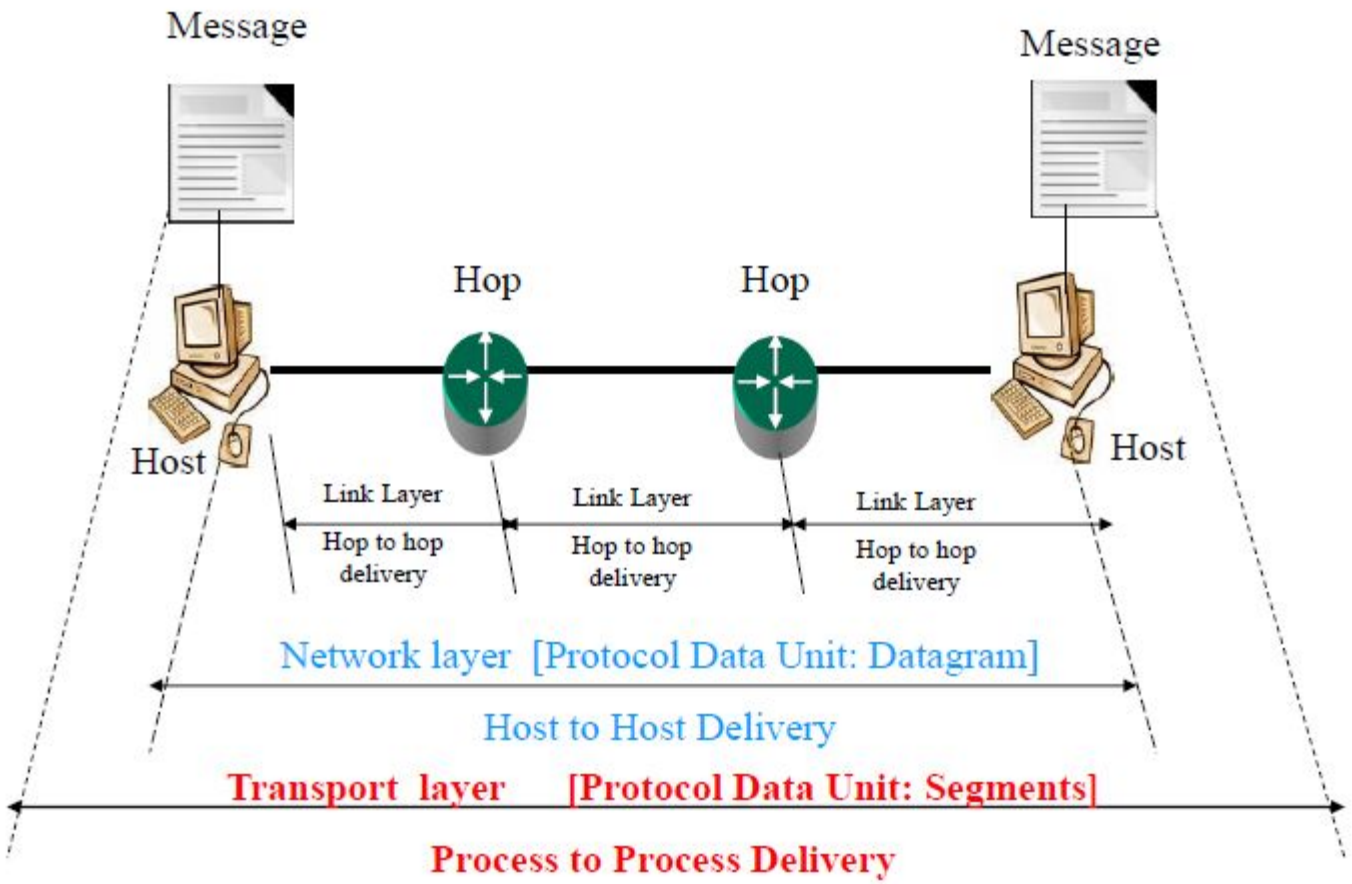
Computer Networks & IOT (18B11CS311)

Even Semester_2024

Transport Layer


Message
Message
Hop
Hop
Host
Host
Link Layer
Hop to hop delivery
Link Layer
Hop to hop delivery
Link Layer
Hop to hop delivery
Network layer [Protocol Data Unit: Datagram]
Host to Host Delivery
Transport layer [Protocol Data Unit: Segments]
Process to Process Delivery

Note:

Some points about TCP's sliding windows:

- *The size of the window is the lesser of rwnd and cwnd.*
- *The source does not have to send a full window's worth of data.*
- *The window can be opened or closed by the receiver, but should not be shrunk.*
- *The destination can send an acknowledgment at any time as long as it does not result in a shrinking window.*
- *The receiver can temporarily shut down the window; the sender, however, can always send a segment of one byte after the window is shut down.*

15-8 ERROR CONTROL

TCP is a reliable transport layer protocol. This means that an application program that delivers a stream of data to TCP relies on TCP to deliver the entire stream to the application program on the other end in order, without error, and without any part lost or duplicated. Error control in TCP is achieved through the use of three tools: *checksum*, *acknowledgment*, and *time-out*.

Error Control

TCP provides reliability using error control

- Error Control :
  - Error DETECTION + Error CORRECTION

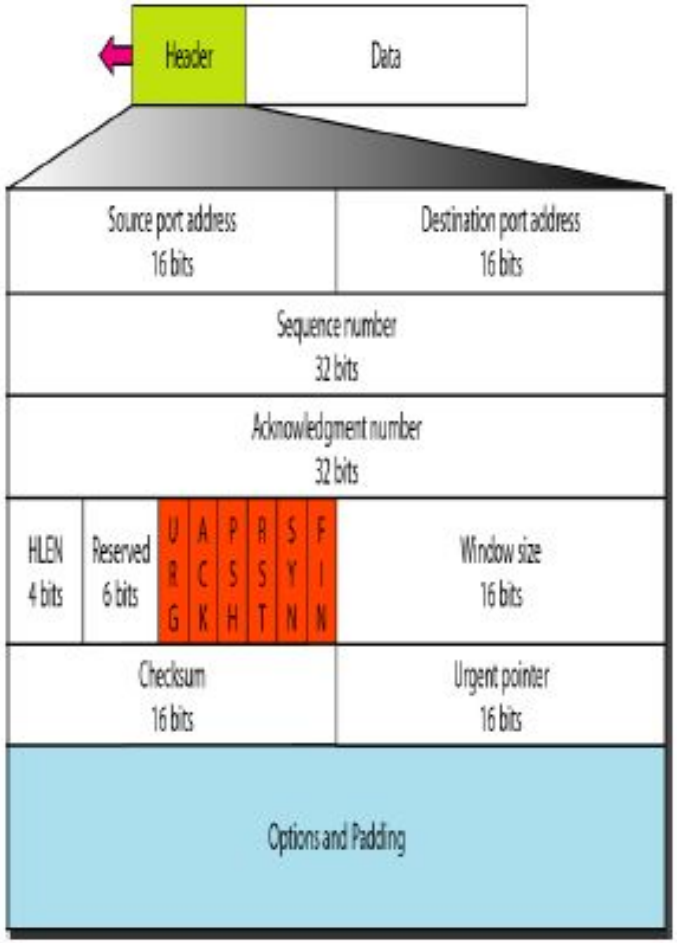
- Error control includes mechanisms for detecting:
  - corrupted segments.
  - lost segments.
  - out-of-order segments.
  - Duplicated segments.

Error Control

- Error detection and correction in TCP is achieved through the use of three simple tools:
  - Checksum.
  - Acknowledgment.
  - Time-out.

1. Checksum

- Each segment includes a checksum field which is used to check for a corrupted segment.
- If the segment is corrupted, it is discarded by the destination TCP and is considered as lost.
- TCP uses a 16-bit checksum that is mandatory in every segment.



Acknowledgement Type

- In the past, TCP used only one type of acknowledgement: Accumulative Acknowledgement (ACK), also namely accumulative positive acknowledgement
- More and more implementations are adding another type of acknowledgement: Selective Acknowledgement (SACK), SACK is implemented as an option at the end of the TCP header.

Rules for Generating ACK (1)

- 1. When one end sends a data segment to the other end, it must include an ACK. That gives the next sequence number it expects to receive. (Piggyback)
- 2. The receiver needs to delay sending (until another segment arrives or 500ms) an ACK segment if there is only one outstanding in-order segment. It prevents ACK segments from creating extra traffic.
- 3. There should not be more than 2 in-order unacknowledged segments at any time. It prevent the unnecessary retransmission

Rules for Generating ACK (2)

- 4. When a segment arrives with an out-of-order sequence number that is higher than expected, the receiver immediately sends an ACK segment announcing the sequence number of the next expected segment. (for fast retransmission)
- 5. When a missing segment arrives, the receiver sends an ACK segment to announce the next sequence number expected.
- 6. If a duplicate segment arrives, the receiver immediately sends an ACK.

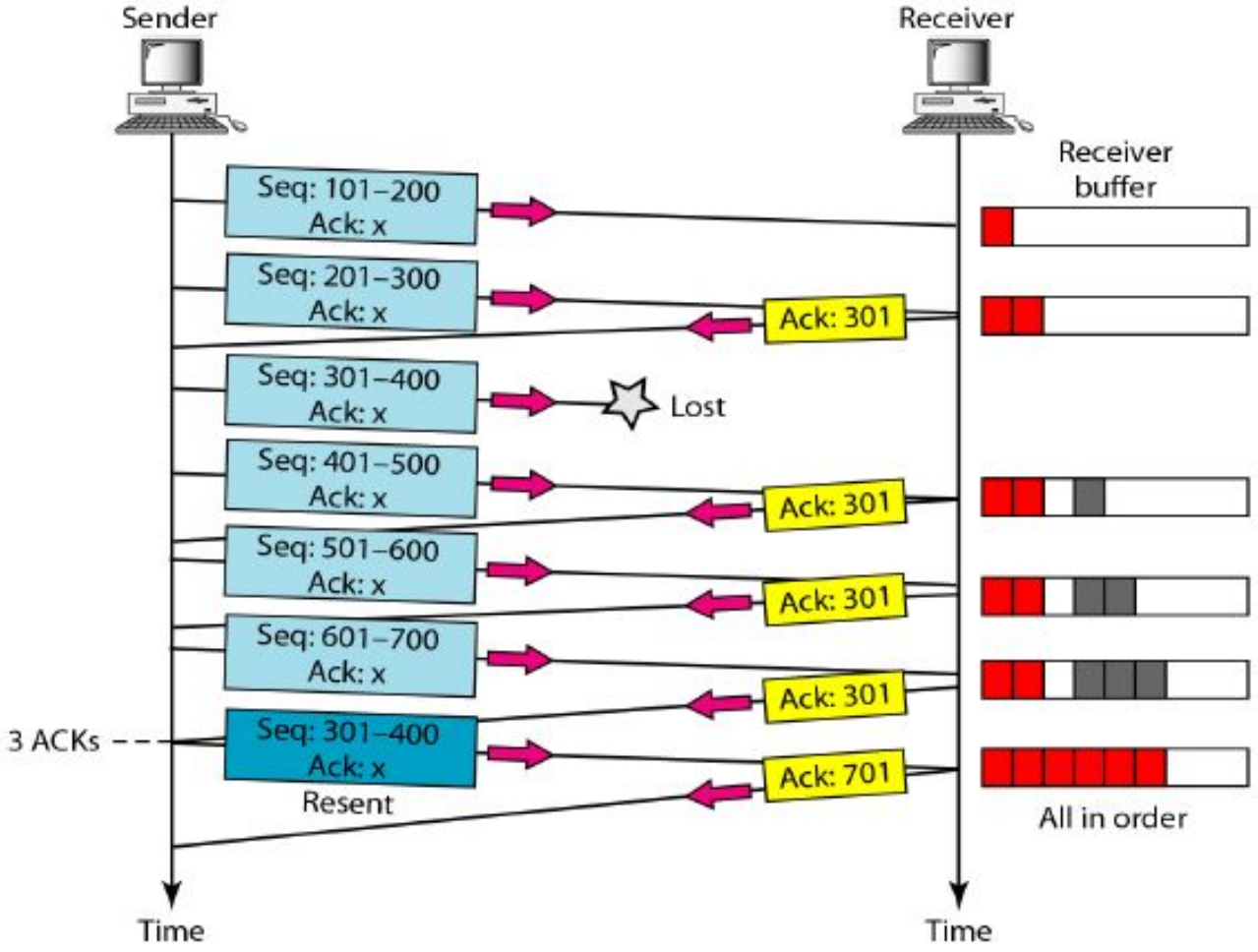
3.Retransmission(Time out)

- A recent implementation of TCP maintains one retransmission time-out (RTO) timer for all outstanding segments.
- When the timer matures, the earliest outstanding segment is retransmitted.
- the lack of a received ACK can be due to a delayed segment, a delayed ACK, or a lost acknowledgment.
- Note that no time-out timer is set for a segment that carries only an acknowledgment.
- The value of RTO is dynamic in TCP and is updated based on the round-trip time (RTT) of segments.
- An RTT is the time needed for a segment to reach a destination and for an acknowledgment to be received

ERROR CONTROL
fast retransmission

- The previous rule about retransmission of a segment is sufficient if the value of RTO is not very large.
- Sometimes one segment is lost and the receiver receives so many out-of-order segments that they cannot be saved (limited buffer size).
- To alleviate this situation, most implementations today follow the three-duplicate-ACKs rule and retransmit the missing segment immediately.

Fast retransmission



ERROR CONTROL
out-of-order segment

- Data may arrive out of order and be temporarily stored by the receiving TCP.
- TCP store them temporarily and flag them as out-of-order segments until the missing segment arrives.
- TCP guarantees that no out-of-order segment is delivered to the process.

Note

Data may arrive out of order and be temporarily stored by the receiving TCP, but TCP guarantees that no out-of-order data are delivered to the process.

Few Scenarios

Figure 15.29 *Normal operation*

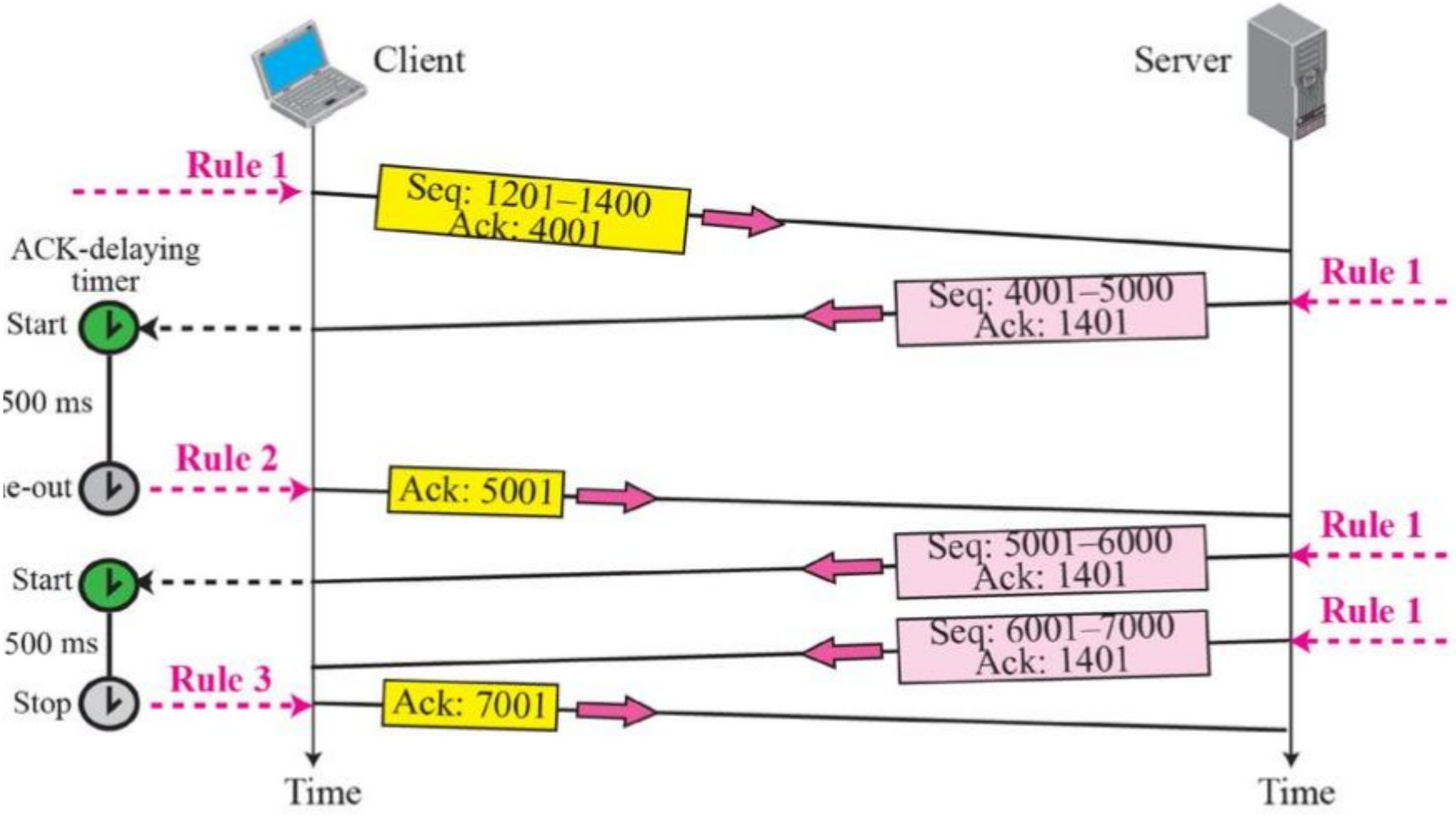
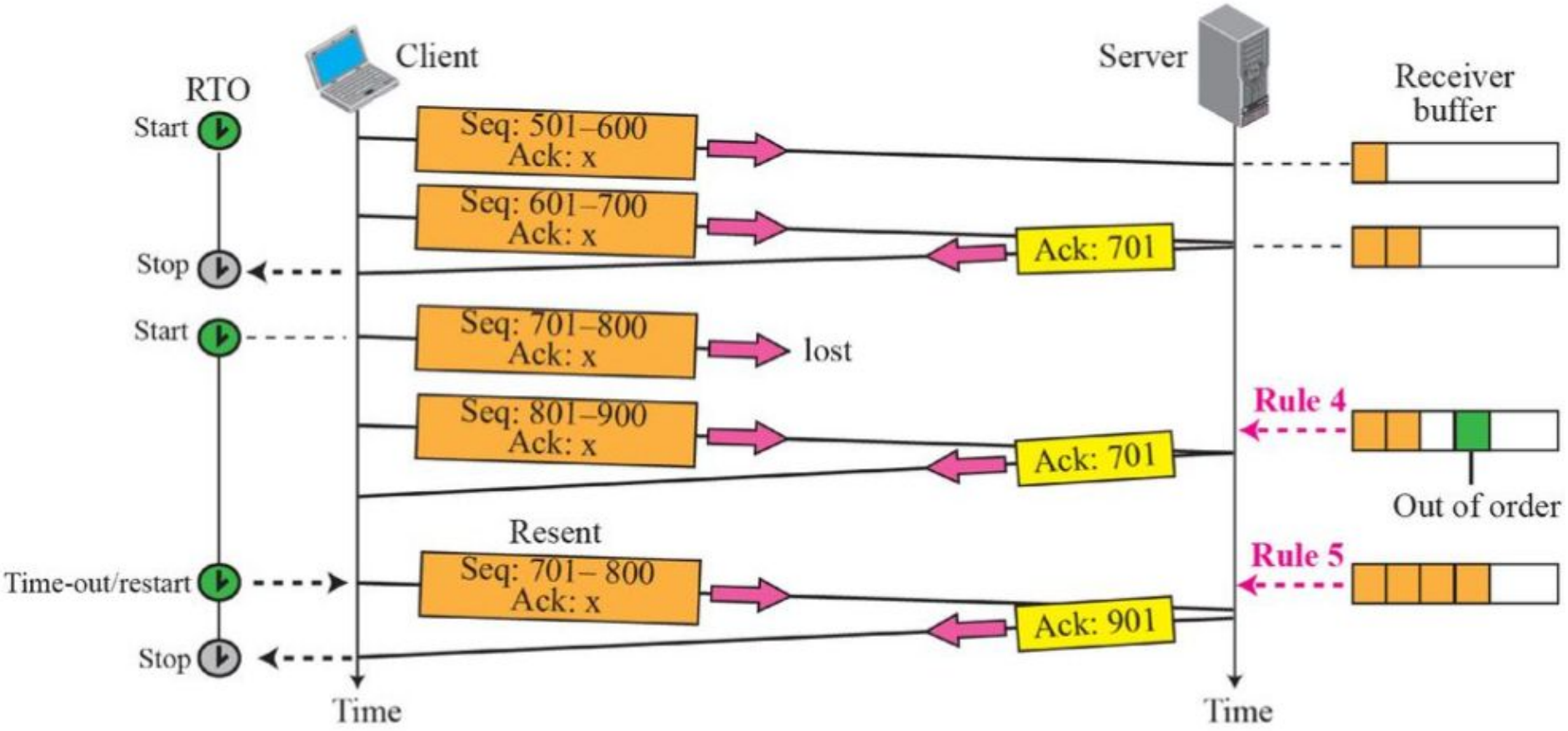


Figure 15.30 *Lost segment*



Note

The receiver TCP delivers only ordered data to the process.

Figure 15.32 *Lost acknowledgment*

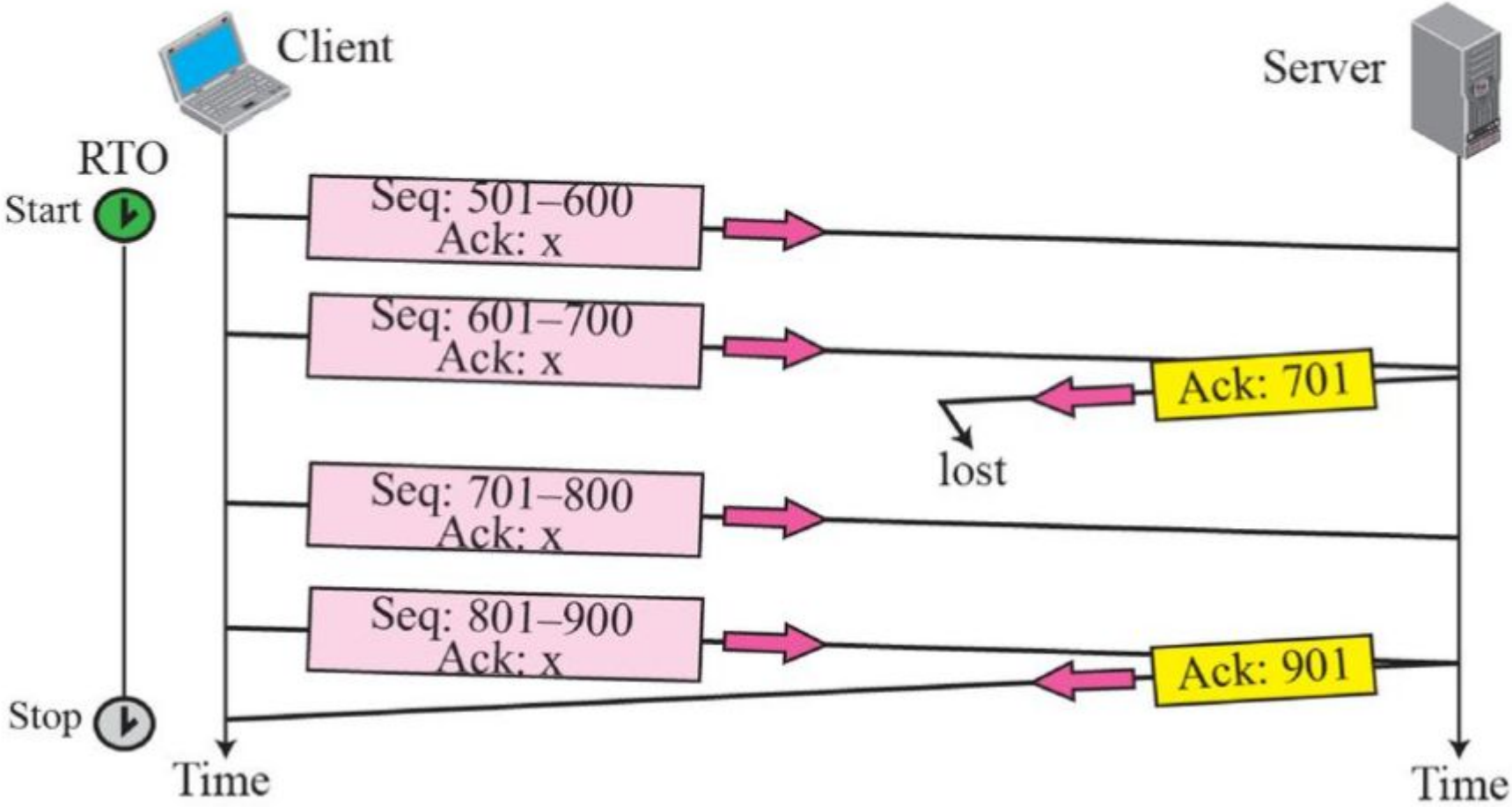
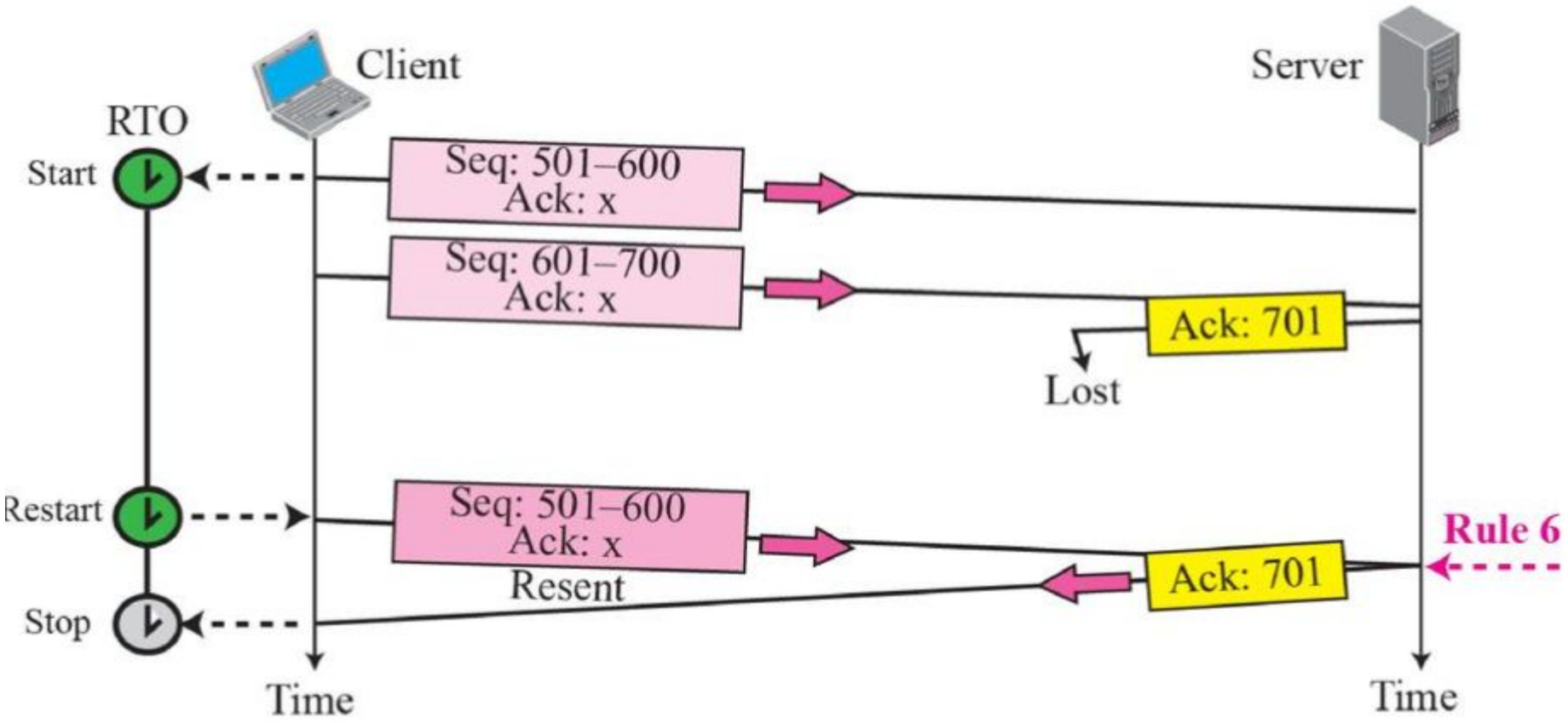


Figure 15.33 *Lost acknowledgment corrected by resending a segment*



Note

Lost acknowledgments may create deadlock if they are not properly handled.

- Reciever: rwnd=0 and sender shutdown
- Receiver: rwnd=nonzero and this ack lost

15-9 CONGESTION CONTROL

Congestion control in TCP is based on both open loop and closed-loop mechanisms. TCP uses a congestion window and a congestion policy that avoid congestion and detect and alleviate congestion after it has occurred.

- ✓ Congestion Window
- ✓ Congestion Policy

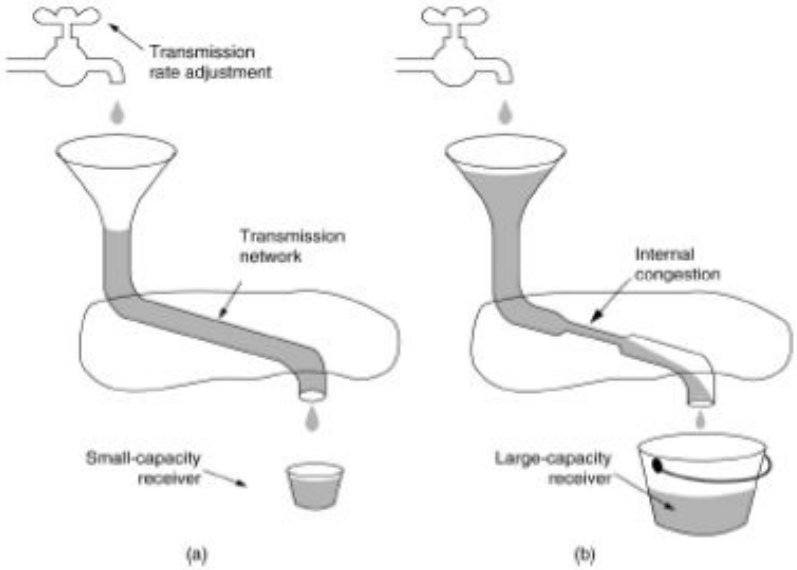
Congestion Control

Congestion control refers to the mechanisms and techniques to keep the load below the capacity.

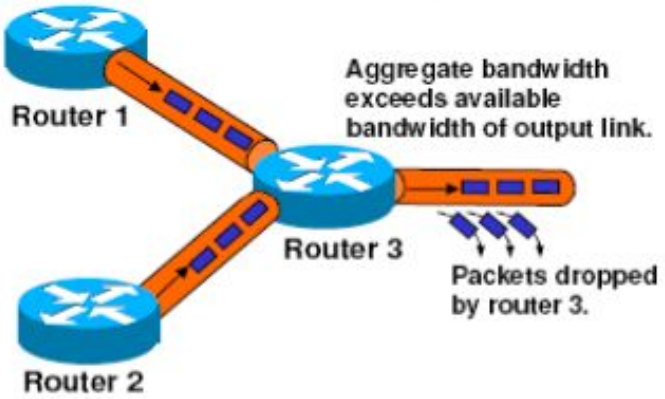
TCP Congestion Control

➔ Congestion control is a control (feedback control) mechanism to prevent congestion (congestion avoidance).

Problem (a): A fast network feeding a low capacity receiver (congestion in receiver).
Problem (b): A slow network feeding a high-capacity receiver (congestion in network).



Congestion can always occur (aggregate ingress Bandwidth exceeds egress bandwidth).



- Prior to 1988 TCP did not define a congestion control mechanism.
- TCP stacks just sent as many TCP segments as the receiver's buffer could hold (based on advertise window).
- With the growth of the Internet router links became congested and thus buffers got full which resulted in packet loss.
- Packet loss lead to retransmissions which in turn aggravated the congestion problem.
- It became necessary to avoid congestion in the first place (it's always better to avoid a problem in the first place rather than cure it!).

How to avoid congestion efficiently (ideas of Van Jacobson):

1. Sender can detect packet loss and interpret it as network congestion (this is in most cases true; packet loss can however also be due to bit errors).
2. When a sender detects packet loss it should reduce the transmission rate quickly.
3. When there is no (more) congestion (no more packet loss) the sender can slowly increase the transmission rate.

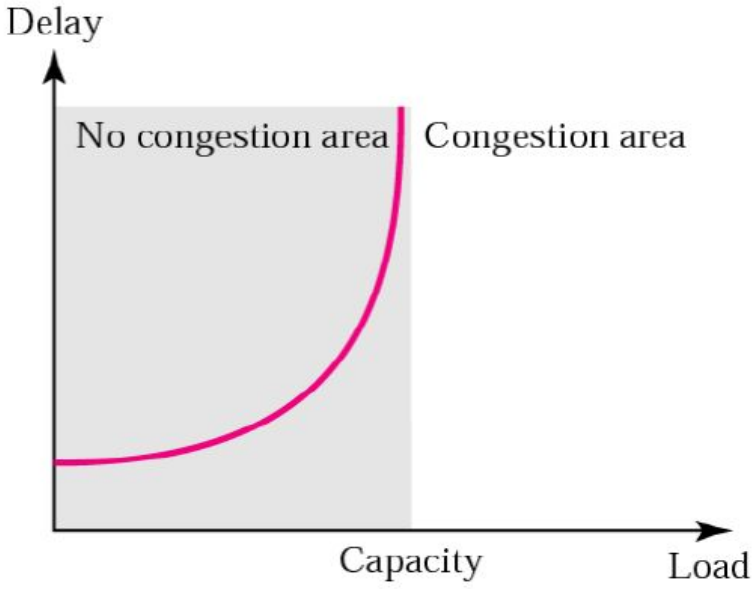
What does TCP do?

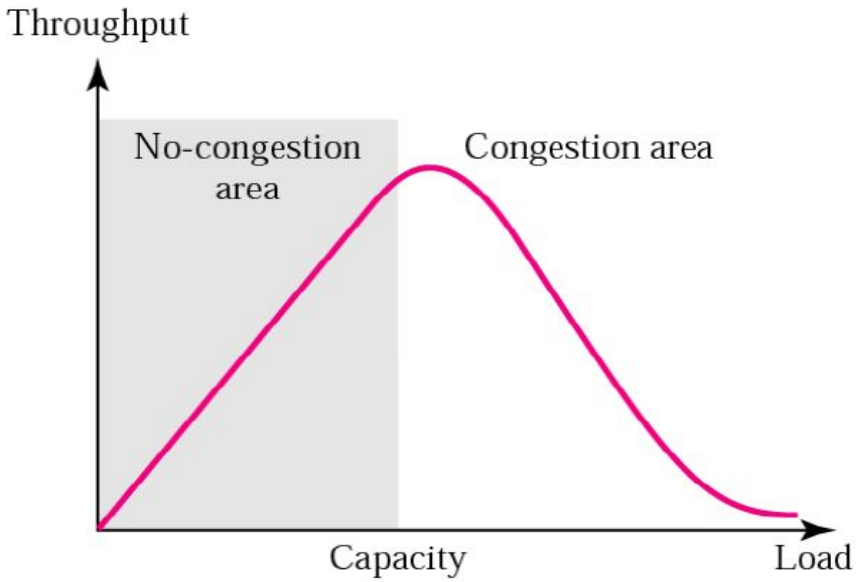
- At the very beginning of a TCP session a sender does not know the maximum possible transmission rate yet. Thus it should start slowly with sending segments (slow start procedure).
- If a sender receives an ACK packet it knows that a segment has been received and is no longer on the network (in transmission). This fact is used for the slow start algorithm. This procedure is self-clocking.

Network Performance

Congestion control involves two factors that measure the performance of a network:

1. Delay
2. Throughput


Delay
No congestion area
Congestion area
Capacity
Load


Throughput
No-congestion area
Congestion area
Capacity
Load

Network Performance

- Congestion control involves two factors that measures the performance of the network:
  - delay and throughput
- Delay versus load
  - When load is much less than capacity, the delay is at minimum. Minimum delay is due to propagation delay and processing delay.
  - When load reaches the network capacity, the delay increases sharply due to addition of waiting time in the queues.
  - Delay has negative effect on the load and consequently the congestion. When a packet is delayed, the source, not receiving the acknowledgement, retransmits the packet, which makes the delay, and the congestion, worse.

Performance: Throughput vs network load

- We can define throughput in a network as the number of packets passing through the network in a unit of time.
- When the load is below the capacity of the network, the throughput increases proportionally with the load.
- The throughput declines sharply after the load reaches to its capacity due to discarding of packets by routers.
- When the load exceeds the capacity, the queues become full and routers have to discard some packets.
- Discarding packets does not reduce the number of packets in the network because the sources retransmit the packets, using time-out mechanisms, when the packets do not reach the destination.

Congestion Control

- Congestion control refers to techniques and mechanisms that can either prevent congestion, before it happens, or remove congestion, after it has happened.

1. Open-loop congestion control [Prevention]
2. Closed-loop Congestion Control [Removal]

- Open-loop congestion control [Prevention]
  - Policies are applied to prevent congestion before it happens.
  - Congestion control is handled by either the source or the destination.
  - Retransmission policy: to optimize efficiency and reduce congestion set proper retransmission policy and timers.

- Open-loop congestion control [Prevention]
  - Windows policy: Type of window at sender may also affect congestion. Selective repeat is better than Go-Back-N.
  - Acknowledgement policy: Policy set by receiver may also affect congestion. If receiver does not acknowledge every packet it receives, it may slow down the sender and help prevent congestion.
  - Discard policy: Discard less sensitive packets [in audio transmission] at routers.
  - Admission policy: Switches in a flow first check the resource requirement of a flow before admitting it to the network.

http://www.brainkart.com/article/Congestion-Control--Open-Loop-and-Closed-Loop_13488/

Closed-loop congestion control [Removal]

- Back Pressure: When a router is congested, it can inform the previous upstream router to reduce the rate of outgoing packets. The action can be recursive all the way to the router before the source.
- Choke Point: A packet sent by a router to the source to inform it of congestion. This type of control is similar to ICMP's source quench packet.
- Implicit Signaling: Source can detect an implicit signal concerning congestion and slow down its sending rate. For example, the mere delay in receiving an acknowledgement can be a signal that the network is congested.
- Explicit Signaling: Routers that experience congestion can send an explicit signal, the setting of a bit in a packet, for example, to inform the sender or the receiver of congestion.
  - Backward Signaling: Bit can be set in a packet moving in the direction opposite to the congestion; indicate the source.
  - Forward Signaling: Bit can be set in a packet moving in the direction of the congestion; indicate the destination.

Congestion Control in TCP

- Sender's window size is not only determined by the receiver, but also by the congestion in the network
- Sender maintains 2 window sizes:
  - Receiver-advertised window (RWND)
  - Congestion window (CWND)
- Actual window size = min(RWND, CWND)

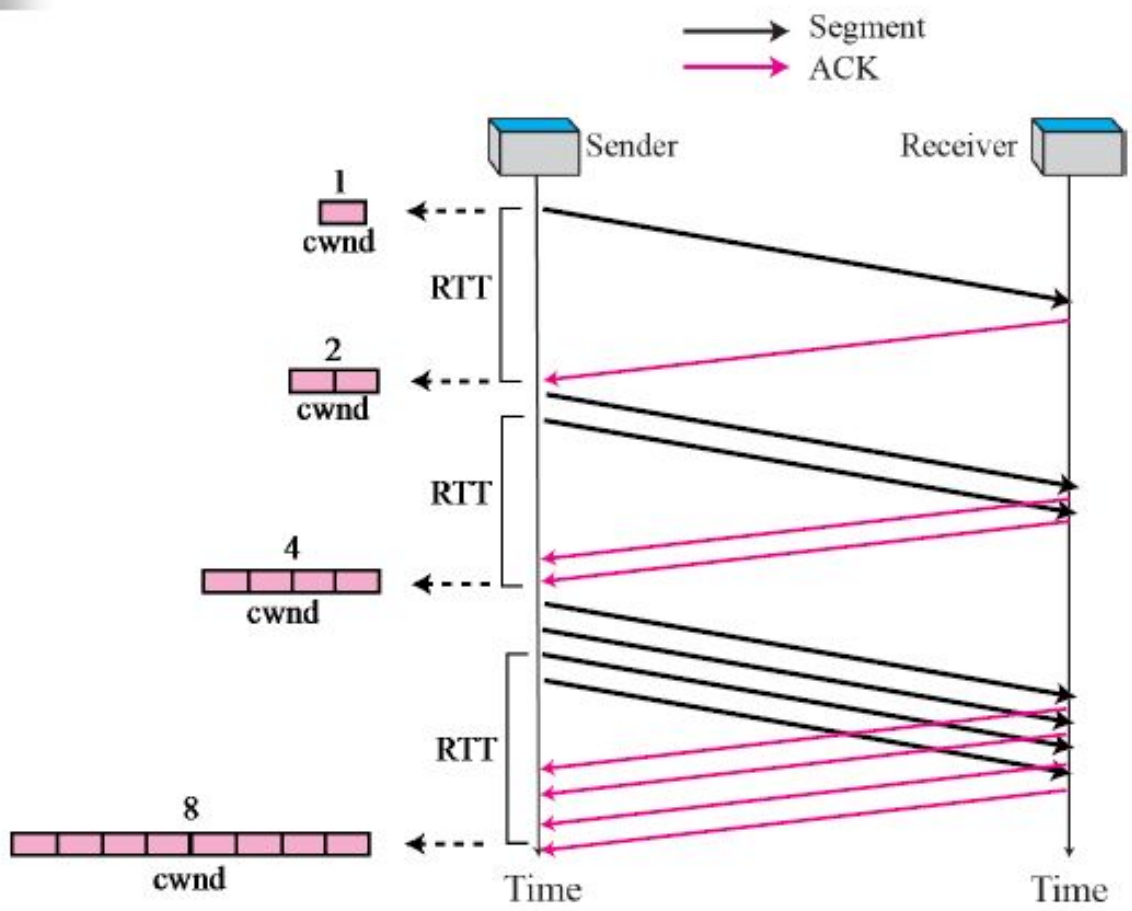
- To deal with congestion, TCP is based on several strategies or phases:
  - Slow Start
  - Additive Increase of CWND : Congestion Avoidance
  - Multiplicative Decrease of CWND: Congestion Detection

1. Slow Start

All TCP implementations are now required to implement slow start.

- At beginning of connection, CWND=MSS
- For each ACKed segment, CWND is increased by one MSS
  - Until a threshold of half allowable window size is reached
- Process not slow at all ! window size grows exponentially !
  - CWND = 1
  - Send 1 segment
  - Receive 1 ACK, and increase CWND to 2
  - Send 2 segments
  - Receive ACK for 2 segments, and increase CWND to 4

Slow start, exponential increase



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

Size of cwnd in terms of RTT

Start	⟹	cwnd = 1	
After 1 RTT	⟹	cwnd = 1 × 2 = 2	⟹ 2^1
After 2 RTT	⟹	cwnd = 2 × 2 = 4	⟹ 2^2
After 3 RTT	⟹	cwnd = 4 × 2 = 8	⟹ 2^3

- Slow start cannot continue indefinitely.

There must be a threshold to stop this phase i.e. ssthresh (slow start threshold).

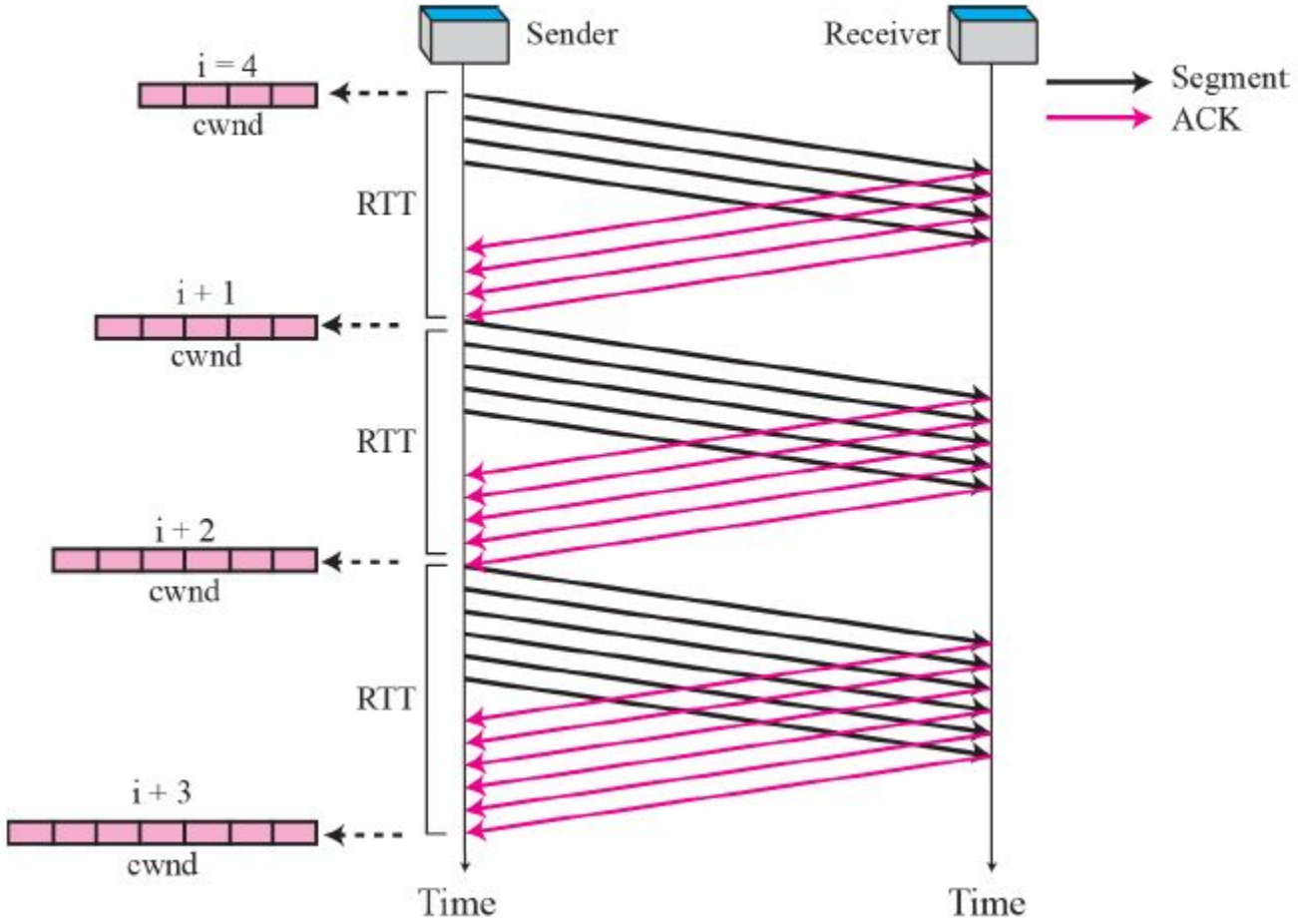
- When the size of the window reaches this threshold slow start stops and next phase starts.

  *Value of ssthresh is 65535 bytes.

2. Congestion Avoidance: Additive Increase

- Slow start will increase CWND size exponentially until threshold value is reached
- To avoid congestion this exponential growth must slow down:
  - After the threshold is reached, CWND will be increased additively (one segment per ACK, even if several segments are ACKed)
  - This linear growth will continue until either:
    - Receiver-advertised window size is reached, or
    - ACK timeout occurs → TCP assumes congestion

Congestion avoidance, additive increase



After the cwnd reaches the threshold, the size of the congestion window increases additively until the congestion is detected

Start	→ cwnd=1
After 1 RTT	→ cwnd=2
After 2 RTTs	→ cwnd =3

....

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

3. Congestion Detection: Multiplicative Decrease

- If congestion occurs, the congestion window size must be decreased.
- Sender guess the congestion has occurred if there is a need to retransmit a segment.
- Retransmission can occurs in one of two cases:
  1. When RTO timer times out
  2. When three ACKs are received.

- When RTO timer times out:

a. It sets the value of the threshold to half of the current window size.
b. It sets cwnd to the size of one segment.
c. It starts the slow start phase again.

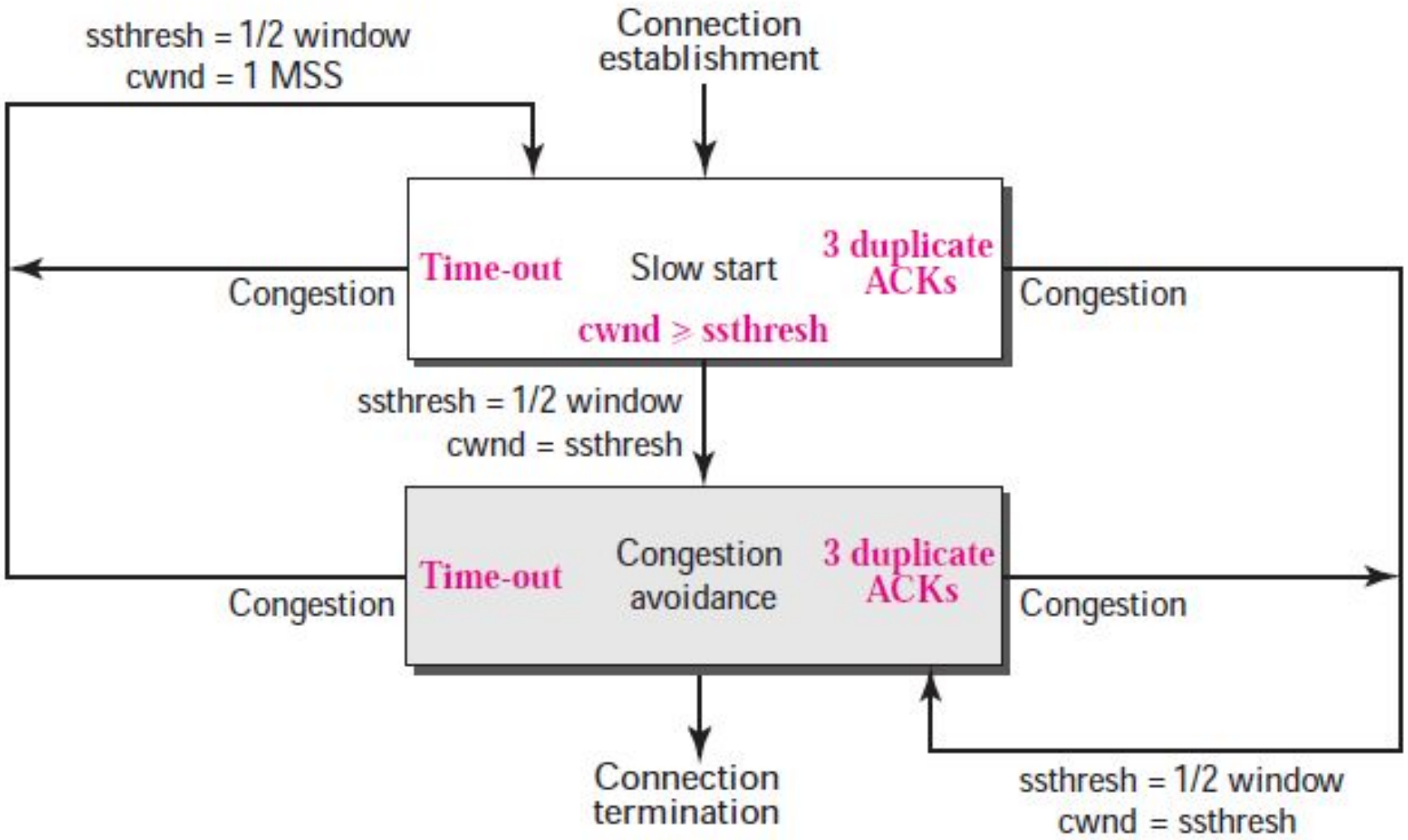
- When 3 ACKS are received:

a. It sets the value of the threshold to half of the current window size.
b. It sets cwnd to the value of the threshold
c. It starts the congestion avoidance phase.

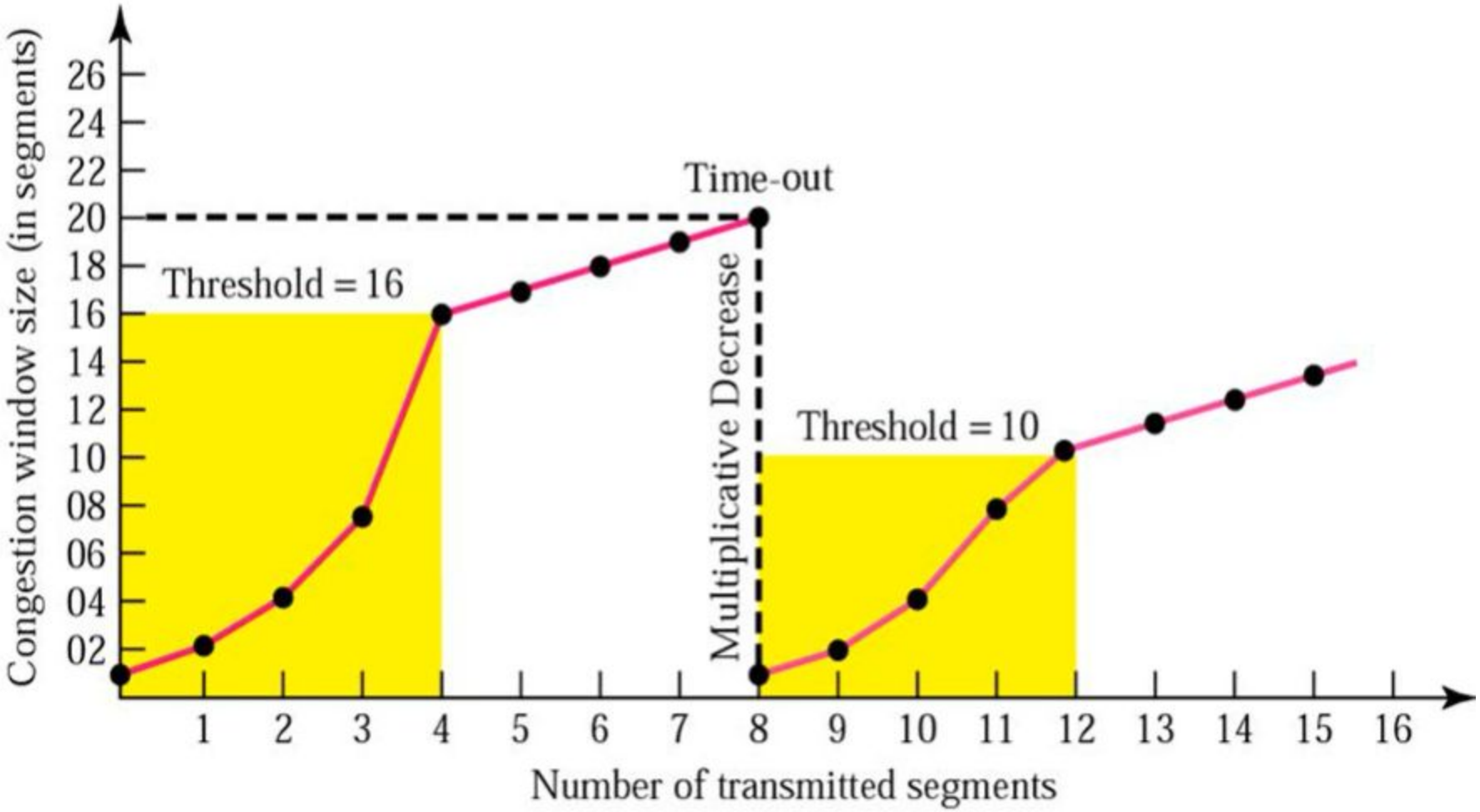
Most implementations react differently to congestion detection:

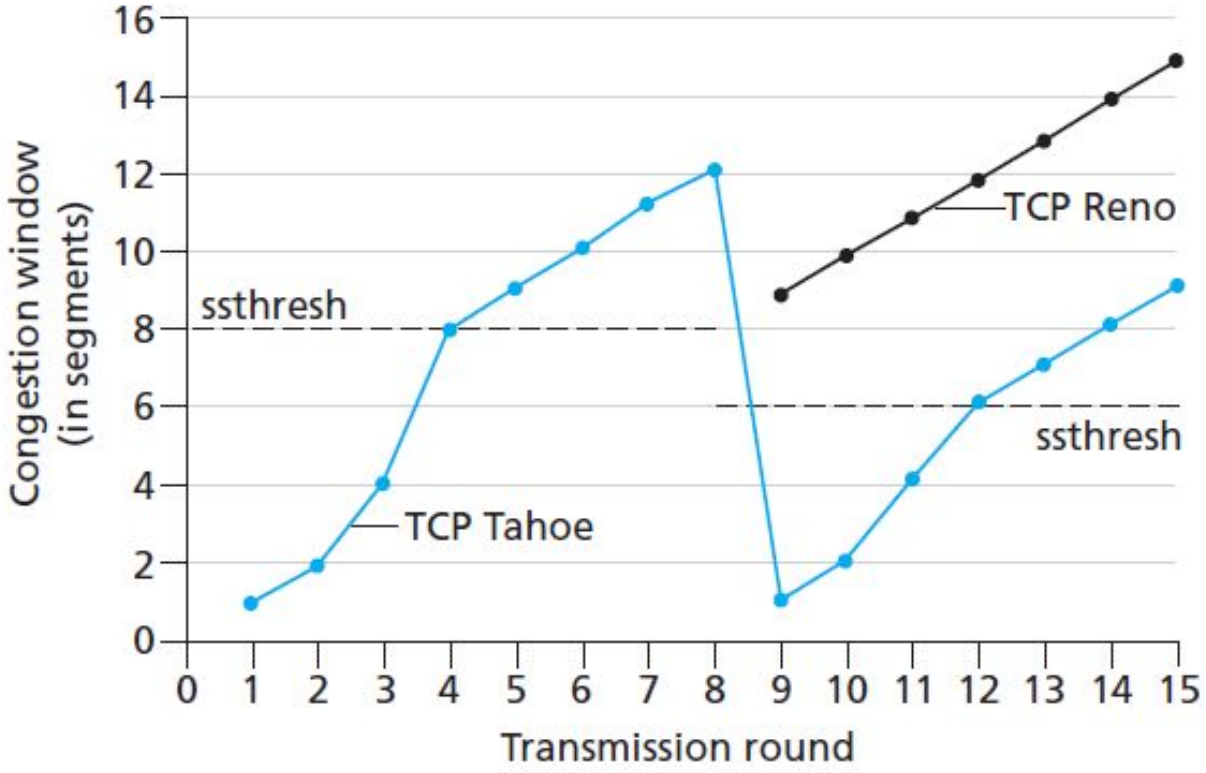
- *If detection is by time-out, a new slow start phase starts.*
- *If detection is by three ACKs, a new congestion avoidance phase starts.*

6.36 TCP congestion policy summary



Multiplicative decrease



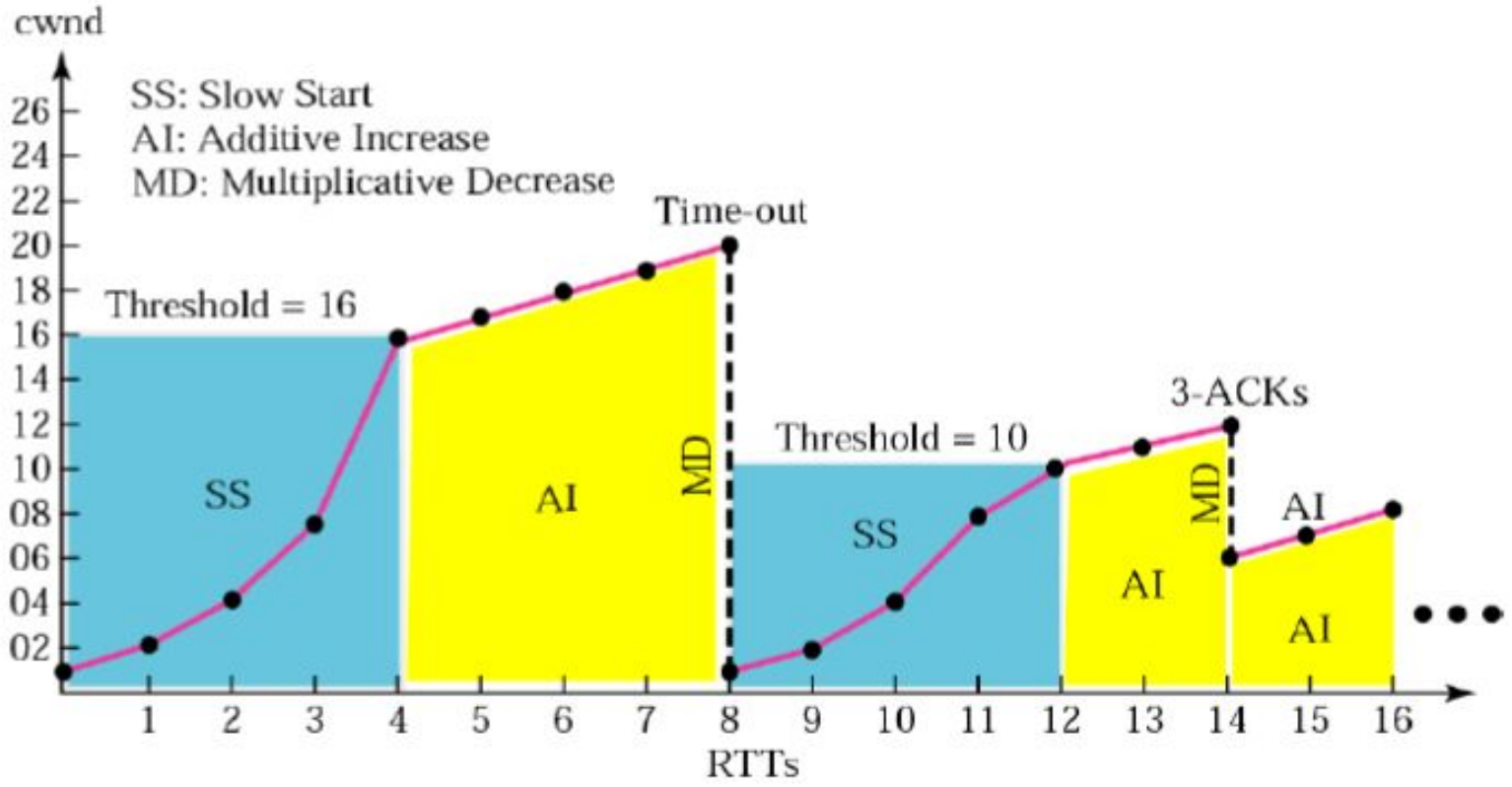


3.53 ♦ Evolution of TCP's congestion window (Tahoe and Reno)

Under TCP Reno, the congestion window is set to `cwnd` =6 • MSS and then grows linearly.**(Avoidance)**

Under TCP Tahoe, the congestion window is set to 1 MSS and grows exponentially **(slow start)** until it reaches the value of `ssthresh`, at which point it grows linearly

Congestion example



Exercise

Consider the following plot of TCP window size as a function of time.

a. Identify the intervals of time when TCP slow start is operating.

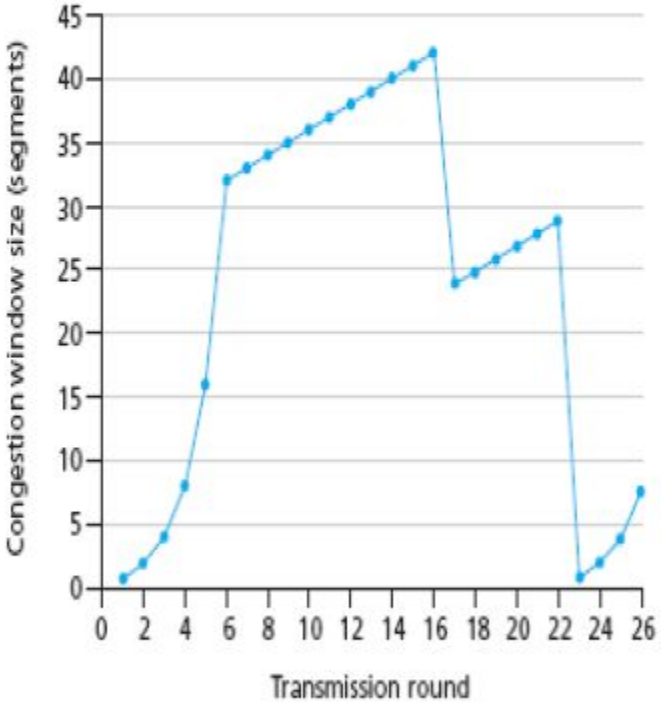
b. Identify the intervals of time when TCP congestion avoidance is operating.

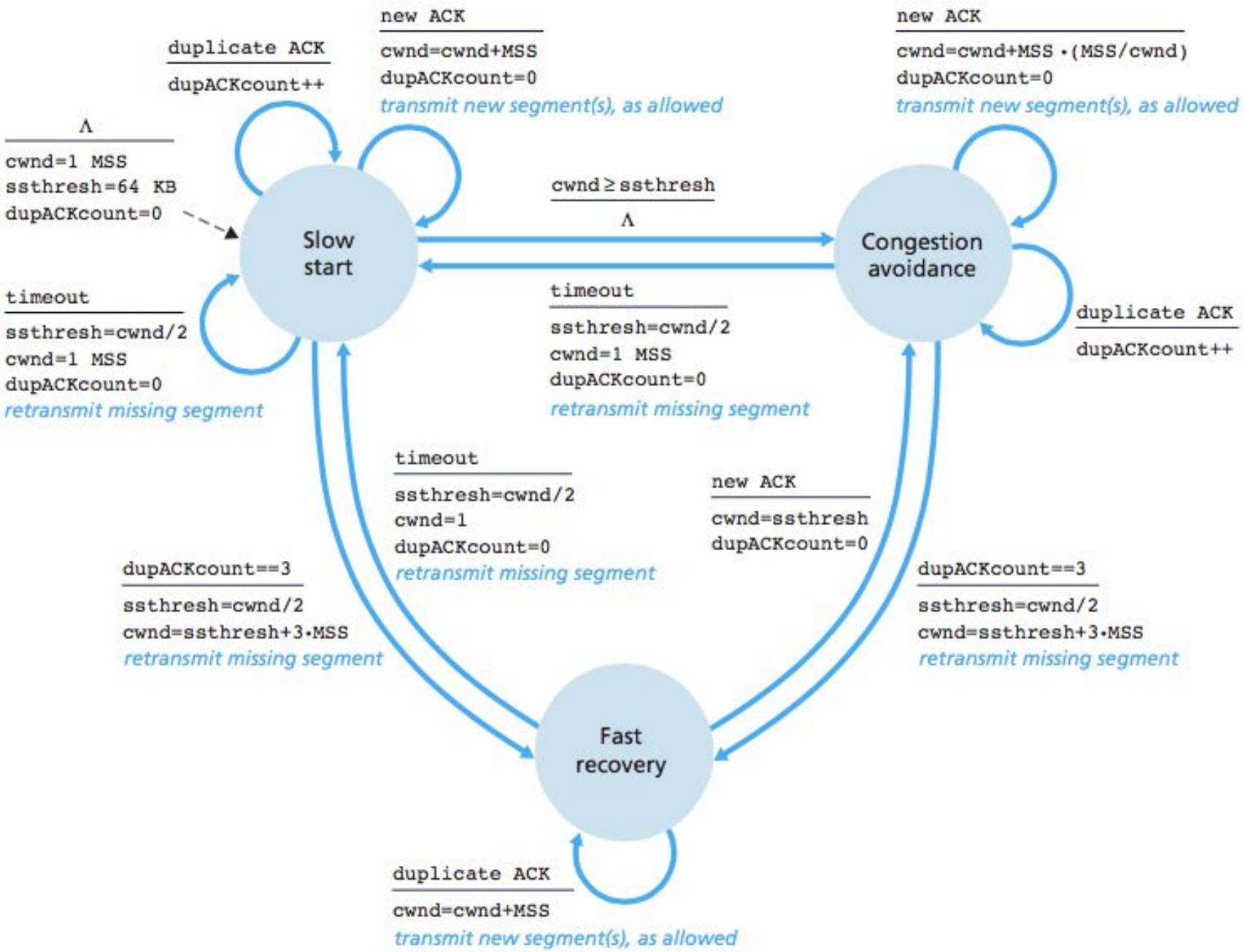
c. After the 16th and 22nd transmission round, is segment loss detected by a triple duplicate ACK or by a timeout?

d. What is the initial value of ssthresh at the first transmission round?

e. During what transmission round is the 70th segment sent?

i. Assuming a packet loss is detected after the 26th round by the receipt of a triple duplicate ACK, what will be the values of the congestion window size and of ssthresh?




new ACK
cwnd=cwnd+MSS
dupACKcount=0
transmit new segment(s), as allowed
duplicate ACK
dupACKcount++
new ACK
cwnd=cwnd+MSS·(MSS/cwnd)
dupACKcount=0
transmit new segment(s), as allowed
Λ
cwnd=1 MSS
ssthresh=64 KB
dupACKcount=0
cwnd≥ssthresh
Λ
Slow start
Congestion avoidance
timeout
ssthresh=cwnd/2
cwnd=1 MSS
dupACKcount=0
retransmit missing segment
timeout
ssthresh=cwnd/2
cwnd=1 MSS
dupACKcount=0
retransmit missing segment
duplicate ACK
dupACKcount++
timeout
ssthresh=cwnd/2
cwnd=1
dupACKcount=0
retransmit missing segment
new ACK
cwnd=ssthresh
dupACKcount=0
dupACKcount==3
ssthresh=cwnd/2
cwnd=ssthresh+3·MSS
retransmit missing segment
dupACKcount==3
ssthresh=cwnd/2
cwnd=ssthresh+3·MSS
retransmit missing segment
Fast recovery
duplicate ACK
cwnd=cwnd+MSS
transmit new segment(s), as allowed

① Fast Recovery

3 ACK (Duplicate) : Congestion Avoidance

Threshold = CWND / 2

CWND = Threshold

= +1 after ~~every~~ ack

② ~~Th~~ Tahoe

→ 3 Ack
→ Time-out (RTO)] → go in slow-start

③ Reno

→ 3 Ack :- use fast Recovery
→ Congestion avoidance

Summary: TCP Congestion Control

- When `CongWin` is below `Threshold`, sender in slow-start phase, window grows exponentially.
- When `CongWin` is above `Threshold`, sender is in congestion-avoidance phase, window grows linearly.
- When a triple duplicate ACK occurs, `Threshold` set to `CongWin/2` and `CongWin` set to `Threshold`.
- When timeout occurs, `Threshold` set to `CongWin/2` and `CongWin` is set to 1 MSS.

There is a Twist!
Fast Retransmit and Fast Recovery

Van Jacobson introduced this modification to the congestion avoidance algorithm in 1990.

TCP is required to generate an immediate ACK (a duplicate ACK) when an out-of-order segment is received, to inform sender what segment number that is expected.

Cause of duplicate ACK could be

- lost segment, or
- reordering of segments
- but data is still flowing between the two ends!!

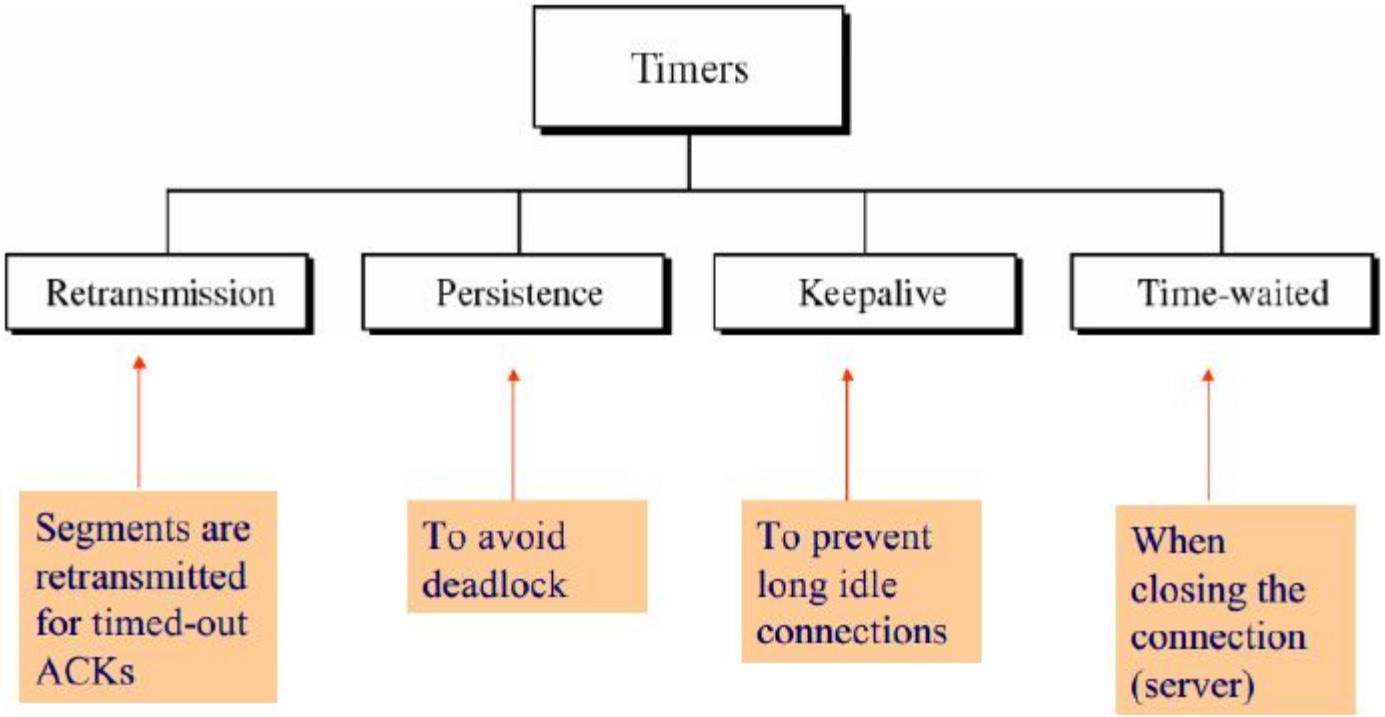
If 3 (or more) duplicate ACKs are received in a row:

- Retransmit immediately (before time-out) - *Fast Retransmit*
- Do congestion avoidance (not slow start) - *Fast Recovery*

TCP sender congestion control

State	Event	TCP Sender Action	Commentary
Slow Start (SS)	ACK receipt for previously unacked data	CongWin = CongWin + MSS, If (CongWin > Threshold) set state to "Congestion Avoidance"	Resulting in a doubling of CongWin every RTT
Congestion Avoidance (CA)	ACK receipt for previously unacked data	CongWin = CongWin+MSS * (MSS/CongWin)	Additive increase, resulting in increase of CongWin by 1 MSS every RTT
SS or CA	Loss event detected by triple duplicate ACK	Threshold = CongWin/2, CongWin = Threshold, Set state to "Congestion Avoidance"	Fast recovery, implementing multiplicative decrease. CongWin will not drop below 1 MSS.
SS or CA	Timeout	Threshold = CongWin/2, CongWin = 1 MSS, Set state to "Slow Start"	Enter slow start
SS or CA	Duplicate ACK	Increment duplicate ACK count for segment being acked	CongWin and Threshold not changed

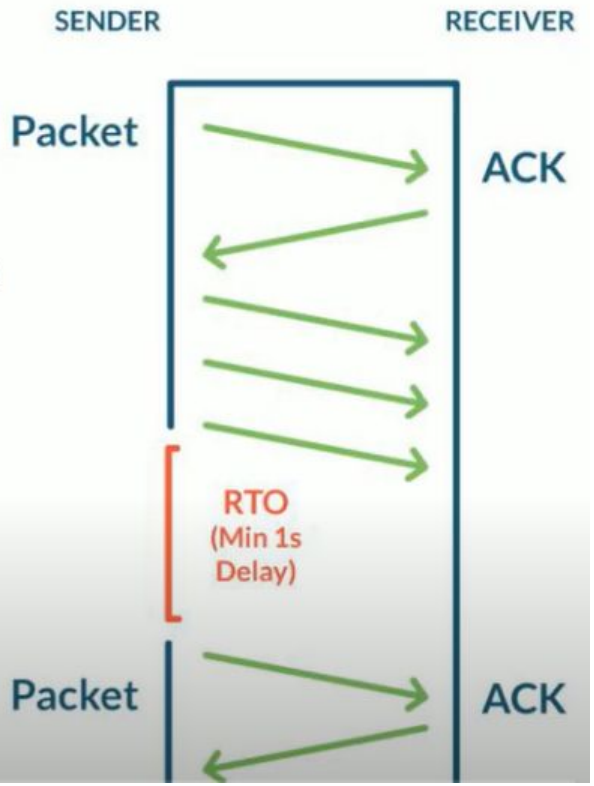
TCP timers


Timers
Retransmission
Persistence
Keepalive
Time-waited
Segments are retransmitted for timed-out ACKs
To avoid deadlock
To prevent long idle connections
When closing the connection (server)

Retransmission Timer

– The most important is the *retransmission timer*

- When a segment is sent, a retransmission timer is started
- If the segment is acknowledged before this timer expires, the timer is stopped
- If the timer goes off before the segment is acknowledged, then the segment gets retransmitted (and the timer restarted)
- The big question is how long this timer interval should be?



Persistent Timer

- *Persistence timer* is designed to prevent deadlock
  - Receiver sends a packet with window size 0
  - Latter, it sends another packet with larger window size, letting the sender know that it can send data, but this segment gets lost
  - Both the receiver and transmitter are waiting for the other
  - Solution: persistence timer on the sender end, that goes off and produces a probe packet to go to the receiver and make it to advertise again its window

Keep Alive Timer

- *Keepalive timer* is designed to check for connection integrity
  - When goes off (because a long time of inactivity), causing one side to check if the other side is still there

Time-Wait Timer

- Wait in time-wait for 2*MSL (maximum segment lifetime)
  - Helps clear out older packets in the network; prevents them from interfering with new connection
  - Time spent in time-wait range from 30sec to 2 min

Exercise

Consider the following plot of TCP window size as a function of time.

a. Identify the intervals of time when TCP slow start is operating.

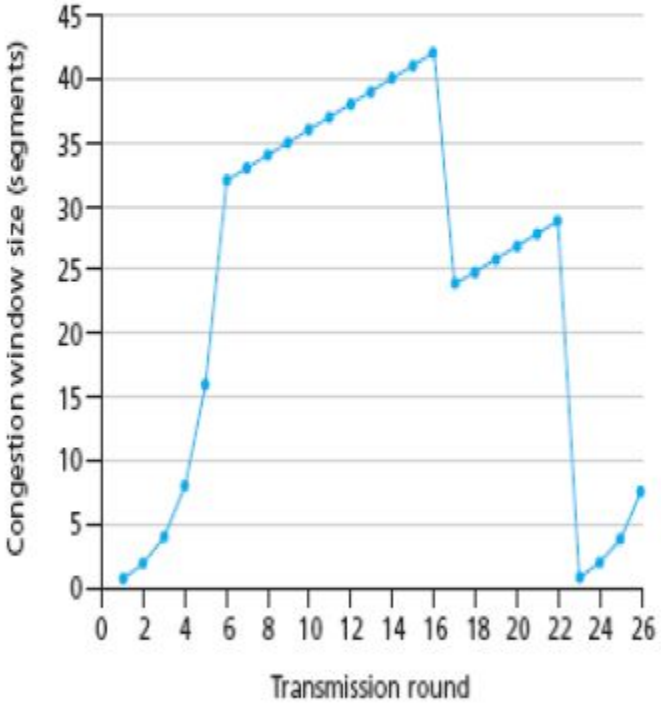
b. Identify the intervals of time when TCP congestion avoidance is operating.

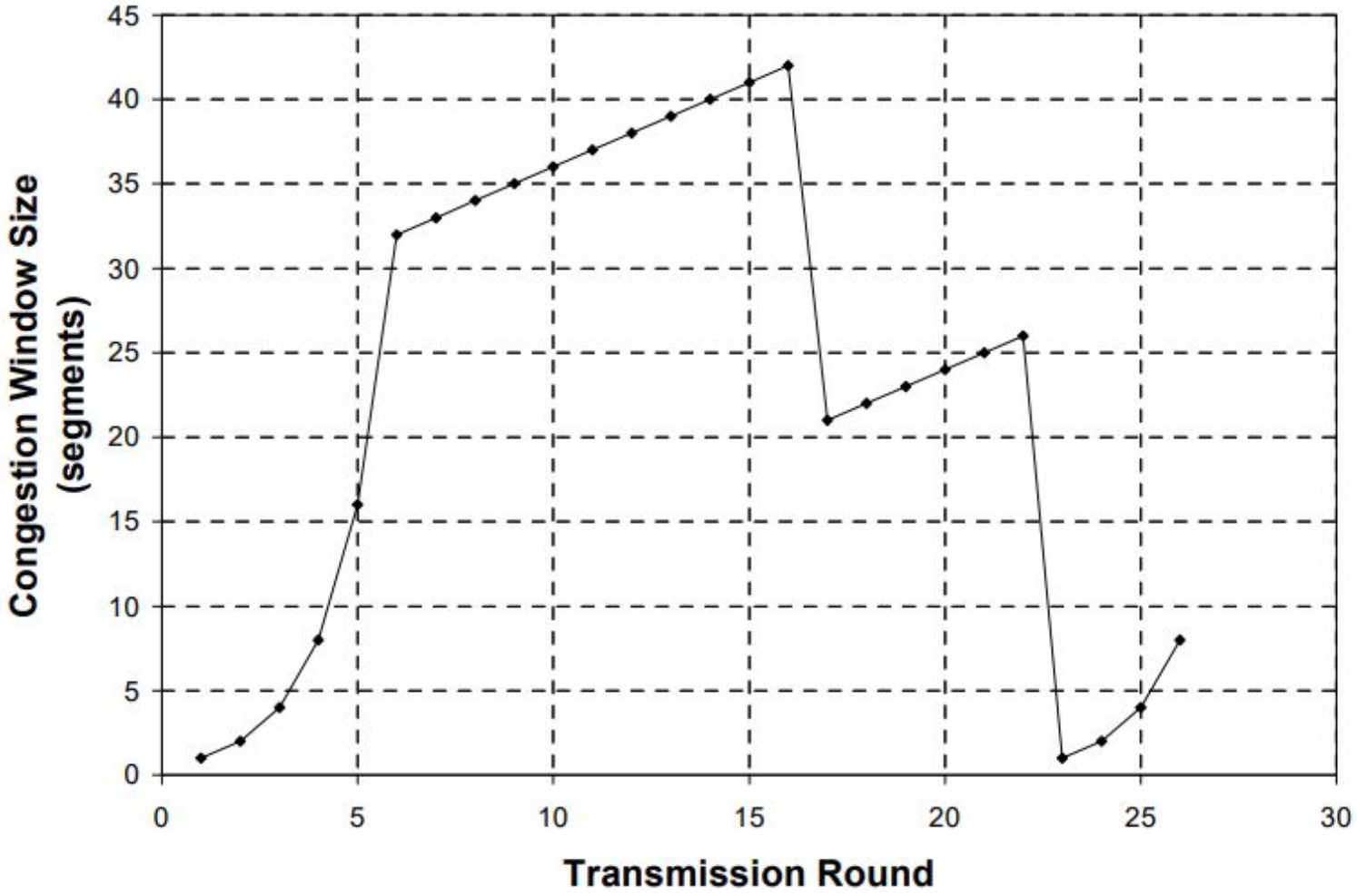
c. After the 16th and 22nd transmission round, is segment loss detected by a triple duplicate ACK or by a timeout?

d. What is the initial value of ssthresh at the first transmission round?

e. During what transmission round is the 70th segment sent?

i. Assuming a packet loss is detected after the 26th round by the receipt of a triple duplicate ACK, what will be the values of the congestion window size and of ssthresh?




Congestion Window Size
(segments)
45
40
35
30
25
20
15
10
5
0
0
5
10
15
20
25
30
Transmission Round

Solution:

a. Identify the intervals of time when TCP slow start is operating.

The slow start times are from 1 to 6 and from 23 to 26. These are slow start times because the congestion window size is picking up at a slope, slower at first and becoming exponentially faster until it gets to a linear section and is no longer in slow start.

b. Identify the intervals of time when TCP congestion avoidance is operating.

This is between transmission rounds 6 and 16 and between rounds 17 and 23. This is when it is linearly rising.

c. After the 16th and 22nd transmission round, is segment loss detected by a triple duplicate ACK or by a timeout?

- **If it had been a timeout, the congestion window would have dropped to 0 and had a slow start, but neither of those are true so it was detected by a triple ACK after 16th transmission round .**
- **After the 22nd transmission round detected by a timeout for the reasons listed above.**

d. What is the initial value of ssthresh at the first transmission round?

The initial value of ssthresh is 33 segments because that is the congestion window size in segments when it starts it's linear rise which indicates congestion avoidance at transmission round 6. It remains at 33 segments until congestion avoidance ends.

e. During what transmission round is the 70th segment sent?

I find this out by adding up the segments that were sent in each transmission round. The congestion window is the segments. So in round 1, 1 segment is sent. Round 2, 2 segments. Round 3, segments 4-7. Round 4, segments 8-15. Round 5, segments 16-31. Round 6, segments 32-63. Round 7, segments 64-99. Therefore, the 70th segment is sent during the 7th transmission round.

f. Assuming a packet loss is detected after the 26th round by the receipt of a triple duplicate ACK, what will be the values of the congestion window size and of ssthresh?

The congestion window size would be 4 segments, and ssthresh would be 4.

TCP Tahoe and Reno

While both consider retransmission timeout (RTO) and duplicate ACKs as packet loss events, the behaviour of Tahoe and Reno differ primarily in how they react to duplicate ACKs:

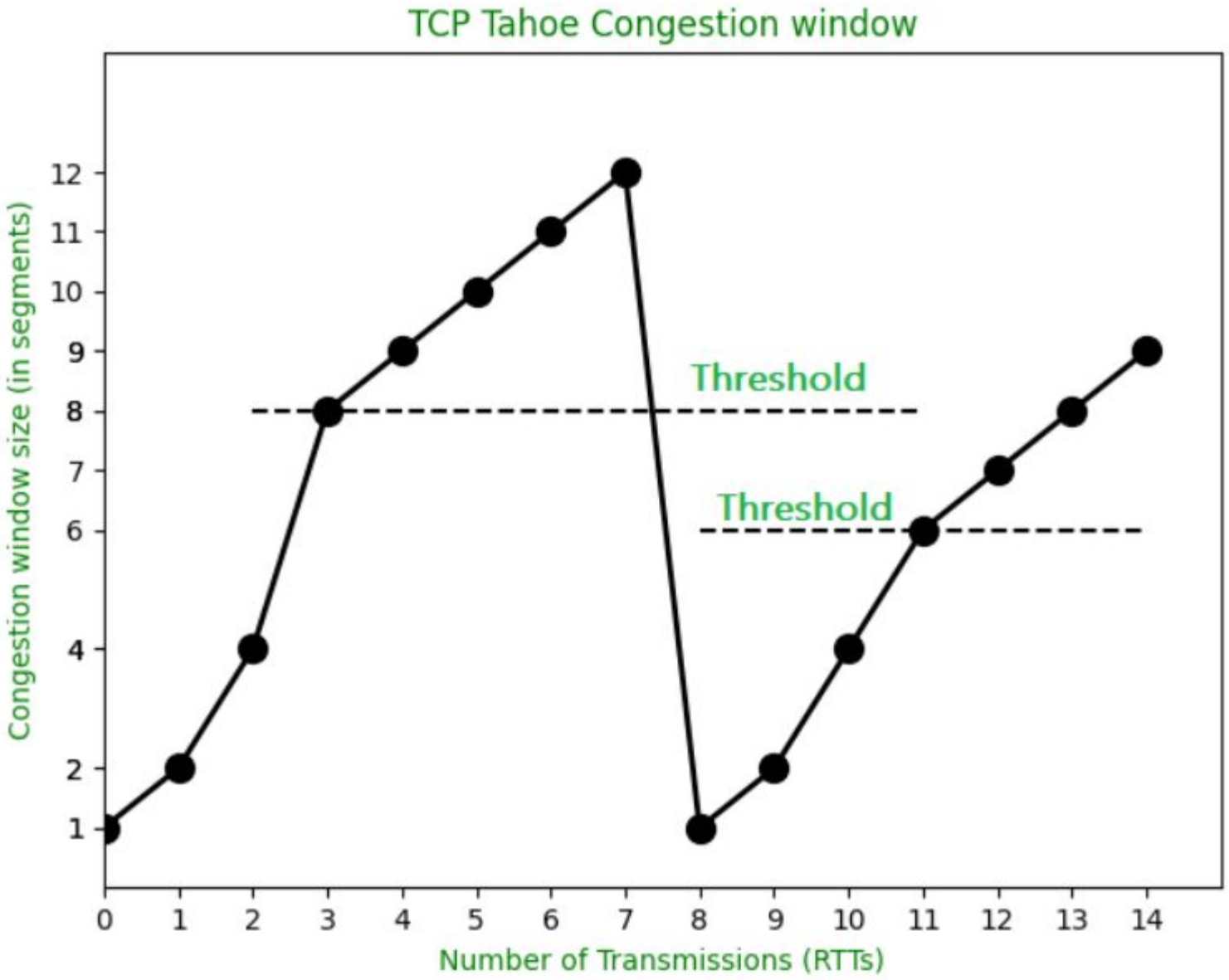
Tahoe: if three duplicate ACKs are received (i.e. four ACKs acknowledging the same packet, which are not piggybacked on data and do not change the receiver's advertised window), Tahoe performs a fast retransmit, sets the slow start threshold to half of the current congestion window, reduces the congestion window to 1 MSS, and resets to slow start state.

Reno: if three duplicate ACKs are received, Reno will perform a fast retransmit and skip the slow start phase by instead halving the congestion window (instead of setting it to 1 MSS like Tahoe), setting the slow start threshold equal to the new congestion window, and enter a phase called *fast recovery*.

In both Tahoe and Reno, if an ACK times out (RTO timeout), slow start is used, and both algorithms reduce congestion window to 1 MSS.

TCP Tahoe = Slow Start + AIMD + Fast Retransmit

- **Slow Start phase:** It lasts until congestion window size reaches up to "**Slow Start Threshold (ssthresh)**". The slow start algorithm doubles the congestion window size(cwnd) in one RTT. Initially, ssthresh is set to infinite(∞). Subsequently, it adapts depending on the packet loss events. When cwnd becomes equal to ssthresh, slow start stops. After that AIMD phase takeovers.
- **AIMD phase:** It starts when the slow start stops. Additive Increase increases cwnd by 1 and Multiplicative Decrease reduces ssthresh to 50% of cwnd. Note that cwnd is 'not' reduced by 50% but ssthresh. This is the point to note that cwnd again resets to the initial window size.
- **Fast Retransmit phase:** It is the loss detection algorithm. It is triggered by 3 duplicate acknowledgments. On packet loss detection, it resets cwnd to initcwnd.

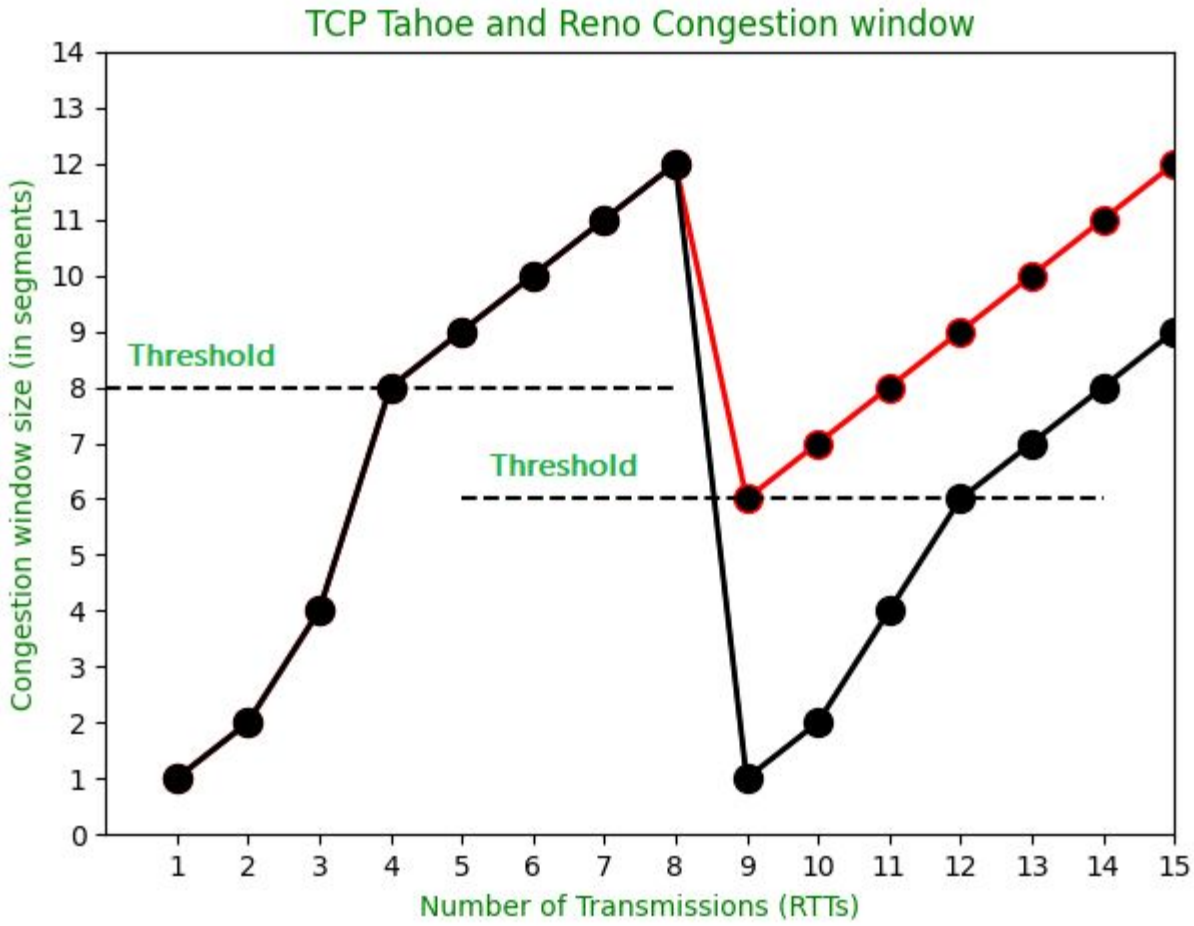

TCP Tahoe Congestion window
Congestion window size (in segments)
Number of Transmissions (RTTs)
Threshold
Threshold
1
2
4
6
7
8
9
10
11
12
0
1
2
3
4
5
6
7
8
9
10
11
12
13
14

TCP Reno = TCP Tahoe + Fast Recovery

Since TCP Reno is the extension of TCP Tahoe, the slow start and AIMD phase are the same.

Fast Recovery phase

- It makes use of both RTO and Fast Retransmit. If the packet loss detection is triggered by 3 duplicate acknowledgments then it is a fast retransmit algorithm in action. On packet loss detection through Fast Retransmit, cwnd is reduced by 50% (cwnd = cwnd/2). 3 duplicate ACKs are received that means network is performing well because ACKs are being received which means packets are getting delivered to the receiver. So, cwnd is reduced by 50% in order to allow the network to come out of congested state. On packet loss detection through RTO, reset cwnd to initcwnd. If the RTO timer expires that means the network is badly congested. So, the cwnd has to be reduced to the initial value in order to recover the network from congestion.


TCP Tahoe and Reno Congestion window
Congestion window size (in segments)
Number of Transmissions (RTTs)
Threshold
Threshold
0
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15

Sample Question 1: Congestion Control

- Consider a congestion control algorithm that works in units of packets and that starts each connection with a congestion window equal to one packet. Assume an ACK is sent for each packet received in-order, and when a packet is lost, ACKs are not sent for the lost packet and the subsequent packets that were transmitted. The lost packet and the subsequent packets have to be retransmitted by the sender. Whenever there is a packet loss and the sender times out in a RTT, the congestion window size in the next RTT has to be reduced to half of its size in the current RTT.

- For simplicity, assume a perfect timeout mechanism that detects a lost packet exactly 1 RTT after it is transmitted. Also, assume the congestion window is always less than or equal to the advertised window, so flow control need not be considered.
- Consider the loss of packets with sequence numbers **5, 15, 22 and 27** in their first transmission attempt. Assume these packets are delivered successfully in their first retransmission attempt.
- Fill the following table to indicate the RTTs and the sequence numbers of the packets sent. The sequence numbers of the packets sent range from **1 to 30**.
- Compute the effective throughput achieved by this connection to send packets with sequence numbers **1 to 30**, each packet holds 1KB of data and that the RTT = 100ms.

Sample Question: AIMD

RTT	Sequence Numbers of Packets Sent
1	1
2	2, 3
3	4, 5, 6
4	5
5	6, 7
6	8, 9, 10
7	11, 12, 13, 14
8	15, 16, 17, 18, 19
9	15, 16
10	17, 18, 19
11	20, 21, 22, 23
12	22, 23
13	24, 25, 26
14	27, 28, 29, 30
15	27, 28
16	29, 30

5, 15, 22, 27
- Lost packets

It takes 16 RTTs to send 30 packets.
The throughput = (30 packets * 1 KB/packet) / (16 RTTs * 100 ms /RTT)
= 153600 bits/sec

Sample Question: Slow Start

RTT	Sequence Numbers of Packets Sent	
1	1	
2	2, 3	
3	4, 5, 6, 7	
4	5	Cong. Threshold = 2
5	6, 7	
6	8, 9, 10	
7	11, 12, 13, 14	
8	15, 16, 17, 18, 19	Cong. Threshold = 2
9	15	
10	16, 17	
11	18, 19, 20	
12	21, 22, 23, 24	Cong. Threshold = 2
13	22	
14	23, 24	
15	25, 26, 27	Cong. Threshold = 1
16	27	
17	28, 29	
18	30	

5, 15, 22, 27
- **Lost packets**

It takes 18 RTTs to send 30 packets.
The throughput = (30 packets * 1 KB/packet) / (18 RTTs * 100 ms /RTT)
= 136533 bits/sec

Sample Question: Fast Recovery

RTT	Sequence Numbers of Packets Sent	
1	1	
2	2, 3	
3	4, 5, 6, 7	Cong. Threshold = 2
4	5, 6	
5	7, 8, 9	
6	10, 11, 12, 13	
7	14, 15, 16, 17, 18	Cong. Threshold = 2
8	**15, 16**	
9	**17, 18**, 19	
10	20, 21, **22**, 23	Cong. Threshold = 2
11	**22, 23**	
12	24, 25, 26	
13	**27**, 28, 29, 30	Cong. Threshold = 2
14	**27, 28**	
15	**29, 30**	

5, 15, 22, 27
- **Lost packets**

It takes 15 RTTs to send 30 packets.
The throughput = (30 packets * 1 KB/packet) / (15 RTTs * 100 ms /RTT)
= 163840 bits/sec

Sample Q2: Congestion Control

- Consider a congestion control algorithm that works in units of packets and that starts each connection with a congestion window equal to one packet. Assume an ACK is sent for each packet received in-order, and when a packet is lost, ACKs are not sent for the lost packet and the subsequent packets that were transmitted. The lost packet and the subsequent packets have to be retransmitted by the sender. Whenever there is a packet loss and the sender times out in a RTT, the congestion window size in the next RTT has to be reduced to half of its size in the current RTT.

- For simplicity, assume a perfect timeout mechanism that detects a lost packet exactly 1 RTT after it is transmitted. Also, assume the congestion window is always less than or equal to the advertised window, so flow control need not be considered.
- Consider the loss of packets with sequence numbers **10, 25, 34 and 45** in their first transmission attempt. Assume these packets are delivered successfully in their first retransmission attempt.
- Fill the following table to indicate the RTTs and the sequence numbers of the packets sent. The sequence numbers of the packets sent range from **1 to 50**.
- Compute the effective throughput achieved by this connection to send packets with sequence numbers **1 to 50**, each packet holds 1KB of data and that the RTT = 100ms.

Sample Question 2: AIMD

RTT	Sequence #
1	1
2	2, 3
3	4, 5, 6
4	7, 8, 9, 10
5	10, 11
6	12, 13, 14
7	15 , 16, 17, 18
8	19, 20, 21, 22, 23
9	24, 25, 26, 27, 28, 29
10	25, 26, 27
11	28, 29, 30, 31
12	32, 33, 34, 35, 36
13	34, 35
14	36, 37, 38
15	39, 40, 41, 42
16	43, 44, 45, 46, 47
17	45, 46
18	47, 48, 49
19	50

10, 25, 34, 45
Lost packets

$$\text{Throughput} = \frac{50 \text{ packets} * 1024 \text{ bytes} * 8 \text{ bits/byte}}{100 \text{ ms/RTT} * 19 \text{ RTTs}} = 50*1024*8/ (19*0.1\text{sec}) = 215,578 \text{ bits/sec}$$

Sample Question 2: Slow Start

RTT	Sequence #
1	1
2	2, 3
3	4, 5, 6, 7
4	8, 9, 10, 11, 12, 13, 14, 15 (congestion window = 8; congestion threshold = 4)
5	10
6	11, 12
7	13, 14, 15, 16
8	17, 18, 19, 20, 21
9	22, 23, 24, 25, 26, 27 (congestion window = 6; congestion threshold = 3)
10	25
11	26, 27
12	28, 29, 30
13	31, 32, 33, 34 (congestion window = 4; congestion threshold = 2)
14	34
15	35, 36
16	37, 38, 39
17	40, 41, 42, 43
18	44, 45, 46, 47, 48 (congestion window = 5; congestion threshold = 2)
19	45
20	46, 47
21	48, 49, 50

10, 25, 34, 45
Lost packets

$$\text{Throughput} = \frac{50 \text{ packets} * 1024 \text{ bytes} * 8 \text{ bits/byte}}{100 \text{ ms/RTT} * 21 \text{ RTTs}} = 50*1024*8/\ (21*0.1\text{sec}) = 195,047 \text{ bits/sec}$$

Sample Question 2: Fast Recovery

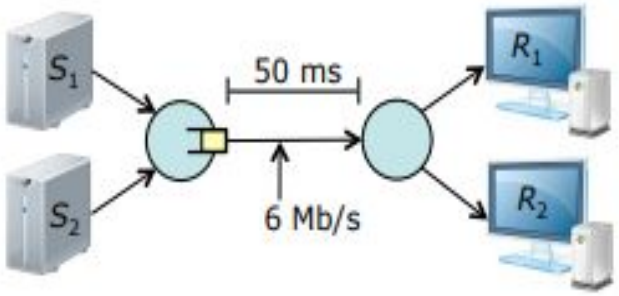
RTT	Sequence #
1	1
2	2, 3
3	4, 5, 6, 7
4	8, 9, 10, 11, 12, 13, 14, 15 (congestion window = 8; congestion threshold = 4)
5	10, 11, 12, 13
6	14, 15, 16, 17, 18
7	19, 20, 21, 22, 23, 24
8	25, 26, 27, 28, 29, 30, 31 (congestion window = 7; congestion threshold = 3)
9	25, 26, 27
10	28, 29, 30, 31
11	32, 33, 34, 35, 36 (congestion window = 5; congestion threshold = 2)
12	34, 35
13	36, 37, 38
14	39, 40, 41, 42
15	43, 44, 45, 46, 47 (congestion window = 5; congestion threshold = 2)
16	45, 46
17	47, 48, 49
18	50

10, 25, 34, 45
Lost packets

$$\text{Throughput} = \frac{50 \text{ packets} * 1024 \text{ bytes} * 8 \text{ bits/byte}}{100 \text{ ms/RTT} * 18 \text{ RTTs}} = 50*1024*8/\ (18*0.1\text{sec}) = 227,\ 555 \text{ bits/sec}$$

Question:

The diagram given below shows two TCP senders at left and the corresponding receivers at right. The first sender uses TCP Tahoe, the second uses Reno. Assume that the MSS is 1 KB, that the one-way propagation delay for both connections is 50 ms and that the link joining the two routers has a bandwidth of 6 Mb/s. Let cwnd1 and cwnd2 be the values of the senders' congestion windows. What is the smallest value of cwnd1+ cwnd2 for which the link joining the two routers stays busy all the time?



Assume that the link buffer overflows whenever cwnd1+cwnd2≥150 KB and that at time 0, cwnd1=30 KB and cwnd2=120 KB. Approximately, what are the values of cwnd1 and cwnd2 one RTT later? Also, what are the values of ssthresh for each of the two connections? Assume that all losses are detected by triple duplicate ACKs.

References:

- https://slideplayer.com/slide/5028490/
- https://slideplayer.com/slide/13219209/
- https://www.geeksforgeeks.org/congestion-control-techniques-in-computer-networks/
- http://www.cs.newpaltz.edu/~easwaran/CCN/Week9/tcpCongestionExercises_Solutions.html
- http://www.brainkart.com/article/Congestion-Control--Open-Loop-and-Closed-Loop_13488/
- https://bedfordsarah.wordpress.com/2013/11/12/chapter-3-homework/#:~:text=Round%207%2C%20segments%2068%2D101,during%20the%207th%20transmission%20round

Problems

- https://hkn.eecs.berkeley.edu/examfiles/ee122_fa03_mt1_sol.pdf
- https://www.jsums.edu/nmeghanathan/files/2015/05/CSC435-Sp2014-Module-7-Transport-Layer.pdf