

TCP Congestion Control

Congestion Window

- TCP uses **different policies** to **handle** the **congestion** in the network.
- **size** of the **send window** is **controlled** by the **receiver** using the **value of *rwnd***
- **receive window** is **never overflowed** with the **received bytes** (no end **congestion**).

Congestion Window

- TCP needs to worry about congestion in middle.
Intermediate buffers in the routers, do become congested.
- A router may receive data from more than one sender.
- no congestion at the other end, but there may be congestion in the middle.

Congestion Window

- TCP needs to worry about congestion - **worsening the congestion, and finally the collapse of the communication.**
- TCP **cannot send small segments either – underutilizing bandwidth**
- TCP needs to **define policies that accelerate the data transmission when there is no congestion and decelerate the transmission when congestion is detected.**

Congestion Window

- TCP uses another variable called ***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.

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

Congestion Detection

- The TCP sender uses the occurrence of two events as signs of congestion in the network:
 - ✓ time-out event
 - ✓ receiving three duplicate ACKs.
- ## 1. Time-out event
- sender does not receive an ACK for a segment or a group of segments before the time-out occurs

Congestion Detection

2. Receiving Three Duplicate Acks

- TCP receiver sends a duplicate ACK, it is the sign that a segment has been delayed
- sending three duplicate ACKs is the sign of a missing segment, which can be due to congestion in the network.
- congestion in the case of three duplicate ACKs can be less severe than in the case of time-out

Congestion Detection

- **Taho and Reno TCP**
- earlier version of TCP, called Taho TCP, treated both events (time-out and three duplicate ACKs) similarly
- but the later version of TCP, called Reno TCP, treats these two signs differently.

Congestion Detection

- **Strong Congestion** – no ACKs results into timeout
- **Weak Congestion** - receiving of three duplicate ACKs

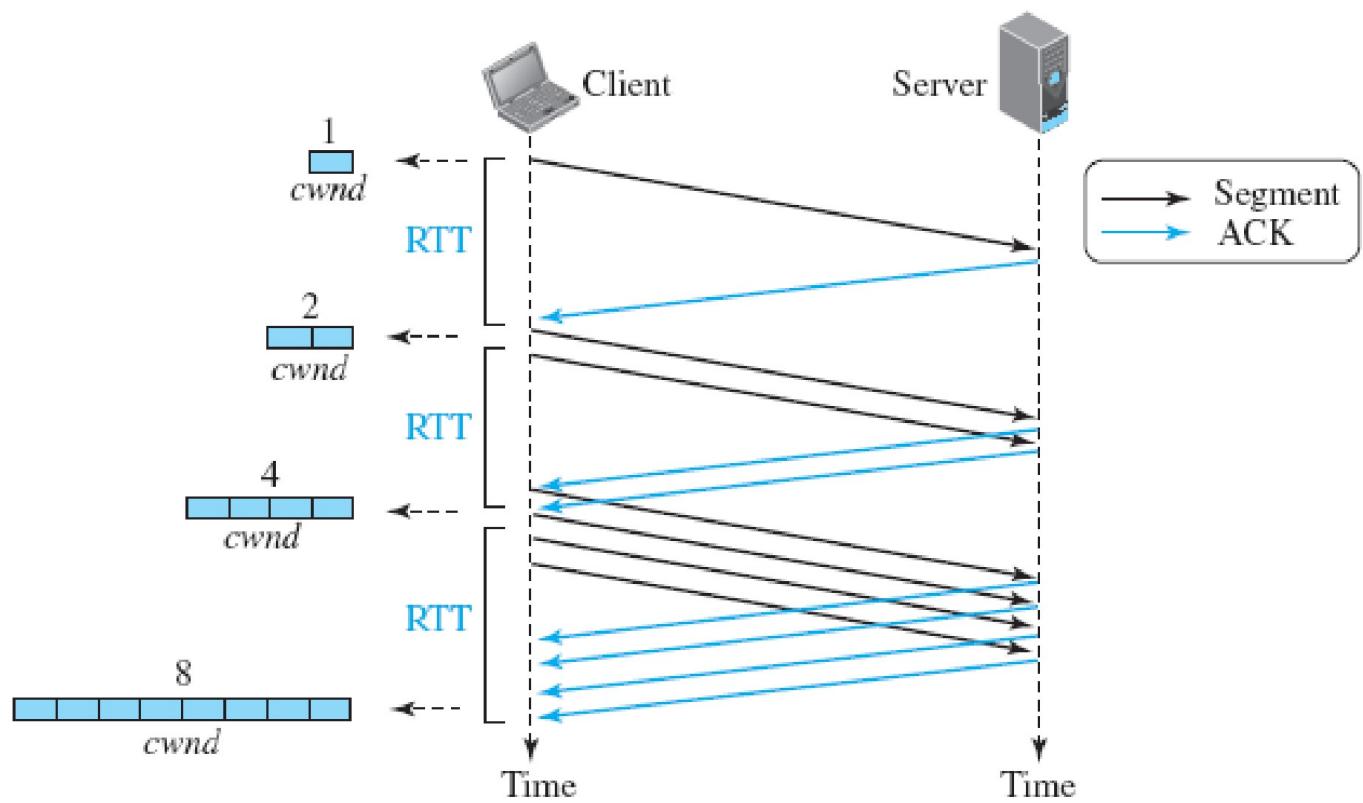
Congestion Policies

- TCP's general policy for handling congestion is based on three algorithms:
 - ✓ slow start : Exponential Increase
 - ✓ congestion avoidance : Additive Increase
 - ✓ fast recovery

Slow Start: Exponential Increase

- congestion window (*cwnd*) starts with one maximum segment size (MSS) ,but it increases one MSS each time an acknowledgment arrives.
- assume that *rwnd* is much larger than *cwnd*, so that the sender window size always equals *cwnd*.

Figure 24.29 Slow start, exponential increase



Slow Start: Exponential Increase

- If an ACK arrives, $cwnd = cwnd + 1$.

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

Slow Start: Exponential Increase

- In the slow-start algorithm, the size of the congestion window increases exponentially until it reaches a threshold.
- **ssthresh (slow-start threshold).**
- 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.

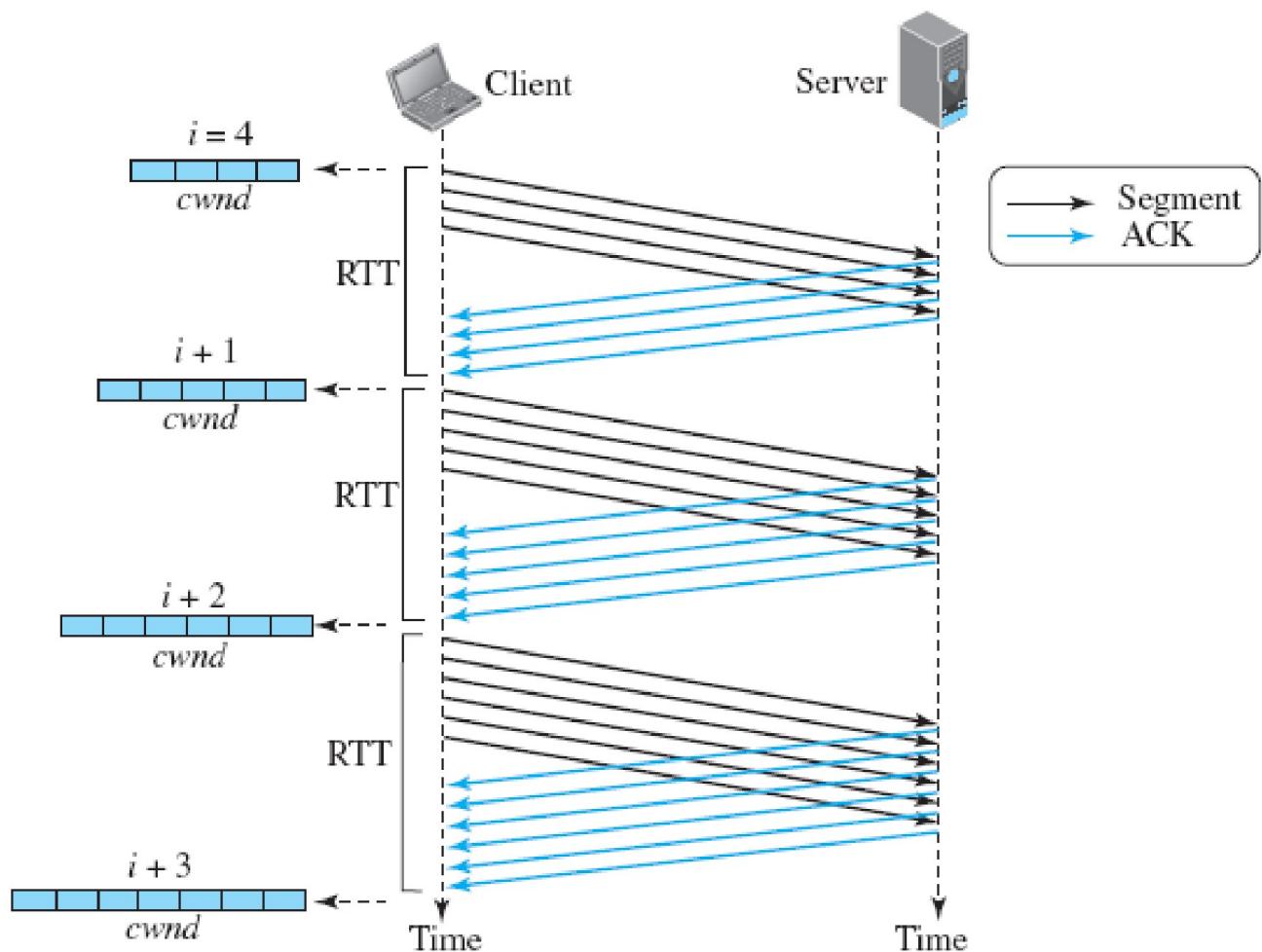
Congestion Avoidance: Additive Increase

- continue with the slow-start algorithm, the size of the congestion window increases exponentially.
- Avoid congestion before it happens – control the size of congestion window.
- **congestion avoidance** - which increases the cwnd additively instead of exponentially.

Congestion Avoidance: Additive Increase

- 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.
- each time the whole “window” of segments is acknowledged, the **size of the congestion window** is increased by **one**.

Figure 24.30 Congestion avoidance, additive increase



Congestion Avoidance: Additive Increase

- If an ACK arrives, $cwnd \leftarrow cwnd + 1/cwnd$.

Start	\rightarrow	$cwnd = i$
After 1 RTT	\rightarrow	$cwnd = i + 1$
After 2 RTT	\rightarrow	$cwnd = i + 2$
After 3 RTT	\rightarrow	$cwnd = i + 3$

Congestion Avoidance: Additive Increase

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

Congestion Detection: Multiplicative Decrease

- If congestion occurs, the congestion window size must be decreased.
- guess that congestion has occurred is the need to retransmit a segment
- retransmission can occur in one of two cases:
 - ✓ RTO timer times out
 - ✓ three duplicate ACKs are received.

Congestion Detection: Multiplicative Decrease

- 1. If a time-out occurs, there is a stronger possibility of congestion:**

A segment has probably been dropped in the network and there is no news about the sent segments.

- In this case TCP reacts strongly:**

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

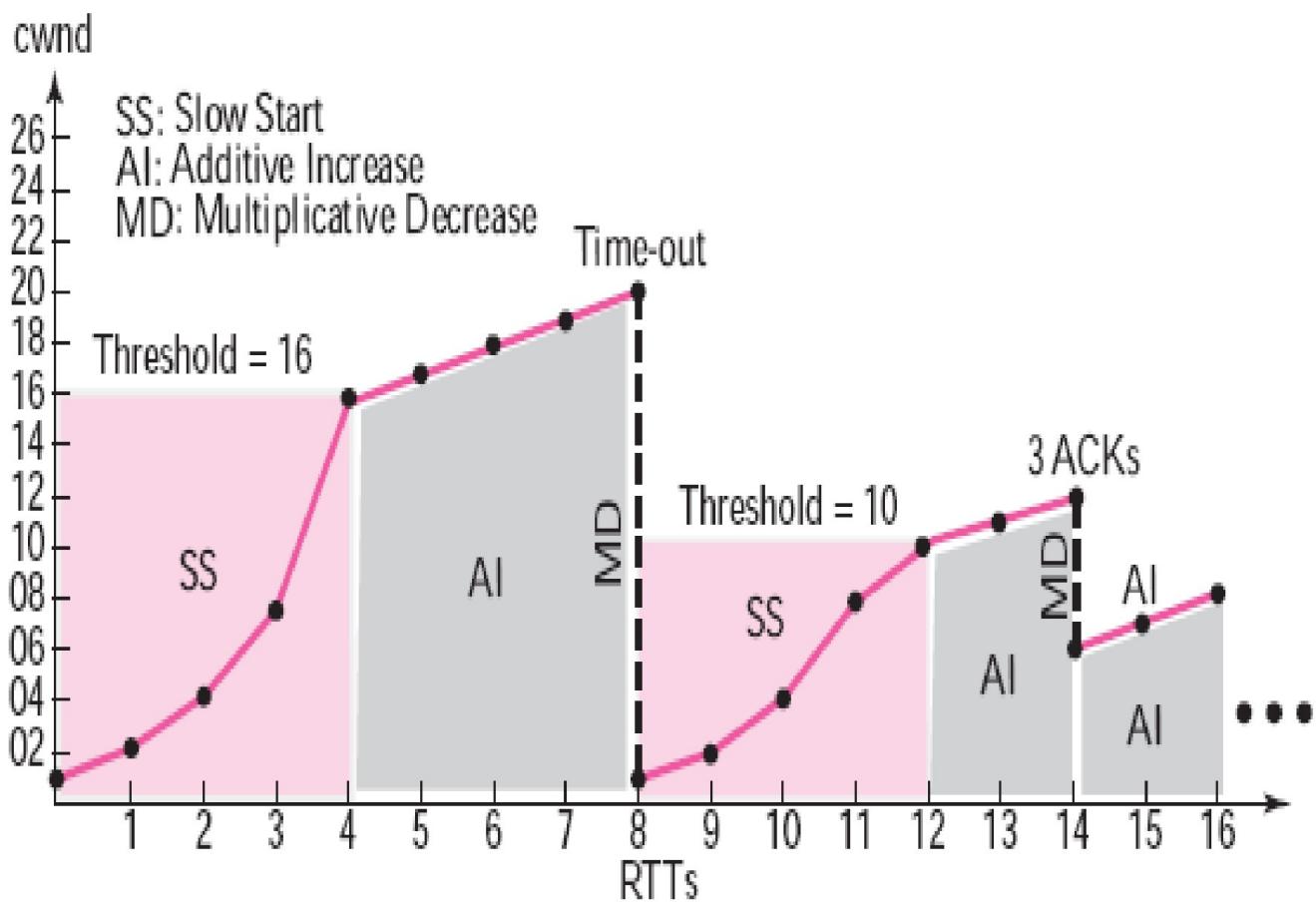
Congestion Detection: Multiplicative Decrease

- 2. If three duplicate ACKs are received, there is a weaker possibility of congestion:**

A segment may have been dropped but some segments after that have arrived safely since three duplicate ACKs are received - **Fast Recovery**

- In this case, TCP has a weaker reaction as shown below:**
 - 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.

Figure 15.37 Congestion example



Fast Recovery

- Optional in TCP
- When 3 duplicate ACKs arrive, which is interpreted as light congestion.
- Like congestion avoidance, this algorithm is also additive increase.

TCP Timers

- Retransmission Timer
- Persistence Timer
- Keepalive Timer
- TIME-WAIT Timer

Retransmission Timer

- To retransmit lost segments, TCP employs one retransmission timer (for the whole connection period).
- Handles the retransmission time-out (RTO), the waiting time for an acknowledgment of a segment.
- When TCP sends the segment in front of the sending queue, it starts the timer.
- If the queue is empty, TCP stops the timer; otherwise, TCP restarts the timer.

Persistence Timer

- To deal with a zero-window-size advertisement, TCP needs another timer.
- Receiving TCP announces a window size of zero, sending TCP stop transmitting segments until receiving TCP sends an ACK announcing non zero window size.
- ACK can be lost.
- Sending TCP has not received an acknowledgment and waits for the other TCP to send an acknowledgment advertising the size of the window.
- Both TCPs might continue to wait for each other forever (a deadlock).
- To correct this deadlock, TCP uses a **persistence timer** for each connection

Persistence Timer

- When the sending TCP receives an acknowledgment with a window size of zero, it starts a persistence timer.
- When the persistence timer goes off, the sending TCP sends a special segment called a *probe*.
- This segment contains only 1 byte of new data.
- The probe causes the receiving TCP to resend the acknowledgment.
- value of the persistence timer is set to the value of the retransmission time.
- If no response received, another probe, value of persistence timer is doubled.

Persistence Timer

- Sender continues sending probe and doubling persistence timer until value reaches threshold (usually 60 s).
- After threshold, probe is send every 60 s until window is reopened.

Keepalive Timer

- Used to prevent long idle connection between two TCP.
- Server Resets Keepalive timer each time it hears from client.
- Timeout is usually 2 hours.
- If the server does not hear from the client after 2 hours, it sends a probe segment.
- If there is no response after 10 probes, each of which is 75 s apart, it assumes that the client is down and terminates the connection.

TIME-WAIT Timer

- The TIME-WAIT (2MSL) timer is used during connection termination.
- MSL (Maximum Segment Life) : amount of time any segment can exist in a network before being discarded.
- Common MSL values are 30 seconds, 1 minute, or even 2 minutes
- The 2MSL timer is used when TCP performs an active close and sends the final ACK.
- The connection must stay open for 2 MSL amount of time to allow TCP to resend the final ACK in case the ACK is lost.