# Computer Networks(CS31006)

## Spring Semester (2021-2022)

## TCP Congestion Control


**Prof. Sudip Misra**

Department of Computer Science and Engineering
Indian Institute of Technology Kharagpur
Email: smisra@sit.iitkgp.ernet.in
Website: http://cse.iitkgp.ac.in/~smisra/
Research Lab: cse.iitkgp.ac.in/~smisra/swan/


# What is Congestion

- The number of packets transmitted on the network is greater than the capacity of the network
- More specifically
- Packets come to some router at a rate higher than the rate at which the router can process packets and send them out
- Causes packets to get queued at router buffers
- Eventually the buffer fills up, causes packets to start getting dropped
- Why is it bad?
  - Retransmissions cause bandwidth wastage
  - Delay is increased
  - Congestion Collapse - retransmissions due to drop due to congestion can further increase congestion!!

# Congestion Control

- 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 first is related to the congestion in the middle (network); the second is related to the congestion at the end.
- The actual size of the window is the minimum of these two.

**Actual window size** 5 **minimum (*rwnd*, *cwnd*)**

# Congestion Detection

- TCP sender can detect the possible existence of congestion in the network.
- The TCP sender uses the occurrence of two events as signs of congestion in the network: time-out and receiving three duplicate ACKs.
- **Time-out:** 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.
- **Receiving of duplicate ACKs :** Receiving of three duplicate ACKs (four ACKs with the same acknowledgment number).
- The congestion in the case of three duplicate ACKs can be less severe than in the case of time-out.
- When a receiver sends three duplicate ACKs, it means that one segment is missing, but three segments have been received.
- The network is either slightly congested or has recovered from the congestion.

# Adjusting Sending Rate

- Based on TCP Congestion Window (cwnd)
- Limits how much data can be in transit (similar to receiver window advertisement, but purpose is different)
- Max. window size at sender at any time = min (cwnd, recvAdvertisedWindow)
- cwnd is varied to control sending rate to address congestion
- Varied by sender, congestion control is sender-side task
- Receiver advertised window varied to address end-to-end flow control
- Basic principle
  - On detecting packet drop, decrease cwnd
  - On receiving ack, increase cwnd

# Congestion Policies

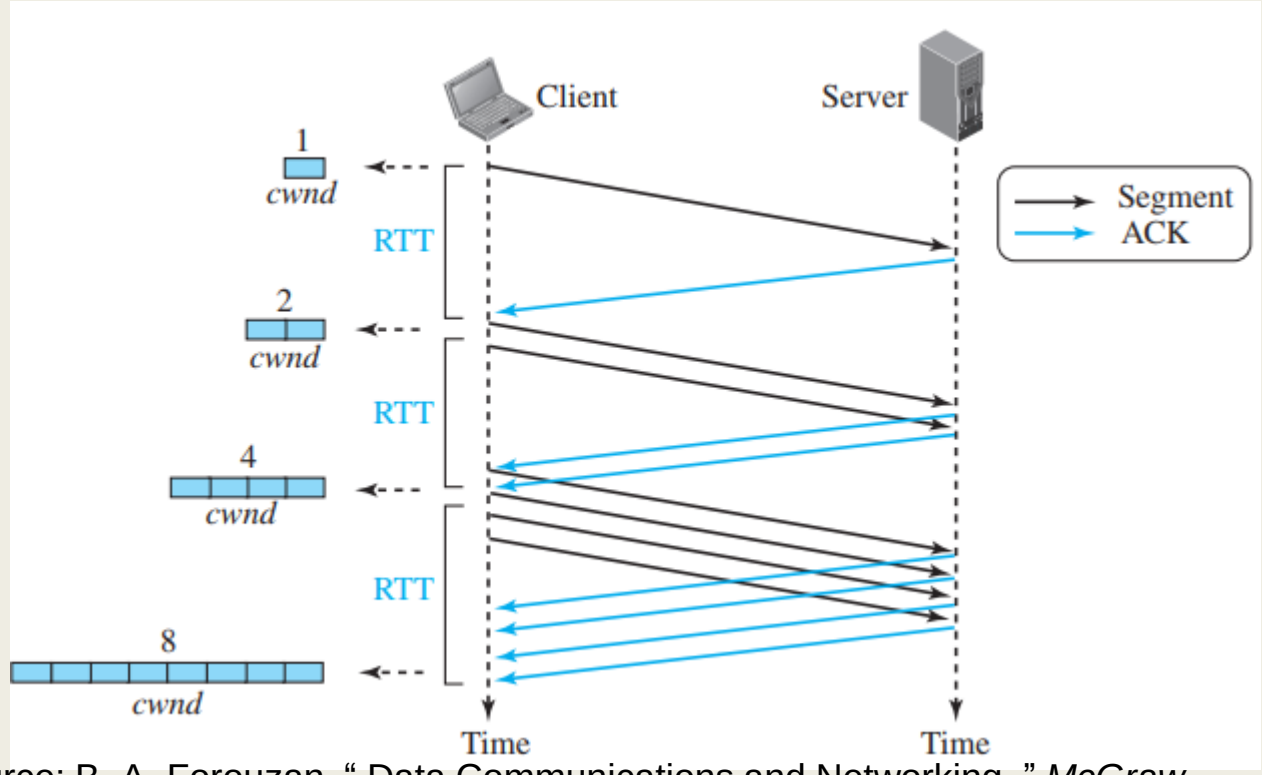
TCP's general policy for handling congestion is based on three algorithms:

- Slow start
- Congestion avoidance
- Fast recovery.

# Slow Start: Exponential Increase

| | | |
|---|---|---|
| **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** | → | $cwnd = cwnd + 4 = 4 + 4 = 8 \rightarrow 2^3$ |

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

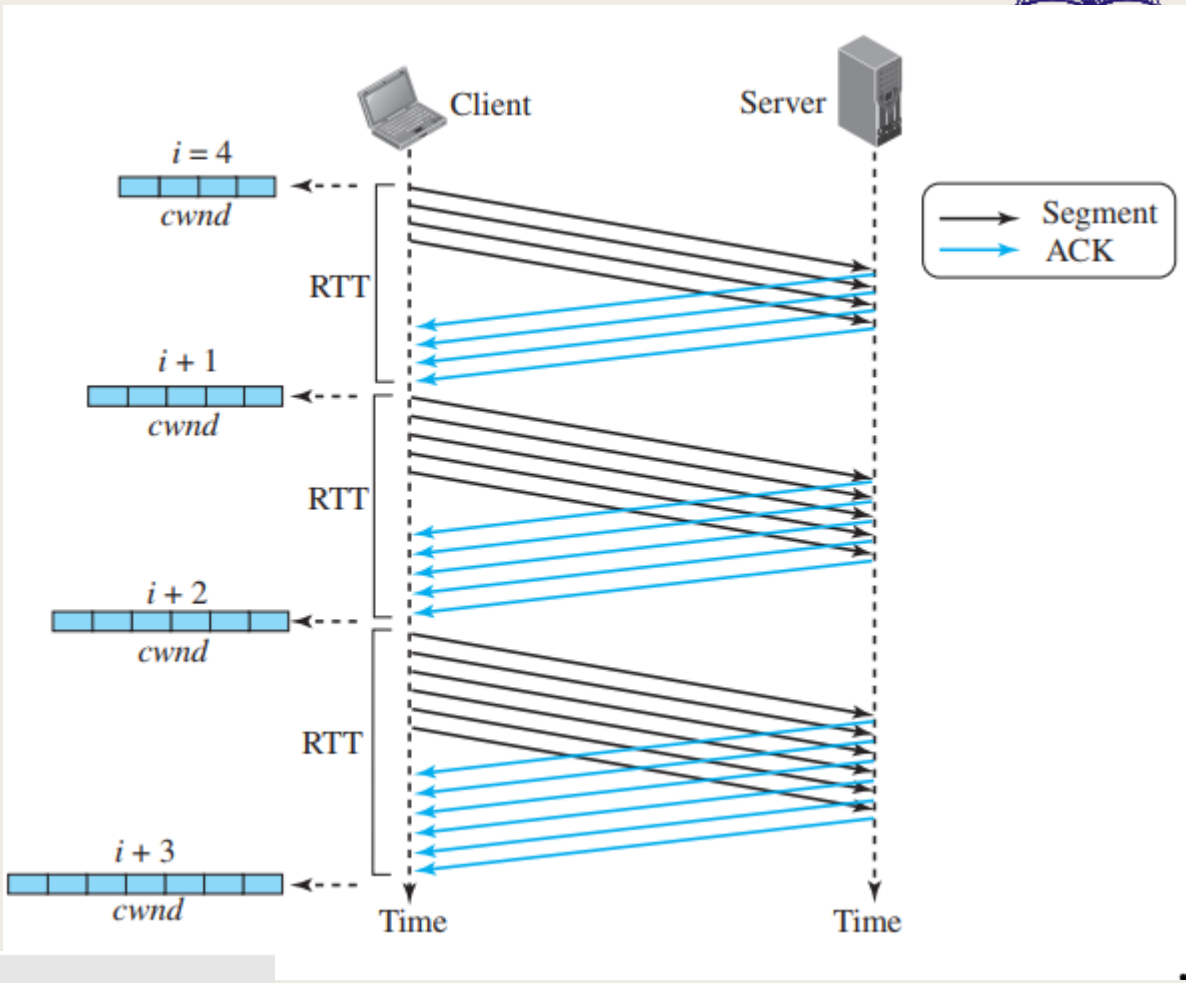



Source: B. A. Forouzan, " Data Communications and Networking ," *McGraw-Hill Forouzan Networking Series*,5E.

# Congestion Avoidance: Additive Increase

- 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.

- **If an ACK arrives, *cwnd* 5 *cwnd* 1 (1/*cwnd*)**




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



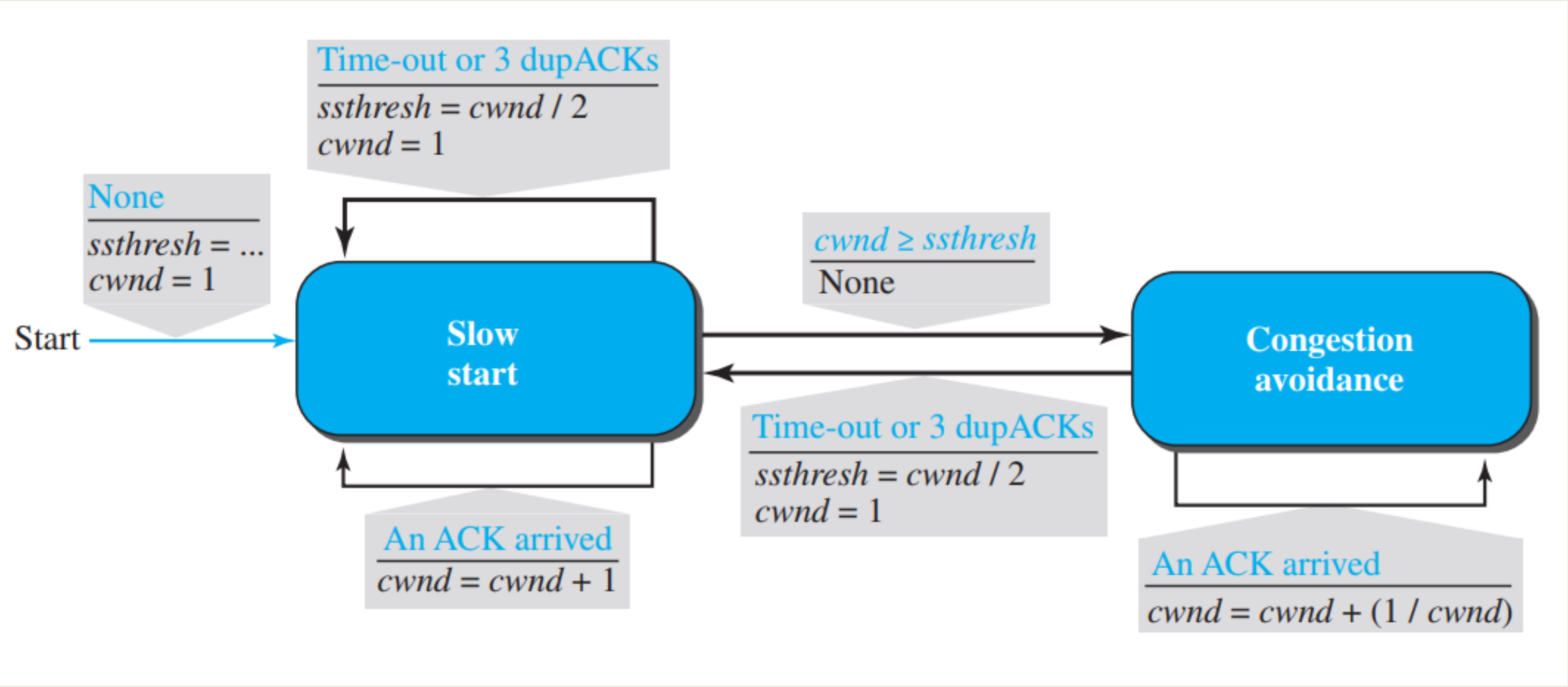
Source: B. A. Forouzan, " Data Communications and Networking ," *McGraw-Hill Forouzan Networking Series*,5E.

# 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 (after the three duplicate ACKs that trigger the use of this algorithm).

**If a duplicate ACK arrives, *cwnd* 5 *cwnd* 1 (1 / *cwnd*).**

# Taho TCP




Source: B. A. Forouzan, " Data Communications and Networking ," *McGraw-Hill Forouzan Networking Series*,5E.

# Congestion Detection in 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.
- In other words, not only does TCP restart from scratch, but it also learns how to adjust the threshold.
- If no congestion is detected while reaching the threshold, TCP learns that the ceiling of its ambition is reached; it should not continue at this speed.
- It moves to the congestion avoidance state and continues in that state.

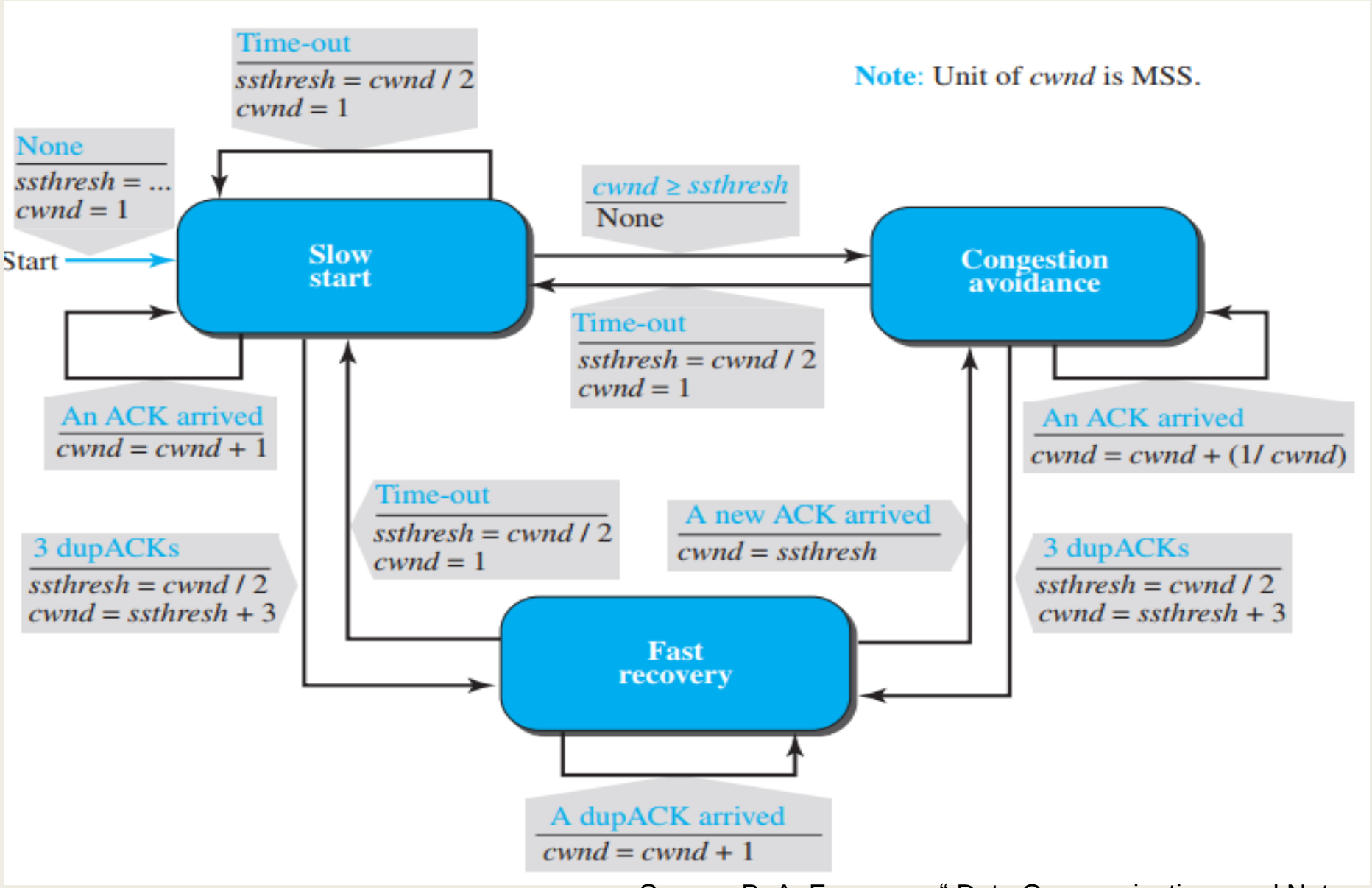
# Congestion Avoidance in Taho TCP

- In the congestion-avoidance state, the size of the congestion window is increased by 1 each time a number of ACKs equal to the current size of the window has been received.
- For example, if the window size is now 5 MSS, five more ACKs should be received before the size of the window becomes 6 MSS.
- There is no ceiling for the size of the congestion window in this state; 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.
- Although in this version of TCP the size of *ssthresh* is continuously adjusted in each congestion detection, this does not mean that it necessarily becomes lower than the previous value.
- For example, if the original *ssthresh* value is 8 MSS and congestion is detected when TCP is in the congestion avoidance state and the value of the *cwnd* is 20, the new value of the *ssthresh* is now 10, which means it has been increased.

# 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 (or starts a new round if it is already in this 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.
- It behaves like the slow start, in which the *cwnd* grows exponentially, but the *cwnd* starts with the value of *ssthresh* plus 3 MSS (instead of 1).
- 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, as though the three duplicate ACKs have not occurred, and transition is from the slow-start state to the congestion-avoidance state.

# Reno TCP

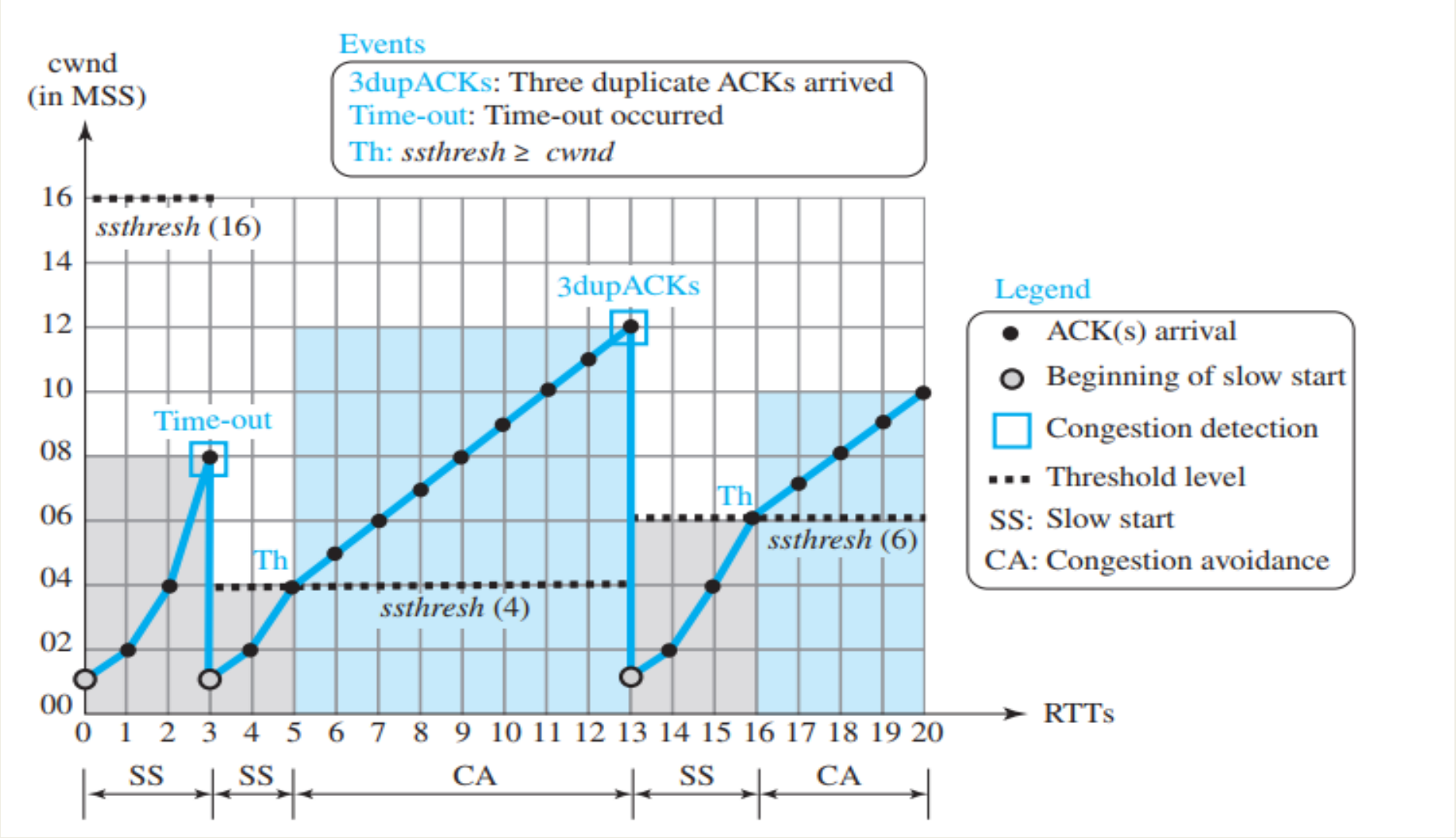



Source: B. A. Forouzan, " Data Communications and Networking ," *McGraw-Hill Forouzan Networking Series*,5E.

# Example

TCP starts data transfer and sets the *ssthresh* variable to an ambitious value of 16 MSS. TCP begins at the slow-start (SS) state with the *cwnd* = 1. The congestion window grows exponentially, but a time-out occurs after the third RTT (before reaching the threshold). TCP assumes that there is congestion in the network. It immediately sets the new *ssthresh* = 4 MSS (half of the current *cwnd,* which is 8) and begins a new slow-start (SA) state with *cwnd* = 1 MSS. The congestion window grows exponentially until it reaches the newly set threshold. TCP now moves to the congestion-avoidance (CA) state and the congestion window grows additively until it reaches *cwnd* = 12 MSS. At this moment, three duplicate ACKs arrive, another indication of congestion in the network. TCP again halves the value of *ssthresh* to 6 MSS and begins a new slow-start (SS) state. The exponential growth of the *cwnd* continues. After RTT 15, the size of *cwnd* is 4 MSS. After sending four segments and receiving only two ACKs, the size of the window reaches the *ssthresh* (6) and TCP moves to the congestion-avoidance state. The data transfer now continues in the congestion avoidance (CA) state until the connection is terminated after RTT 20.

# Example




Source: B. A. Forouzan, " Data Communications and Networking ," *McGraw-Hill Forouzan Networking Series*,5E.

# TCP Throughput

- The throughput for TCP, which is based on the congestion window behavior, can be easily found if the cwnd is a constant (flat line) function of RTT. The throughput with this unrealistic assumption is throughput = cwnd / RTT.

- In this assumption, TCP sends a cwnd bytes of data and receives acknowledgement for them in RTT time. The behavior of TCP, as shown in Figure 24.35, is not a flat line; it is like saw teeth, with many minimum and maximum values.

- If each tooth were exactly the same, we could say that the throughput = [(maximum + minimum) / 2] / RTT.

- However, we know that the value of the maximum is twice the value of the minimum because in each congestion detection the value of cwnd is set to half of its previous value.

- So the throughput can be better calculated as :

**throughput** 5 **(0.75) Wmax / RTT**

in which W max is the average of window sizes when the congestion occurs.

# TCP Timers

To perform their operations smoothly, most TCP implementations use at least four timers:

- Retransmission
- Persistence
- Keepalive
- TIME-WAIT.

# Retransmission Timer

To retransmit lost segments, TCP employs one retransmission timer (for the whole connection period) that handles the retransmission time-out (RTO), the waiting time for an acknowledgment of a segment.

We can define the following rules for the retransmission timer:

**1.** When TCP sends the segment in front of the sending queue, it starts the timer.
**2.** When the timer expires, TCP resends the first segment in front of the queue, and restarts the timer.
**3.** When a segment or segments are cumulatively acknowledged, the segment or segments are purged from the queue.
**4.** If the queue is empty, TCP stops the timer; otherwise, TCP restarts the timer.

# RTT Timer

**Measured RTT**

- The measured round-trip time for a segment is the time required for the segment to reach the destination and be acknowledged, although the acknowledgment may include other segments.
- In TCP only one RTT measurement can be in progress at any time.
- This means that if an RTT measurement is started, no other measurement starts until the value of this RTT is finalized. We use the notation *RTTM* to stand for measured RTT.

**In TCP, there can be only one RTT measurement in progress at any time.**

# RTT Timer: Contd.

**Smoothed RTT**

- The measured RTT, RTTM, is likely to change for each round trip. The fluctuation is so high in today's Internet that a single measurement alone cannot be used for retransmission time-out purposes.
- Most implementations use a smoothed RTT, called *RTTS,* which is a weighted average of RTTM and the previous RTTS, as shown below:

| | | |
|---|---|---|
| **Initially** | → | **No value** |
| **After first measurement** | → | $\mathbf{RTT_S = RTT_M}$ |
| **After each measurement** | → | $\mathbf{RTT_S = (1 - \alpha)\ RTT_S + \alpha \times RTT_M}$ |

- The value of α is implementation-dependent, but it is normally set to 1/8.
- In other words, the new RTTS is calculated as 7/8 of the old RTTS and 1/8 of the current RTTM.



Source: B. A. Forouzan, " Data Communications and Networking ," *McGraw-Hill Forouzan Networking Series*,5E.

# RTT Timer: Contd.

## RTT Deviation

- Most implementations do not just use $RTT_S$; they also calculate the RTT deviation, called $RTT_D$, based on the $RTT_S$ and $RTT_M$, using the following formulas.
- The value of β is also implementation-dependent, but is usually set to 1/4.

| | | |
|---|---|---|
| **Initially** | → | **No value** |
| **After first measurement** | → | $\mathbf{RTT_S = RTT_M}$ |
| **After each measurement** | → | $\mathbf{RTT_S = (1 - \alpha)\, RTT_S + \alpha \times RTT_M}$ |



Source: B. A. Forouzan, " Data Communications and Networking ," *McGraw-Hill Forouzan Networking Series*,5E.

# Retransmission Time-Out

- The value of RTO is based on the smoothed roundtrip time and its deviation.
- Most implementations use the following formula to calculate the RTO:

| | | |
|---|---|---|
| **Original** | $\rightarrow$ | **Initial value** |
| **After any measurement** | $\rightarrow$ | $\mathbf{RTO = RTT_S + 4 \times RTT_D}$ |



Source: B. A. Forouzan, " Data Communications and Networking ," *McGraw-Hill Forouzan Networking Series*,5E.

# Persistent Timer

- TCP uses a **persistence timer** for each connection.
- 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.
- It has a sequence number, but its sequence number is never acknowledged; it is even ignored in calculating the sequence number for the rest of the data.
- The probe causes the receiving TCP to resend the acknowledgment.

# Keep Alive Timer

- A **keepalive timer** is used in some implementations to prevent a long idle connection between two TCPs.
- Suppose that a client opens a TCP connection to a server, transfers some data, and becomes silent. Perhaps the client has crashed.
- In this case, the connection remains open forever.
- To remedy this situation, most implementations equip a server with a keepalive timer.

# Time Wait Timer

- The TIME-WAIT (2MSL) timer is used during connection termination.
- The maximum segment lifetime (MSL) is the amount of time any segment can exist in a network before being discarded.
- The implementation needs to choose a value for MSL. Common 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.
- This requires that the RTO timer at the other end times out and new FIN and ACK segments are resent.

# Thank You!!!