SNS COLLEGE OF ENGINEERING

Kurumbapalayam(Po), Coimbatore – 641 107
Accredited by NAAC-UGC with 'A' Grade
Approved by AICTE, Recognized by UGC & Affiliated to Anna University, Chennai

Department of Information Technology

Course Name – 19IT401 Computer Networks

II Year / IV Semester

Unit 4 – Transport Layer

Topic – Congestion Control

TCP Congestion Control

TCP uses different policies to handle the congestion in the network

Congestion Window

- TCP Flow Control guarantees that the receive window is never overflowed with the received bytes. (No end system congestion)
- This, however, does not mean that the intermediate buffers, buffers in the routers, do not become congested.
- A router may receive data from more than one sender.
- There is no congestion at the other end systems, but there may be congestion in the middle(Network). Due to this many segments lost may seriously affect the error control.
- More segment loss means resending the same segments again, resulting in worsening the congestion, and finally the collapse of the communication.
- To control the number of segments to transmit, TCP uses another variable called a congestion window, cwnd, whose size is controlled by the congestion situation in the network.
- The cwnd variable and the rwnd variable together define the size of the send window in TCP.
- The actual size of the window is the minimum of these two.

Actual window size = minimum (rwnd, cwnd)

TCP Congestion Control

Congestion Detection

The TCP sender uses the occurrence of two events as signs of congestion in the network:

- time-out and
- receiving three duplicate ACKs.

- If a TCP sender does not receive an ACK for a segment or a group of segments before the time-out occurs, it assumes that the corresponding segment or segments are lost and the loss is due to congestion.

- Similarly sending three duplicate ACKs is the sign of a missing segment, which can be due to congestion in the network.

TCP Congestion Control

Congestion Policies

- Slow Start: Exponential Increase
- Congestion Avoidance: Additive Increase
- Fast Recovery

Policy Transition (When to use these three policies)

- Taho TCP,
- Reno TCP, and
- New Reno TCP.

TCP Congestion Control – Slow Start

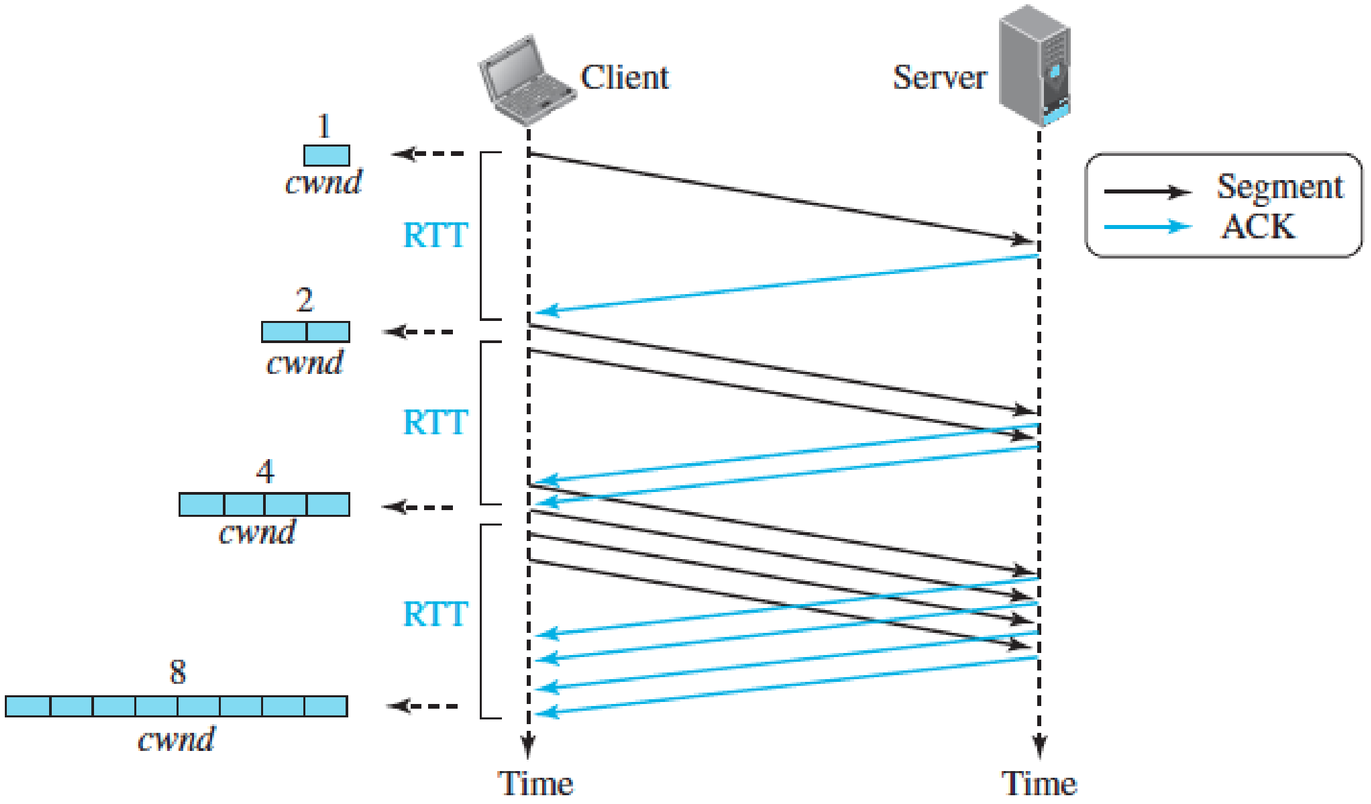
Slow Start: Exponential Increase

- The slow-start algorithm is based on the idea that the size of the congestion window (cwnd) starts with one maximum segment size (MSS), but it increases one MSS each time an acknowledgment arrives.
- The algorithm starts slowly, but grows exponentially.
- The sender starts with cwnd = 1. This means that the sender can send only one segment.
- After the first ACK arrives, the size of the congestion window is also increased by 1, The size of the window is now 2. TCP sends 2 segments now.
- After sending two segments and receiving two individual acknowledgments for them, the size of the congestion window now becomes 4, and so on. TCP can send 4 segments now and expects 4 Ack.
- Thus TCP doubles the number of packets every RTT as shown.

When to stop?

- In the slow-start algorithm, the size of the congestion window increases exponentially until it reaches a threshold.
- The sender keeps track of a variable named **ssthresh** (slow-start threshold). When the size of the window in bytes reaches this threshold, slow start stops and the next phase starts.

TCP Congestion Control - Slow Start



Slow-start strategy is slower in the case of delayed acknowledgments.

If two segments are acknowledged cumulatively, the size of the cwnd increases by only 1, not 2.

The growth is still exponential, but it is not a power of 2. With one ACK for every two segments, it is a power of 1.5.

Start	→	$cwnd = 1 \rightarrow 2^0$
After 1 RTT	→	$cwnd = cwnd + 1 = 1 + 1 = 2 \rightarrow 2^1$
After 2 RTT	→	$cwnd = cwnd + 2 = 2 + 2 = 4 \rightarrow 2^2$
After 3 RTT	→	$cwnd = cwnd + 4 = 4 + 4 = 8 \rightarrow 2^3$

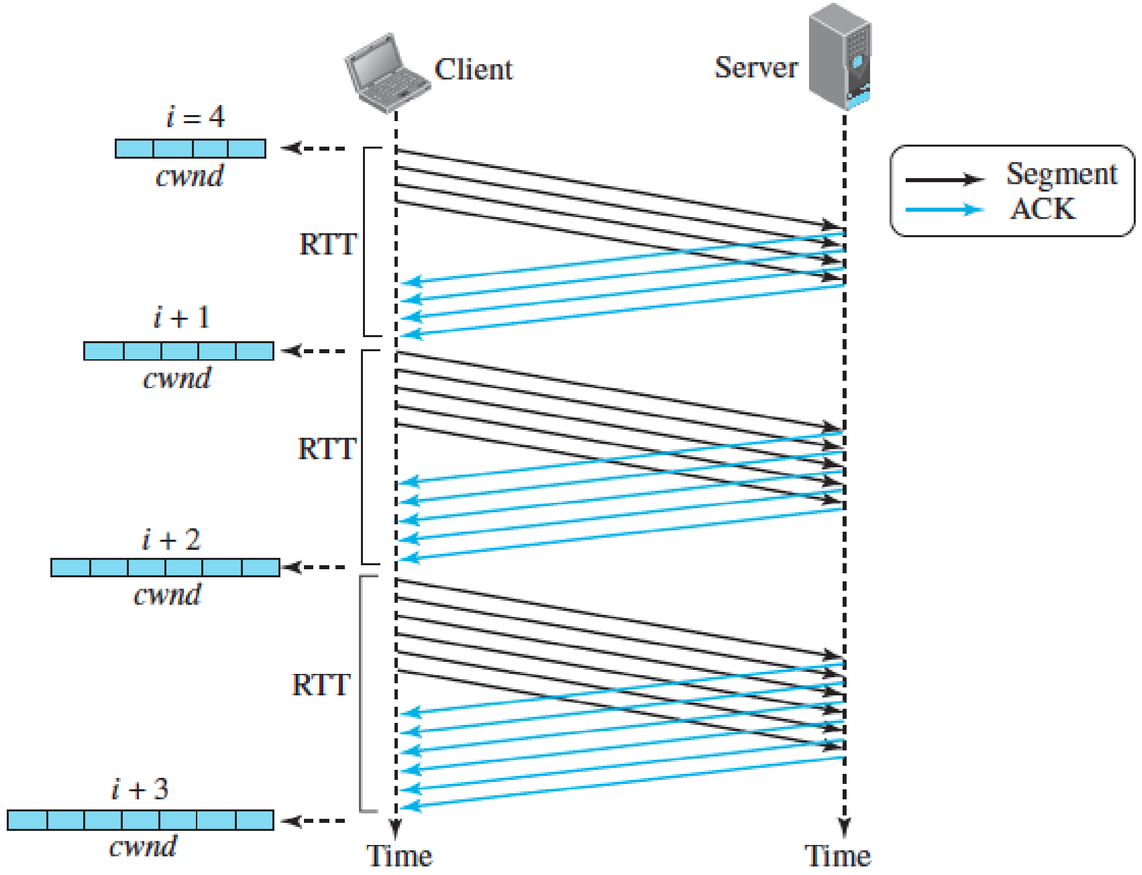
TCP Congestion Control - Congestion Avoidance

Congestion Avoidance: Additive Increase

- Congestion Avoidance increases the cwnd additively instead of exponentially.
- When the size of the congestion window reaches the slow-start threshold in the case where cwnd = i , the slow-start phase stops and the additive phase begins.
- In this algorithm, each time the whole “window” of segments is acknowledged, the size of the congestion window is increased by one. A window is the number of segments transmitted during RTT.
- The sender starts with cwnd = 4. This means that the sender can send only four segments.
- After four ACKs arrive, the size of the congestion window is also increased by 1. The size of window is now 5.
- After sending five segments and receiving five acknowledgments for them, the size of the congestion window now becomes 6, and so on.

If an ACK arrives, cwnd = cwnd + (1/cwnd).

TCP Congestion Control - Congestion Avoidance



Start	→	$cwnd = i$
After 1 RTT	→	$cwnd = i + 1$
After 2 RTT	→	$cwnd = i + 2$
After 3 RTT	→	$cwnd = i + 3$

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

TCP Congestion Control – Fast Recovery

Fast Recovery

- The fast-recovery algorithm is optional in TCP.
- The old version of TCP did not use it, but the new versions try to use it.
- It starts when three duplicate ACKs arrive, which is interpreted as light congestion in the network.
- Like congestion avoidance, this algorithm is also an additive increase, but it increases the size of the congestion window when a duplicate ACK arrives.

If a duplicate ACK arrives, cwnd = cwnd + (1 / cwnd).

TCP Congestion Control – Policy Transition

Policy Transition

When each of these policies is used and when TCP moves from one policy to another.

To answer these questions, we need to refer to three versions of TCP

- Taho TCP,
- Reno TCP, and
- New Reno TCP.

TCP Congestion Control – Taho TCP

Taho TCP

- The early TCP, known as Taho TCP, used only two different algorithms in their congestion policy: slow start and congestion avoidance.
- Taho TCP treats the two signs used for congestion detection, time-out and three duplicate ACKs.
- In this version, when the connection is established, TCP starts the slow-start algorithm and sets the ssthresh variable to a pre-agreed value (normally a multiple of MSS) and the cwnd to 1 MSS.
- In this state, each time an ACK arrives, the size of the congestion window is incremented by 1.
- We know that this policy is very aggressive and exponentially increases the size of the window, which may result in congestion.

TCP Congestion Control – Taho TCP

Taho TCP

- If congestion is detected (occurrence of time-out or arrival of three duplicate ACKs), TCP immediately interrupts this aggressive growth and restarts a new slow start algorithm by limiting the threshold to half of the current cwnd and resetting the congestion window to 1.
- TCP restart from scratch (Restarts slow start from 1), but it also learns how to adjust the threshold.
- If it reaches new threshold which is half of cwnd of previous, It moves to the congestion avoidance state and continues in that state. i.e it linearly increases cwnd by 1 every time it receives ack.
- The conservative additive growth of the congestion window continues to the end of the data transfer phase unless congestion is detected.
- If congestion is detected in this state, TCP again resets the value of the ssthresh to half of the current cwnd and moves to the slow-start state again.

TCP Congestion Control – Reno TCP

Reno TCP

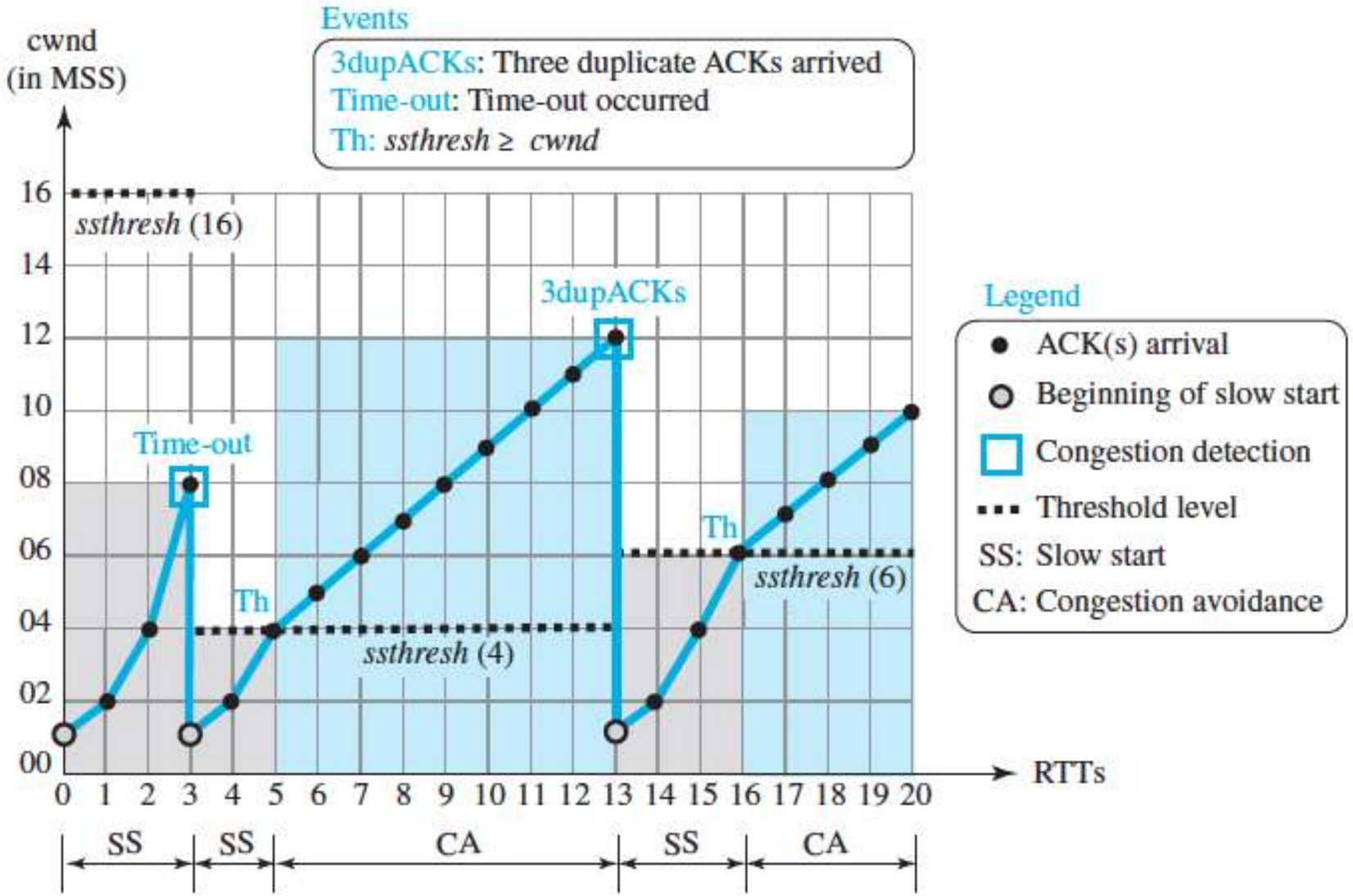
- A newer version of TCP, called Reno TCP, added a new state to the congestion-control FSM, called the fast-recovery state.
- This version treated the two signals of congestion, time-out and the arrival of three duplicate ACKs, differently.
- In this version, if a time-out occurs, TCP moves to the slow-start state, on the other hand, if three duplicate ACKs arrive, TCP moves to the fast-recovery state and remains there as long as more duplicate ACKs arrive.
- The fast-recovery state is a state somewhere between the slow-start and the congestion-avoidance states.

TCP Congestion Control – Reno TCP

Reno TCP

- When TCP enters the fast-recovery state, three major events may occur.
- If duplicate ACKs continue to arrive, TCP stays in this state, but the cwnd grows exponentially.
- If a time-out occurs, TCP assumes that there is real congestion in the network and moves to the slow-start state.
- If a new (nonduplicate) ACK arrives, TCP moves to the congestion-avoidance state, but deflates the size of the cwnd to the ssthresh value.

TCP Congestion Control – Reno TCP

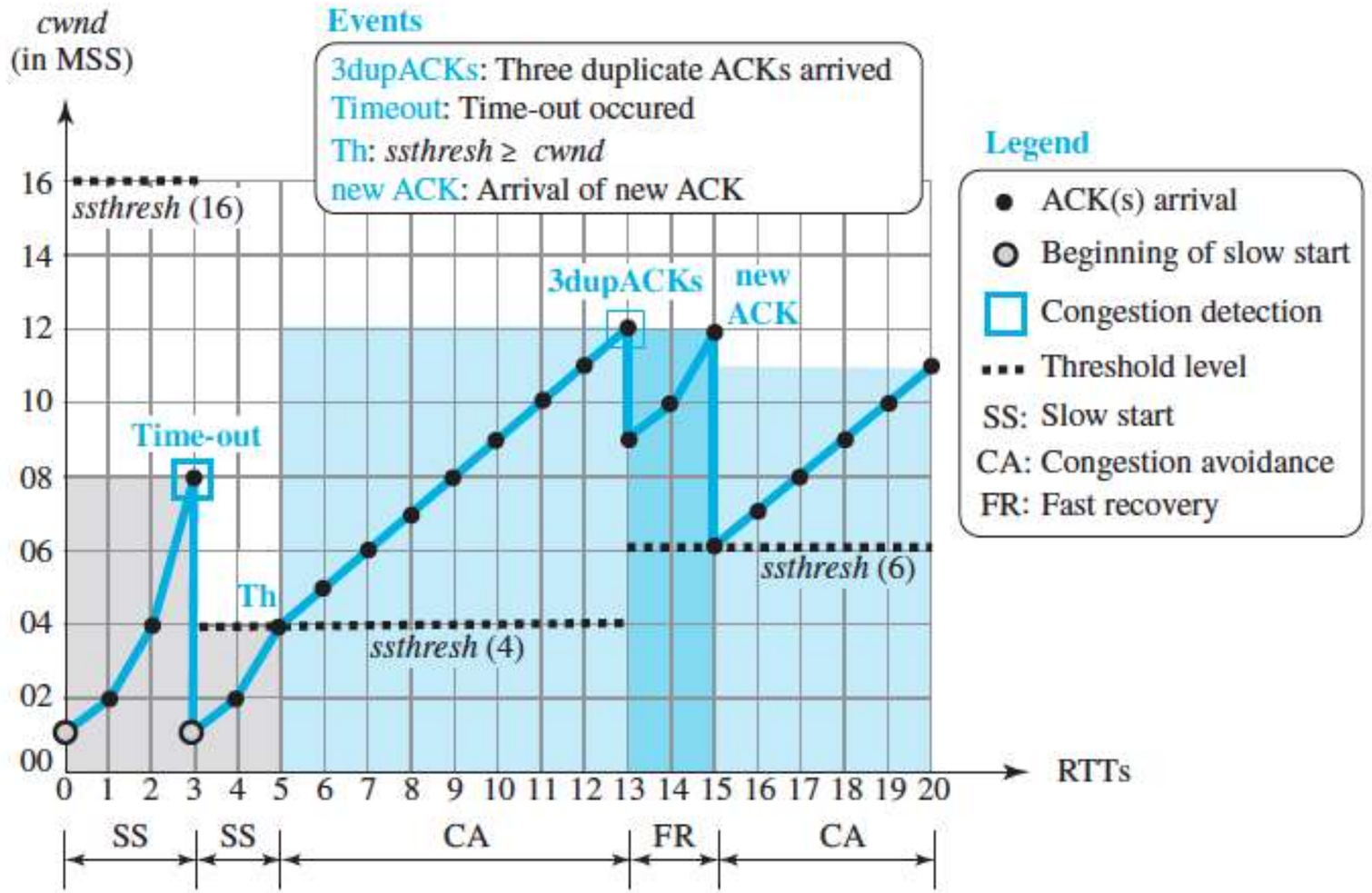


TCP Congestion Control – NewReno TCP

NewReno TCP

- A later version of TCP, called NewReno TCP, made an extra optimization on the Reno TCP.
- In this version, TCP checks to see if more than one segment is lost in the current window when three duplicate ACKs arrive.
- When TCP receives three duplicate ACKs, it retransmits the lost segment until a new ACK (not duplicate) arrives.
- If the new ACK defines the end of the window when the congestion was detected, TCP is certain that only one segment was lost.
- However, if the ACK number defines a position between the retransmitted segment and the end of the window, it is possible that the segment defined by the ACK is also lost.
- NewReno TCP retransmits this segment to avoid receiving more and more duplicate ACKs for it.

TCP Congestion Control – NewReno TCP



THANK YOU