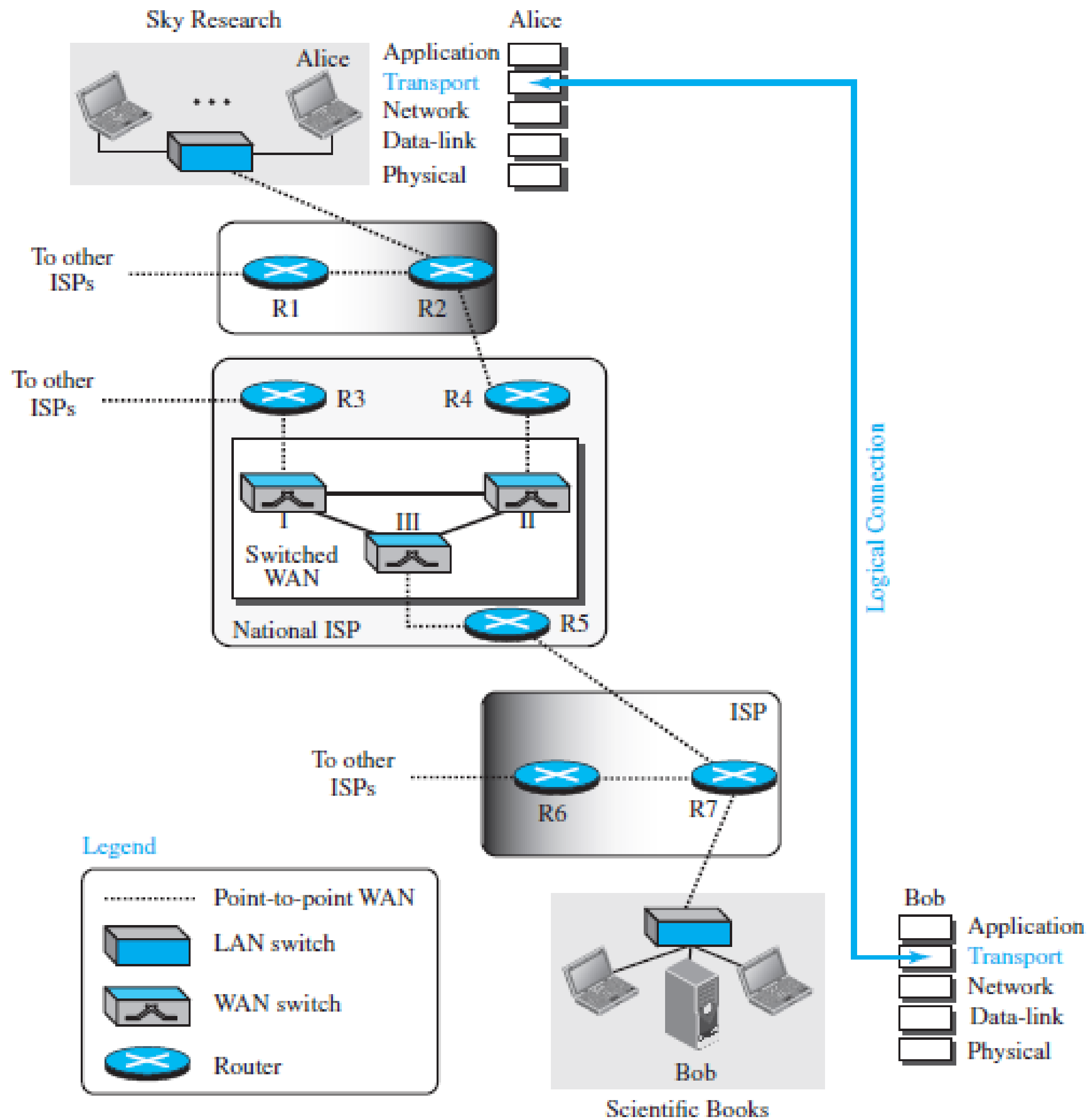
Transport Layer

Dr. Subrat K. Dash

Transport Layer – The Link

- Application layer
  - Communication for specific applications
  - E.g., HyperText Transfer Protocol (HTTP), File Transfer Protocol (FTP), Network News Transfer Protocol (NNTP)
- Transport layer
  - Communication between processes (e.g., socket)
  - Relies on network layer and serves the application layer
  - E.g., TCP and UDP
- Network layer
  - Logical communication between nodes
  - Hides details of the link technology
  - E.g., IP

Transport Layer

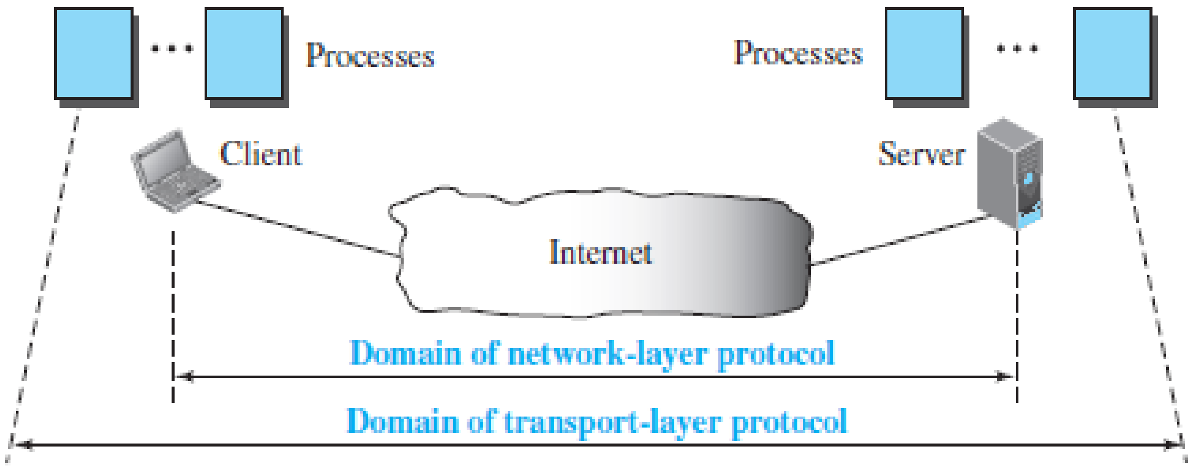


Transport Protocols

- Run on end hosts
  - Sender: breaks application messages into segments,
    and passes to network layer
  - Receiver: reassembles segments into messages, passes to application layer
- Multiple transport protocol available to applications
  - Internet: TCP and UDP

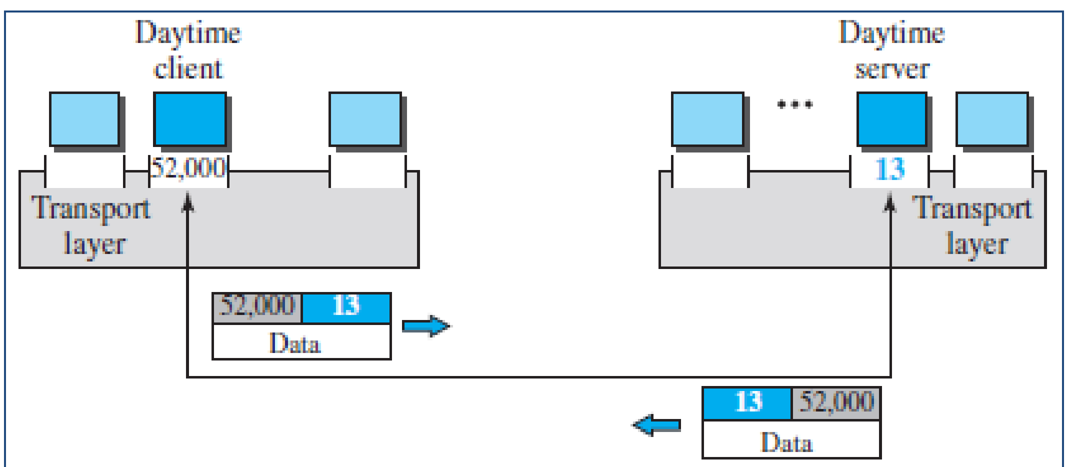
Transport Layer Services

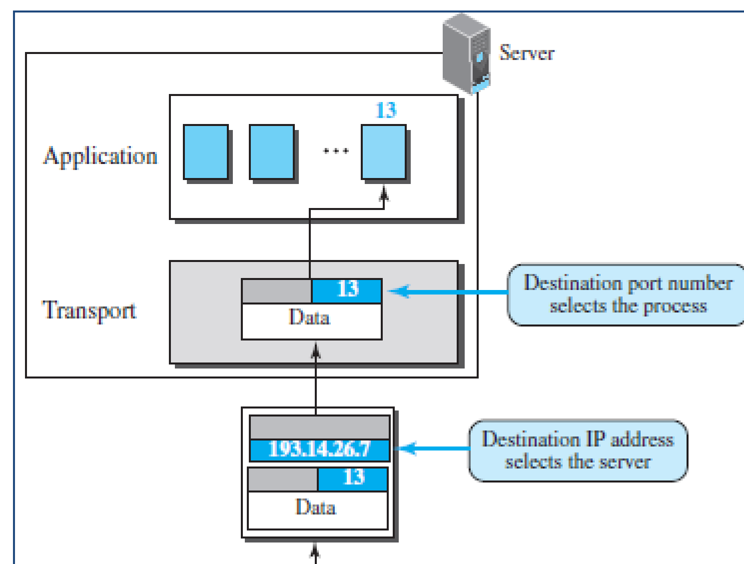
- Process-to-Process Communication



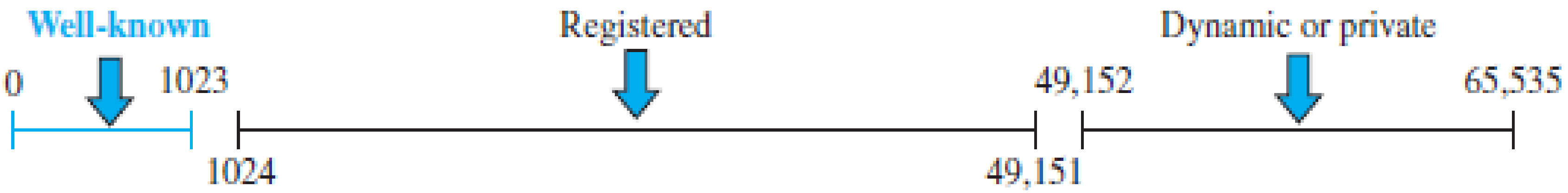
Transport Layer Services

- Addressing: Port Numbers (16 bits)
  - Client-server paradigm
  - Port numbers
    - Client: Ephemeral port (>1023)
    - Server: Well-known port





ICANN Ranges

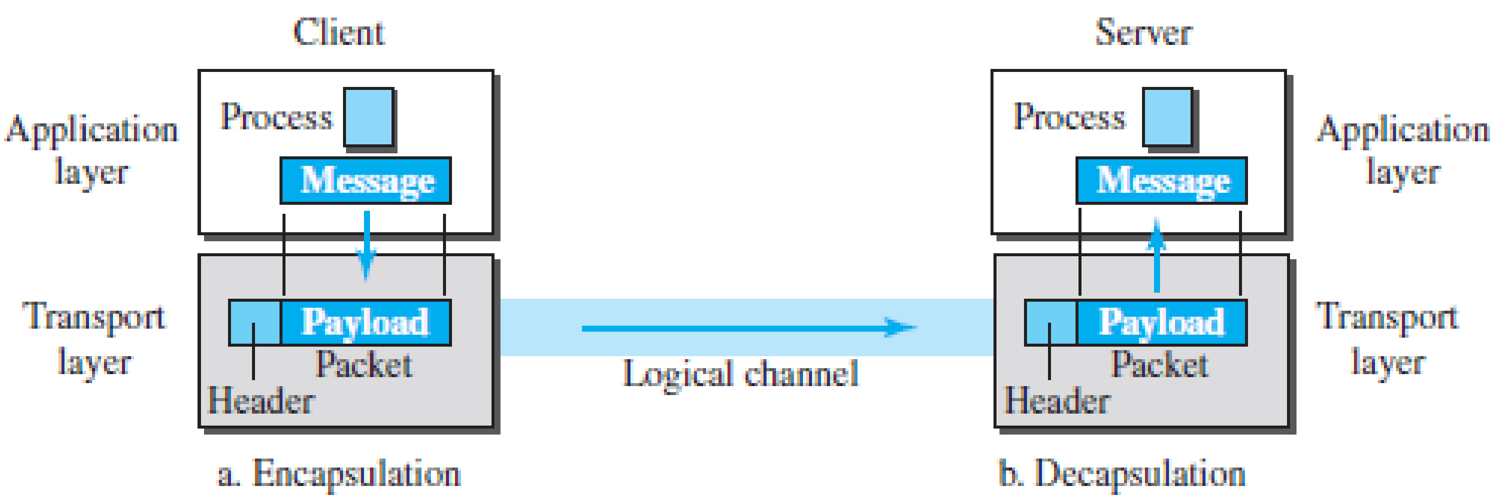



```
$grep     tftp/etc/services
tftp 69/tcp
tftp 69/udp
```

```
$grep     snmp/etc/services
snmp161/tcp#Simple Net Mgmt Proto
snmp161/udp#Simple Net Mgmt Proto
snmptrap162/udp#Traps for SNMP
```

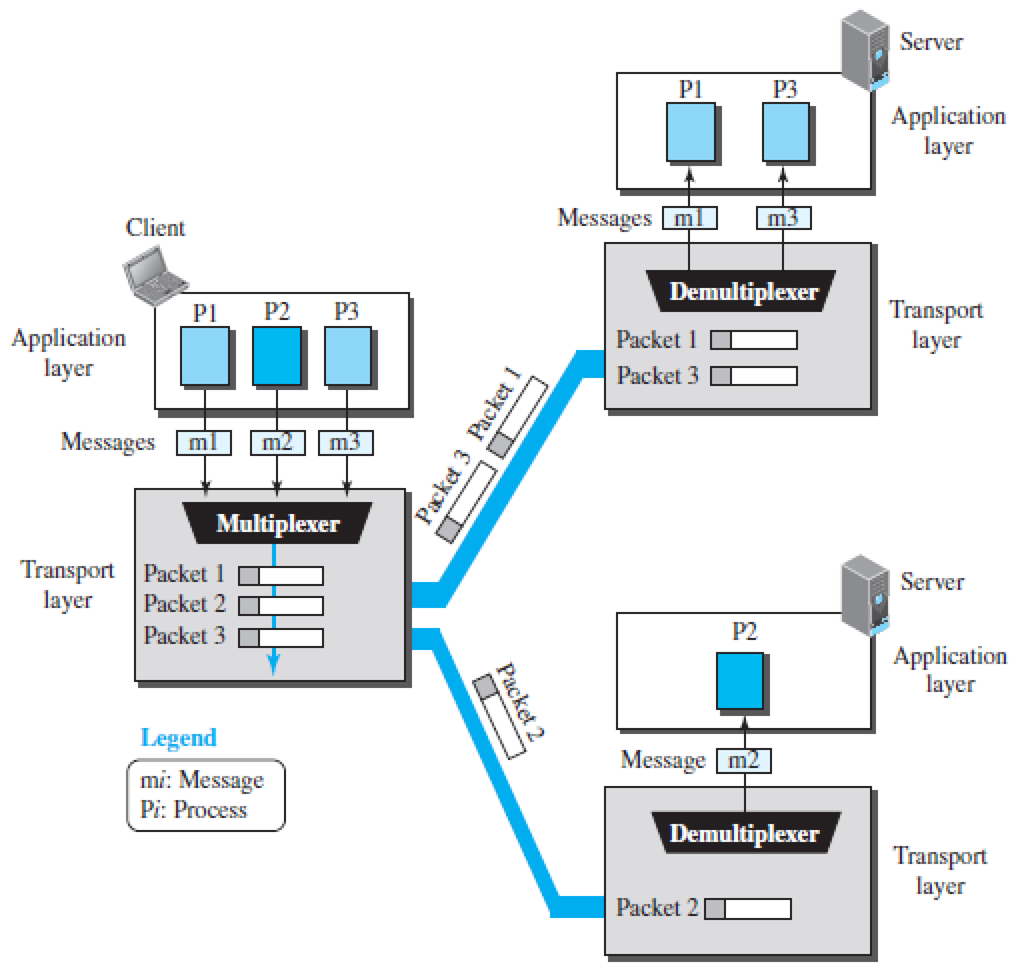
Transport Layer Services

- Encapsulation and Decapsulation
  - User datagrams, Segments, Packets



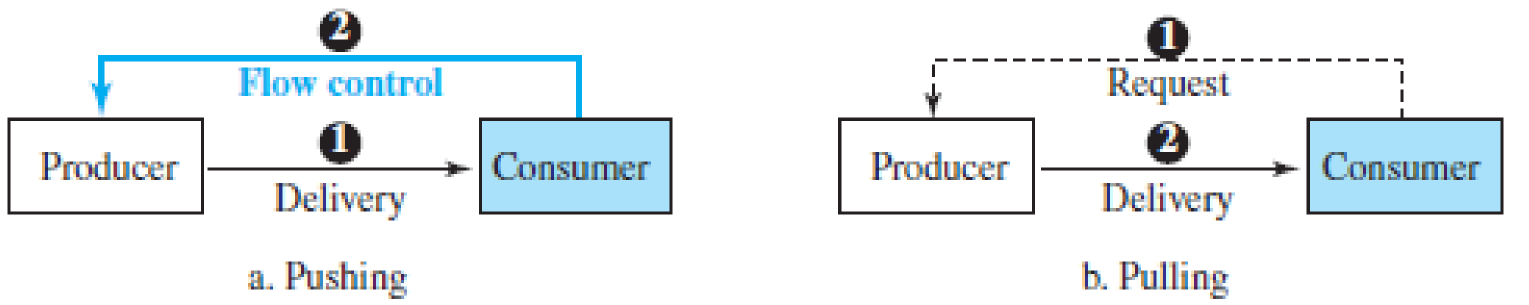
Transport Layer Services

- Multiplexing and Demultiplexing



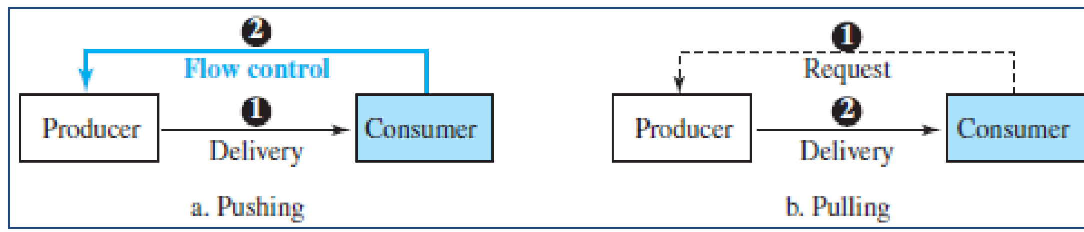
Transport Layer Services

- Flow Control
  - Items are produced faster than they can be consumed.

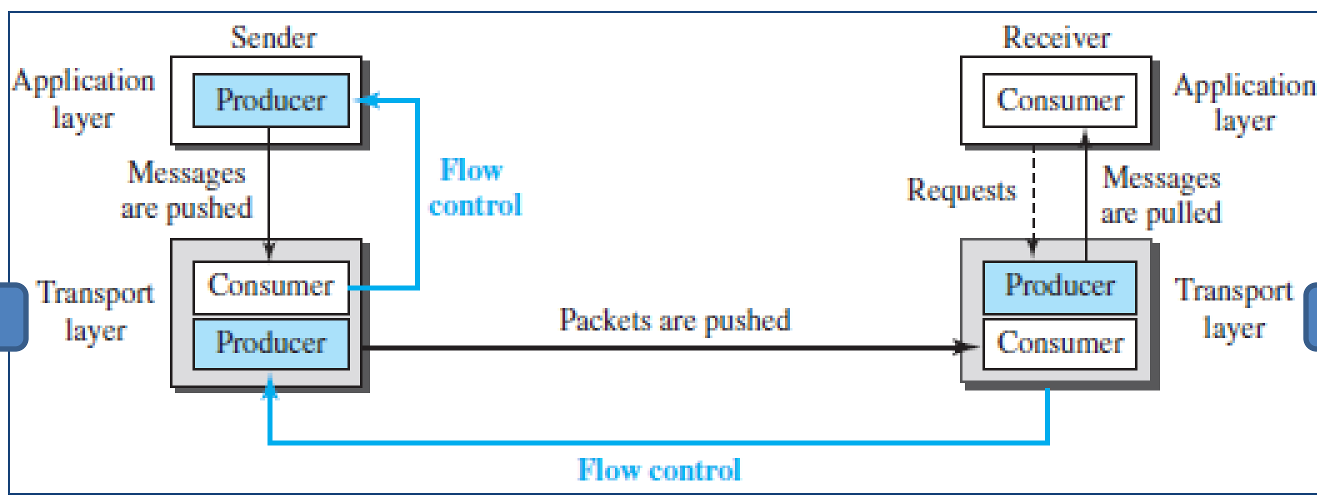


Transport Layer Services

- Flow Control
  - Items are produced faster than they can be consumed.



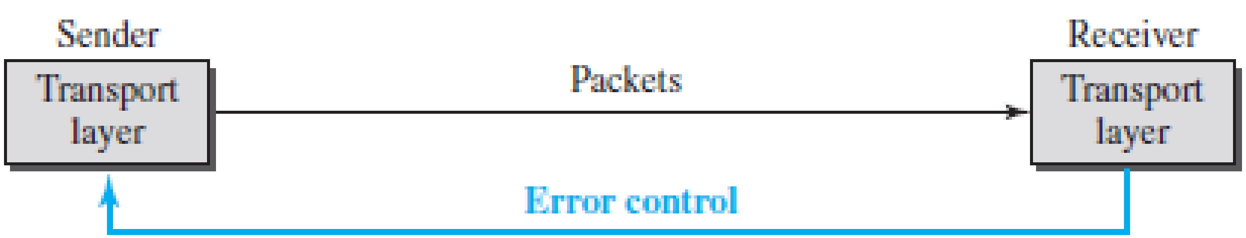
Buffers



Buffers

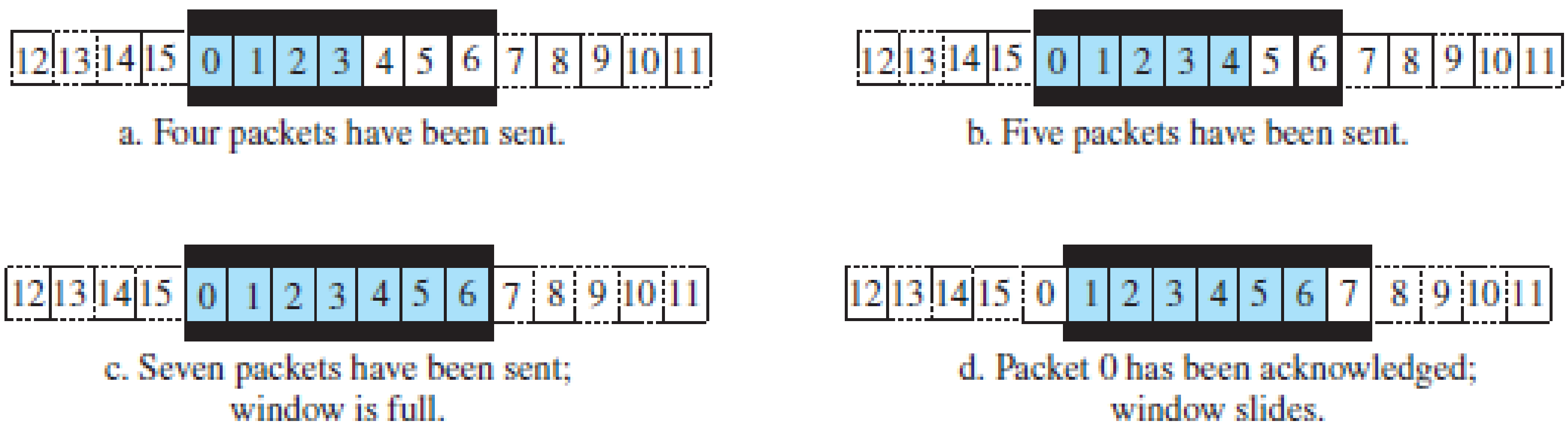
Transport Layer Services

- Error Control
  - Detecting and discarding corrupted packets.
  - Keeping track of lost and discarded packets and resending them.
  - Recognizing duplicate packets and discarding them.
  - Buffering out-of-order packets until the missing packets arrive.



Transport Layer Services

- Error Control
  - Sequence Number
  - Acknowledgement



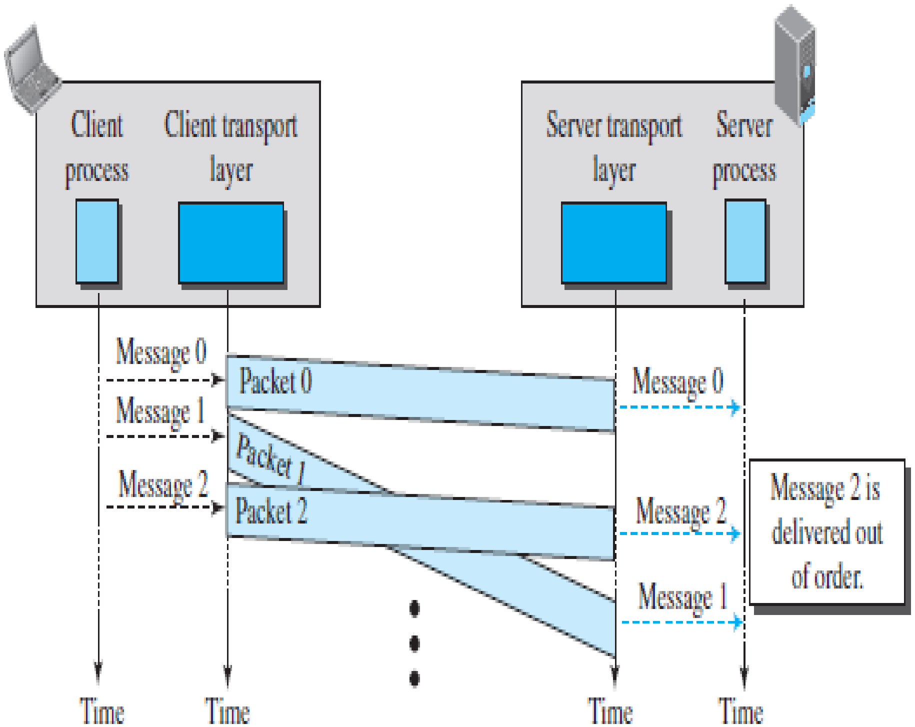
Transport Layer Services

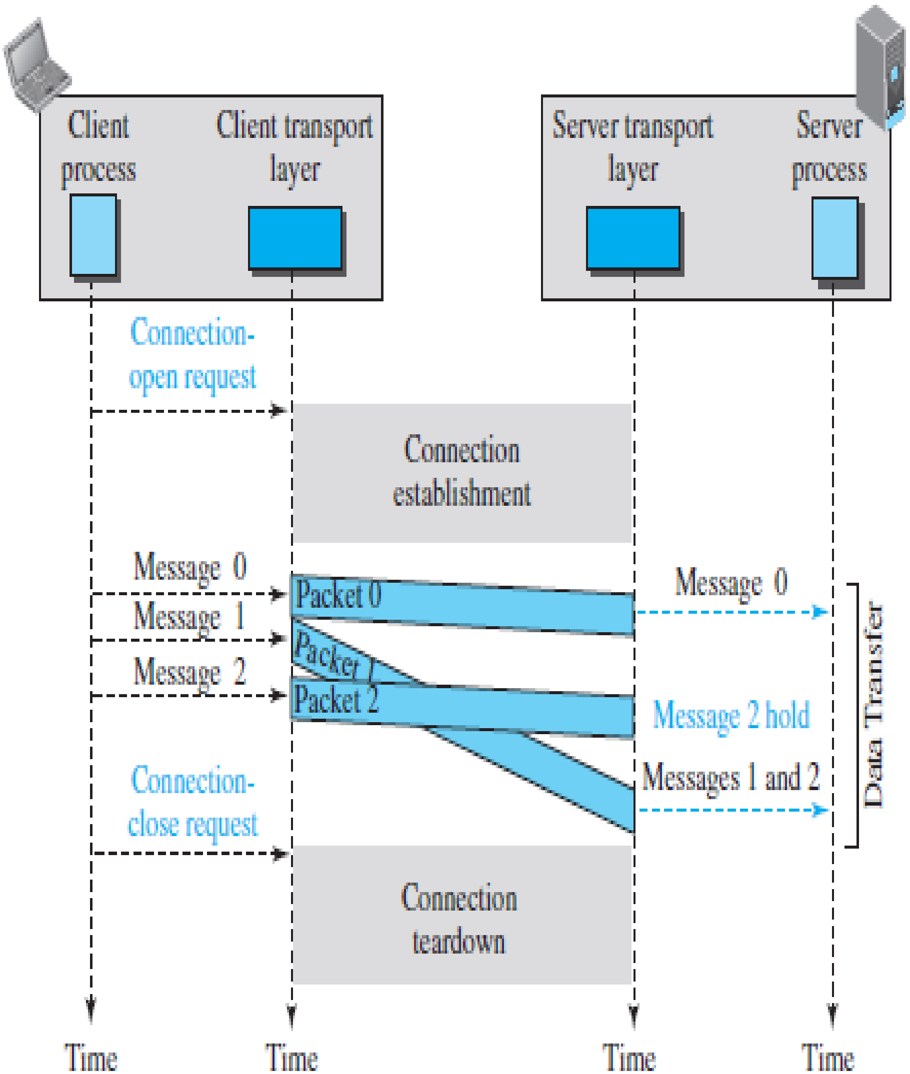
- Congestion Control
  - Congestion
    - Results due to imbalance between load of network and capacity of network
    - Waiting
  - TCP

Protocol Types

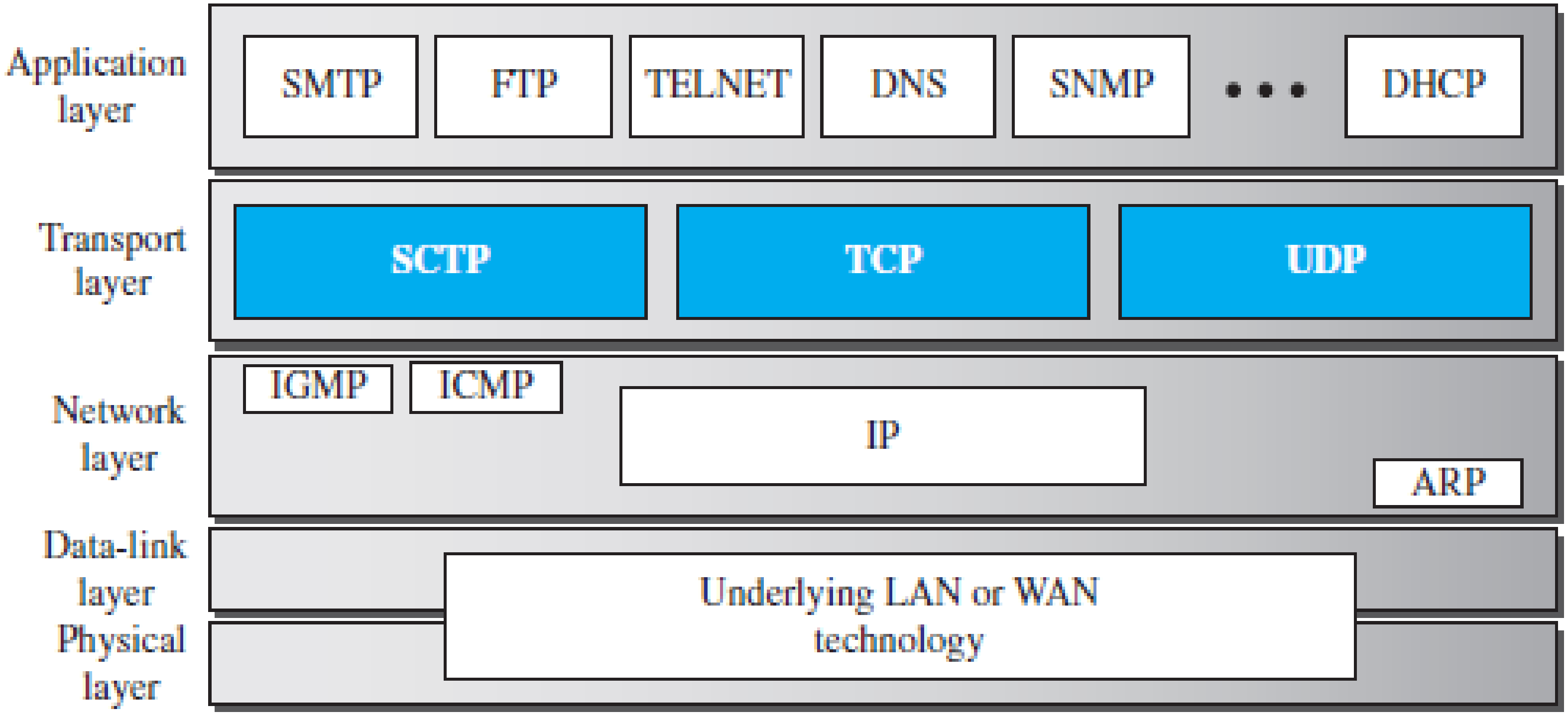
- Connectionless
  - Independency between packets (TL)
  - Different paths for different datagrams belonging to the same message (NL)
- Connection-oriented
  - Dependency between packets (TL)
  - Same path for different datagrams belonging to the same message (NL)

Connectionless vs. Connection-oriented





Transport-Layer Protocols in TCP/IP Protocol Suite



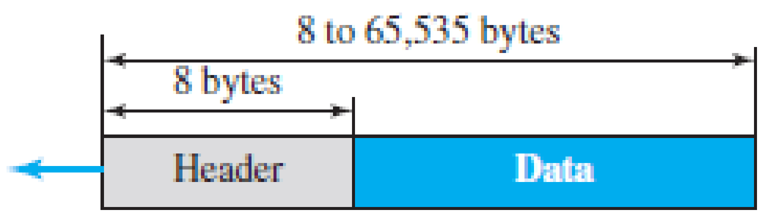
Sample list of well-known ports

Port	Protocol	UDP	TCP	SCTP	Description
7	Echo	√	√	√	Echoes back a received datagram
9	Discard	√	√	√	Discards any datagram that is received
11	Users	√	√	√	Active users
13	Daytime	√	√	√	Returns the date and the time
17	Quote	√	√	√	Returns a quote of the day
19	Chargen	√	√	√	Returns a string of characters
20	FTP-data		√	√	File Transfer Protocol
21	FTP-21		√	√	File Transfer Protocol
23	TELNET		√	√	Terminal Network
25	SMTP		√	√	Simple Mail Transfer Protocol
53	DNS	√	√	√	Domain Name Service
67	DHCP	√	√	√	Dynamic Host Configuration Protocol
69	TFTP	√	√	√	Trivial File Transfer Protocol
80	HTTP		√	√	HyperText Transfer Protocol
111	RPC	√	√	√	Remote Procedure Call
123	NTP	√	√	√	Network Time Protocol
161	SNMP-server	√			Simple Network Management Protocol
162	SNMP-client	√			Simple Network Management Protocol

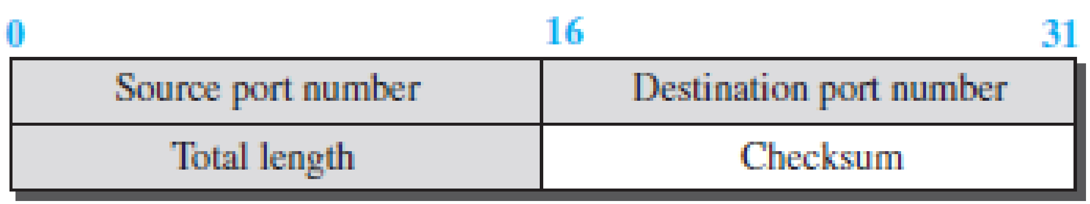
User Datagram Protocol (UDP)

- Connectionless, unreliable TL Protocol

- User datagram packet format



a. UDP user datagram

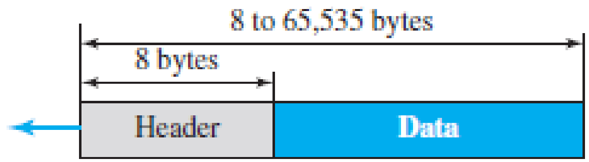


b. Header format

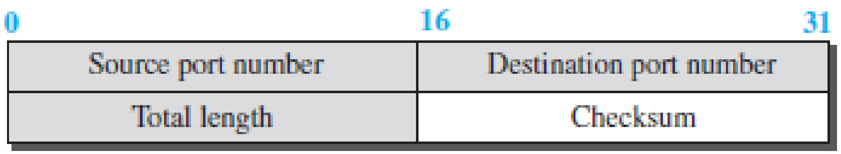
UDP Services

- Process-to-process communication
- Encapsulation and decapsulation
- Multiplexing and demultiplexing
- Flow control
- Error control
- Congestion control
- Connectionless services
- Checksum

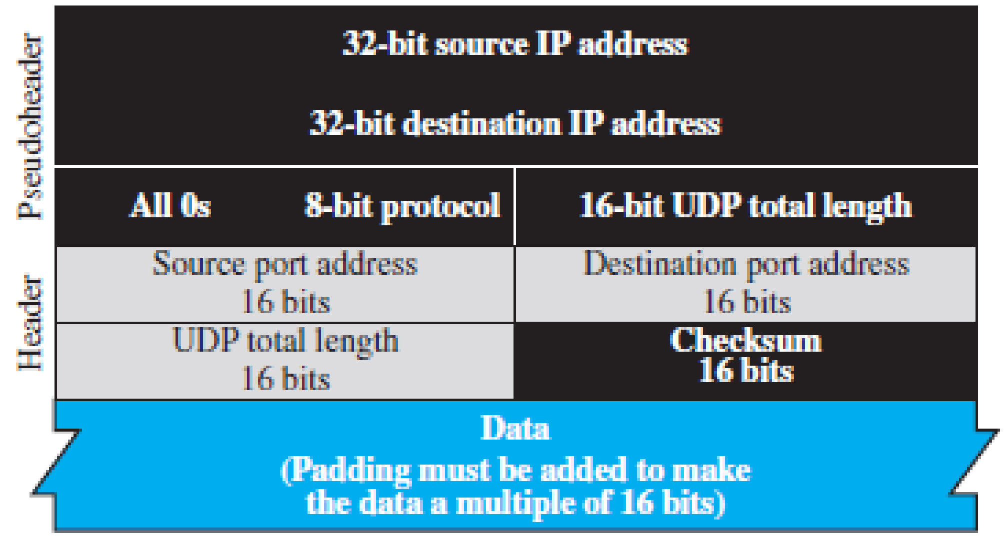
UDP: Checksum



a. UDP user datagram



b. Header format



TCP

- Connection oriented
  - Explicit set-up and tear-down of TCP session
- Stream-of-bytes service
  - Sends and receives a stream of bytes, not messages
- Reliable, in-order delivery
  - Checksums to detect corrupted data
  - Acknowledgments & retransmissions for reliable delivery
  - Sequence numbers to detect losses and reorder data
- Flow control
  - Prevent overflow of the receiver's buffer space
- Congestion control
  - Adapt to network congestion for the greater good

An Analogy: Talking on a Cell Phone

- Alice and Bob on their cell phones
  - Both Alice and Bob are talking
- What if Alice couldn’t understand Bob?
  - Bob asks Alice to repeat what she said
- What if Bob hasn’t heard Alice for a while?
  - Is Alice just being quiet?
  - Or, have Bob and Alice lost reception?
  - How long should Bob just keep on talking?
  - Maybe Alice should periodically say “uh huh”
  - … or Bob should ask “Can you hear me now?”

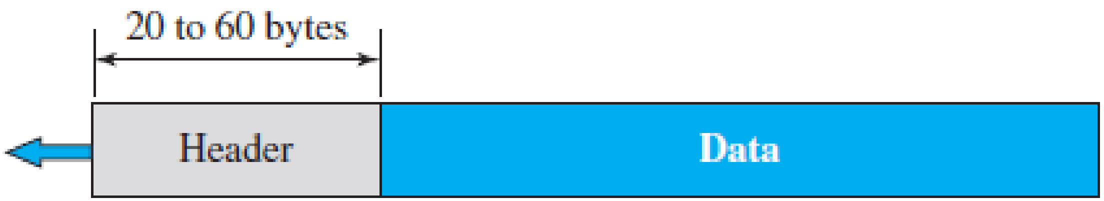
Some Take-Aways from the Example

- Acknowledgments from receiver
  - Positive: “okay” or “ACK”
  - Negative: “please repeat that” or “NACK”
- Timeout by the sender (“stop and wait”)
  - Don’t wait indefinitely without receiving some response
  - … whether a positive or a negative acknowledgment
- Retransmission by the sender
  - After receiving a “NACK” from the receiver
  - After receiving no feedback from the receiver

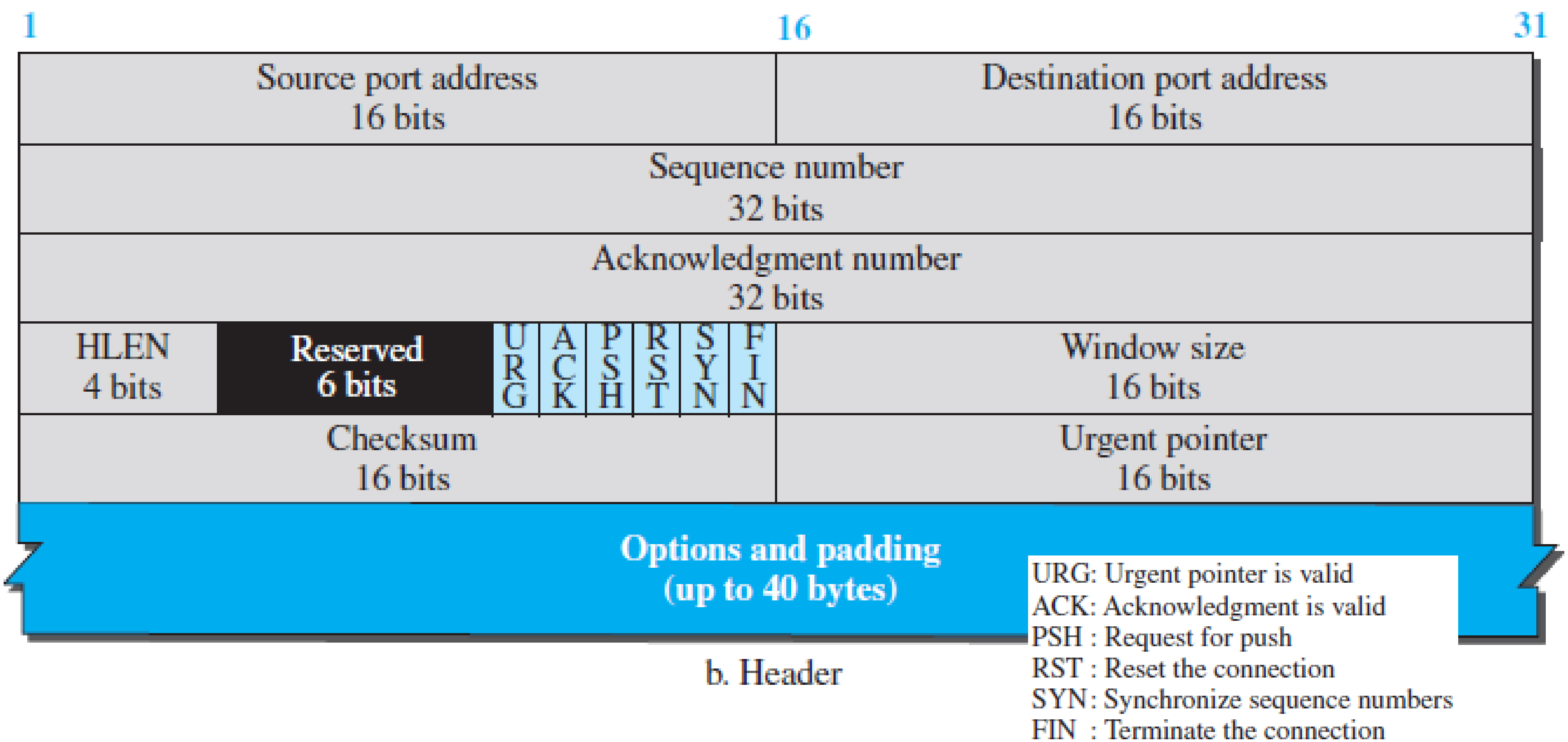
TCP Services

- Process-to-process communication
- Stream delivery service
- Full-Duplex communication
- Multiplexing and demultiplexing
- Connection-oriented service
- Reliable service

TCP Segment

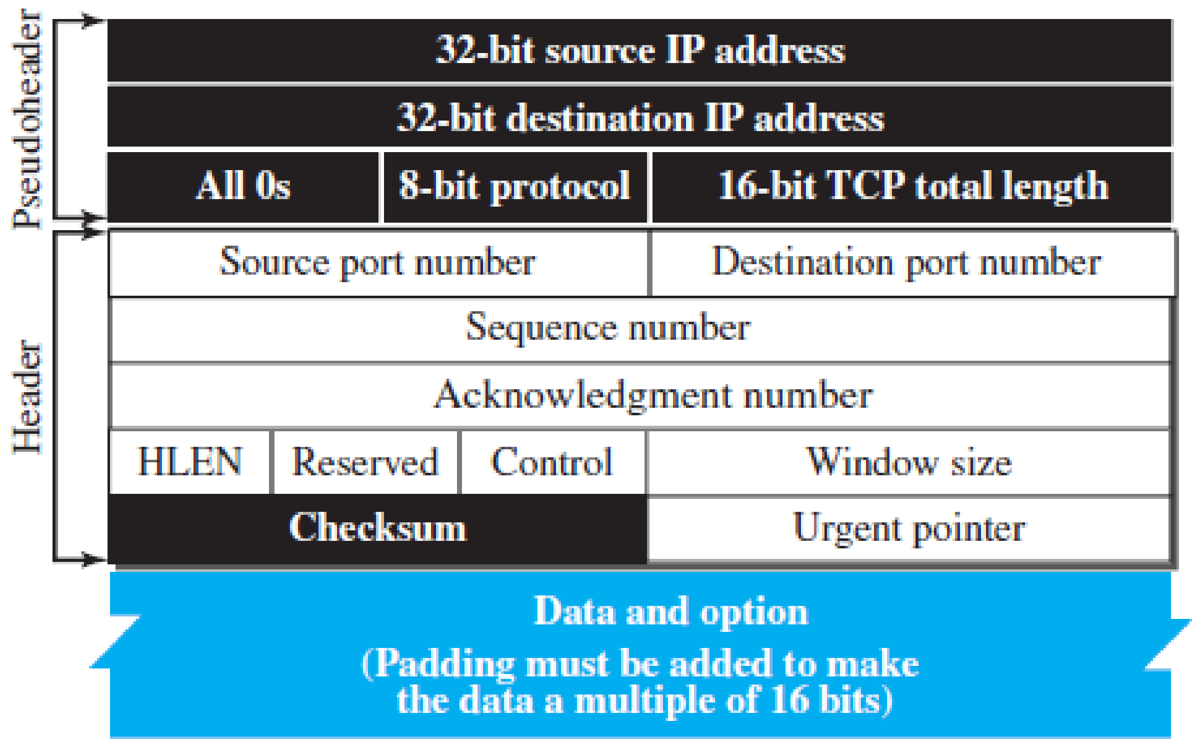


a. Segment



b. Header

TCP – Checksum



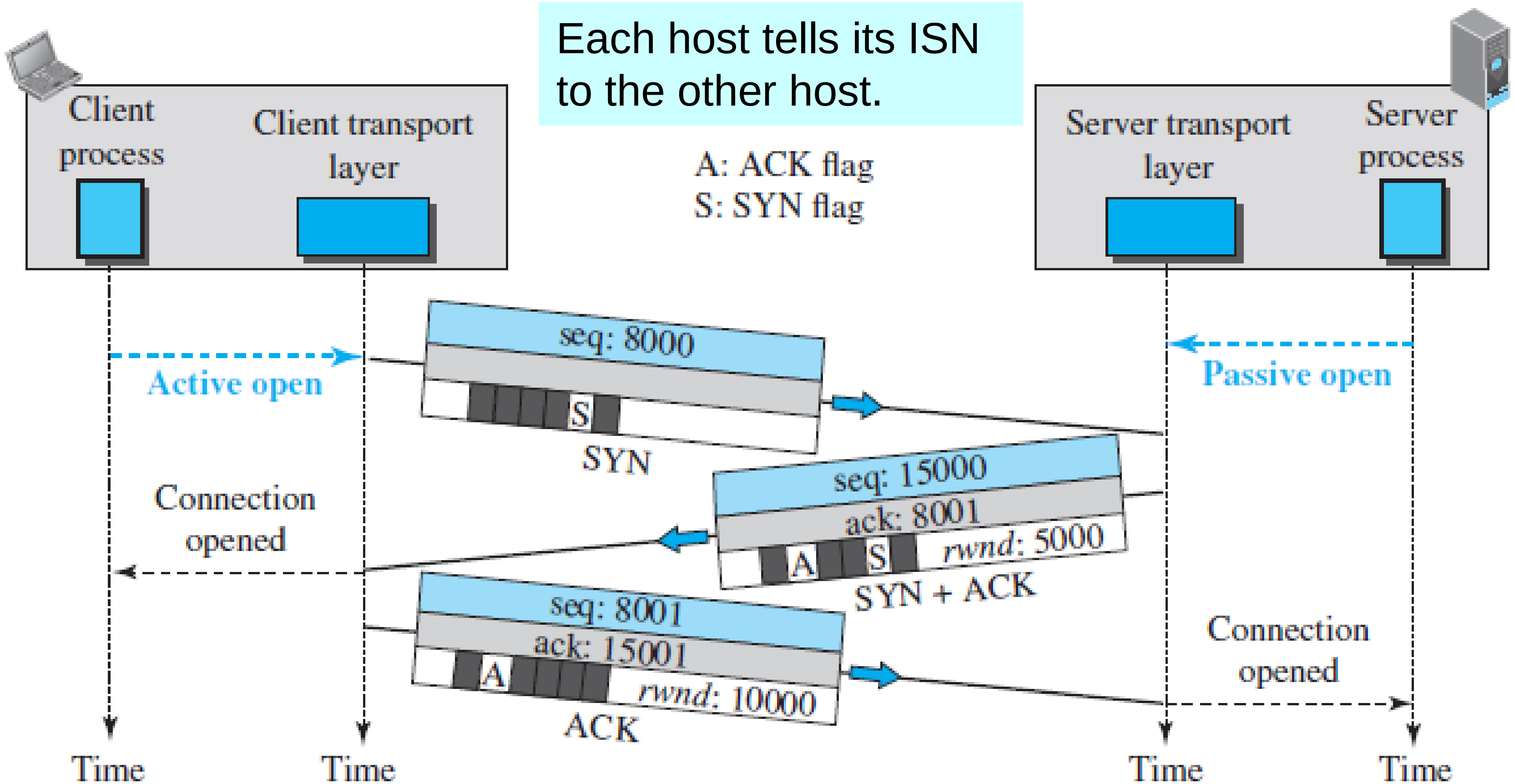
The use of the checksum in TCP is mandatory.

TCP Connection

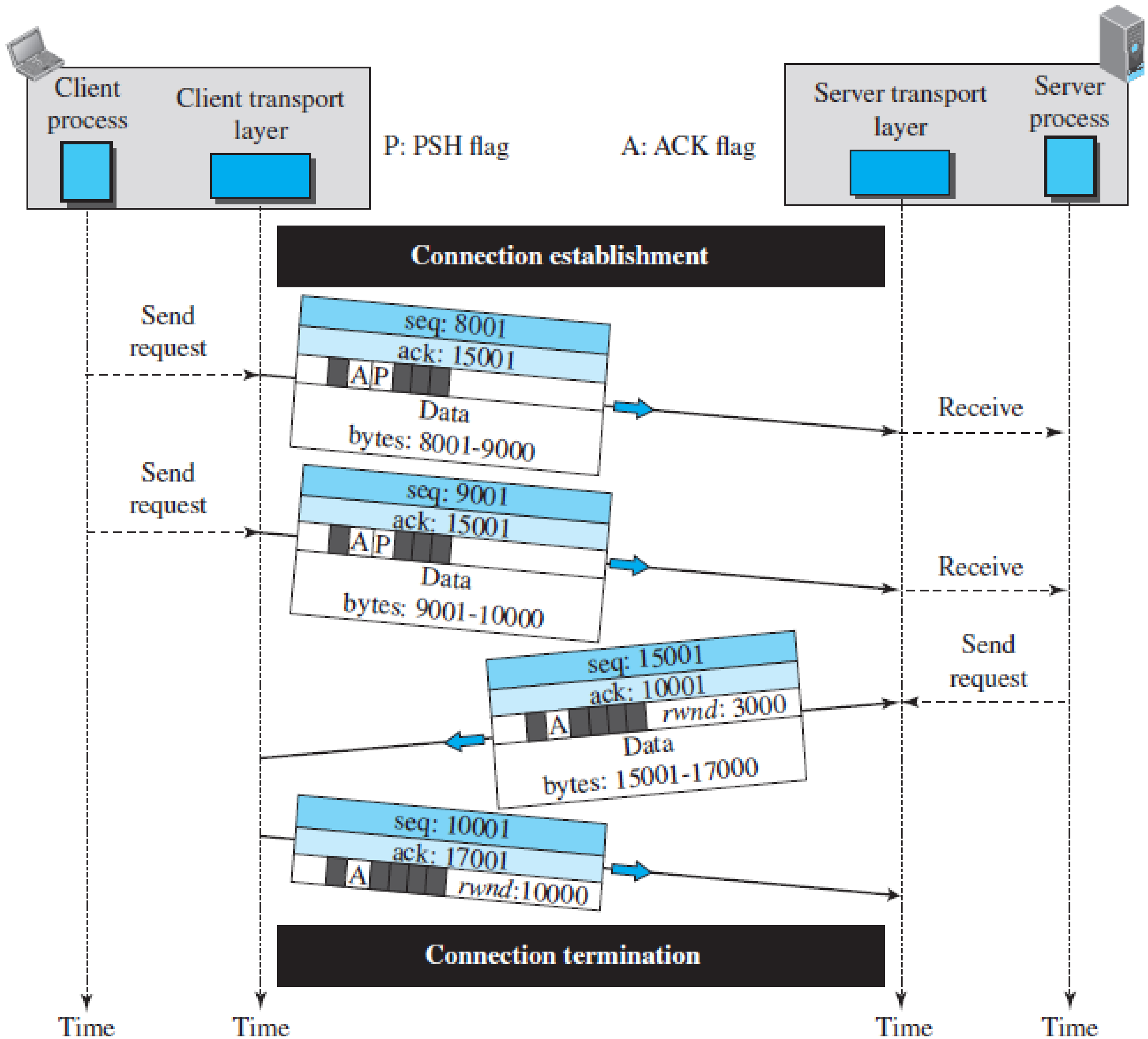
- Connection establishment
  - Three-way handshaking

SYN Flooding Attack

Each host tells its ISN to the other host.

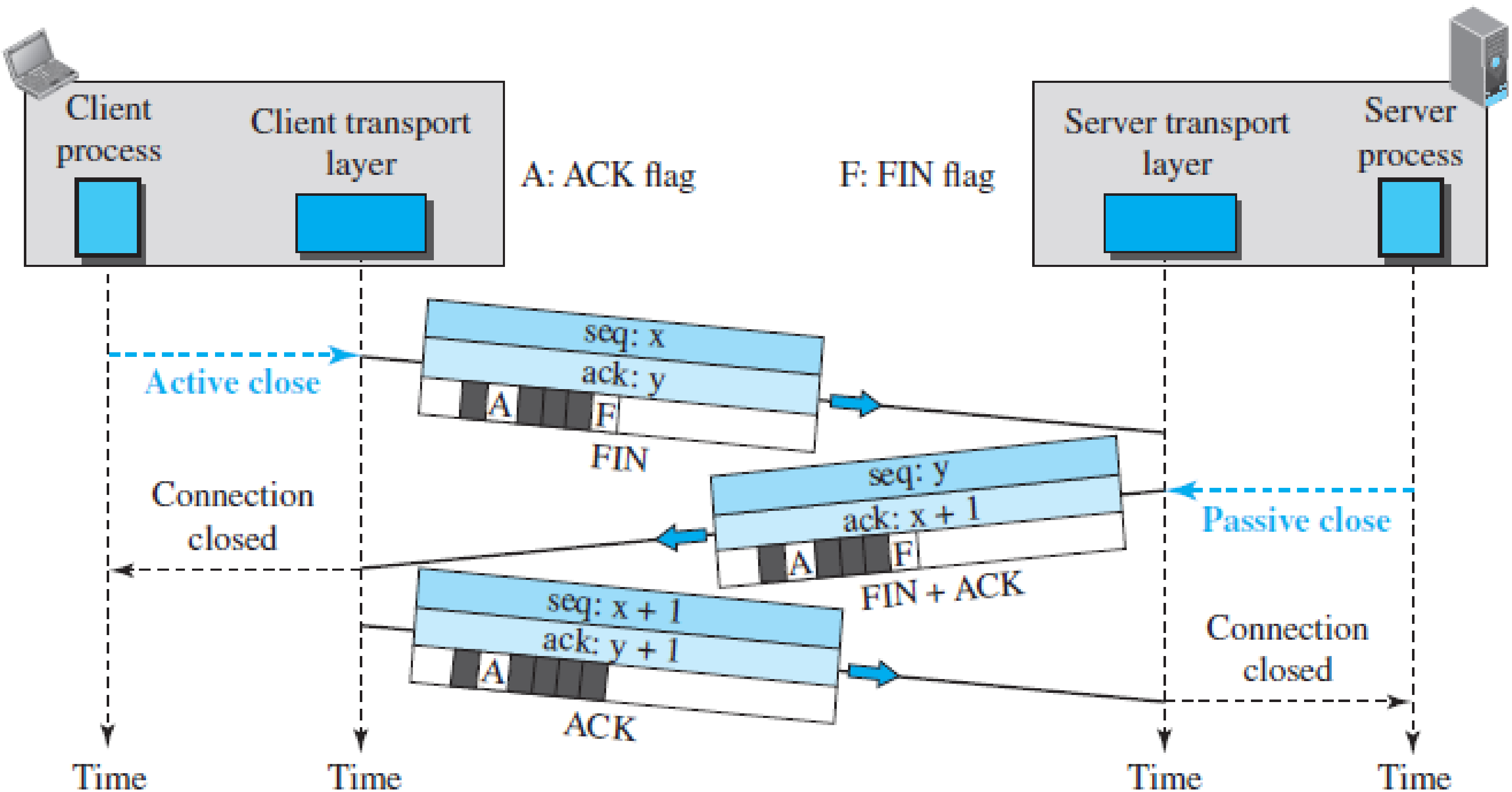


Data Transfer

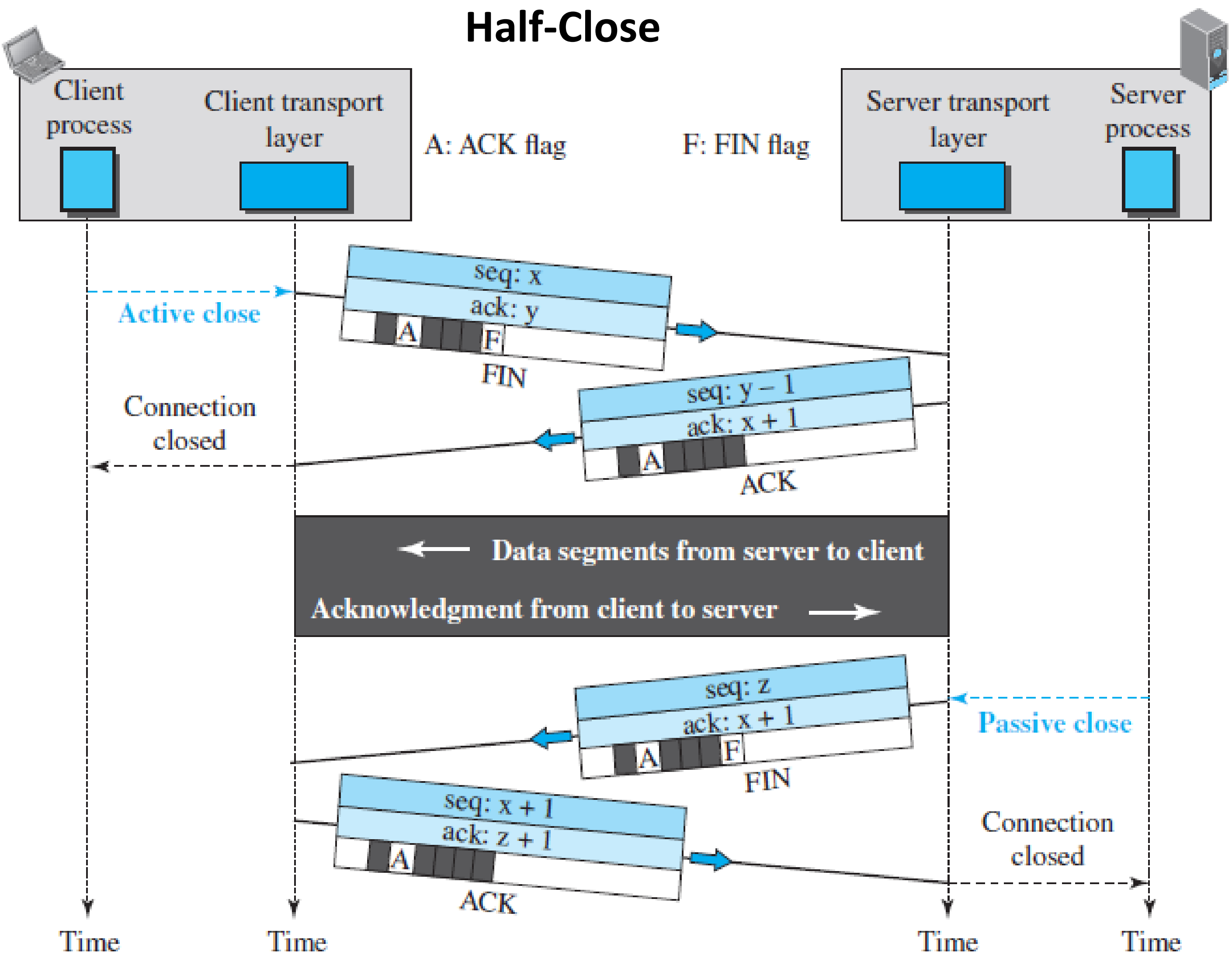


Connection Termination

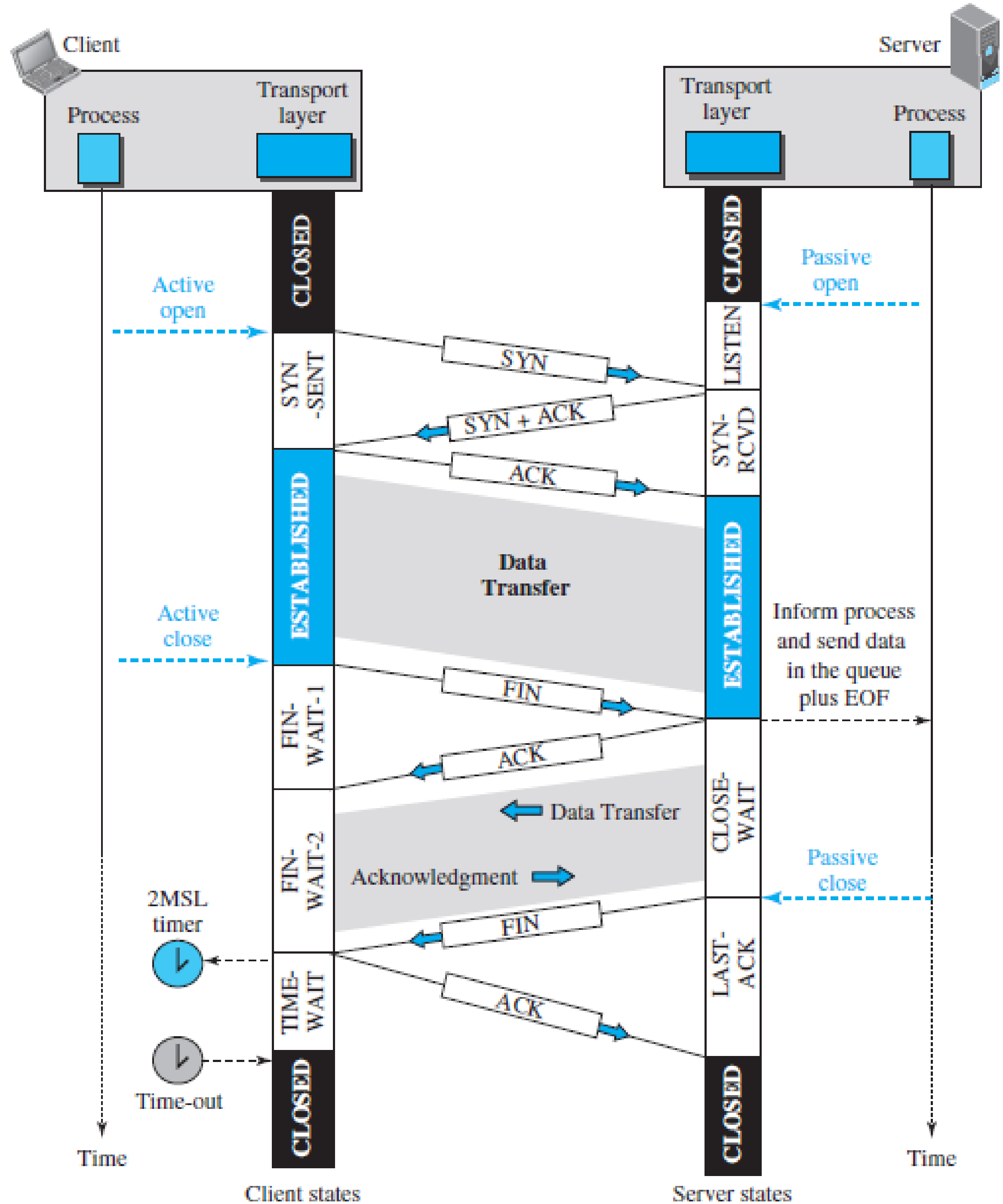
Three-Way Handshaking



Connection Termination

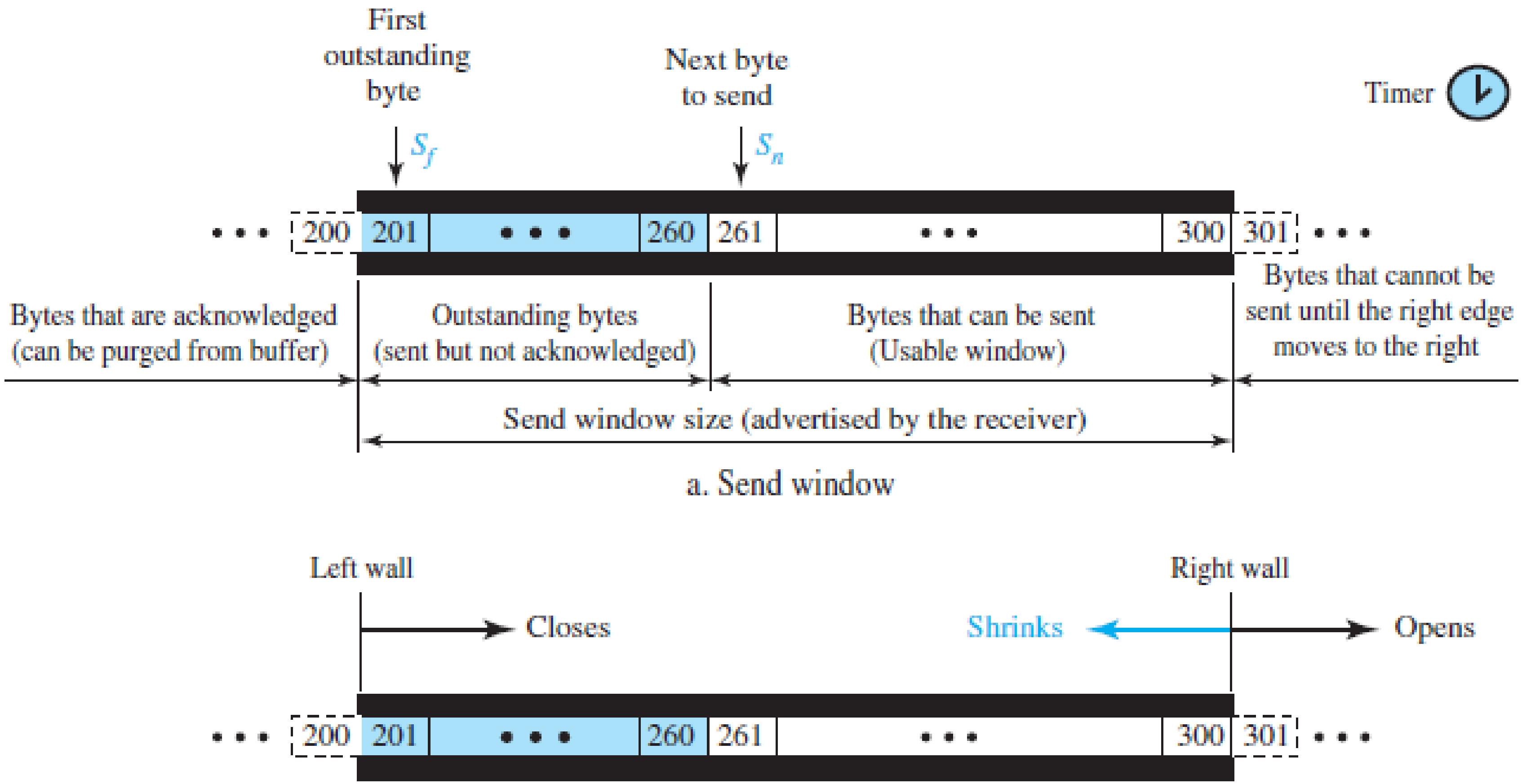


Time-line diagram for a common scenario



Windows in TCP

Send Window

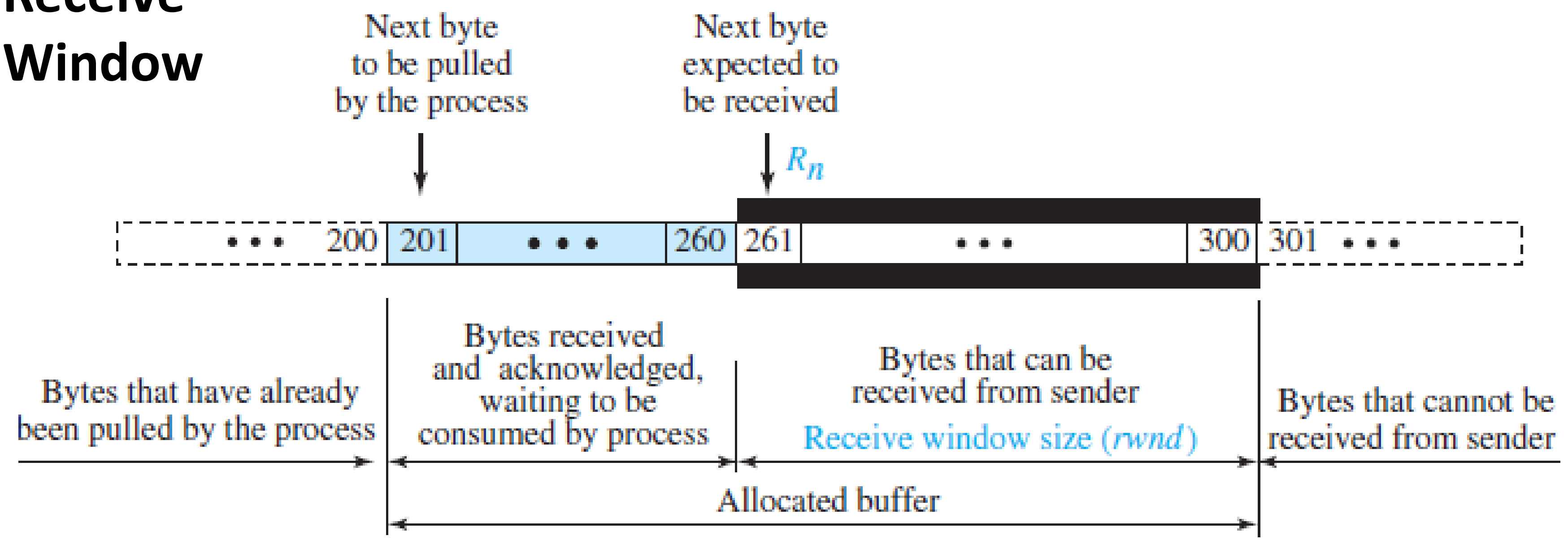


a. Send window

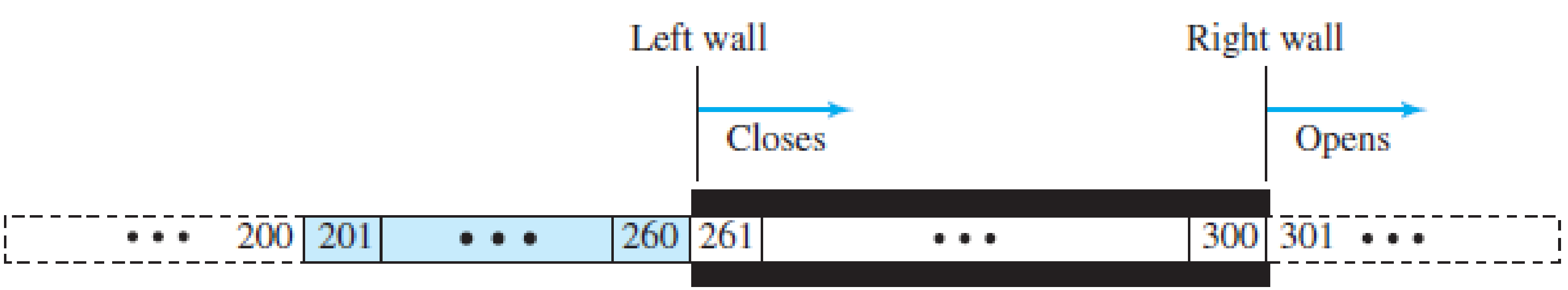
b. Opening, closing, and shrinking send window

Windows in TCP

Receive Window

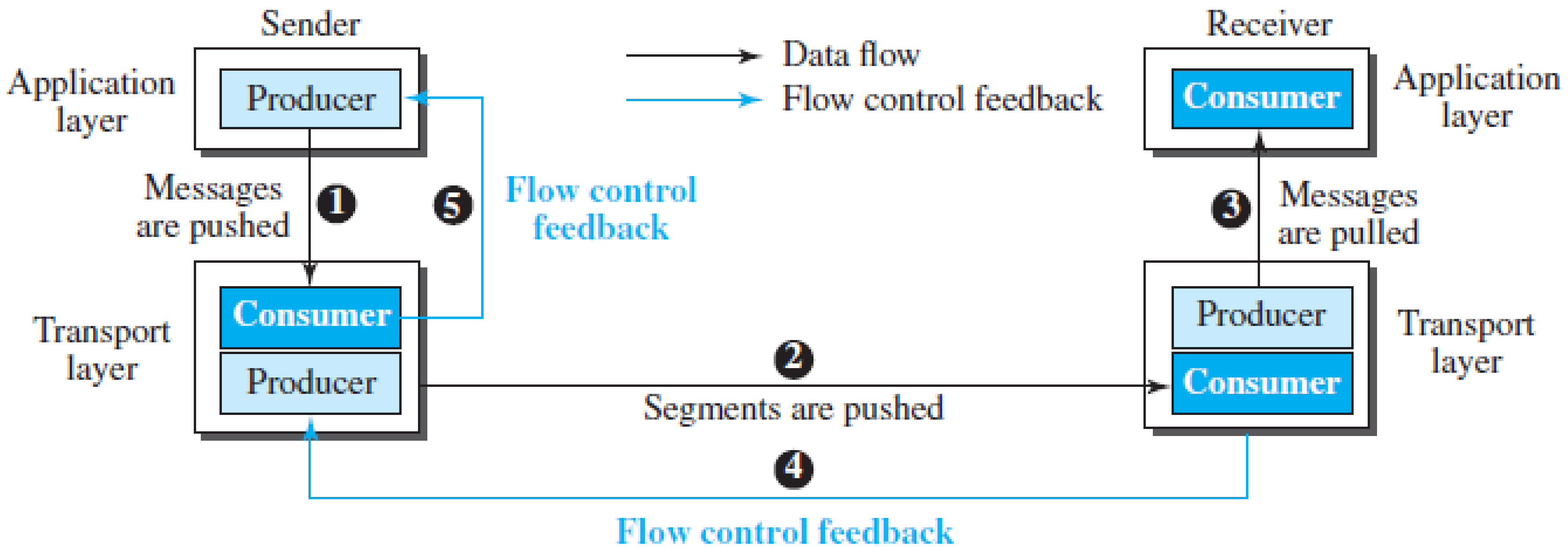


a. Receive window and allocated buffer

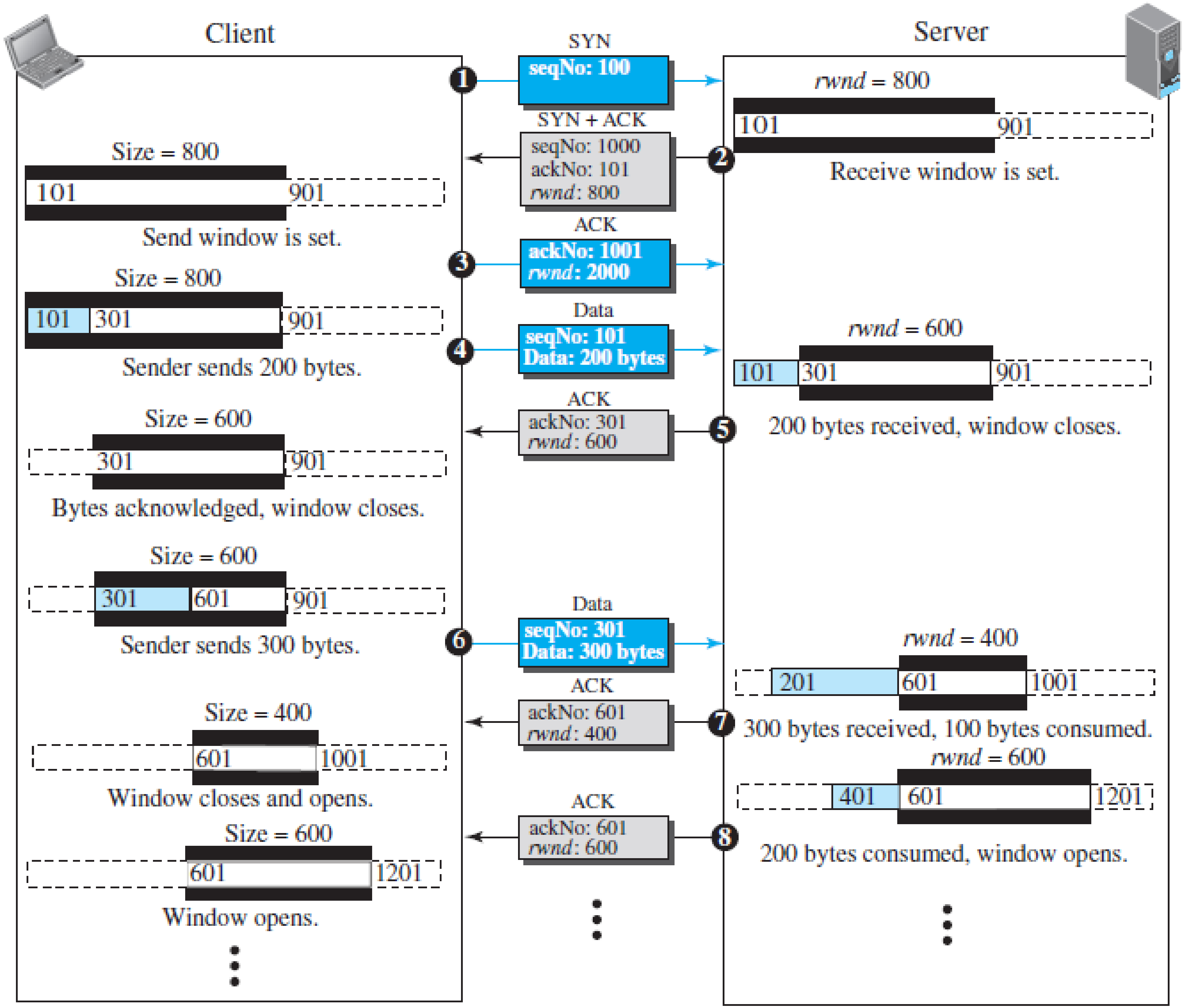


b. Opening and closing of receive window

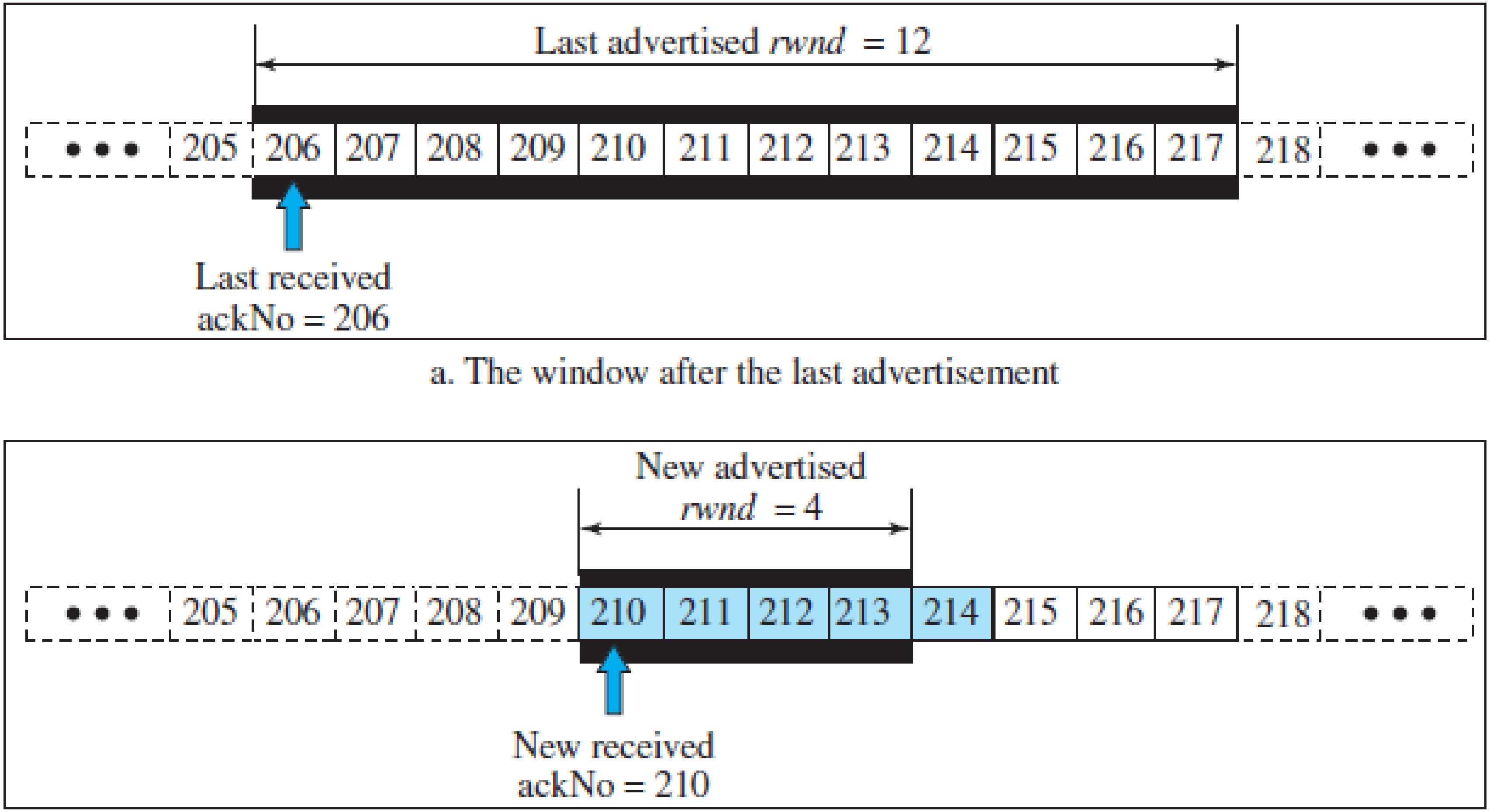
TCP – Flow Control



Data flow and flow control feedbacks in TCP


Client
Server
SYN
seqNo: 100
rwnd = 800
101
901
SYN + ACK
seqNo: 1000
ackNo: 101
rwnd: 800
Receive window is set.
Size = 800
101
901
Send window is set.
ACK
ackNo: 1001
rwnd: 2000
Size = 800
101
301
901
Data
seqNo: 101
Data: 200 bytes
rwnd = 600
101
301
901
Sender sends 200 bytes.
ACK
ackNo: 301
rwnd: 600
200 bytes received, window closes.
Size = 600
301
901
Bytes acknowledged, window closes.
Size = 600
301
601
901
Data
seqNo: 301
Data: 300 bytes
Sender sends 300 bytes.
rwnd = 400
201
601
1001
ACK
ackNo: 601
rwnd: 400
300 bytes received, 100 bytes consumed.
Size = 400
601
1001
rwnd = 600
401
601
1201
Window closes and opens.
ACK
ackNo: 601
rwnd: 600
200 bytes consumed, window opens.
Size = 600
601
1201
Window opens.

Shrinking of Window



a. The window after the last advertisement

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

Window Shutdown

Silly Window Syndrome

- The sending application program creates data slowly or the receiving application program consumes data slowly, or both.
  - Creates problem in sliding window operation and leads to overhead.
- Syndrome created by
  - Sender
  - Receiver

Syndrome created by the Sender

- The sending TCP may create a silly window syndrome if it is serving an application program that creates data slowly
- Solution: prevent the sending TCP from sending the data in small quantity
  - Nagle’s Algorithm
    - Send first piece of data received from sending app.
    - Accumulate data and wait until an ACK is received or until max data is accumulated

Syndrome created by the Receiver

- The receiving TCP may create a silly window syndrome if it is serving an application program that consumes data slowly
- Two Solutions:
  - send an ACK as soon as the data arrive, but to announce a window size of zero until either there is enough space to accommodate a segment of maximum size or until at least half of the receive buffer is empty (Clark's Solution)
  - delay sending the acknowledgment

TCP – Error Control

- Detecting and resending corrupted segments
- Resending lost segments
- Storing out-of order segments until missing segments arrive
- Detecting and discarding duplicated segments

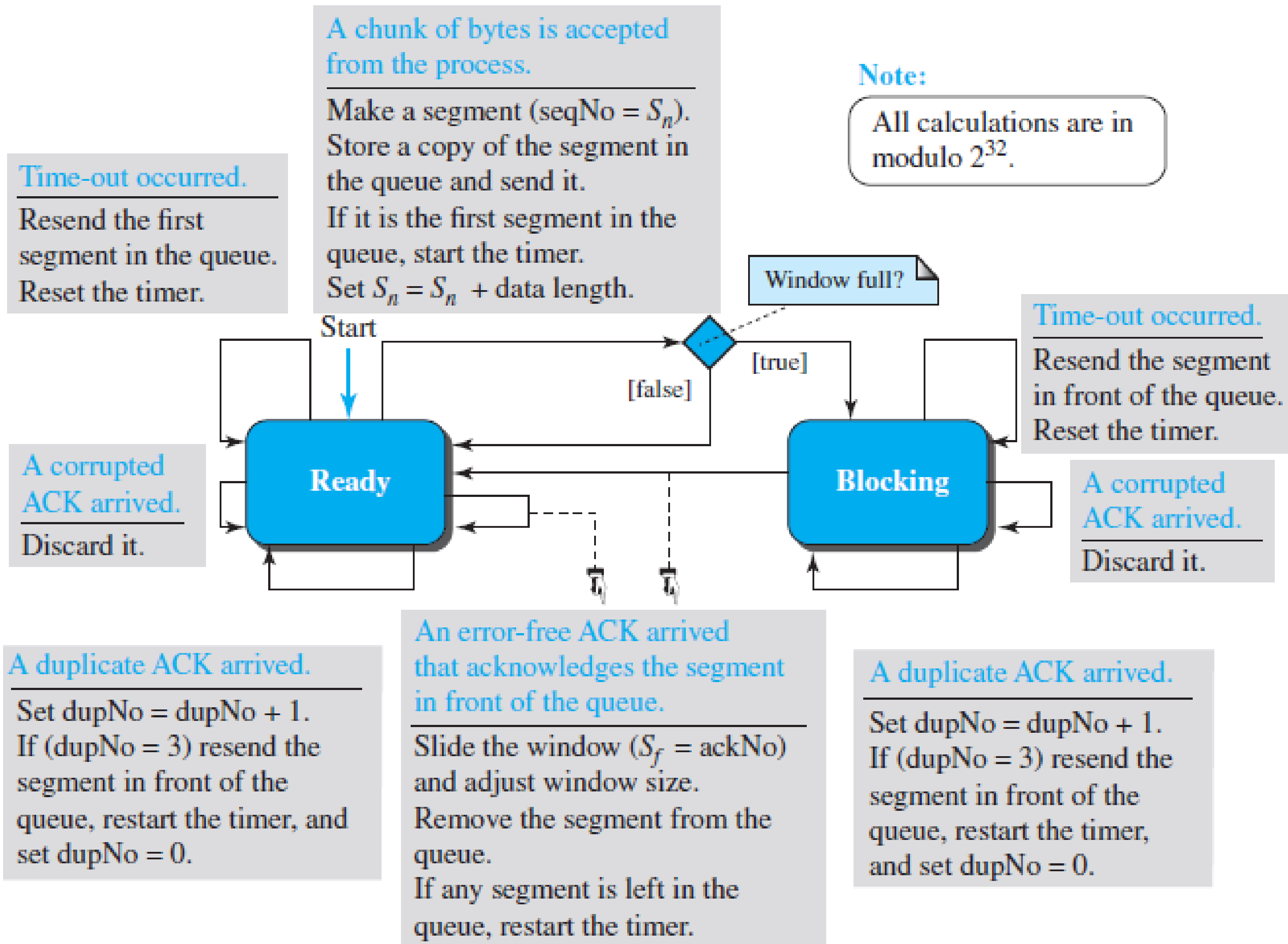
- Solution
  - checksum, acknowledgment (Cumulative, Selective), and time-out

TCP Acknowledgement

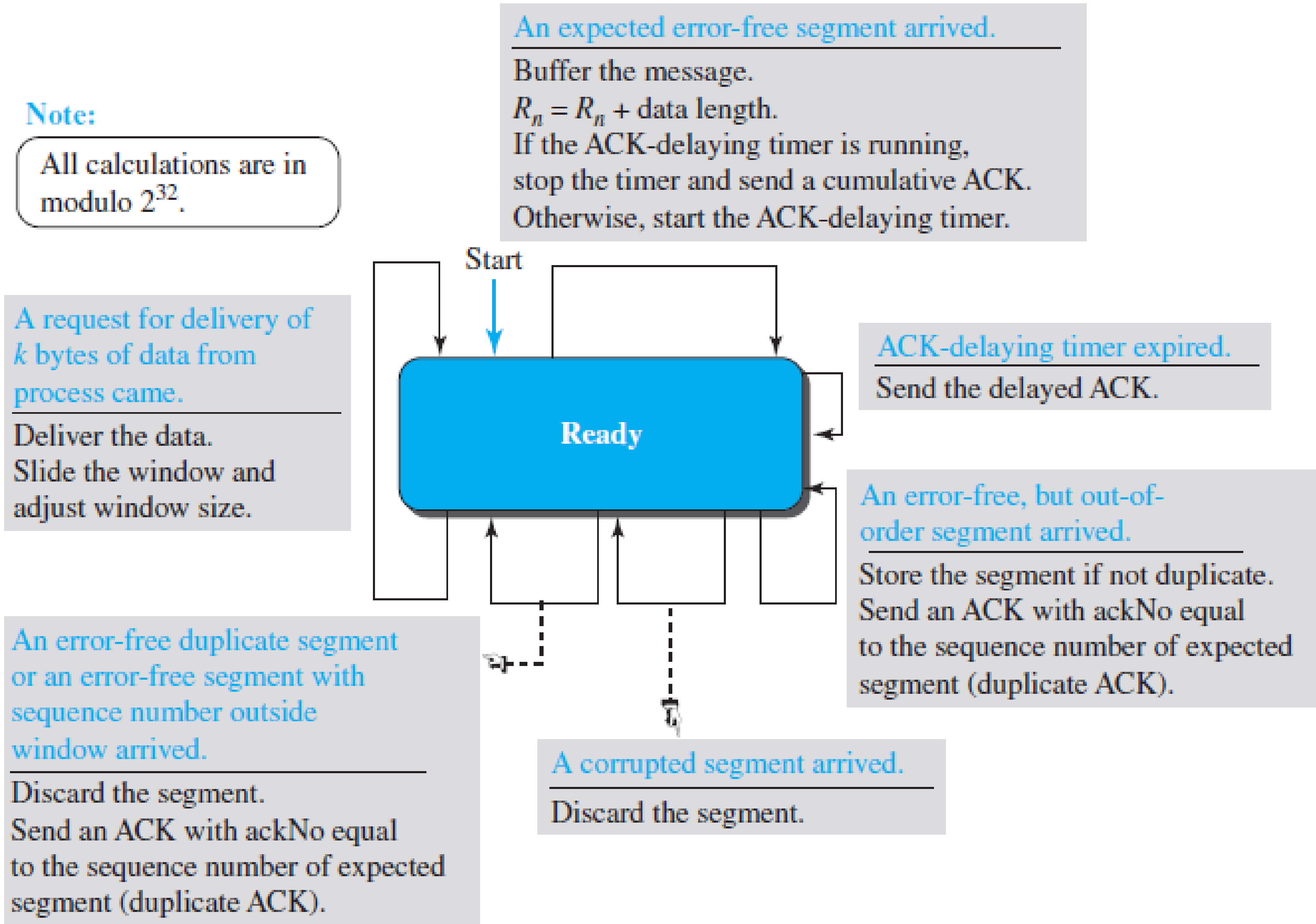
- Acknowledgement
  - Cumulative (ACK)
  - Selective (SACK)

- Retransmission
  - After RTO (Retransmission time-out)
  - After three duplicate ACK segments
    - Fast retransmission

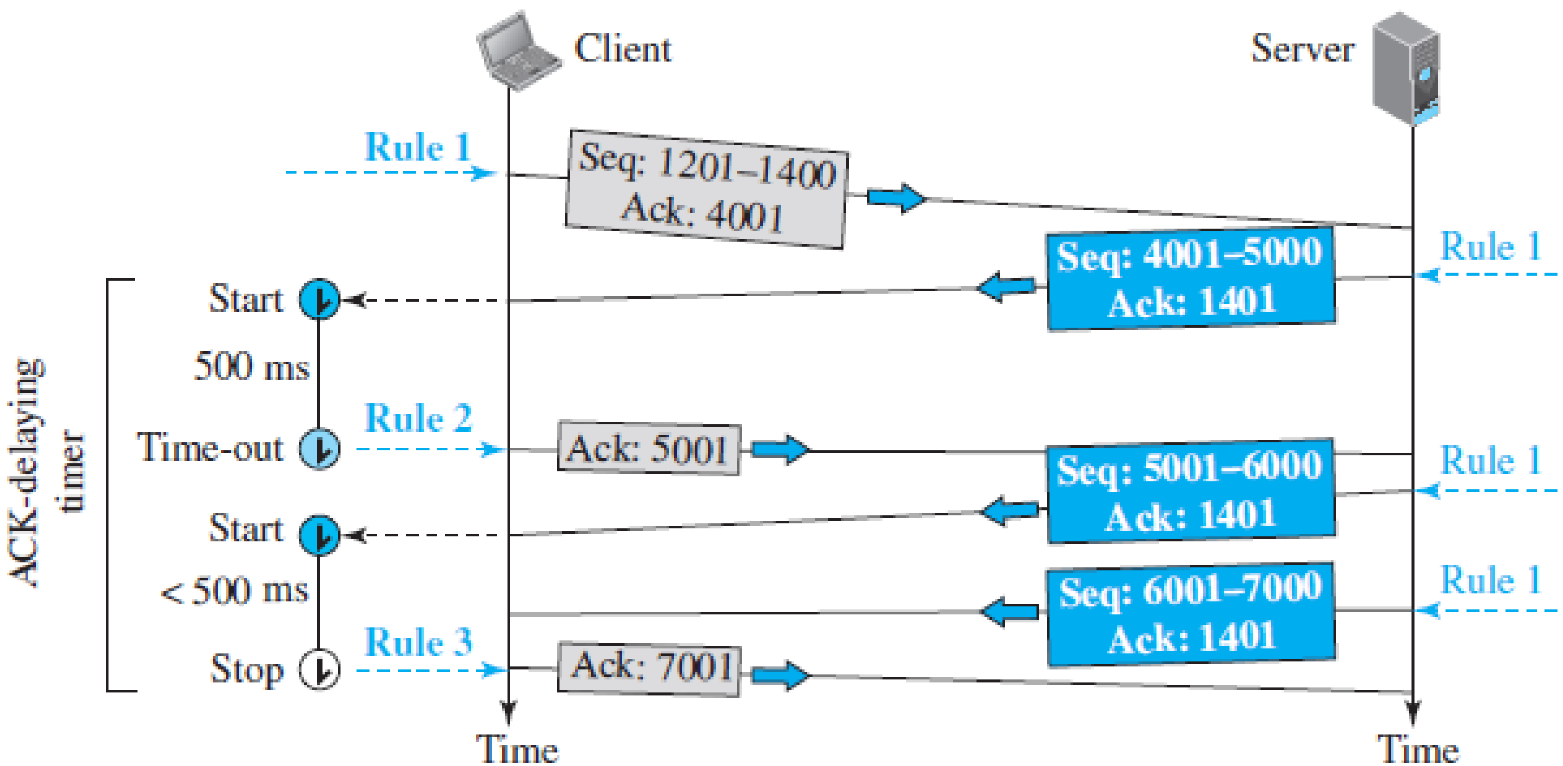
FSM for the TCP sender side



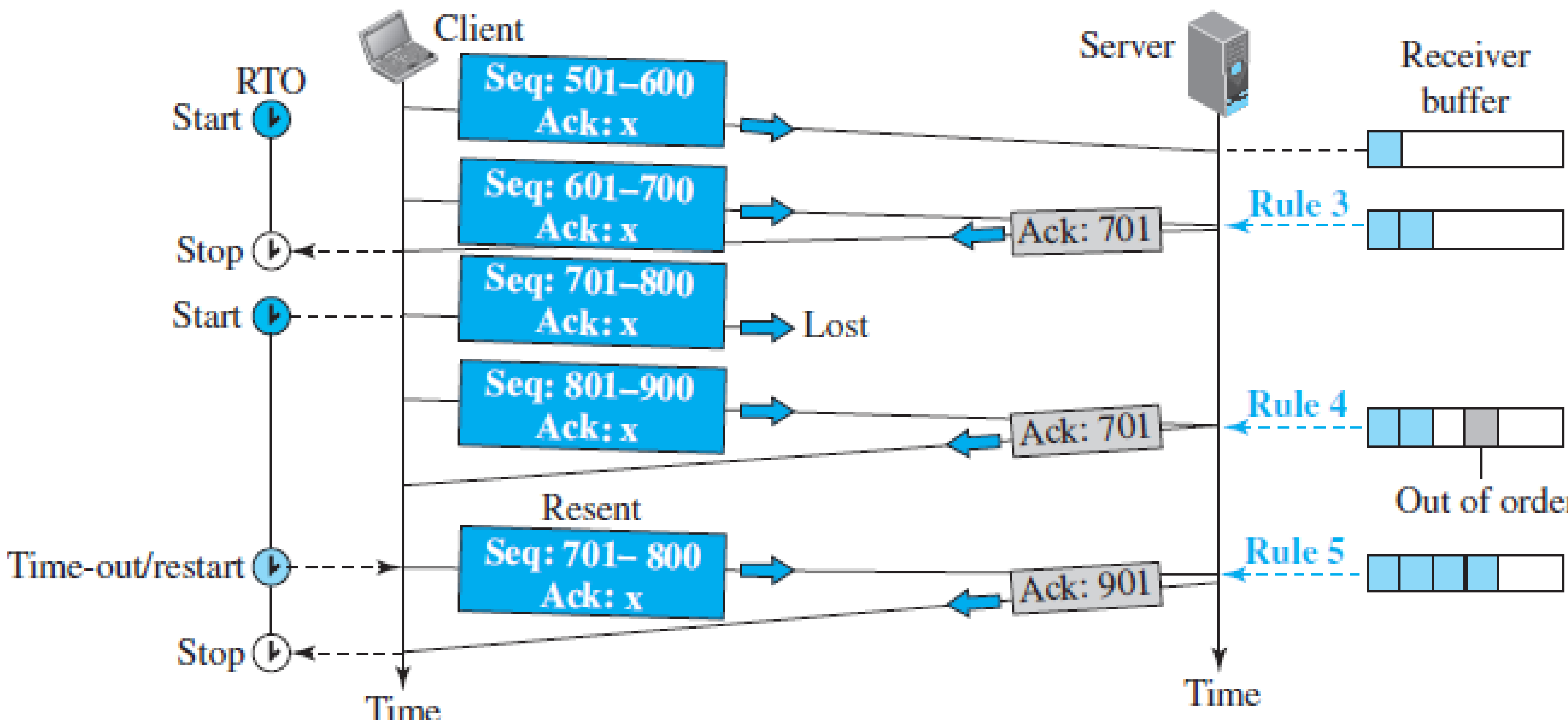
FSM for the TCP receiver side



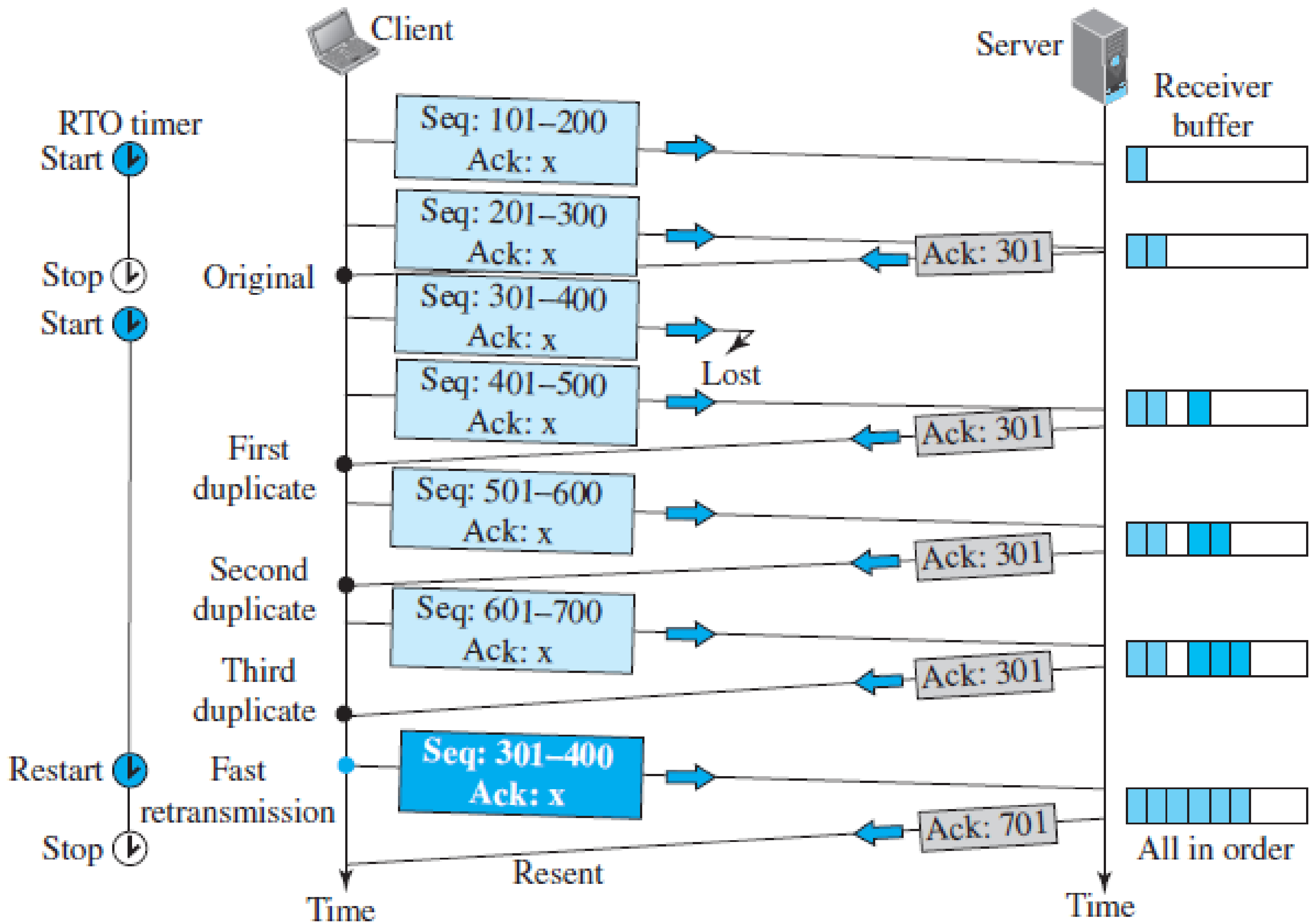
Normal Operations



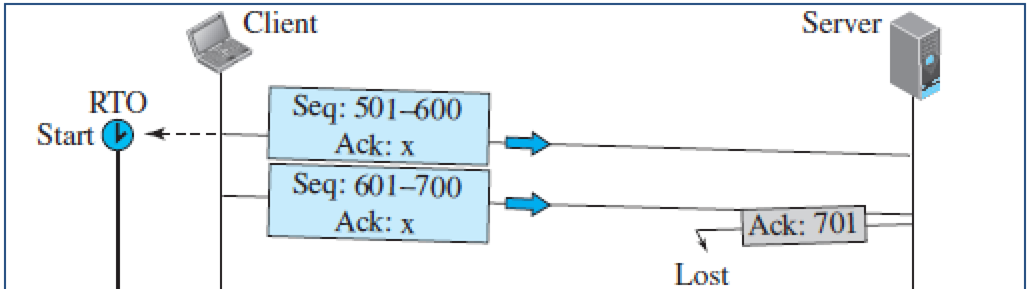
Lost Segment



Fast retransmission

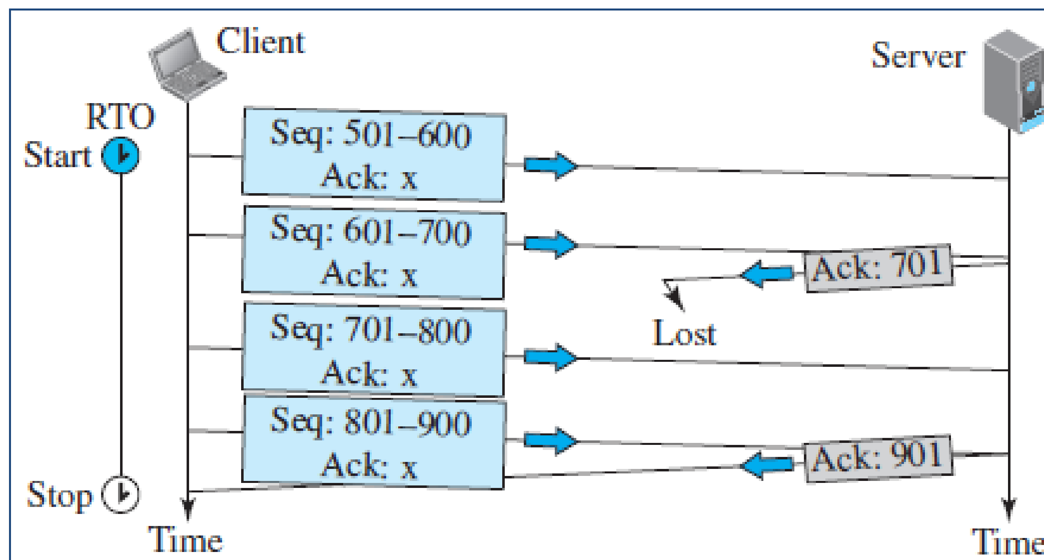


Lost acknowledgement



Next?

Deadlock due to Lost acknowledgement



TCP – Congestion Control

- Implemented at the Sender side
- Send window = MIN(*rwnd*, *cwnd*)
  - *rwnd*: advertised by the receiver
  - *cwnd*: adjusted based on feedback from the network

- Congestion Detection
  - Time-out – Strong congestion
  - Three duplicate ACKs – Weak congestion

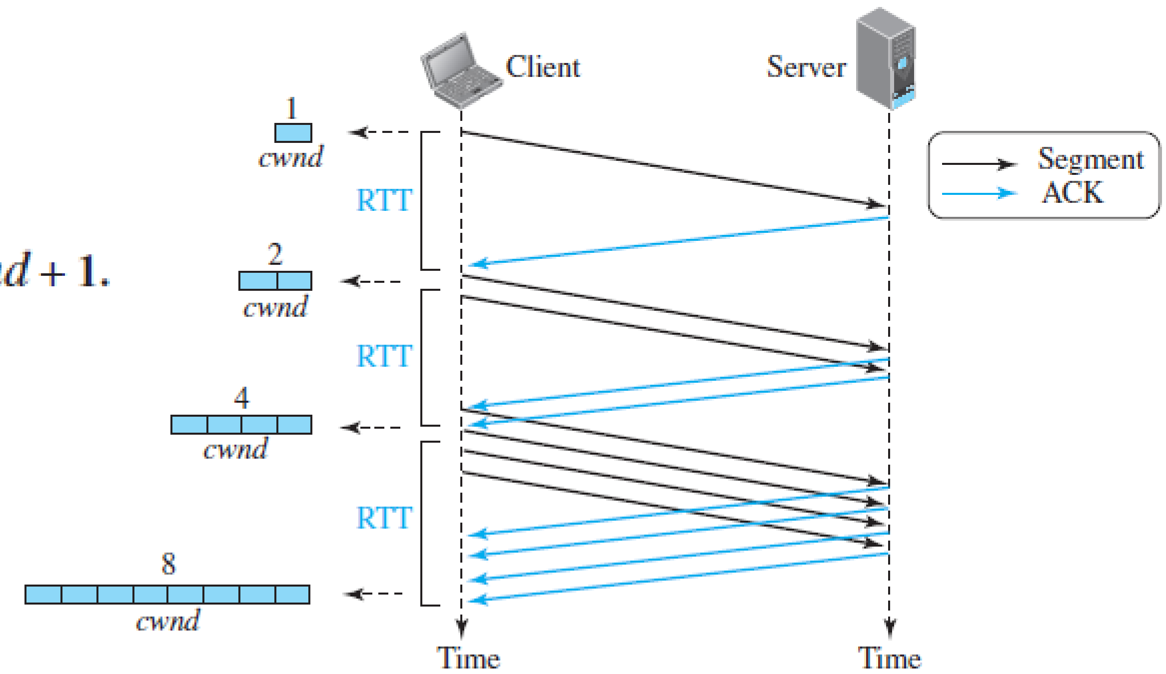
Congestion Policies

- Slow start: Exponential increase
- Congestion avoidance
- Fast recovery

Slow Start

Initially, *cwnd* = 1 MSS

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

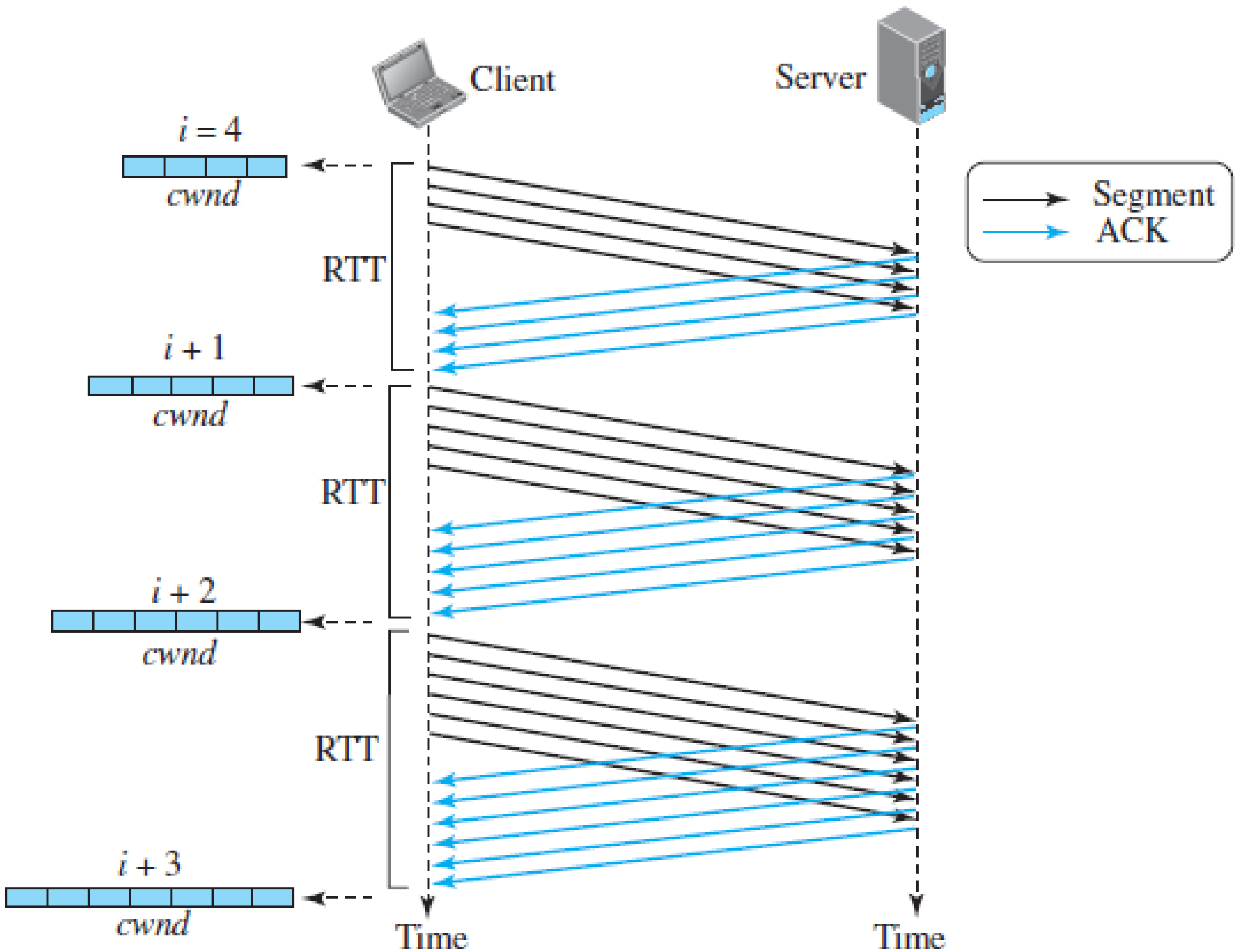


Start	$\rightarrow$	$cwnd = 1 \rightarrow 2^0$
After 1 RTT	$\rightarrow$	$cwnd = cwnd + 1 \ = 1 + 1 = 2 \rightarrow 2^1$
After 2 RTT	$\rightarrow$	$cwnd = cwnd + 2 \ = 2 + 2 = 4 \rightarrow 2^2$
After 3 RTT	$\rightarrow$	$cwnd = cwnd + 4 \ = 4 + 4 = 8 \rightarrow 2^3$

Congestion Avoidance

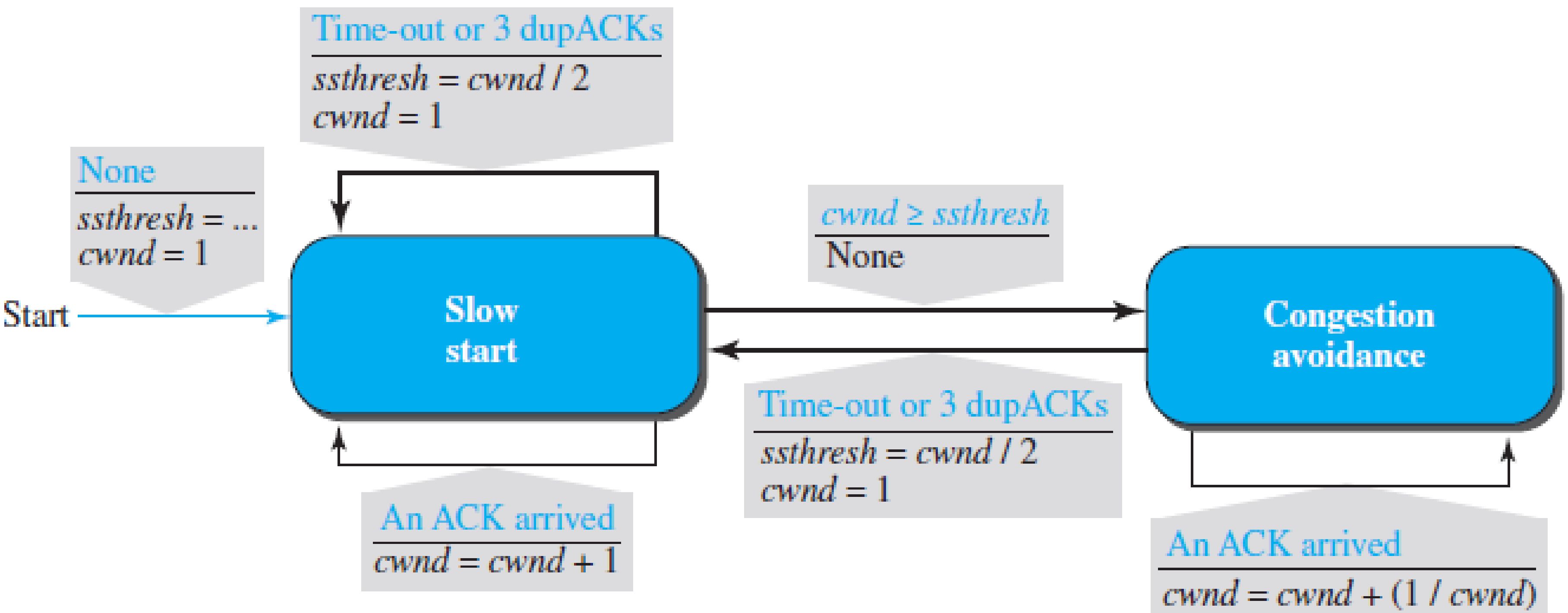
- Congestion avoidance phase is started if *cwnd* has reached the *slow-start threshold* value
- If cwnd >= ssthresh then each time an ACK is received, increment *cwnd* as follows:
  - cwnd = cwnd + 1/cwnd
- So *cwnd* is increased by one segment (=MSS bytes) only if all segments have been acknowledged.

Congestion Avoidance

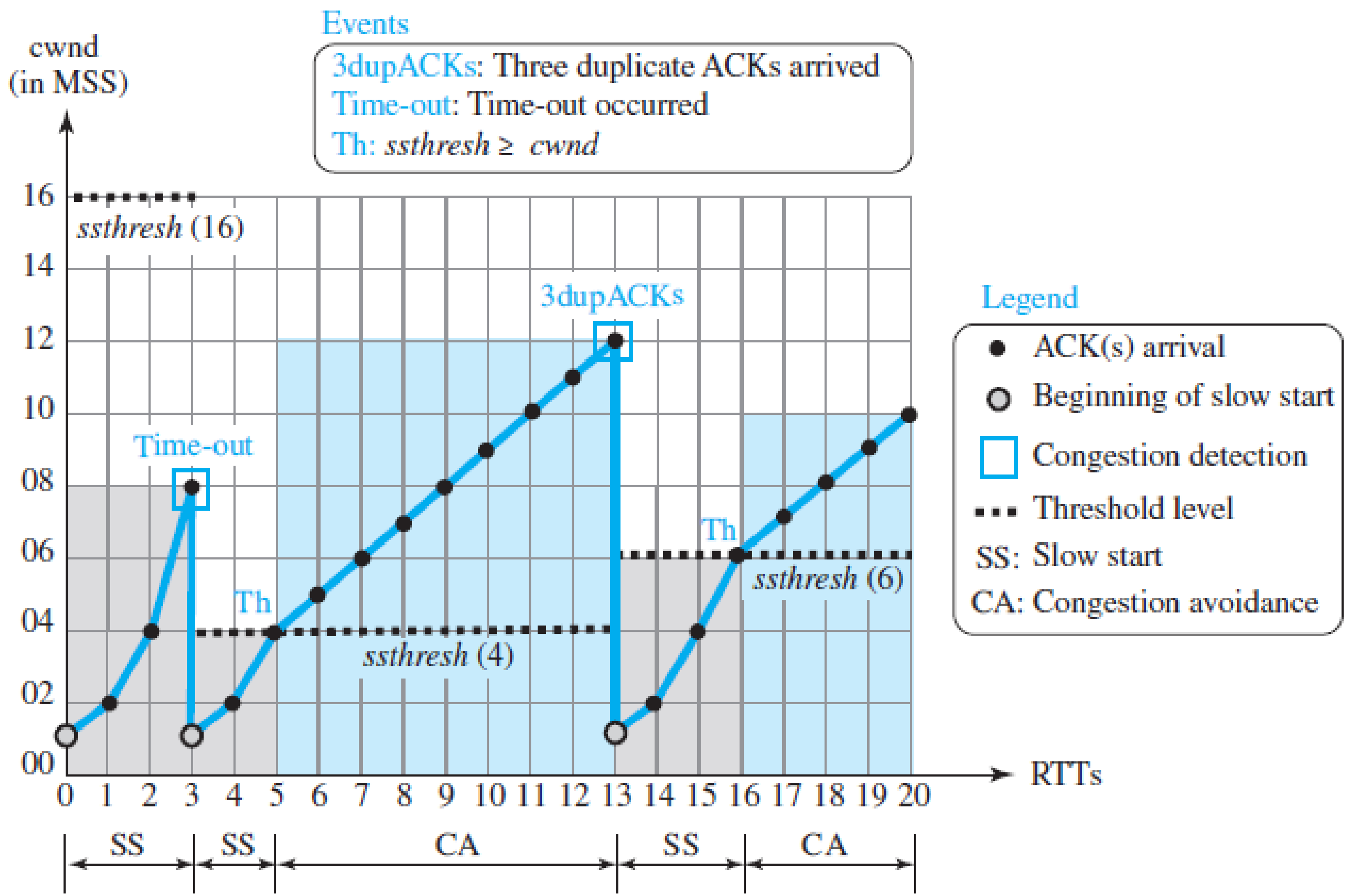


TCP – Congestion Control

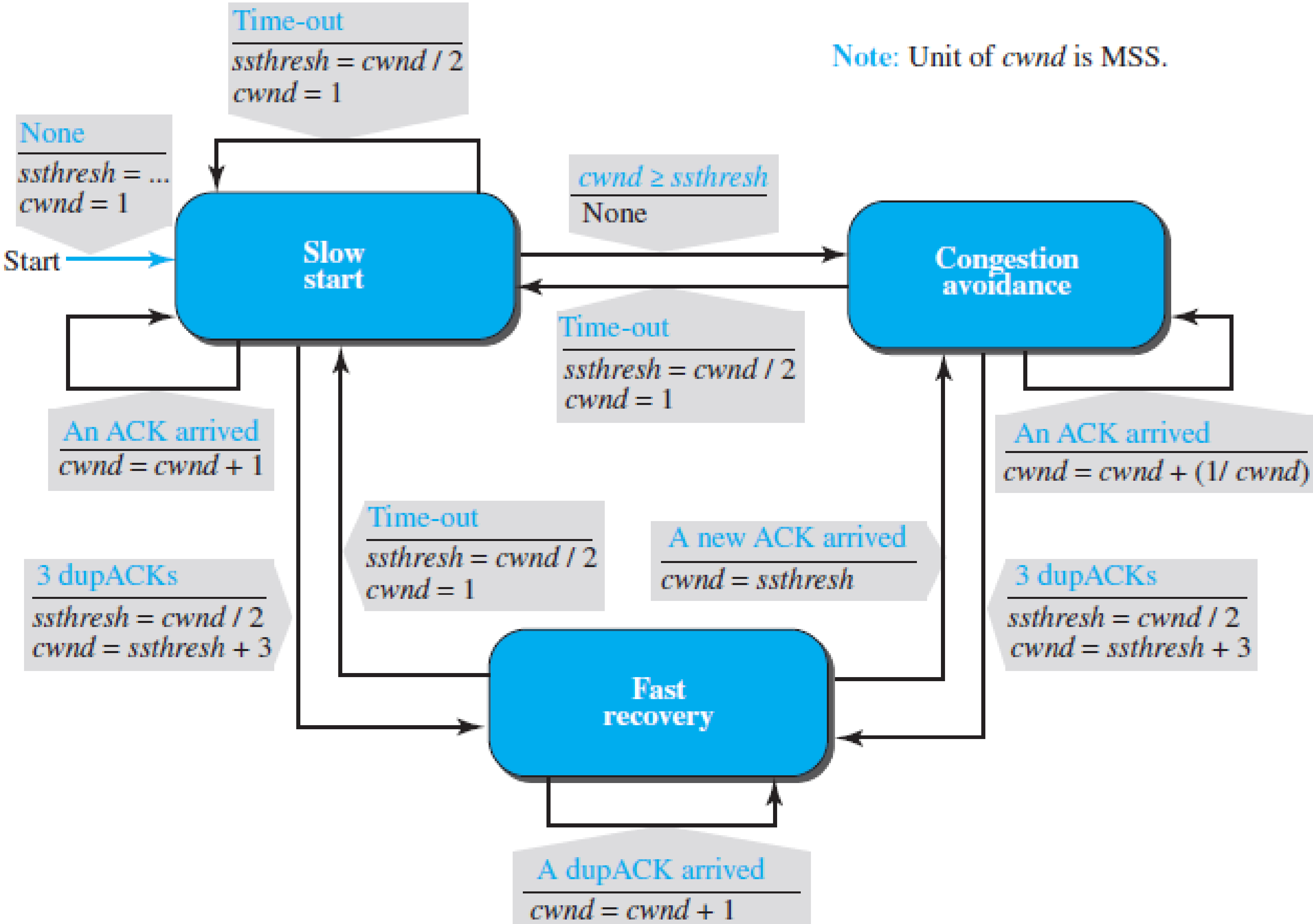
- Taho TCP
  - Slow start and Congestion Avoidance



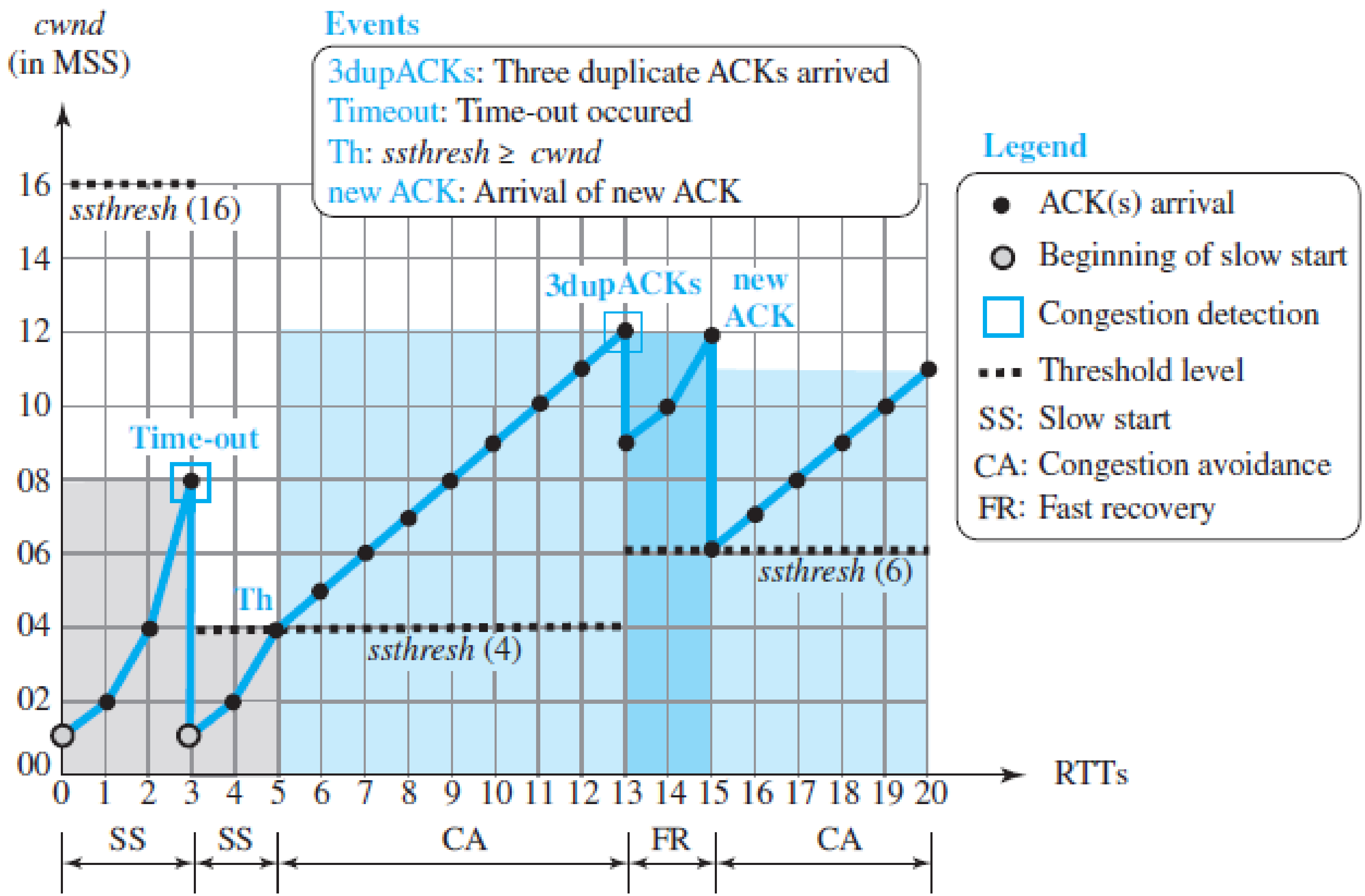
Taho TCP – Example



Reno TCP


Time-out
ssthresh = cwnd / 2
cwnd = 1
Note: Unit of cwnd is MSS.
None
ssthresh = ...
cwnd = 1
Start
Slow start
cwnd ≥ ssthresh
None
Congestion avoidance
Time-out
ssthresh = cwnd / 2
cwnd = 1
An ACK arrived
cwnd = cwnd + 1
An ACK arrived
cwnd = cwnd + (1/ cwnd)
Time-out
ssthresh = cwnd / 2
cwnd = 1
A new ACK arrived
cwnd = ssthresh
3 dupACKs
ssthresh = cwnd / 2
cwnd = ssthresh + 3
3 dupACKs
ssthresh = cwnd / 2
cwnd = ssthresh + 3
Fast recovery
A dupACK arrived
cwnd = cwnd + 1

Reno TCP – Example



AIMD

- Additive increase, multiplicative decrease (AIMD)