Transmission Control Protocol (TCP)

Unit-IV

Session Objectives

- Discuss the TCP, its services and features protocol
- Discuss how TCP provides a connection-oriented service, flow and error control and congestion control

Session Outcomes

At the end of this session, participants will be able to

- Describing the TCP services and protocol operations format

Agenda

Presentation Outline

Transmission Control Protocol (TCP)

- Transmission Control Protocol (TCP) is a connection-oriented, reliable protocol.
- TCP explicitly defines connection establishment, data transfer, and connection teardown phases to provide a connection-oriented service.
- TCP uses a combination of GBN and SR protocols to provide reliability

TCP Services

- Process to Process Communication
- Stream Delivery Service
- Sending and Receiving Buffers
- Segments
- Full Duplex communication
- Connection Oriented Service
- Reliable Service
- Encapsulation and Decapsulation

Stream Delivery Service

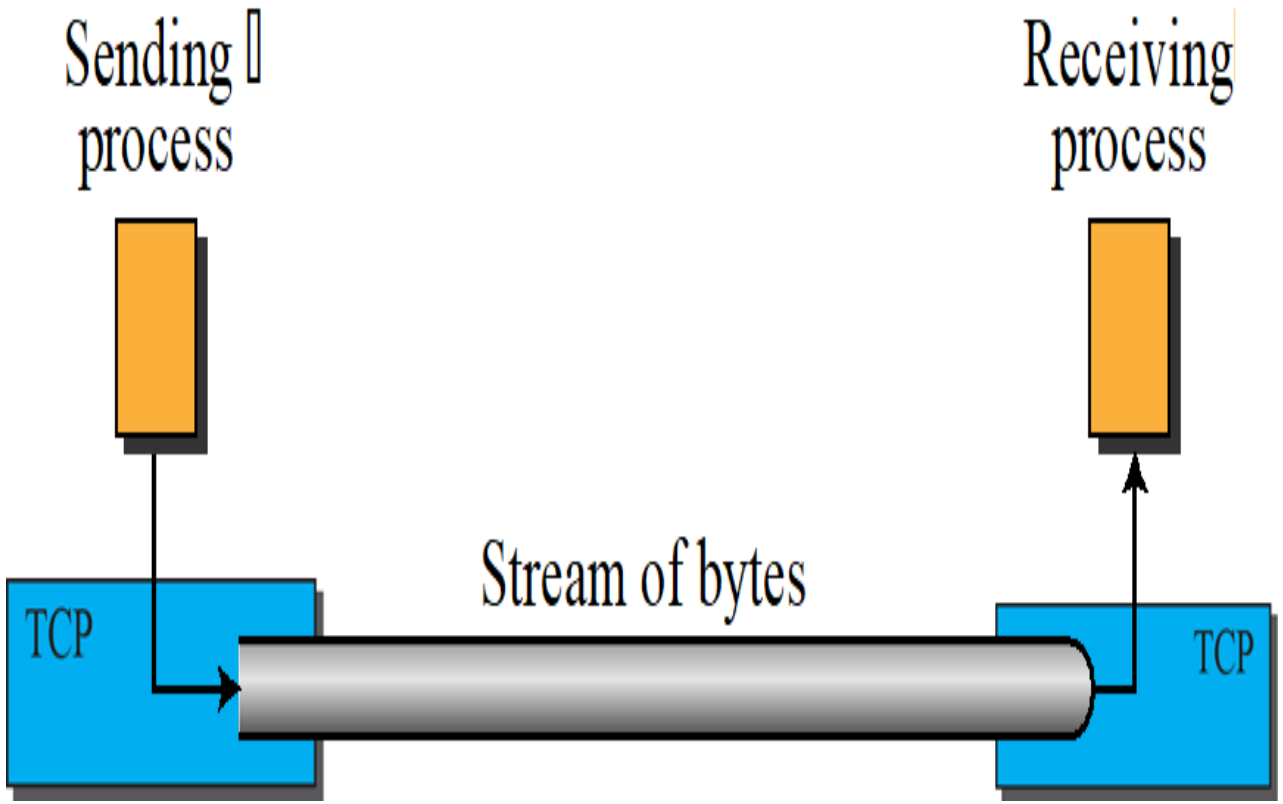
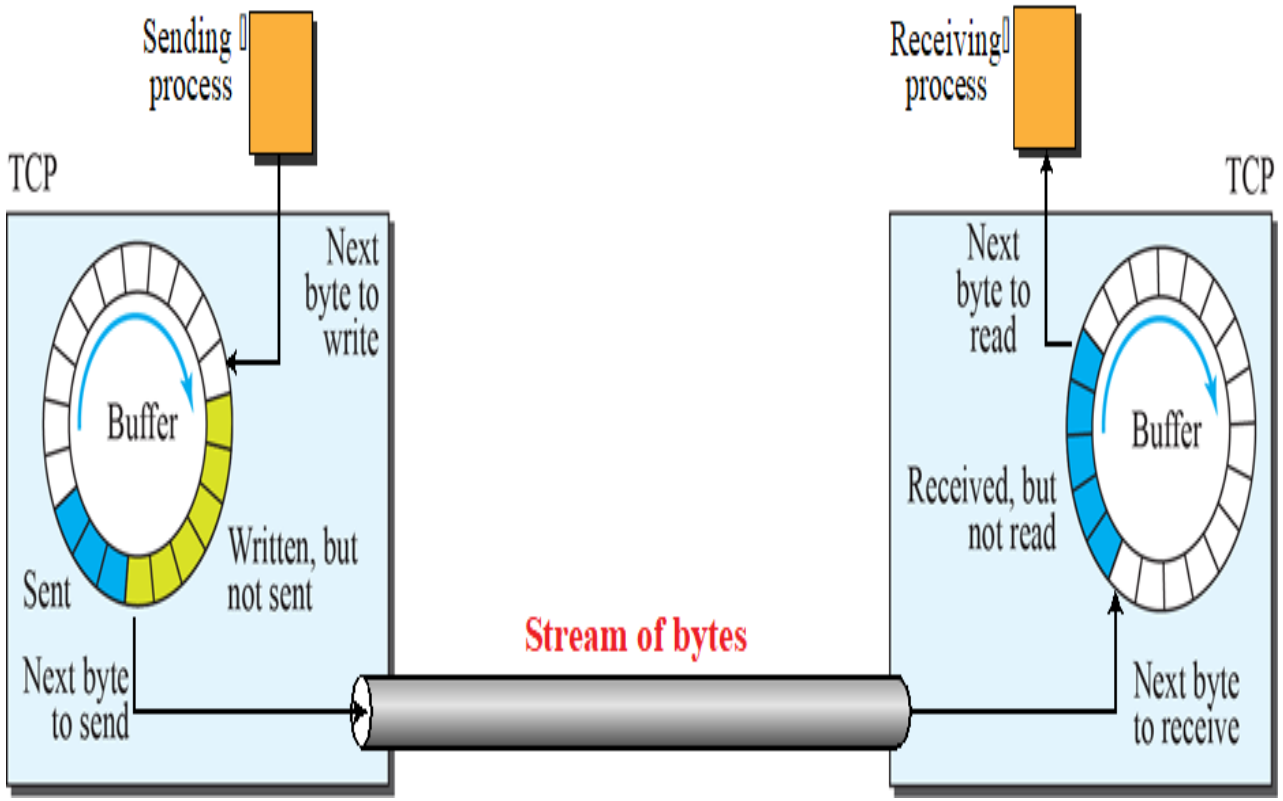
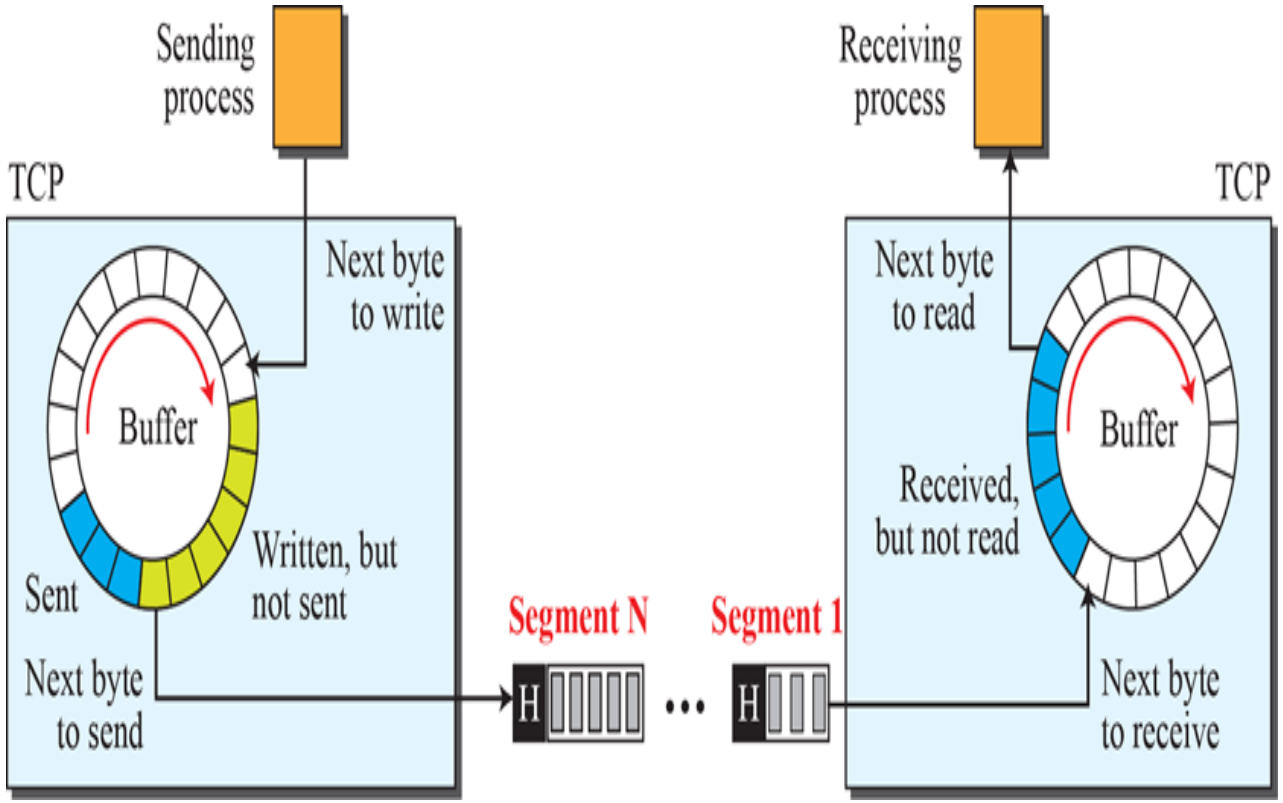


Figure: Forwarding process

Sending and receiving buffers



TCP segments



TCP Features

Numbering system

- Byte Number
- Sequence Number
- Acknowledgement Number

Byte Number

- Each byte should be numbered
- Random number – 1057
- Data contains 6000 bytes
- 1057 – 7056

Sequence Number

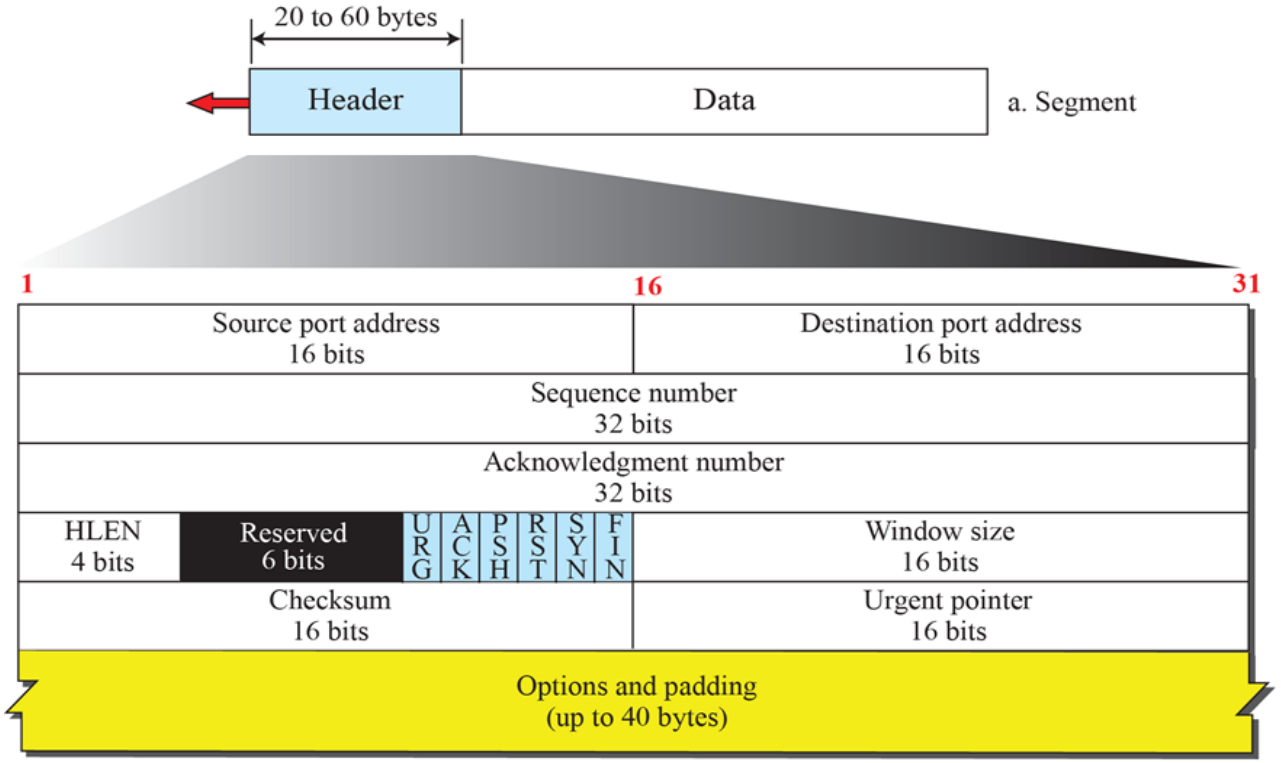
- Sequence number for each segment is the number of the first byte carried in that segment
- Suppose a TCP connection is transferring a file of 5,000 bytes. The first byte is numbered 10,001. What are the sequence numbers for each segment if data are sent in five segments, each carrying 1,000 bytes?
  Solution The following shows the sequence number for each segment

Segment 1	→	Sequence Number:	10,001	Range:	10,001	to	11,000
Segment 2	→	Sequence Number:	11,001	Range:	11,001	to	12,000
Segment 3	→	Sequence Number:	12,001	Range:	12,001	to	13,000
Segment 4	→	Sequence Number:	13,001	Range:	13,001	to	14,000
Segment 5	→	Sequence Number:	14,001	Range:	14,001	to	15,000

Acknowledgement Number

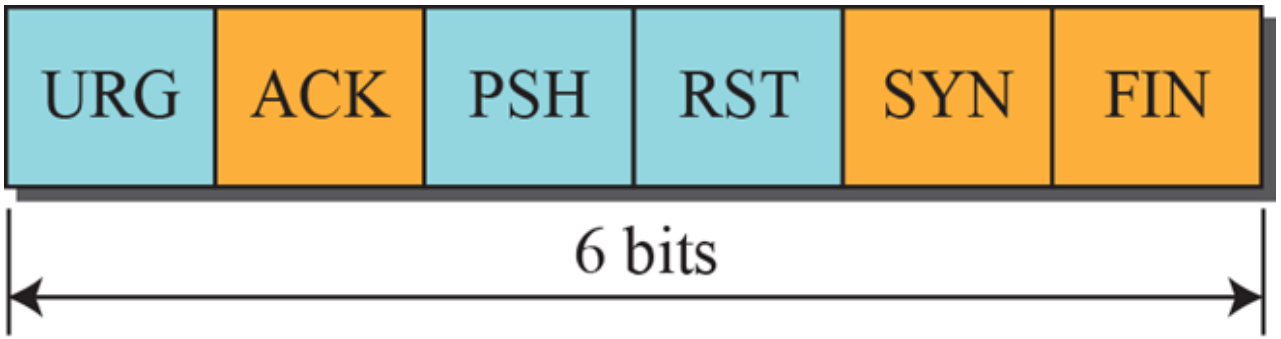
- The value of the acknowledgment field in a segment defines the number of the next byte a party expects to receive.
- The acknowledgement number is cumulative.
- **Segment**: A packet in TCP is called a segment.

TCP segment format



b. Header

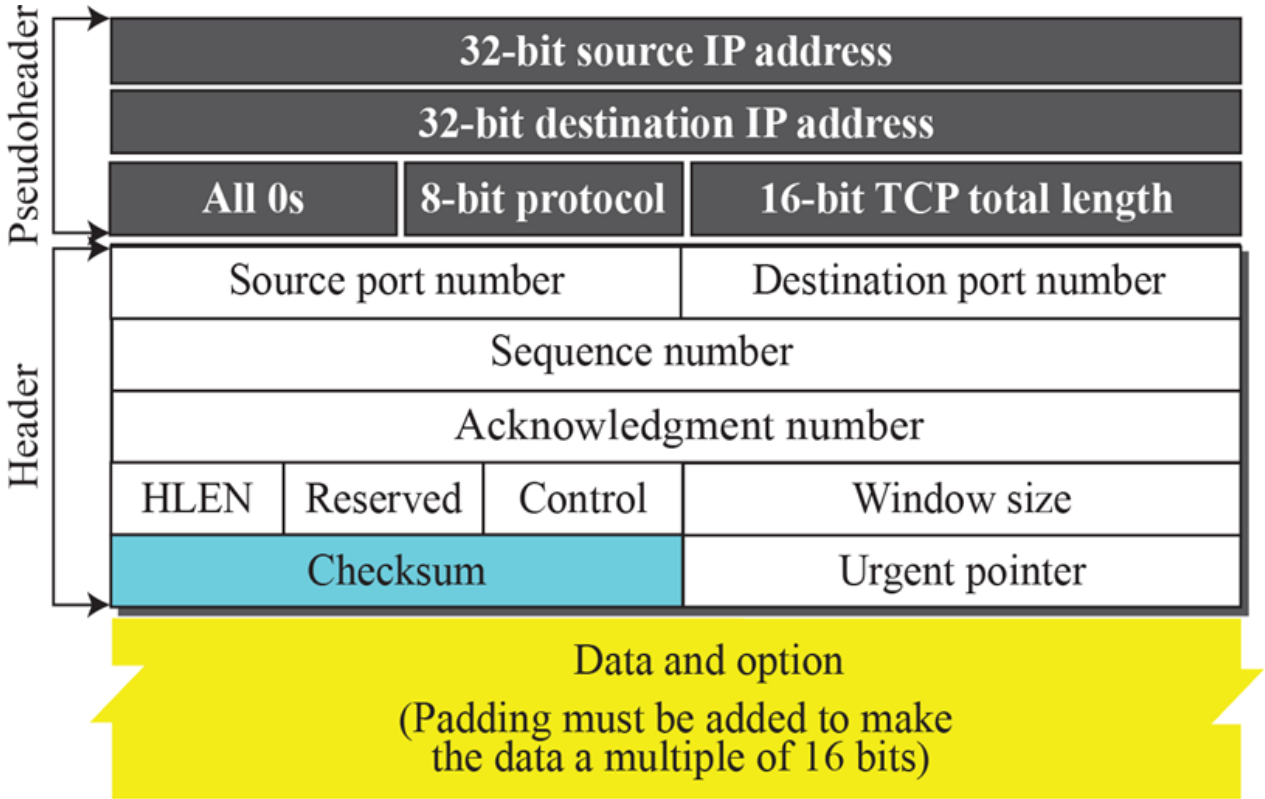
Control field



URG: Urgent pointer is valid
ACK: Acknowledgment is valid
PSH: Request for push
RST: Reset the connection
SYN: Synchronize sequence numbers
FIN: Terminate the connection

The length of the header can be between 20 and 60 bytes. Therefore, the value of this field is always between 5 (5 x 4 = 20) and 15 (15 x 4 = 60).

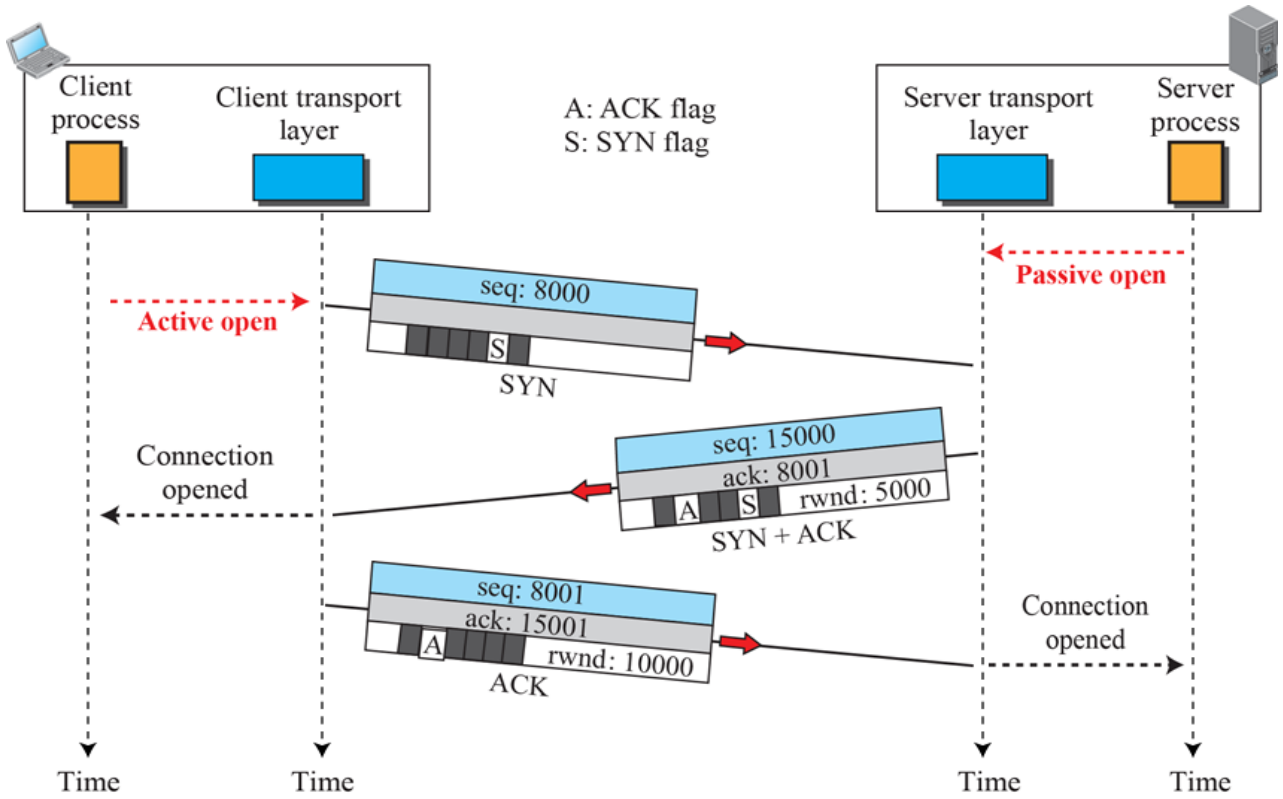
Checksum calculation



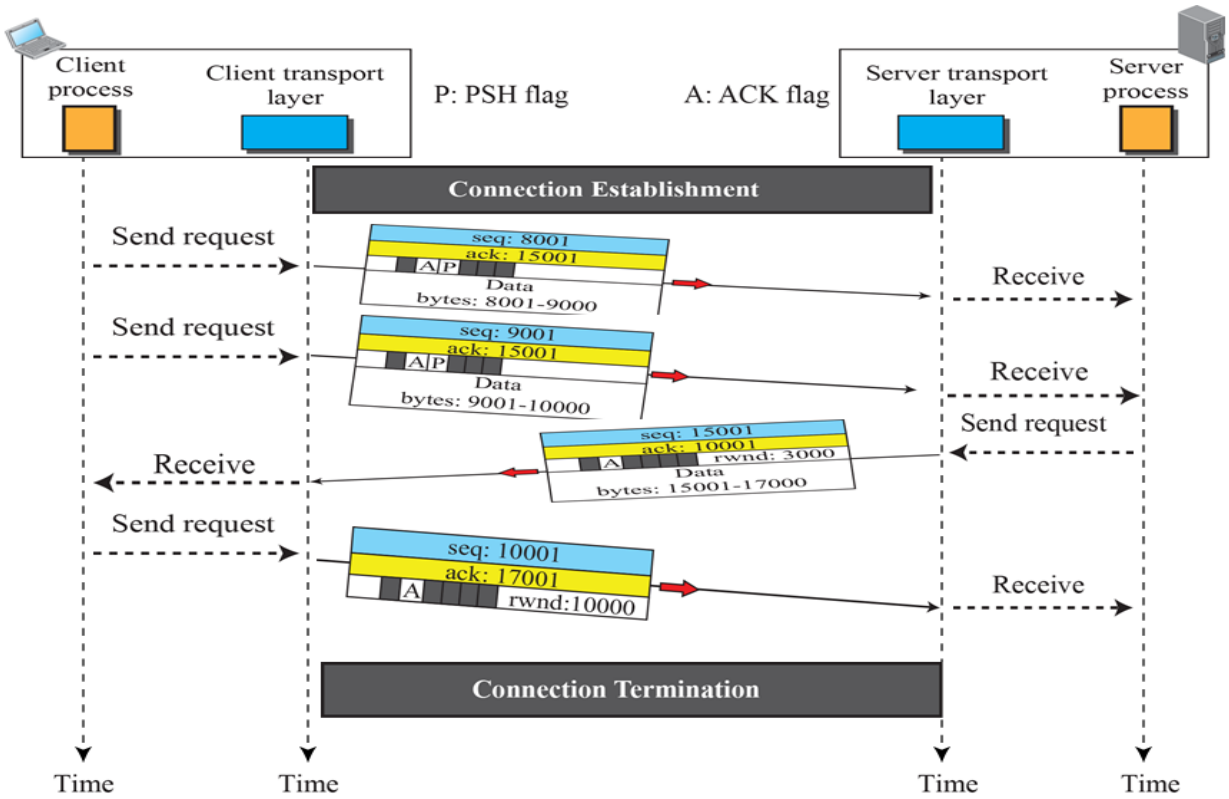
A TCP Connection

- TCP is connection-oriented.
- All of the segments belonging to a message are then sent over this logical path.
- Using a single logical pathway for the entire message facilitates the acknowledgment process as well as retransmission of damaged or lost frames.
- How TCP, which uses the services of IP, a connectionless protocol, can be connection-oriented?.
- The point is that a TCP connection is logical, not physical.
- TCP operates at a higher level.
- TCP uses the services of IP to deliver individual segments to the receiver, but it controls the connection itself.

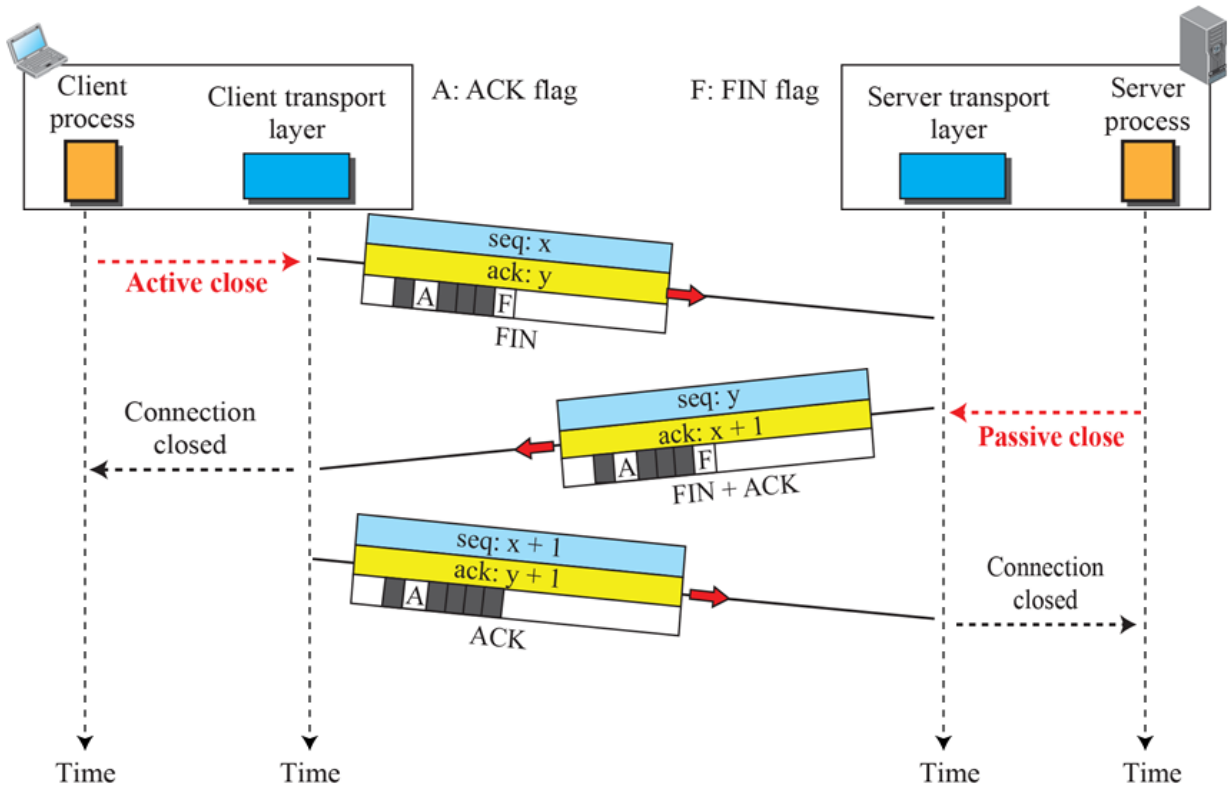
Connection establishment



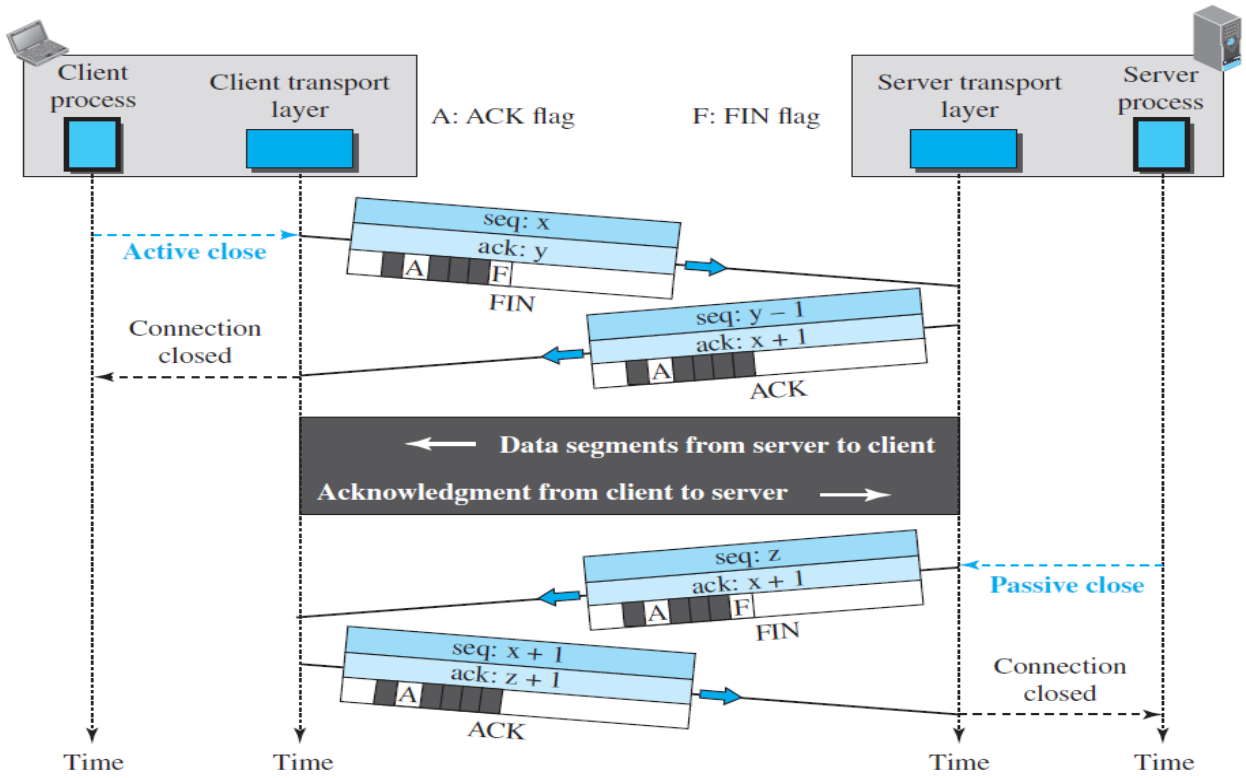
Data transfer



Connection termination



Half-close

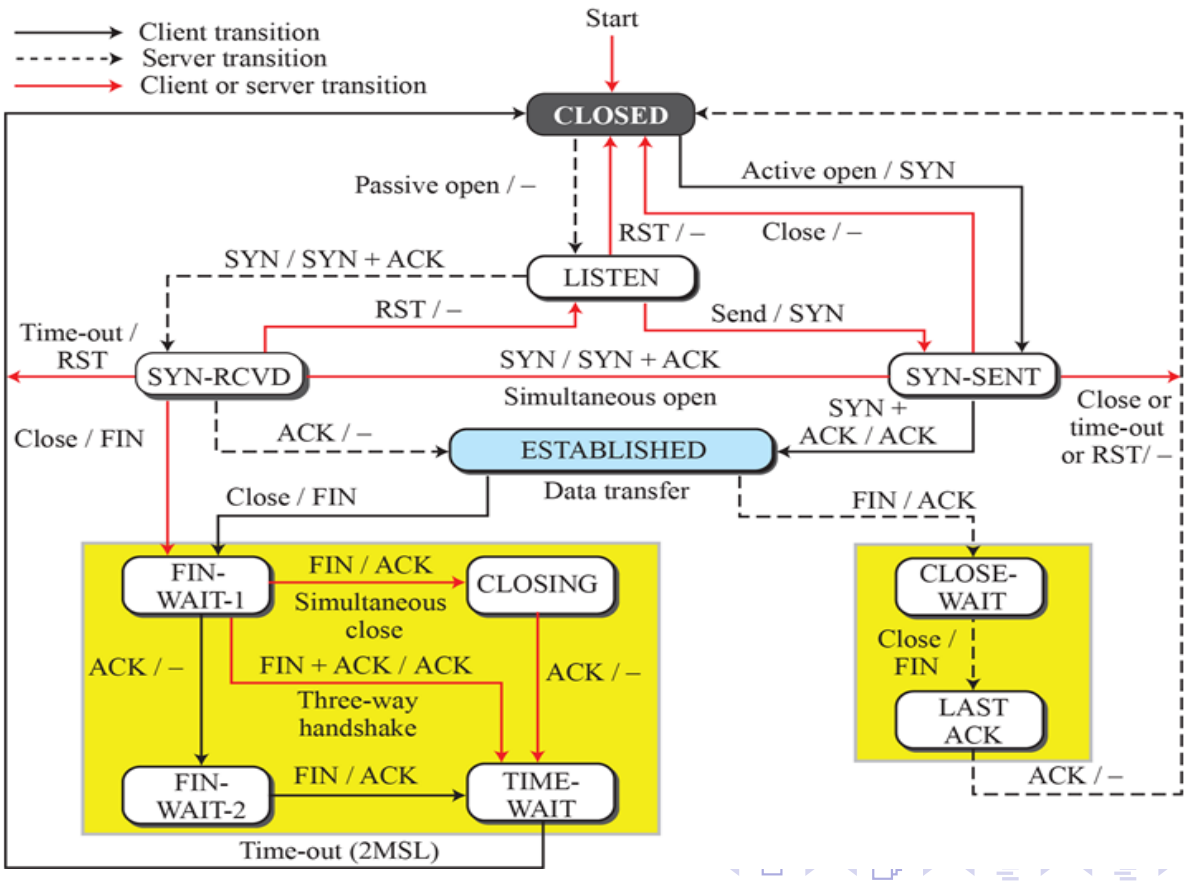

Client process
Client transport layer
A: ACK flag
F: FIN flag
Server transport layer
Server process
Active close
seq: x
ack: y
A
F
FIN
Connection closed
seq: y – 1
ack: x + 1
A
ACK
Data segments from server to client
Acknowledgment from client to server
seq: z
ack: x + 1
A
F
FIN
Passive close
seq: x + 1
ack: z + 1
A
ACK
Connection closed
Time
Time
Time
Time

State Transition Diagram

- To keep track of all the different events happening during connection establishment, connection termination, and data transfer, TCP is specified as the finite state machine (FSM

State Transition Diagram

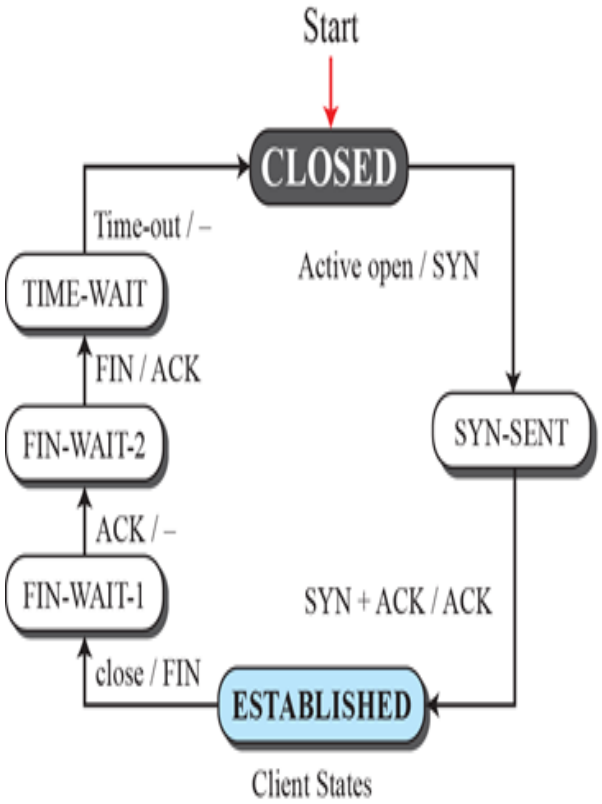
- To keep track of all the different events happening during connection establishment, connection termination, and data transfer, TCP is specified as the finite state machine (FSM

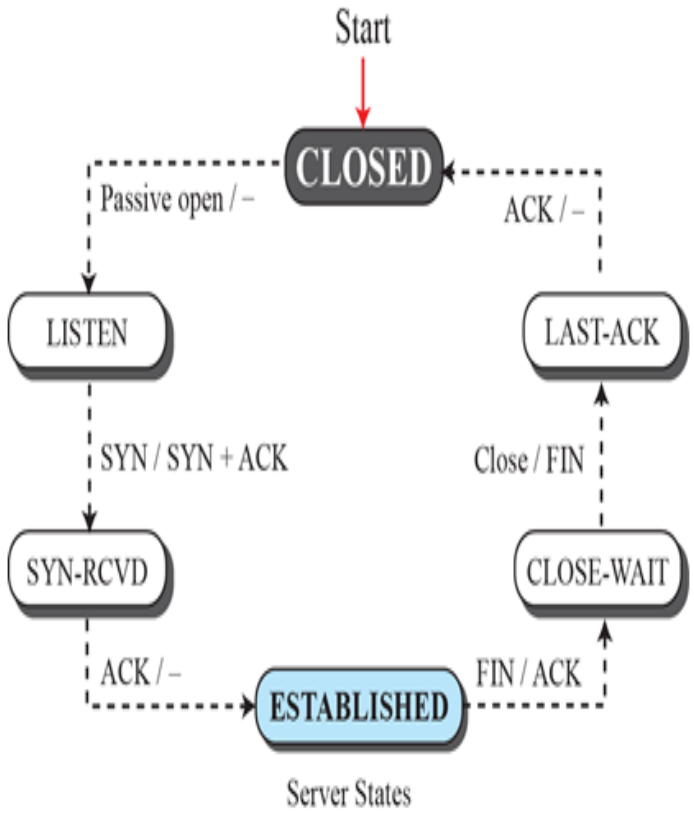


States for TCP

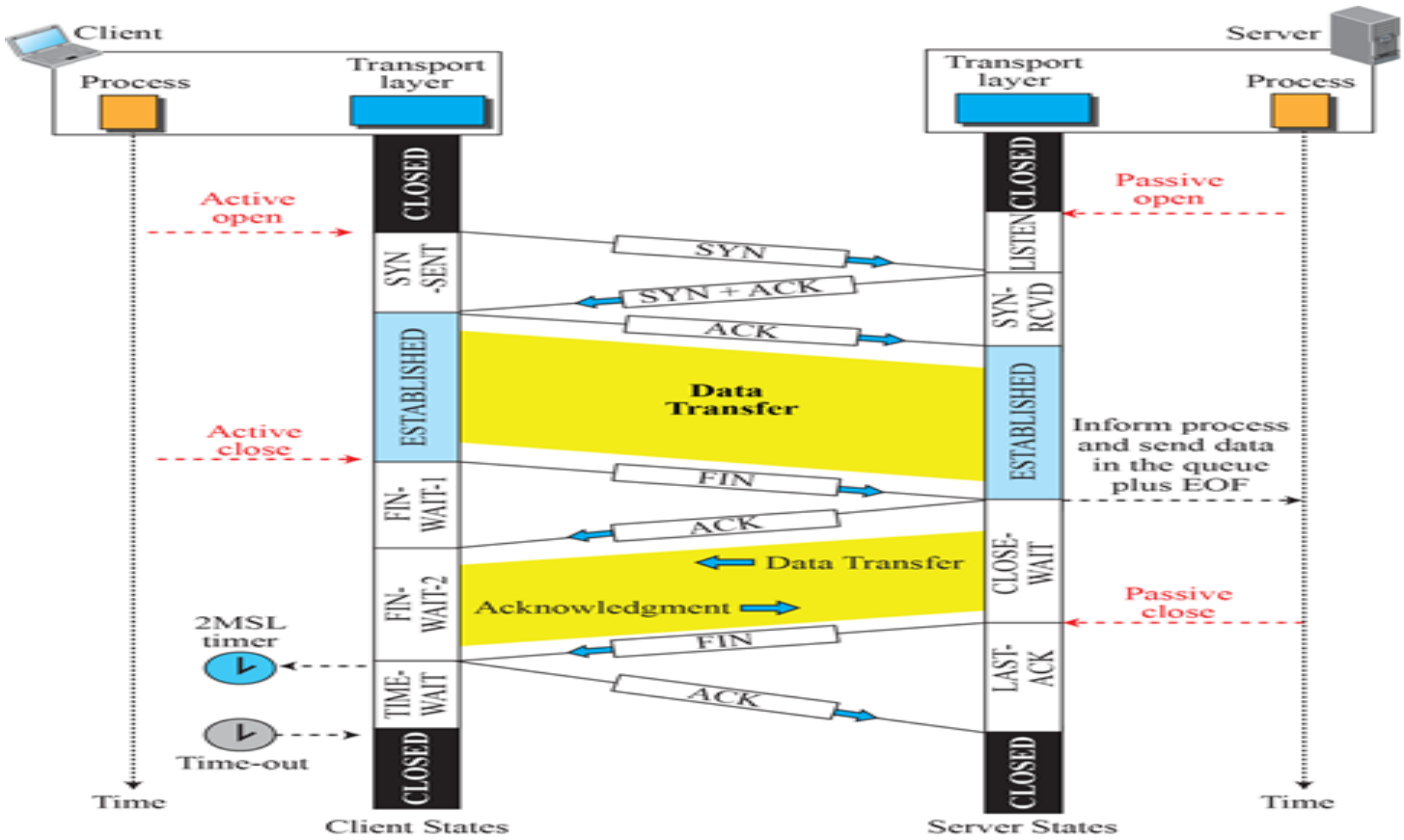
State	*Description*
CLOSED	No connection exists
LISTEN	Passive open received; waiting for SYN
SYN-SENT	SYN sent; waiting for ACK
SYN-RCVD	SYN+ACK sent; waiting for ACK
ESTABLISHED	Connection established; data transfer in progress
FIN-WAIT-1	First FIN sent; waiting for ACK
FIN-WAIT-2	ACK to first FIN received; waiting for second FIN
CLOSE-WAIT	First FIN received, ACK sent; waiting for application to close
TIME-WAIT	Second FIN received, ACK sent; waiting for 2MSL time-out
LAST-ACK	Second FIN sent; waiting for ACK
CLOSING	Both sides decided to close simultaneously

Half-close connection termination





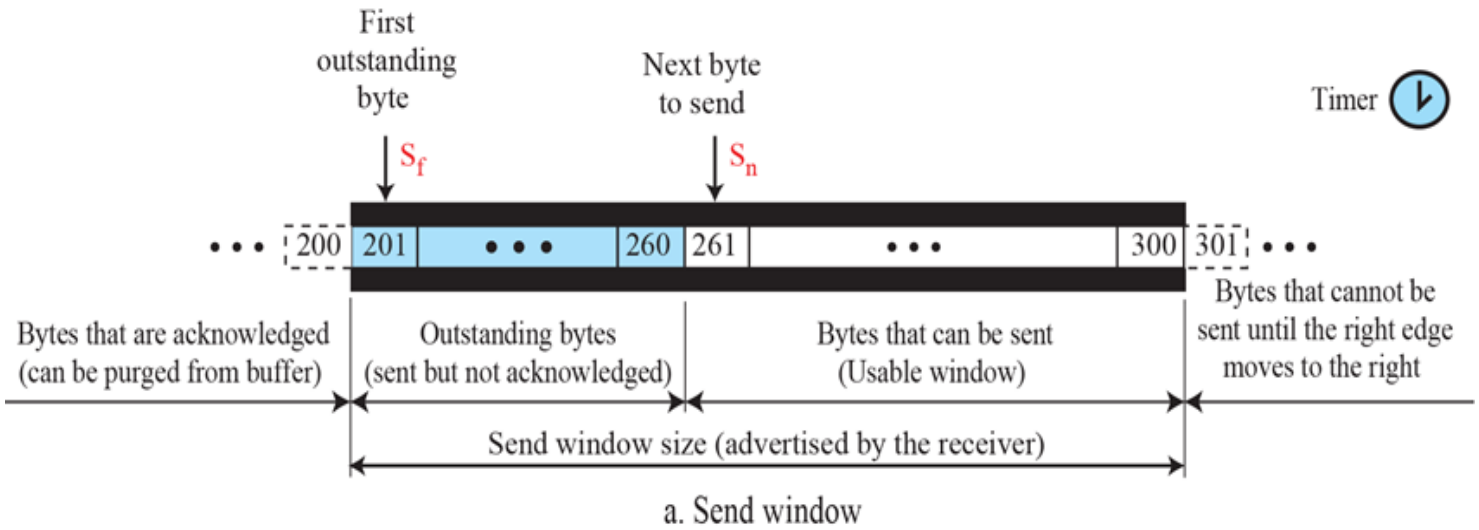
Time-line diagram



Windows in TCP

- TCP uses two windows (send window and receive window) for each direction of data transfer, which means four windows for a bidirectional communication.
- To make the discussion simple, we make an unrealistic assumption that communication is only unidirectional.
- The bidirectional communication can be inferred using two unidirectional communications with piggybacking.

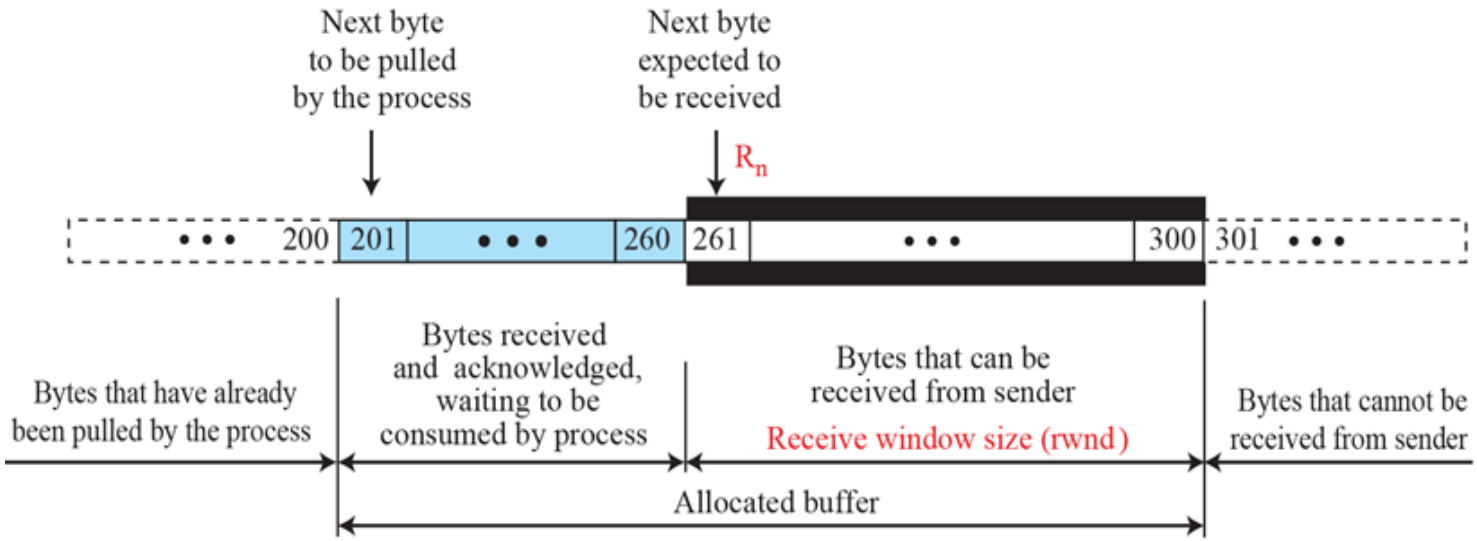
Send window in TCP



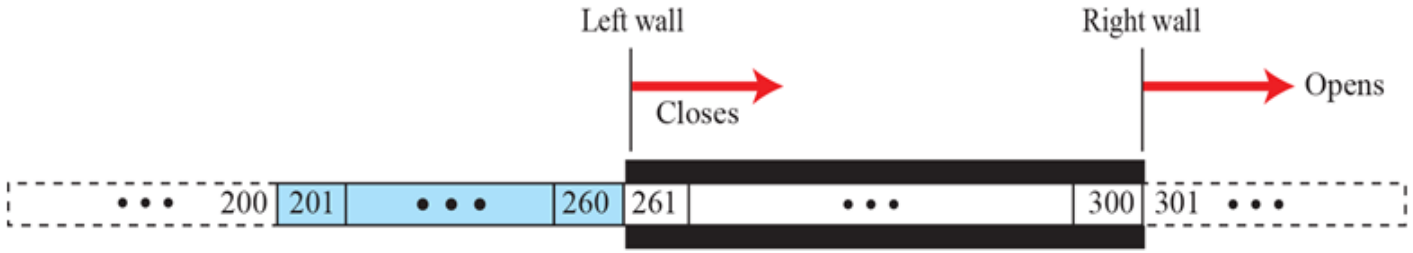
a. Send window

b. Opening, closing, and shrinking send window

Receive window in TCP



a. Receive window and allocated buffer

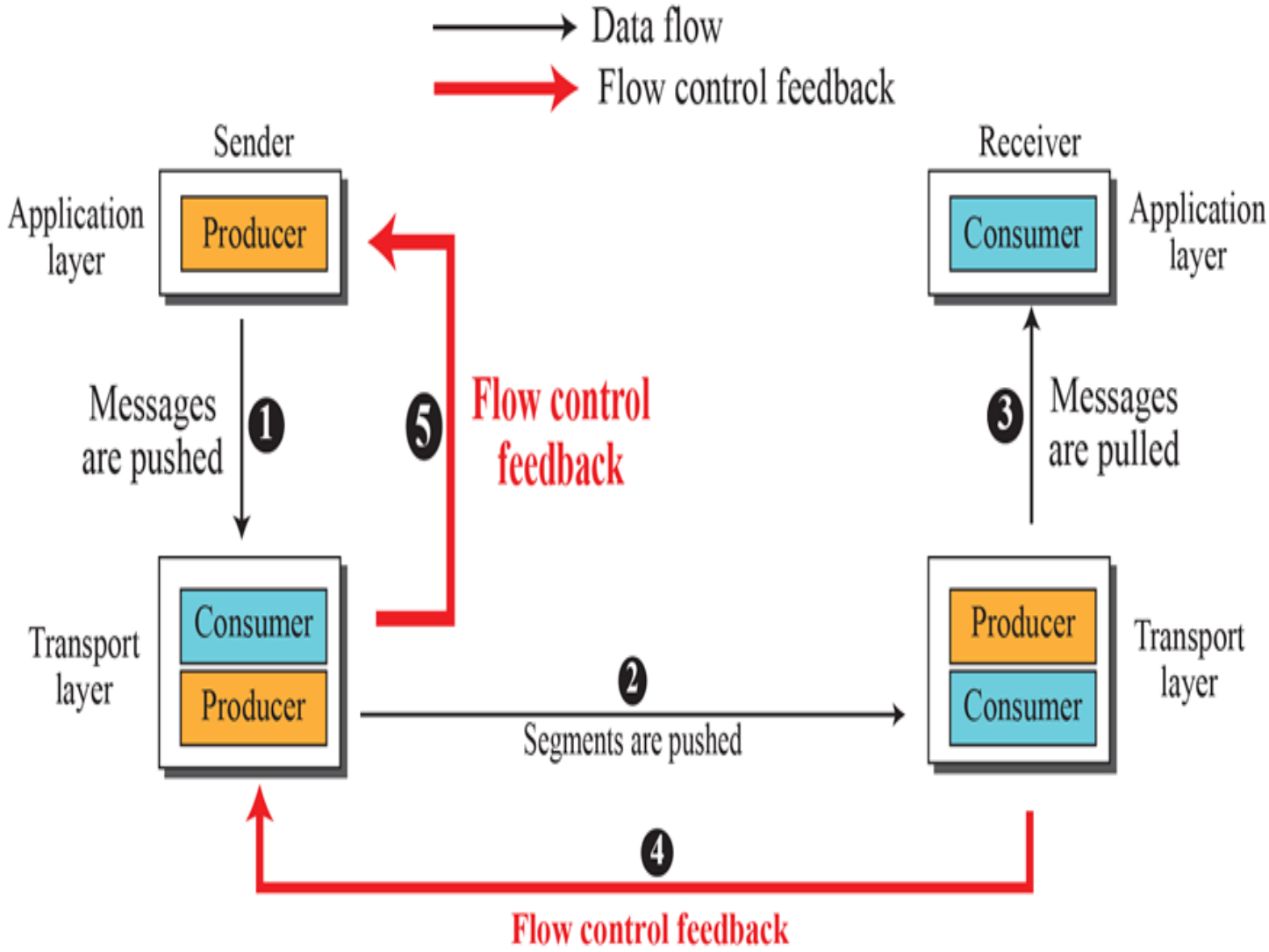


b. Opening and closing of receive window

Flow Control

- Flow control balances the rate a producer creates data with the rate a consumer can use the data.
- TCP separates flow control from error control.
- Assume that the logical channel between the sending and receiving TCP is error-free.

Flow control feedbacks in TCP



Shrinking of Windows

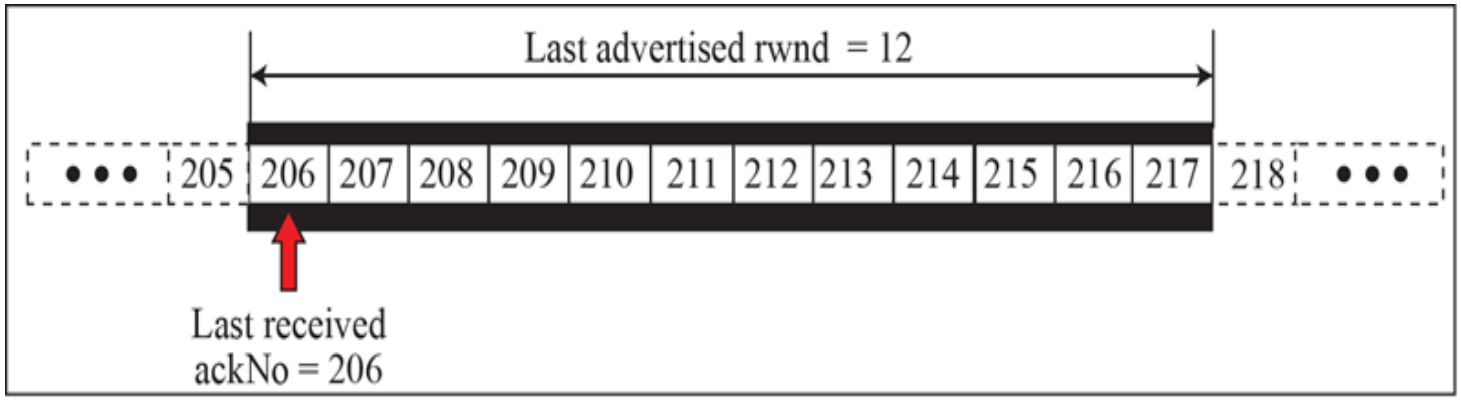
- The receiver needs to keep the following relationship between:
- the last and new acknowledgment and the last and new rwnd values to prevent shrinking of the send window.

  $$\textbf{new ackNo} + \textbf{new rwnd} \geq \textbf{last ackNo} + \textbf{last rwnd}$$

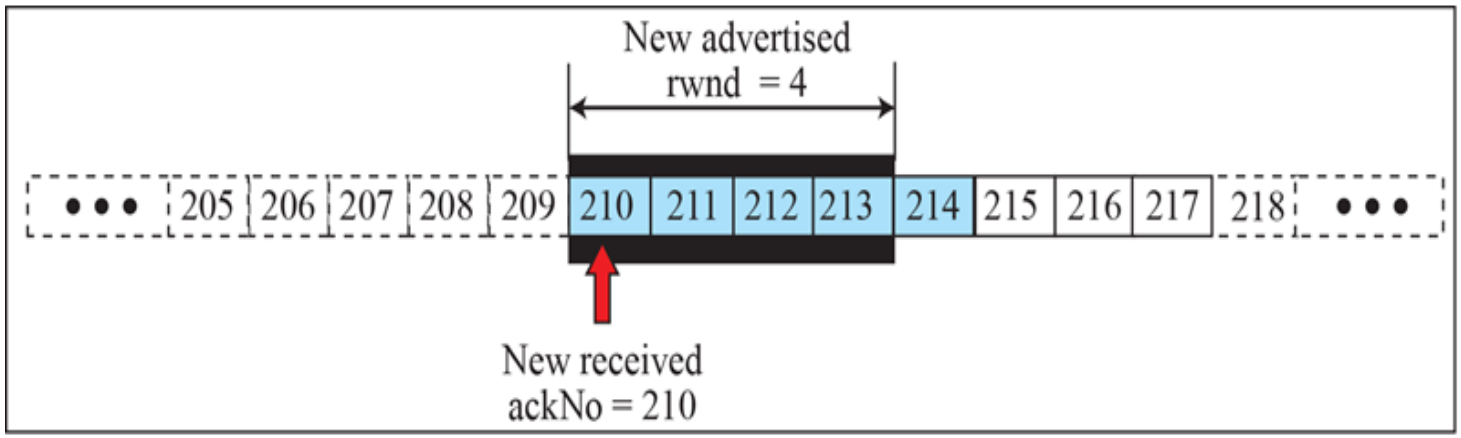
Shrinking of Windows

- Part a of the figure shows the values of the last acknowledgment and rwnd. Part b shows the situation in which the sender has sent bytes 206 to 214. Bytes 206 to 209 are acknowledged and purged. The new advertisement, however, defines the new value of rwnd as 4, in which
- $210 + 4 < 206 + 12$. When the send window shrinks, it creates a problem: byte 214, which has already been sent, is outside the window. The relation discussed before forces the receiver to maintain the right-hand wall of the window to be as shown in part a, because the receiver does not know which of the bytes 210 to 217 has already been sent. described above

Shrinking of Windows



a. The window after the last advertisement

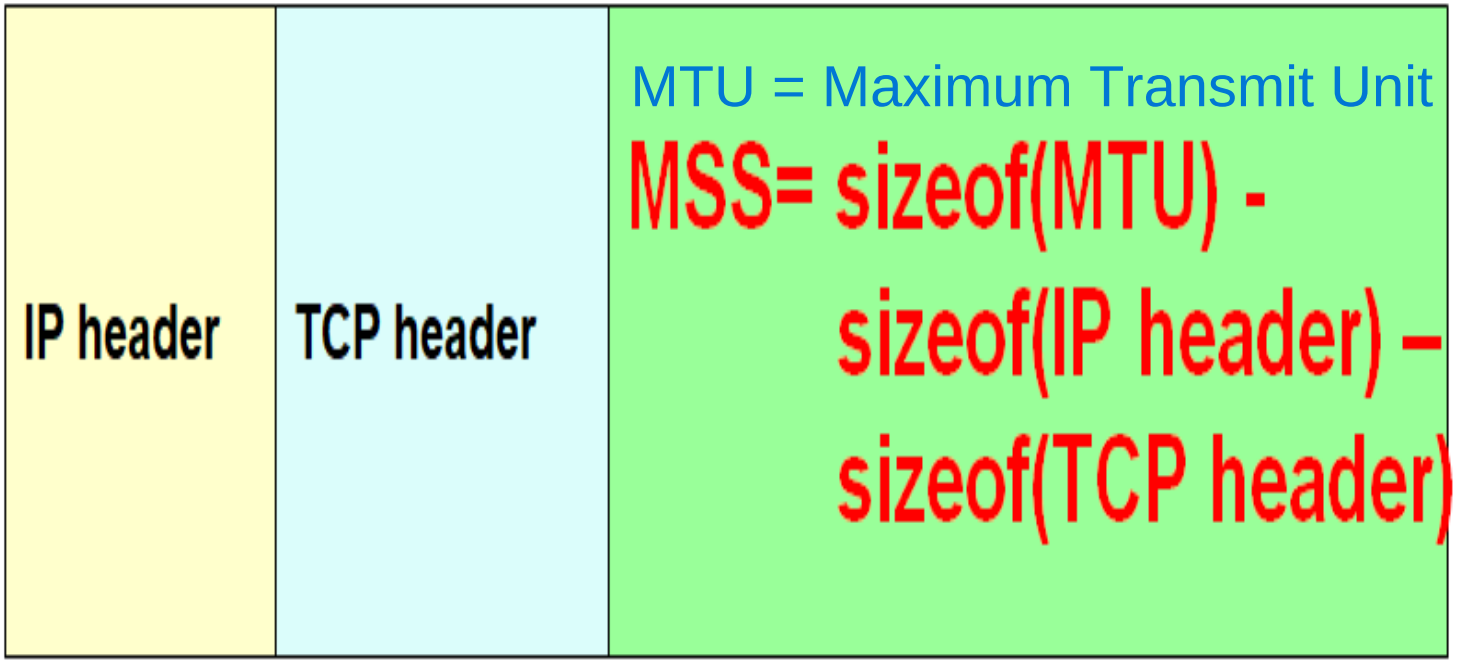


b. The window after the new advertisement; window has shrunk

Window Shutdown

- The receiver can temporarily shut down the window by sending a rwnd of 0.
- This can happen if for some reason the receiver does not want to receive any data from the sender for a while.
- In this case, the sender does not actually shrink the size of the window, but stops sending data until a new advertisement has arrived.

Maximum Segment Size



Silly Window Syndrome

- A serious problem can arise in the sliding window operation when either the sending application program creates data slowly or the receiving application program consumes data slowly, or both.
- Any of these situations results in the sending of data in very small segments, which reduces the efficiency of the operation.
- For example, if TCP sends segments containing only 1 byte of data, it means that a 41-byte datagram (20 bytes of TCP header and 20 bytes of IP header) transfers only 1 byte of user data.
- Here the overhead is 41/1, which indicates that we are using the capacity of the network very inefficiently.

Nagle's algorithm

if there is **new data to send**
  if **the window size** $\geq$ **MSS and available data is** $\geq$ **MSS**
    send **complete MSS segment** now
  Else
    if
      if **there is unACKed data** in flight
        buffer the new data until an ACK arrives
    else
      send data immediately
      end if
  end if
end if

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

- **Checksum**
- **Acknowledgment**

  ACK segments do not consume sequence numbers and are not acknowledged.

  Cumulative Acknowledgment (ACK)

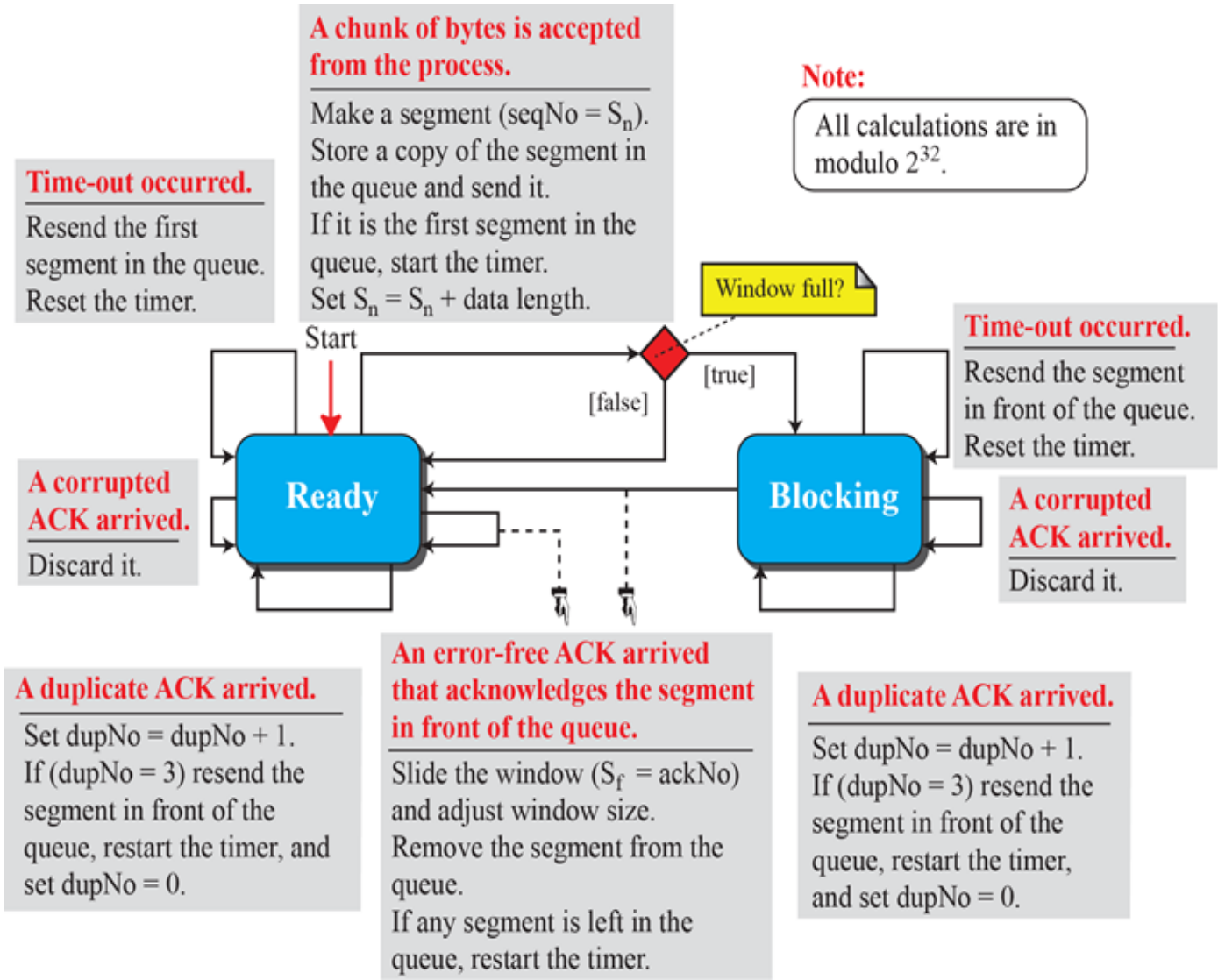
  Selective Acknowledgment (SACK)
- **Retransmission**: Retransmission after RTO

  sending TCP maintains one retransmission time-out (RTO)

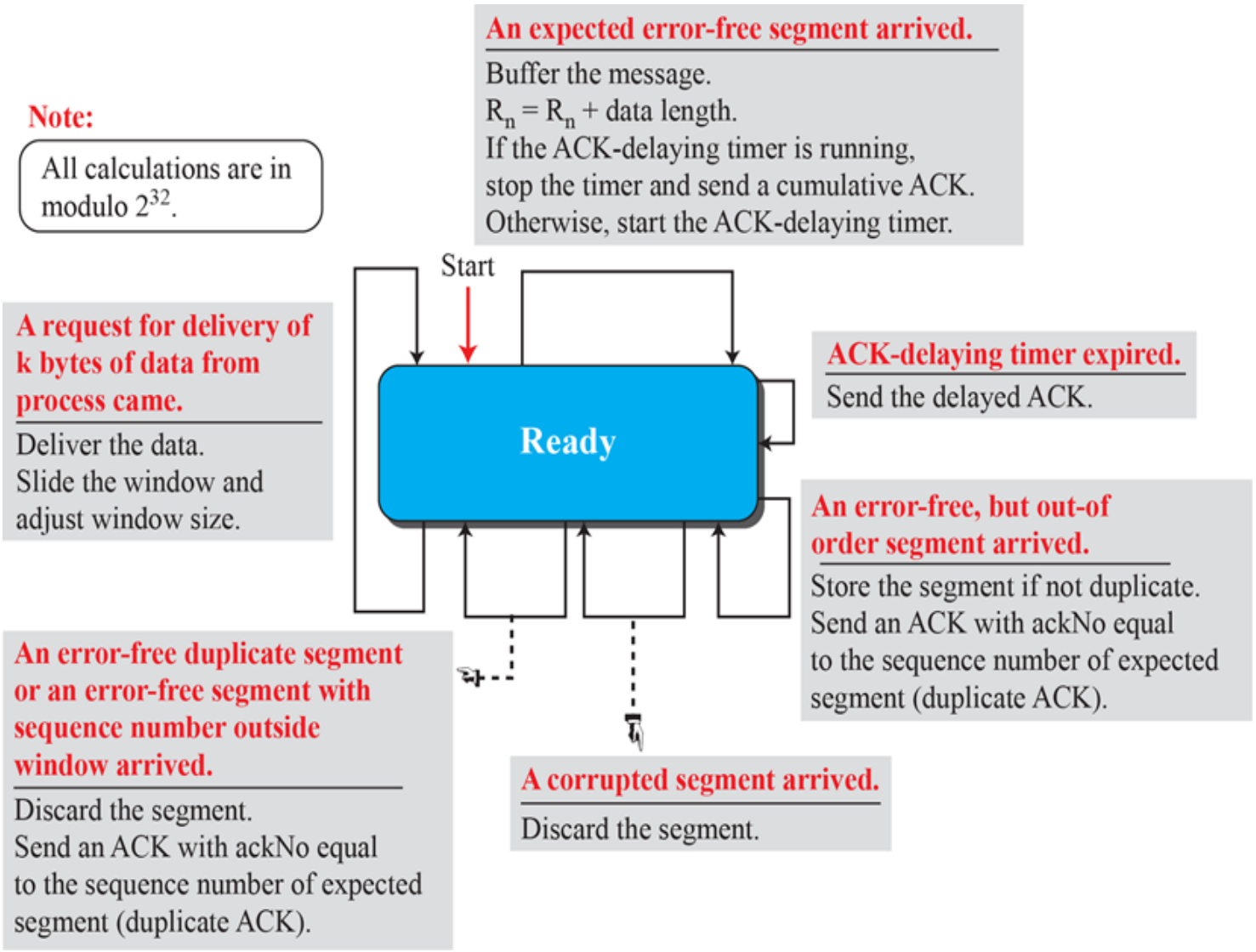
  Retransmission after Three Duplicate ACK Segments
- **Fast retransmission**: Out-of-Order Segments

  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.

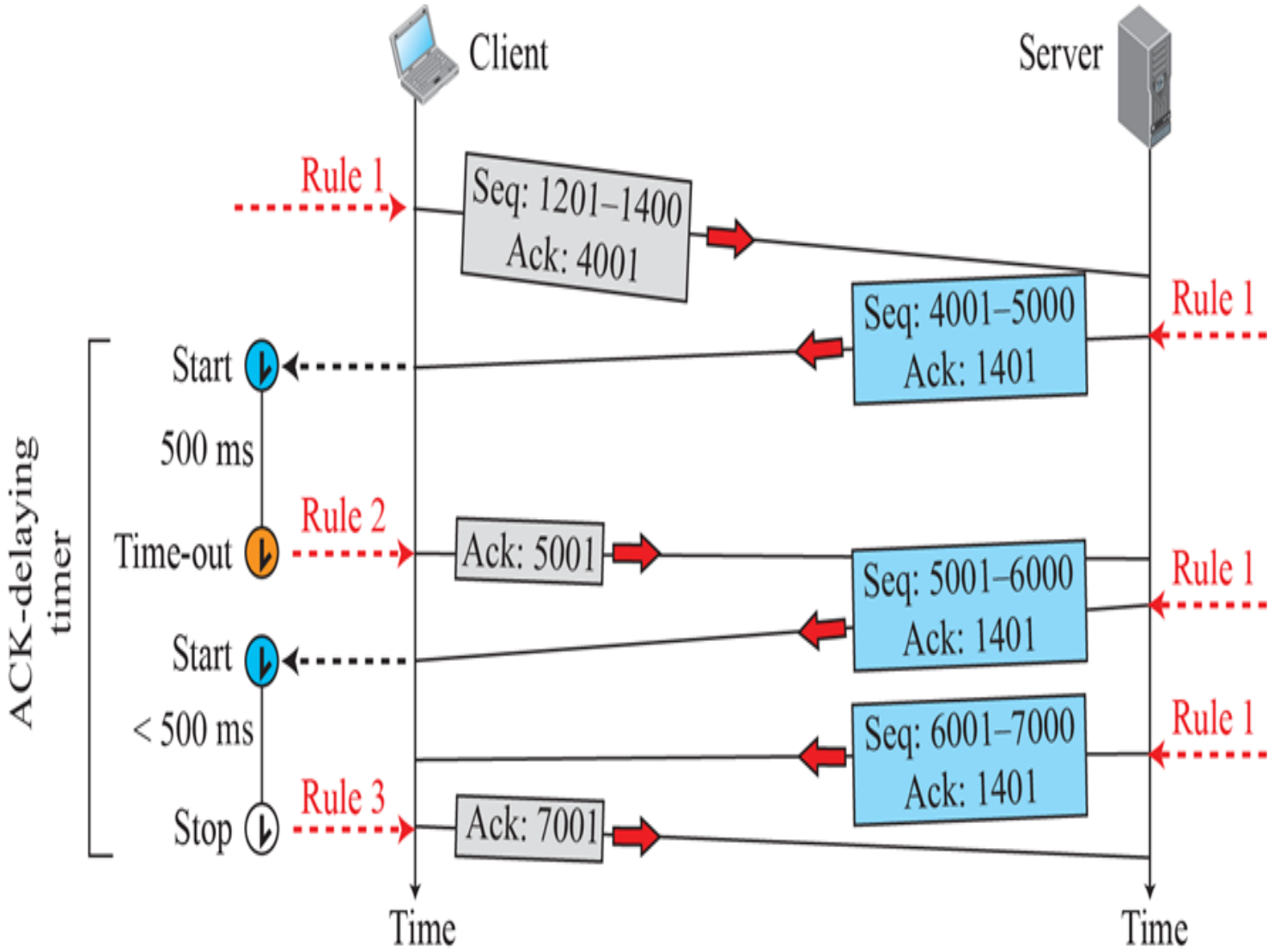
FSM for the TCP sender side



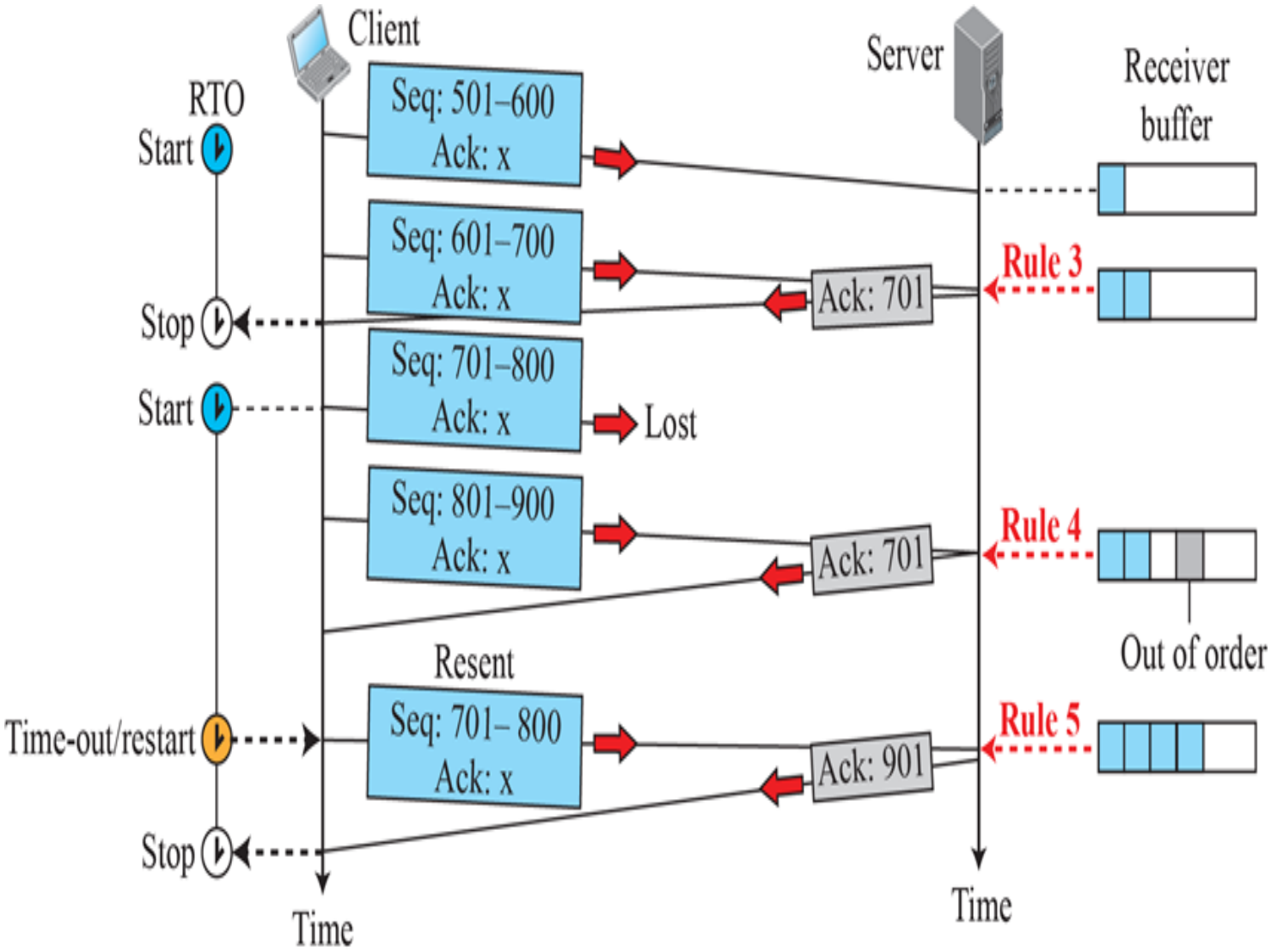
FSM for the TCP receiver side



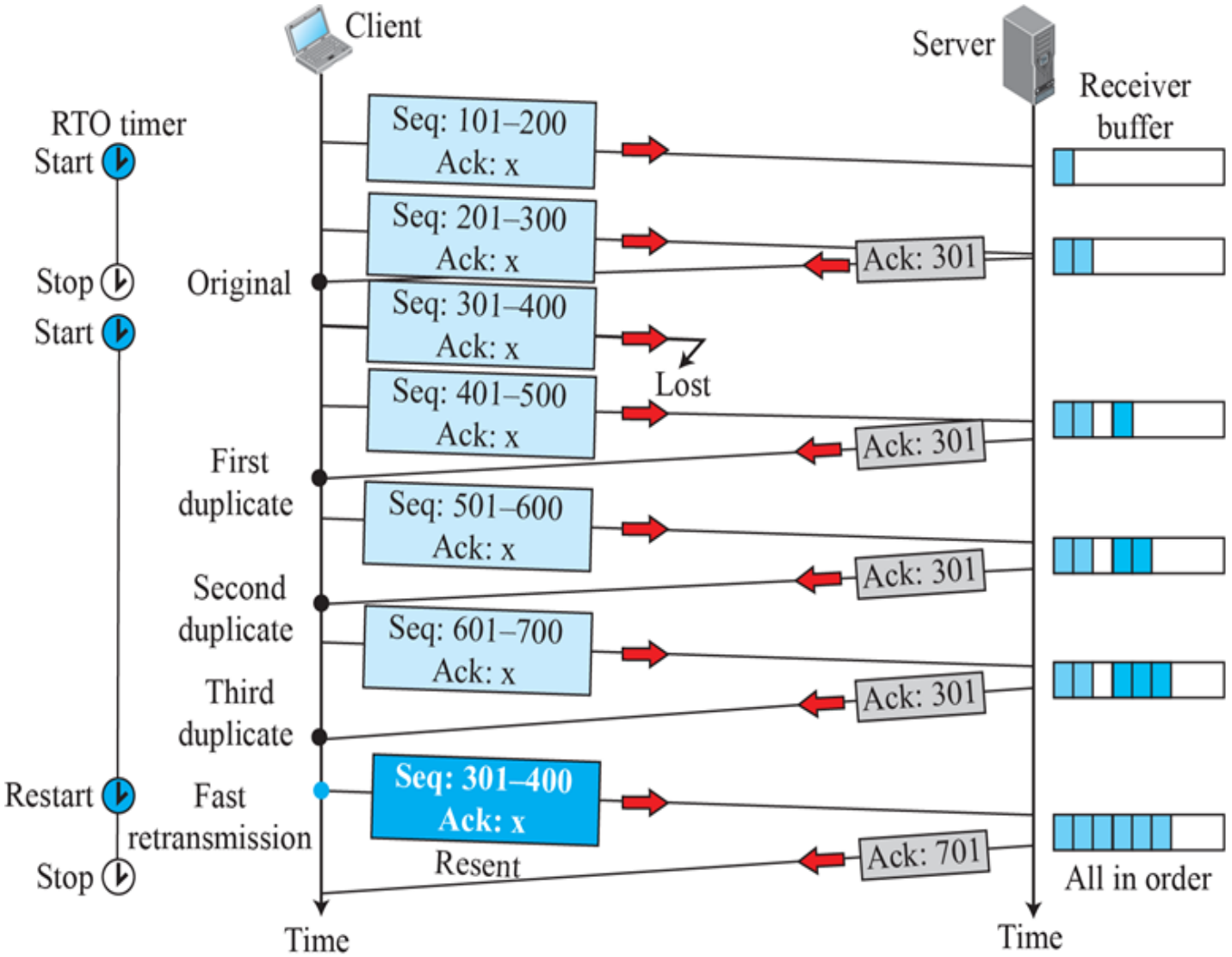
Normal operation



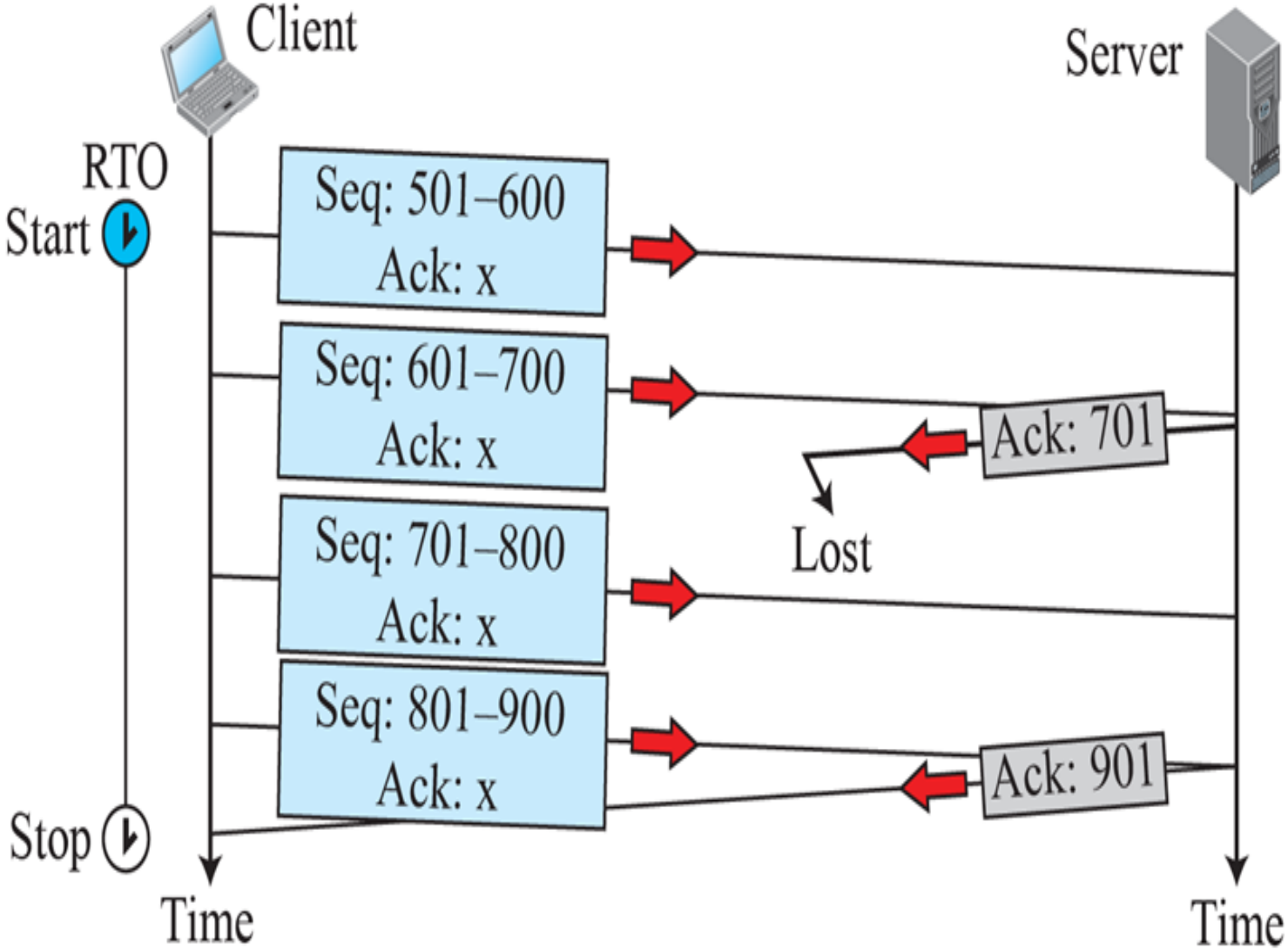
Lost segment



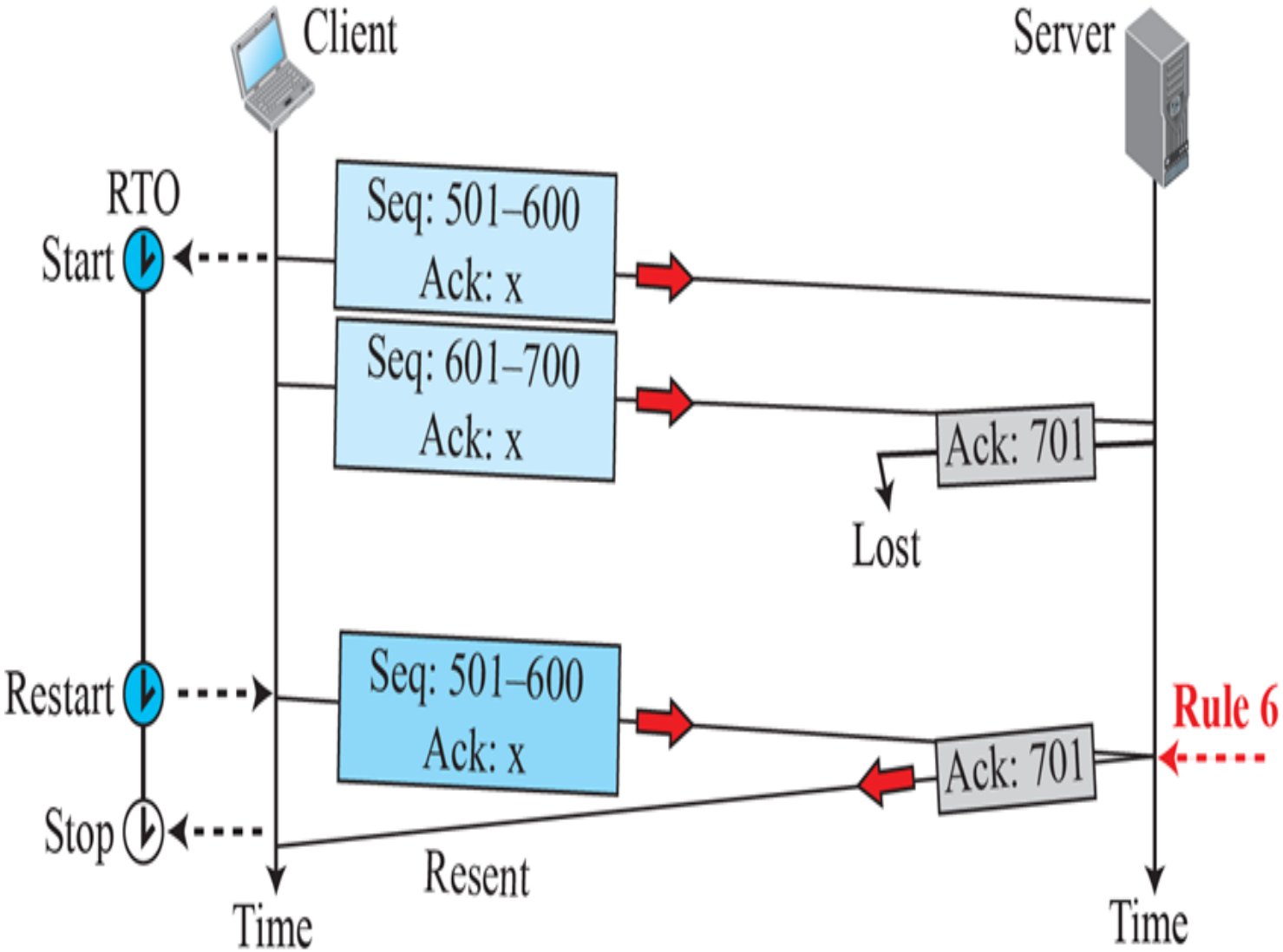
Fast retransmission



Lost acknowledgment



Lost acknowledgment corrected by resending a segment



TCP Congestion Control

TCP uses different policies to handle the congestion in the network.

- Congestion Window
- Congestion Detection
- Congestion Policies

Congestion Window

- The **size of the send window** is controlled by the receiver using the value of **rwnd**, which is advertised in each segment traveling in the opposite direction.
- The use of this strategy guarantees that the receive window is never overflowed with the received bytes (no end congestion).
- However, **does not mean** that the intermediate buffers, **buffers in the routers**, do not become congested.
- Router may be overwhelmed with data which results in dropping some segments sent by a specific TCP sender (there may be congestion in the middle).
- TCP uses **another variable called a congestion window, cwnd**, whose size is controlled by the congestion situation in the network.

Congestion Window

- 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 = minimum (rwnd, cwnd)

Congestion Detection

- 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 the segments are lost and the loss is due to congestion.

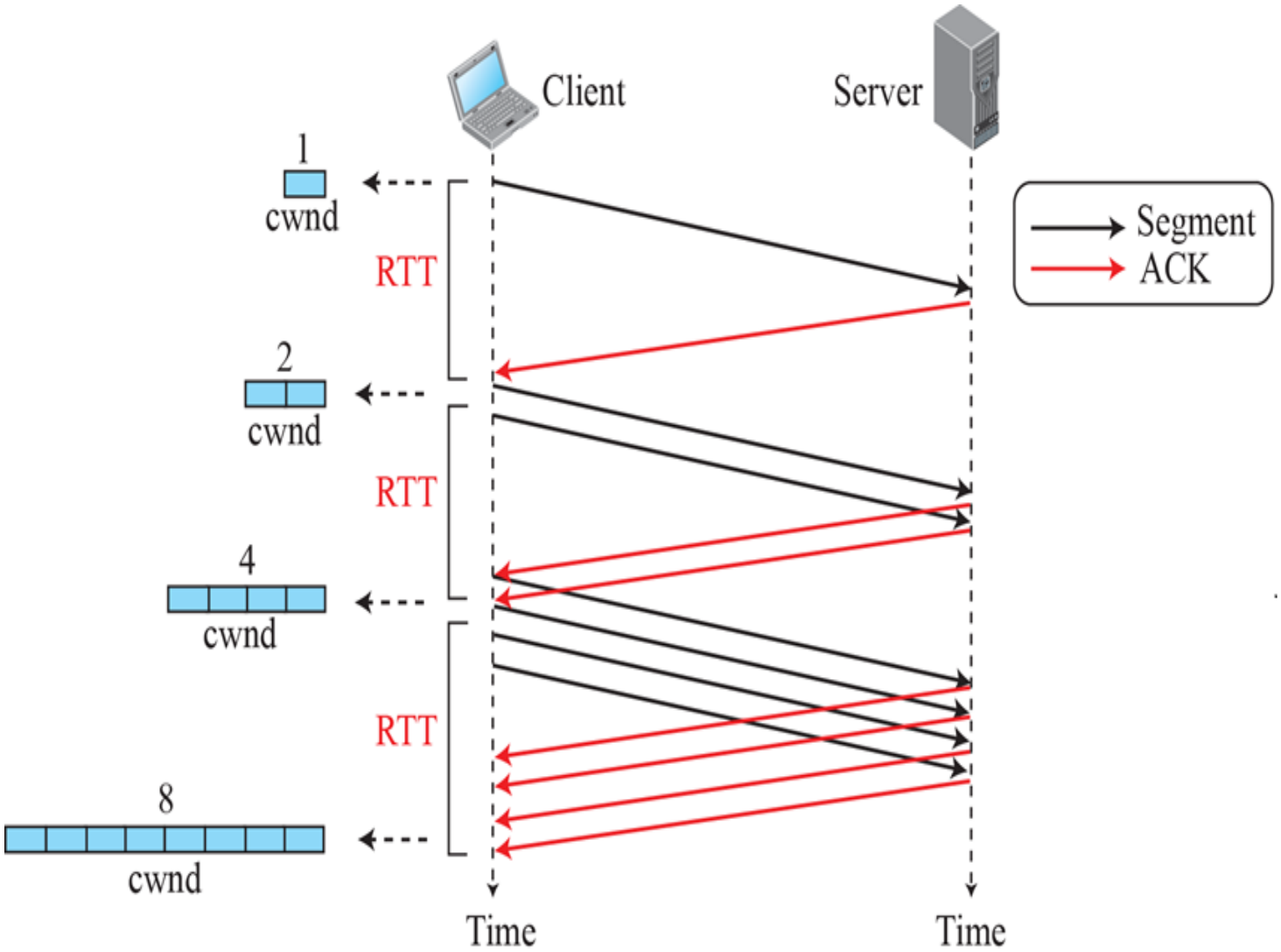
- Receiving three duplicate ACKs.

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

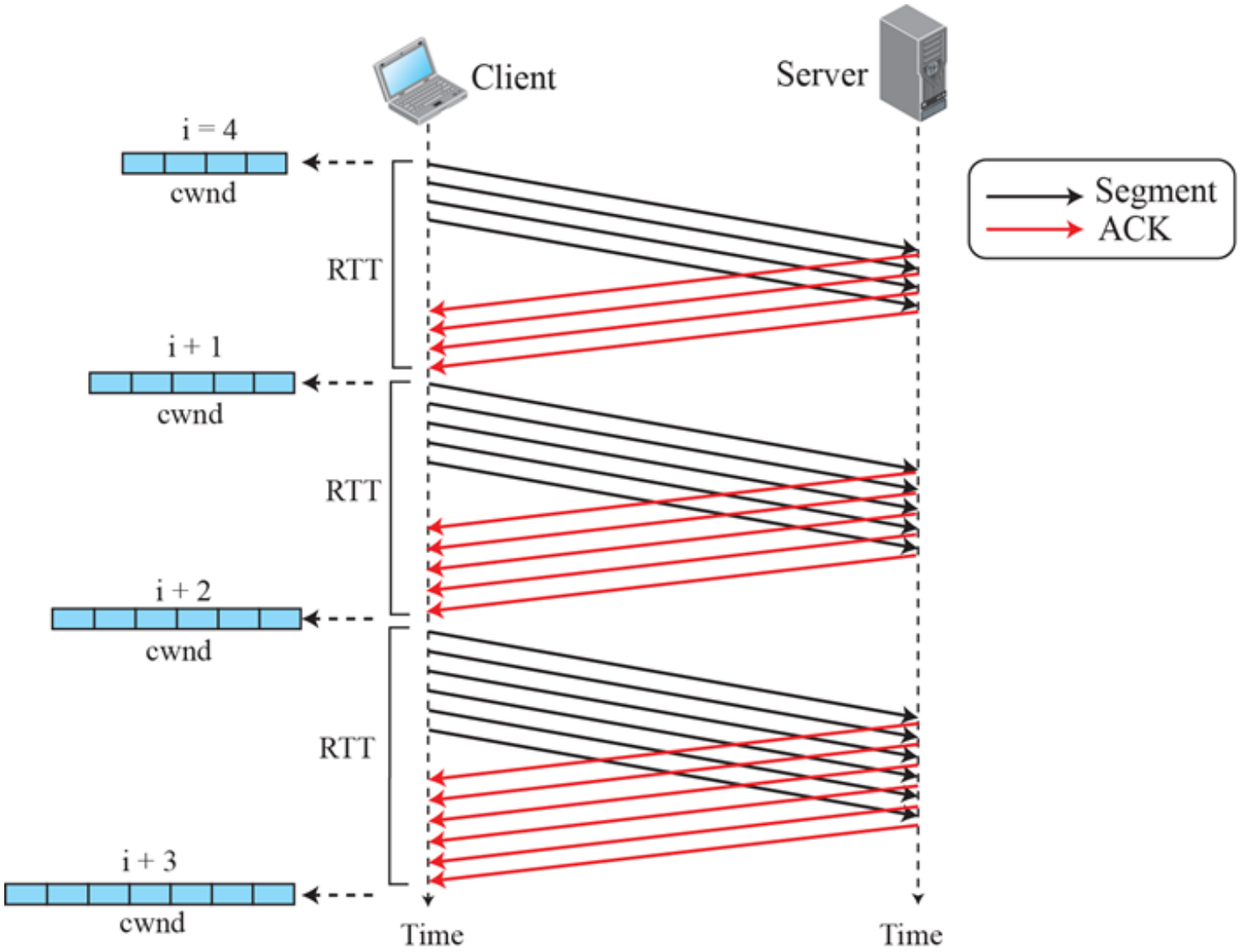
Congestion Policies

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

Slow start, exponential increase



Congestion avoidance, additive increase



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 5 cwnd 1 (1 / cwnd).**

Policy Transition

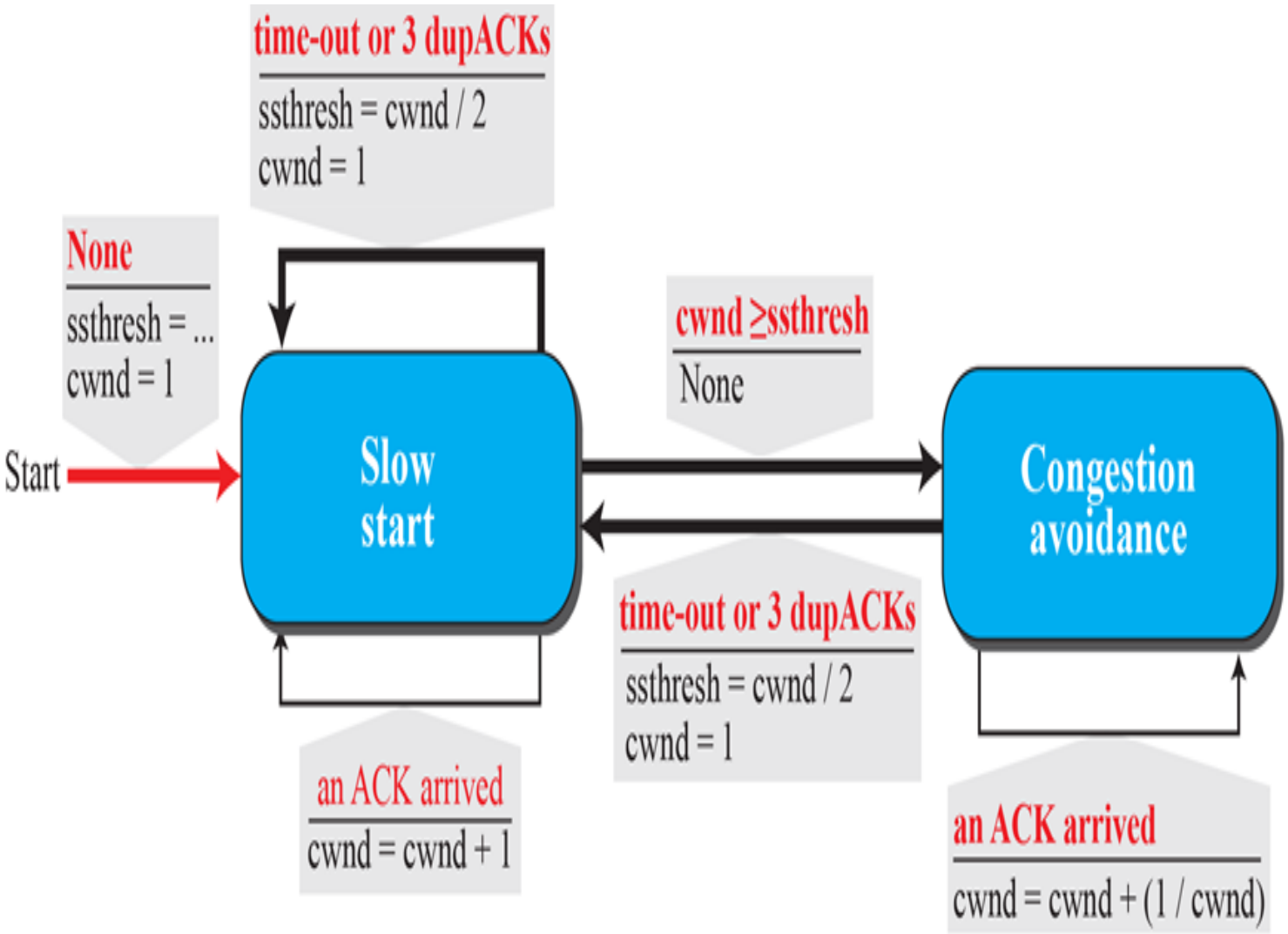
- When each of these policies is used and when TCP moves from one policy to another?
- Three versions of TCP:

  Taho TCP: used only two different algorithms in their congestion policy: slow start and congestion avoidance.

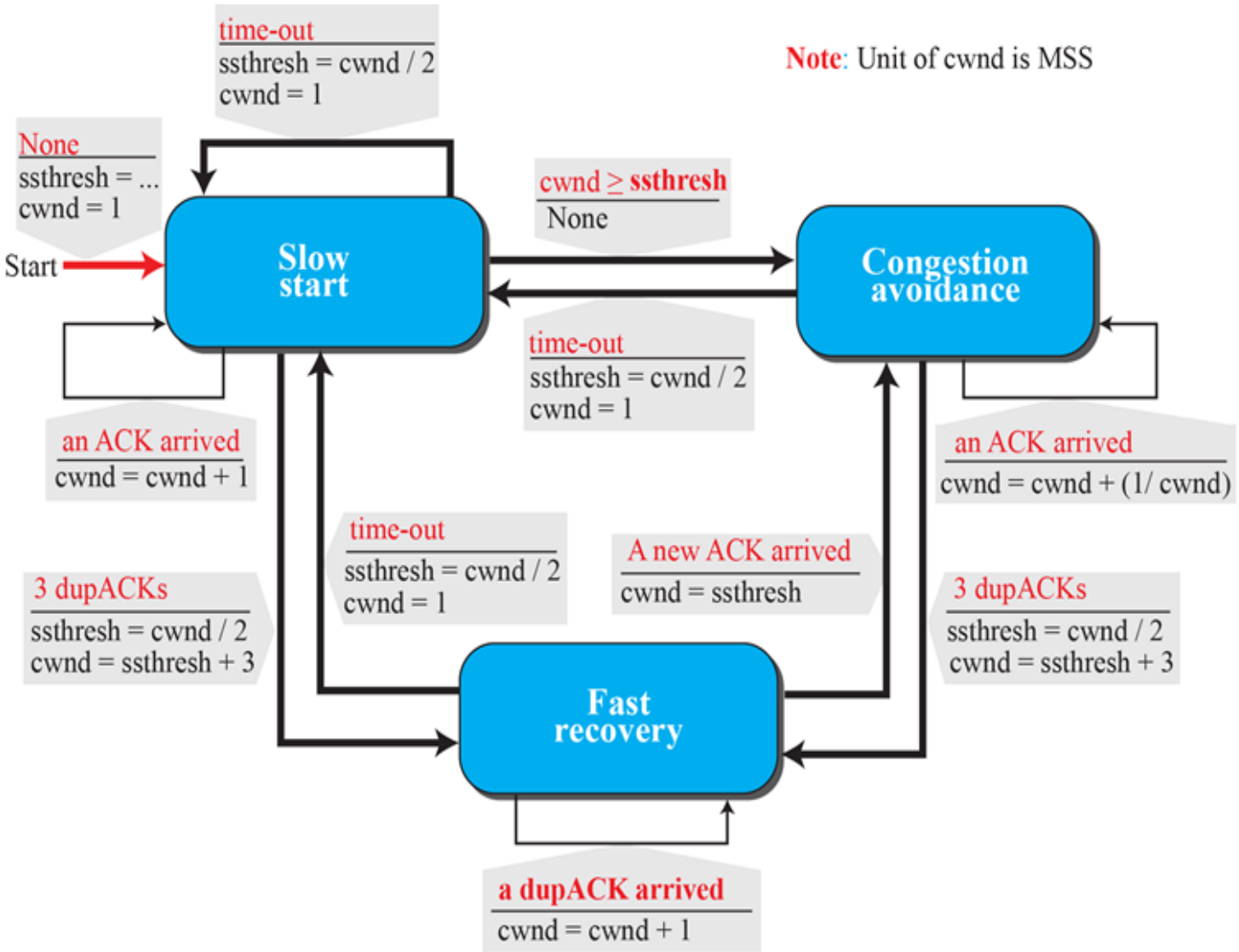
  Reno TCP, and

  New Reno TCP.

FSM for Taho TCP



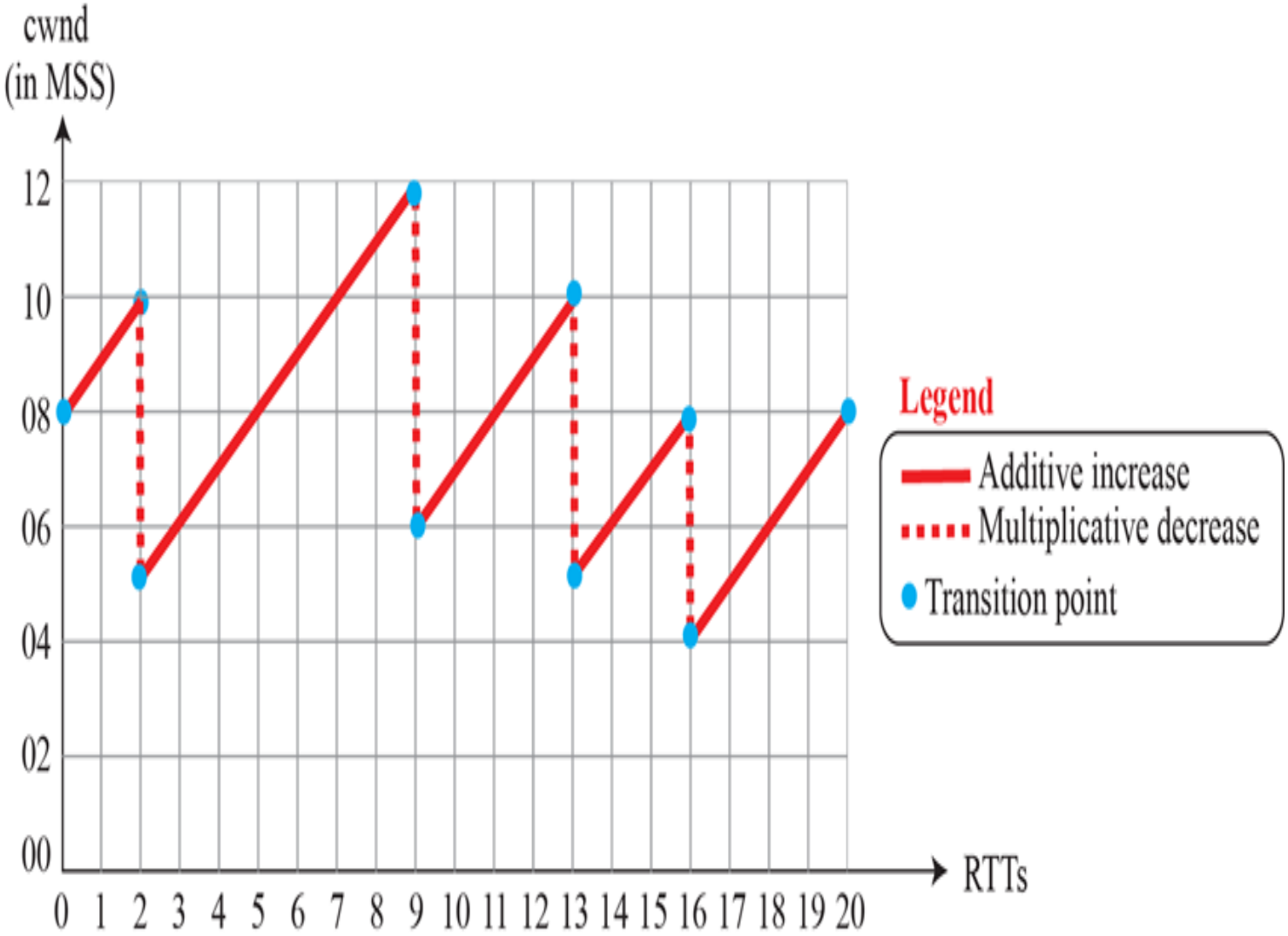
FSM for Reno TCP



New Reno TCP

- 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.

Additive increase, multiplicative decrease (AIMD)



TCP Timers

- To perform their operations smoothly, most TCP implementations use at least four timers:
- Retransmission,
- Persistence,
- Keepalive, and
- TIME-WAIT.

TCP Timers

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

Initially	→	No value
After first measurement	→	$RTT_D = RTT_M / 2$
After each measurement	→	$RTT_D = (1 - \beta)\, RTT_D + \beta \times \lvert RTT_S - RTT_M \rvert$

Original	→	Initial value
After any measurement	→	$RTO = RTT_S + 4 \times RTT_D$

Summary

- TCP Services
- Connection establishment, data transfer, and connection termination.
- TCP window size is determined by the receiver-advertised window size (rwnd) or the congestion window size (cwnd), whichever is smaller.
- Error control is handled by checksums, acknowledgment, and time-outs.
- The slow-start (exponential increase), congestion-avoidance (additive increase), and congestion-detection (multiplicative decrease) strategies are used for congestion control.

Test Your Understanding

- Can you explain why we need four (or sometimes three) segments for connection termination in TCP?
- In TCP, how many sequence numbers are consumed by each of the following segments?
  a. SYN b. ACK c. SYN + ACK d. Data
- In TCP, does a FIN segment close a connection in only one direction or in both directions?
- In TCP, can the sender window be smaller, larger, or the same size as the receiver window?