Chapter 3

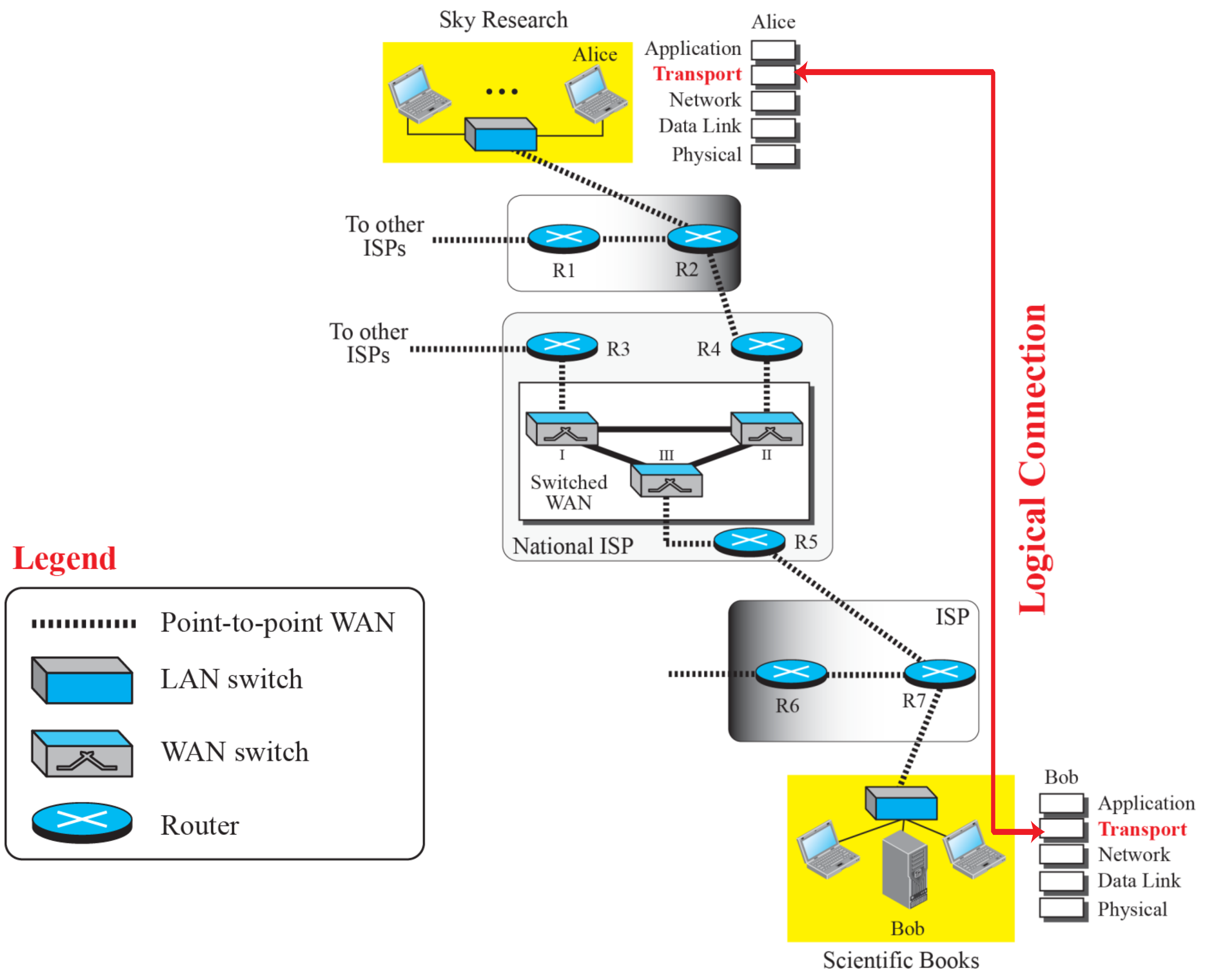
Transport Layer

3-1 INTRODUCTION

- ***provides a process-to-process communication between two application layers***

- ***Communication is provided using a logical connection***

Figure 3.1: Logical connection at the transport layer

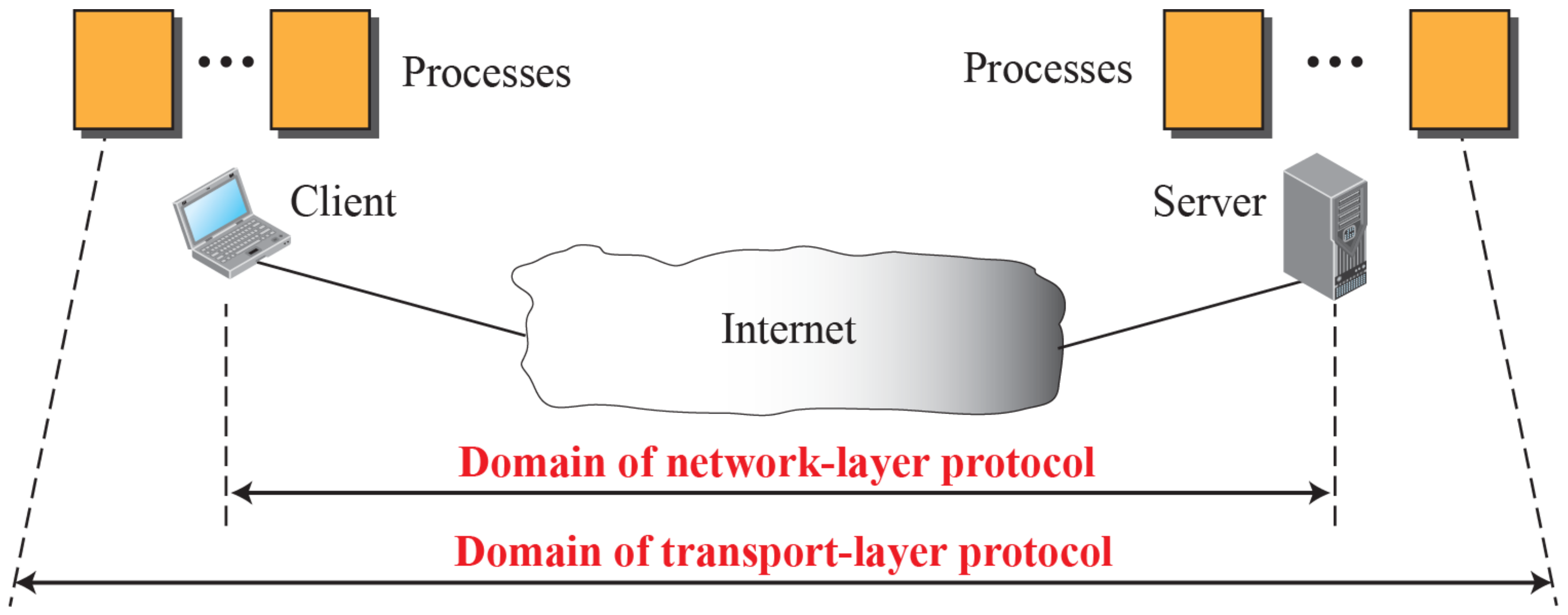


3.1.1 Transport-Layer Services

- ***is located between the network layer and the application layer***

- ***is responsible for providing services to the application layer; it receives services from the network layer***

❑ *Process-to-Process Communication*


Processes
Processes
Client
Server
Internet
Domain of network-layer protocol
Domain of transport-layer protocol

❑ *Addressing: Port Numbers*

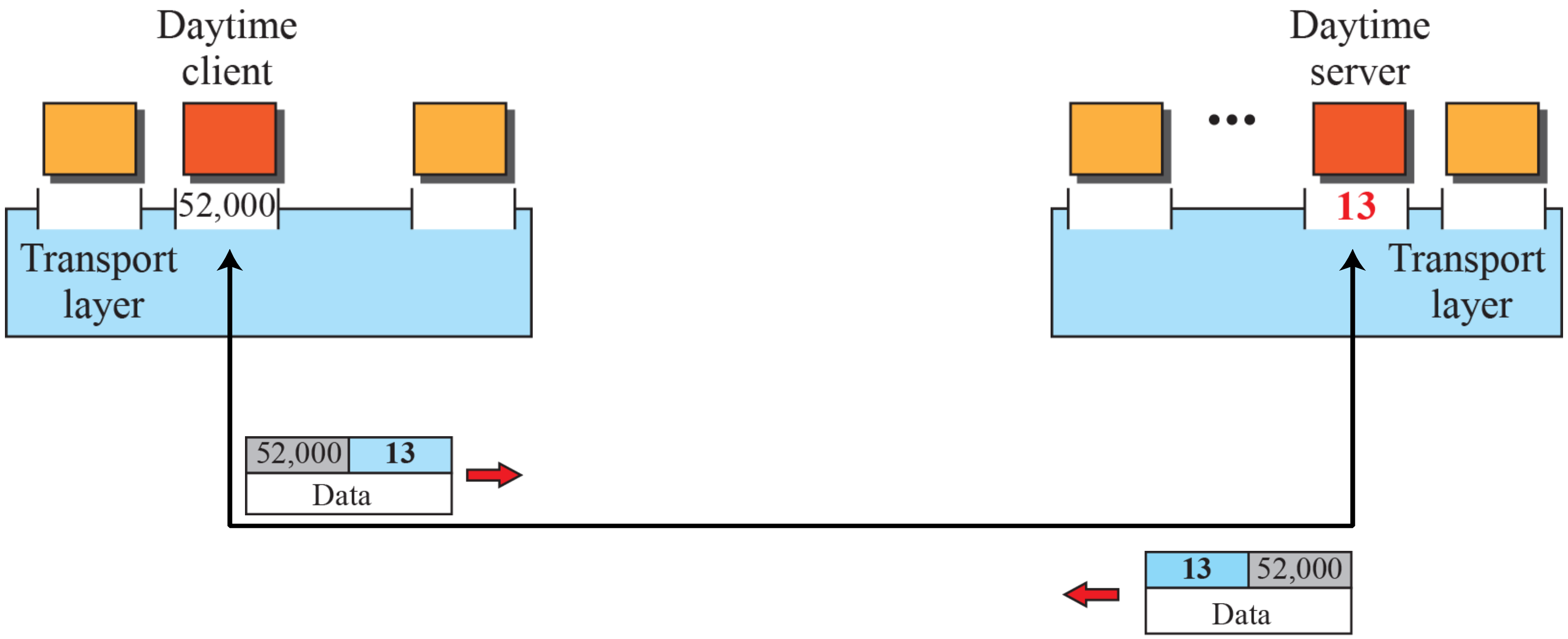
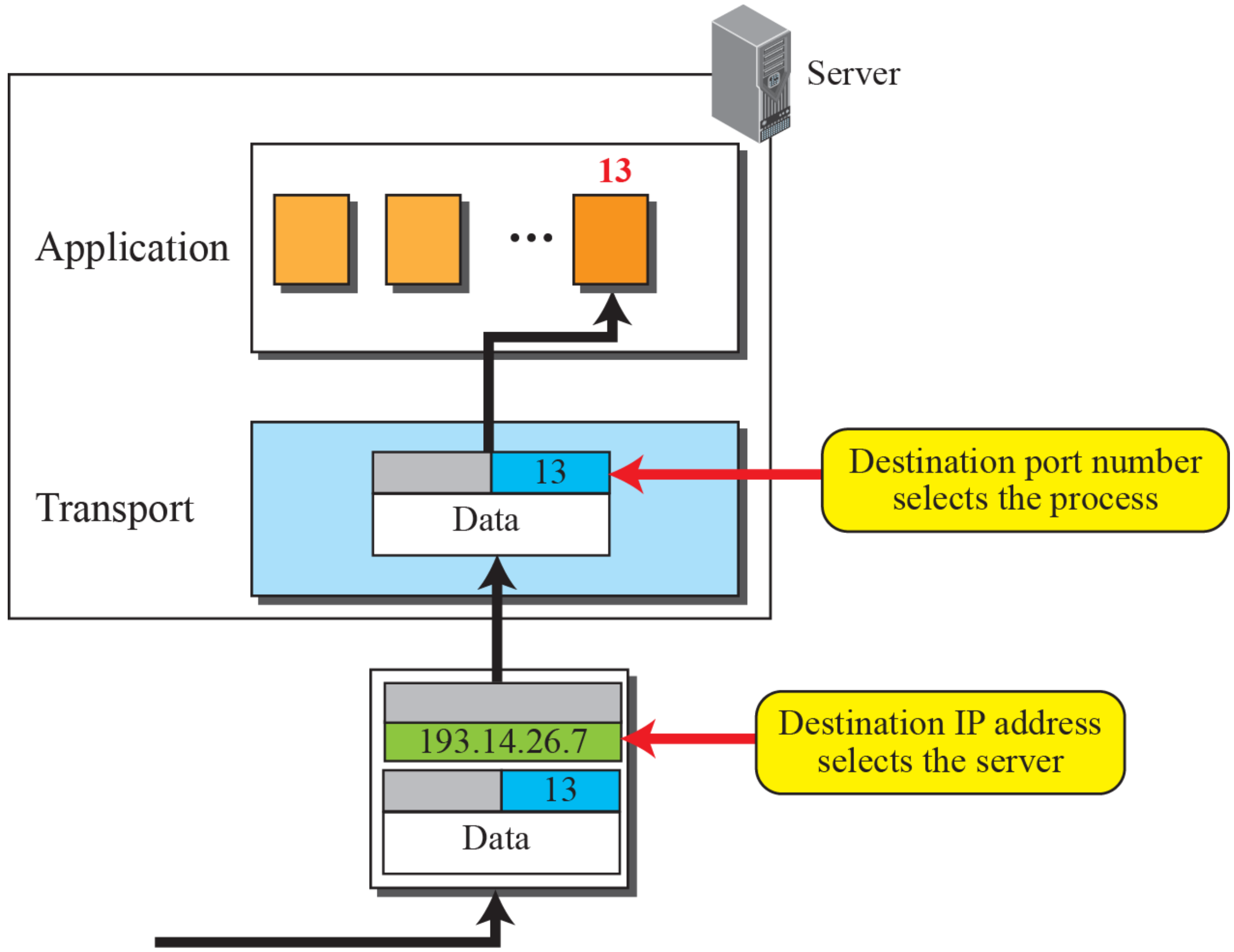


Figure 3.4: IP addresses versus port numbers


Server
Application
13
Transport
13
Data
Destination port number selects the process
193.14.26.7
13
Data
Destination IP address selects the server

❑ ICANN Ranges

- ❖ ***Well-known ports***
- ❖ ***Registered ports***
- ❖ ***Dynamic ports***

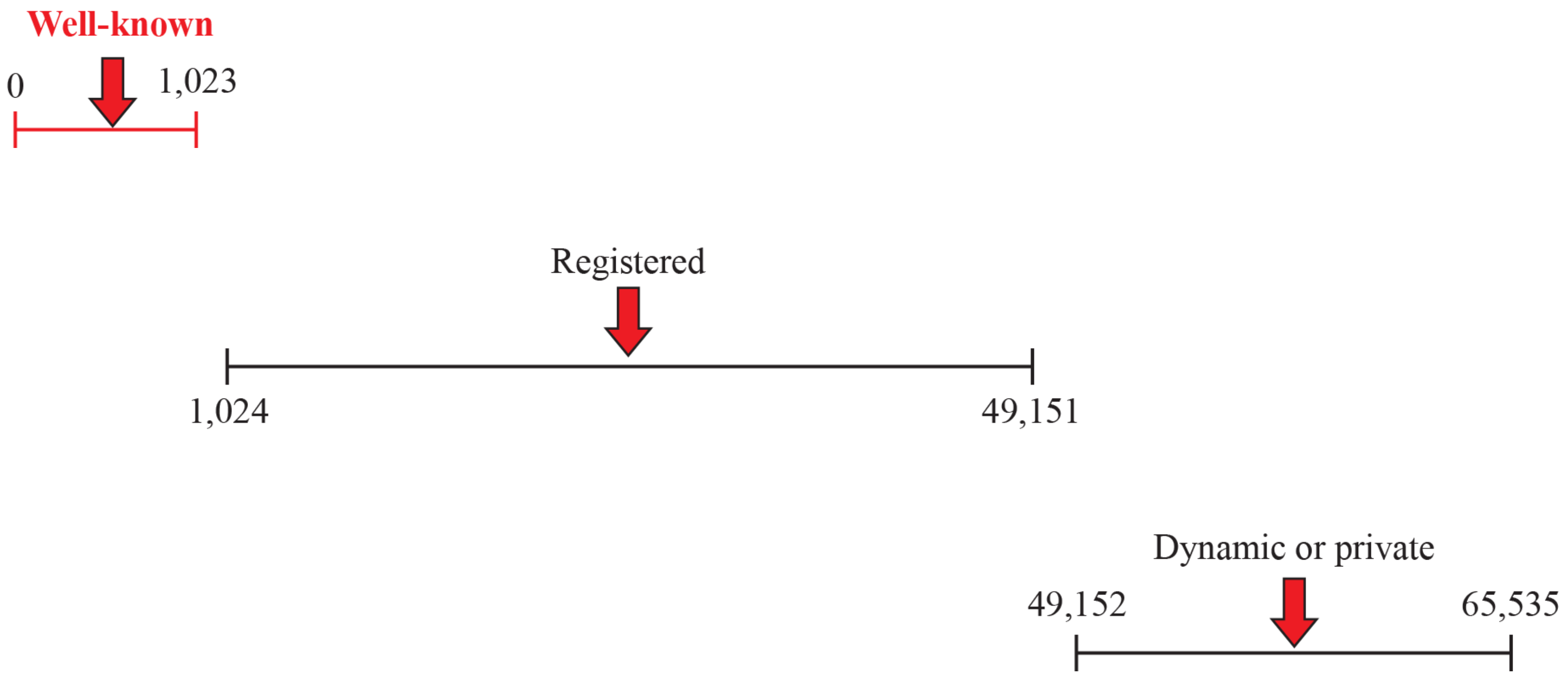
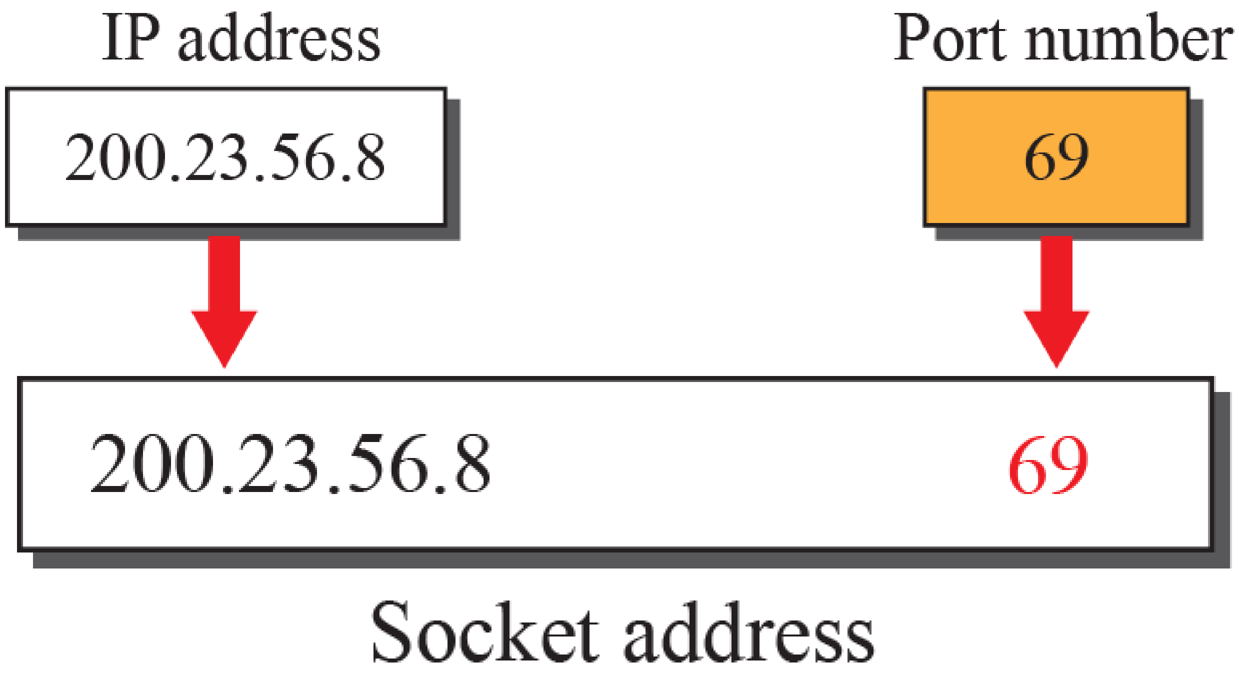
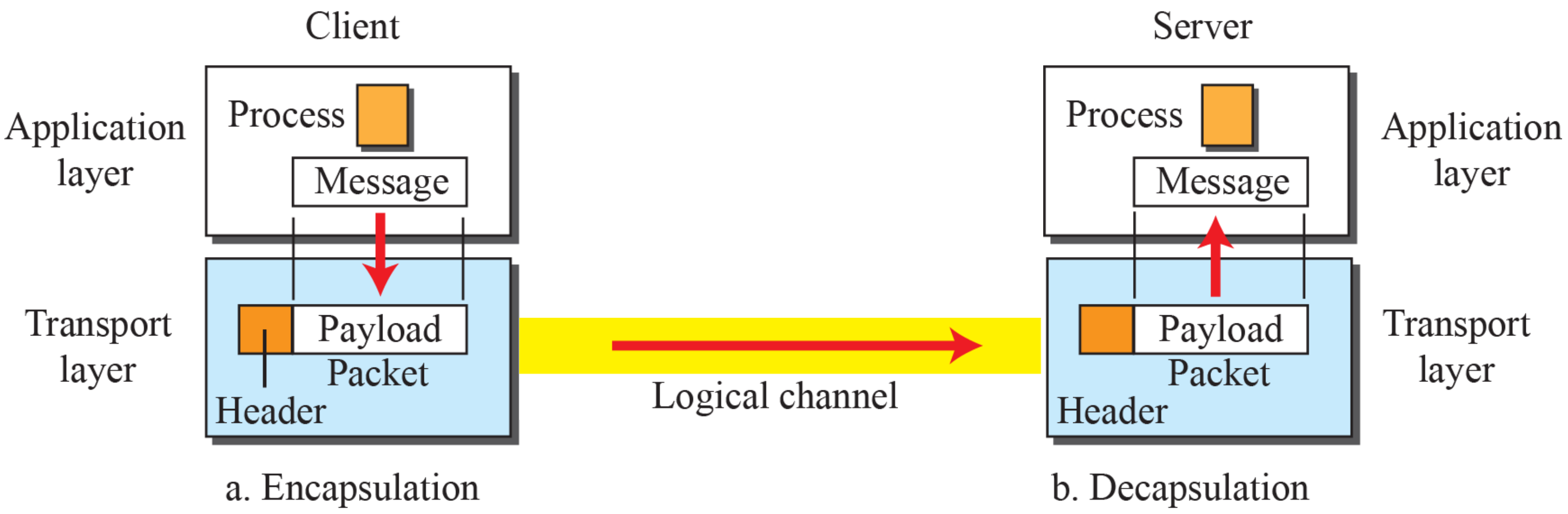


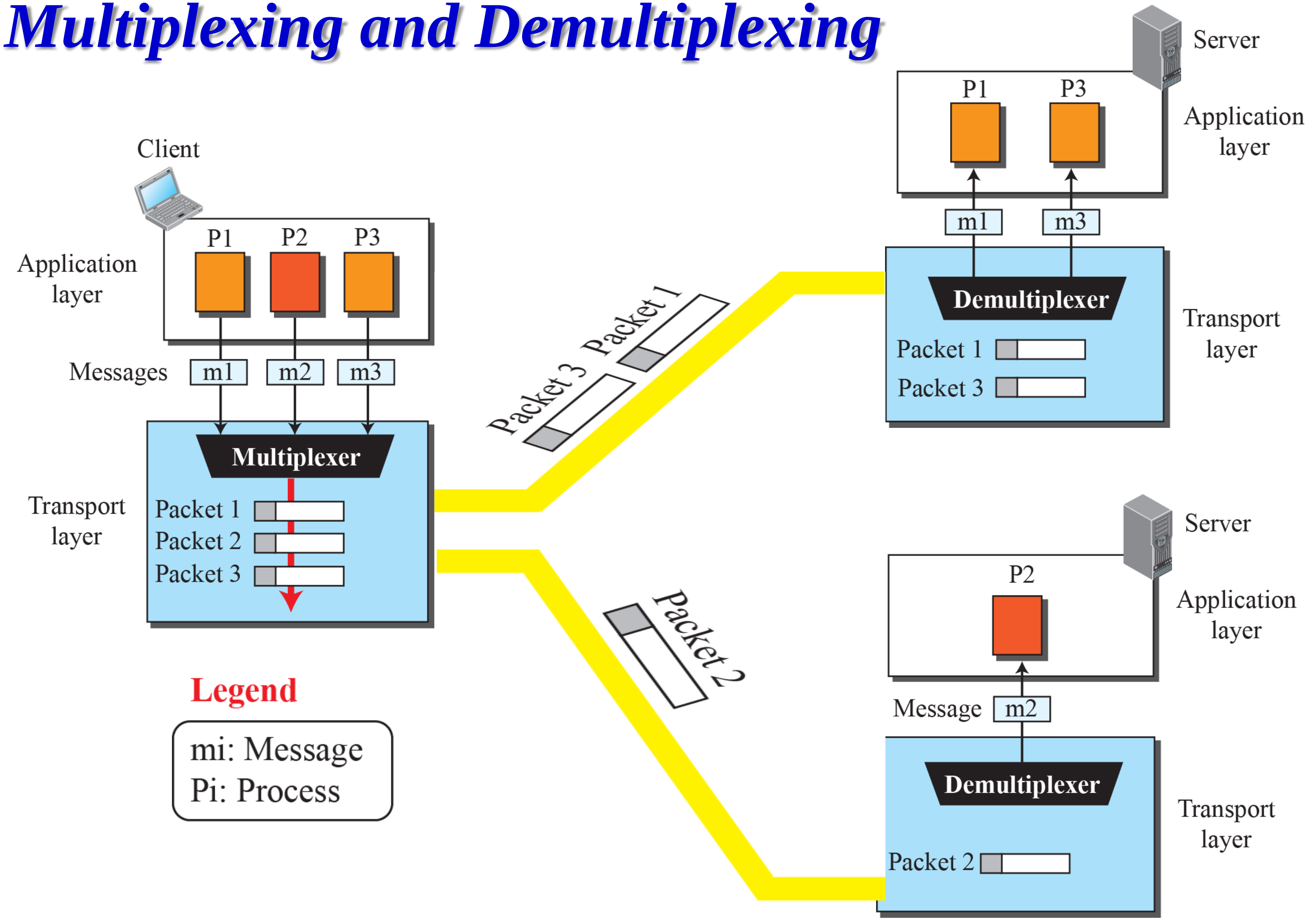
Figure 3.6: Socket address



❑ *Encapsulation and Decapsulation*



❑ *Multiplexing and Demultiplexing*



❑ Flow Control

Figure 3.9: *Pushing or pulling*

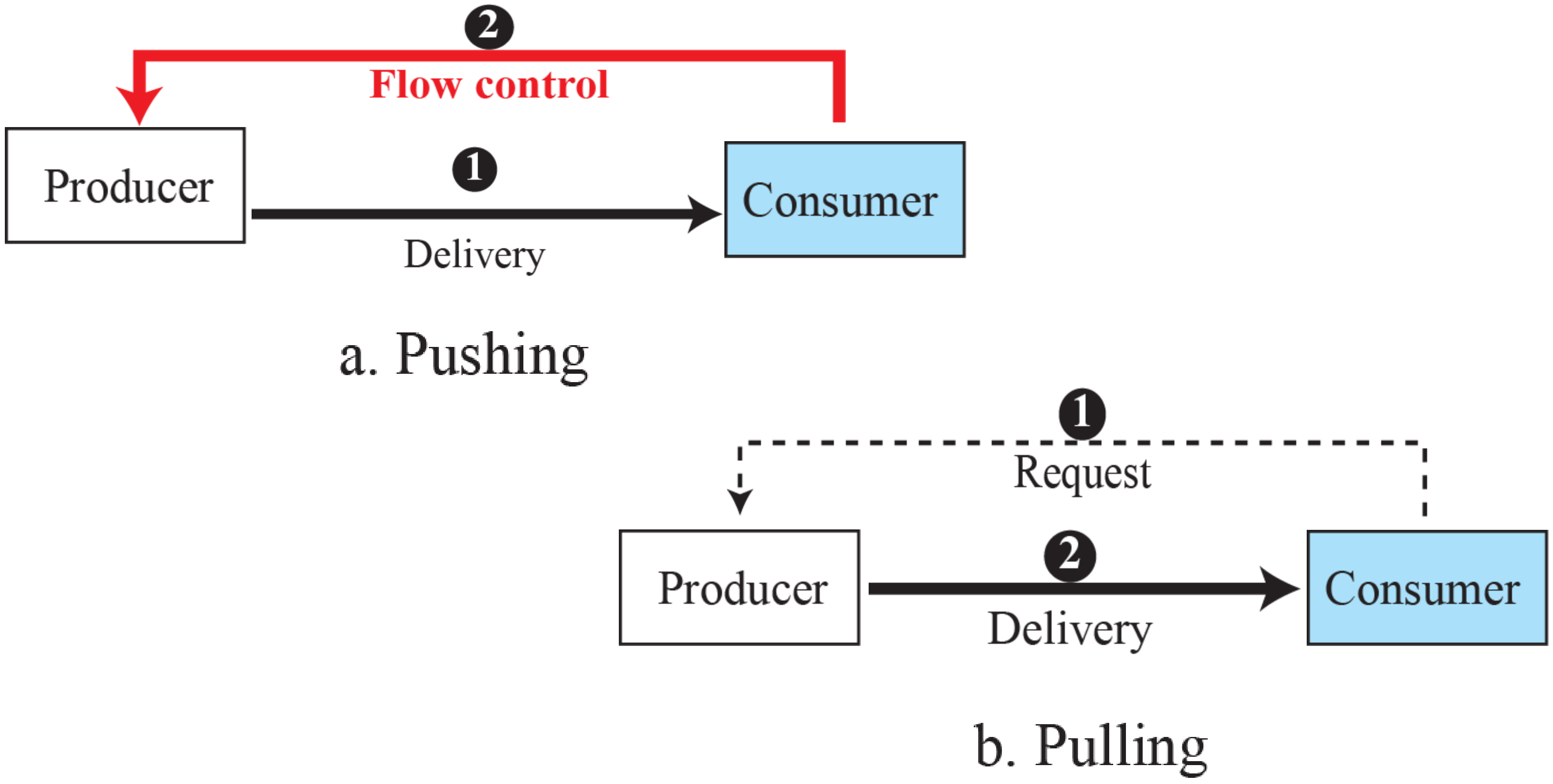
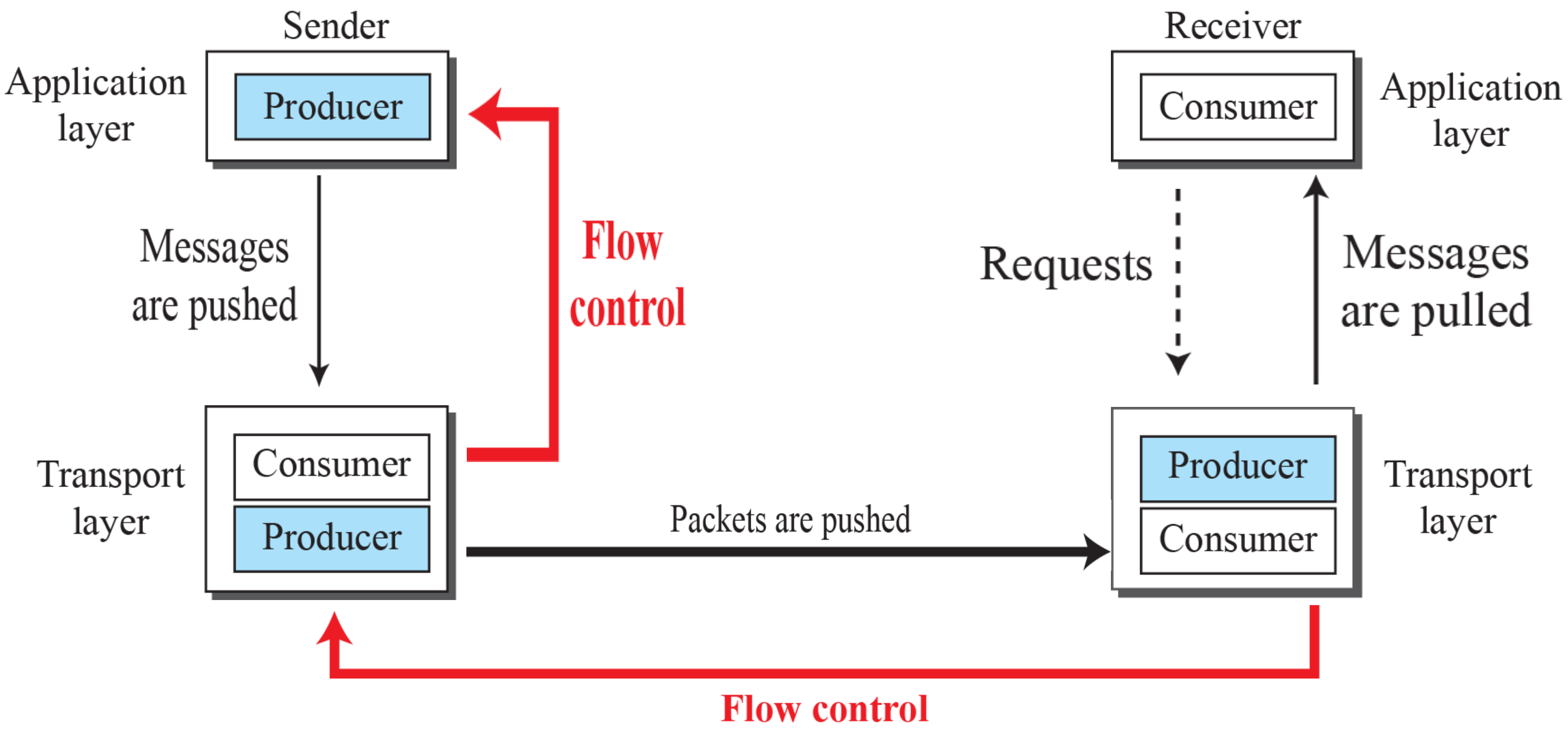


Figure 3.10: Flow control at the transport layer



❑ *Error Control*

- ❖ ***Sequence Numbers***
- ❖ ***Acknowledgment***

Figure 3.11: ***Error control at the transport layer***

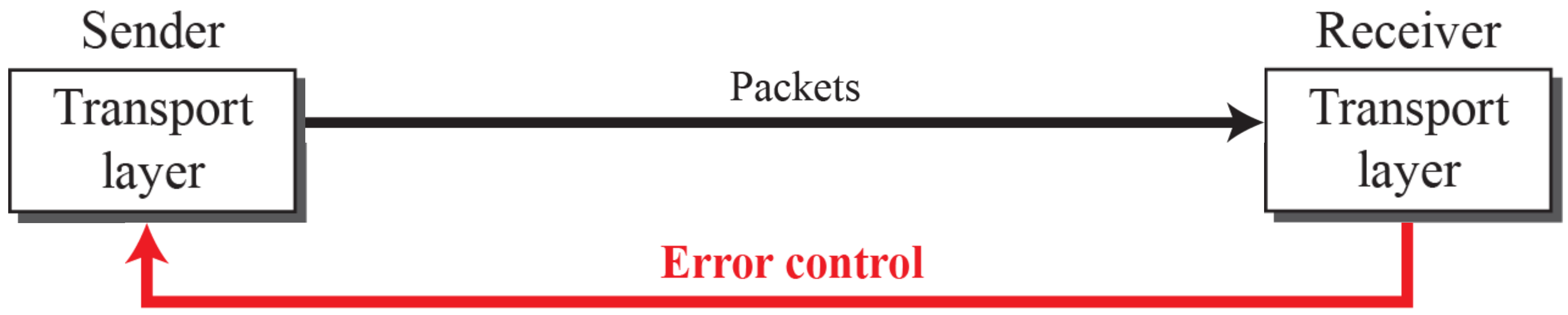


Figure 3.14: *Connectionless service*

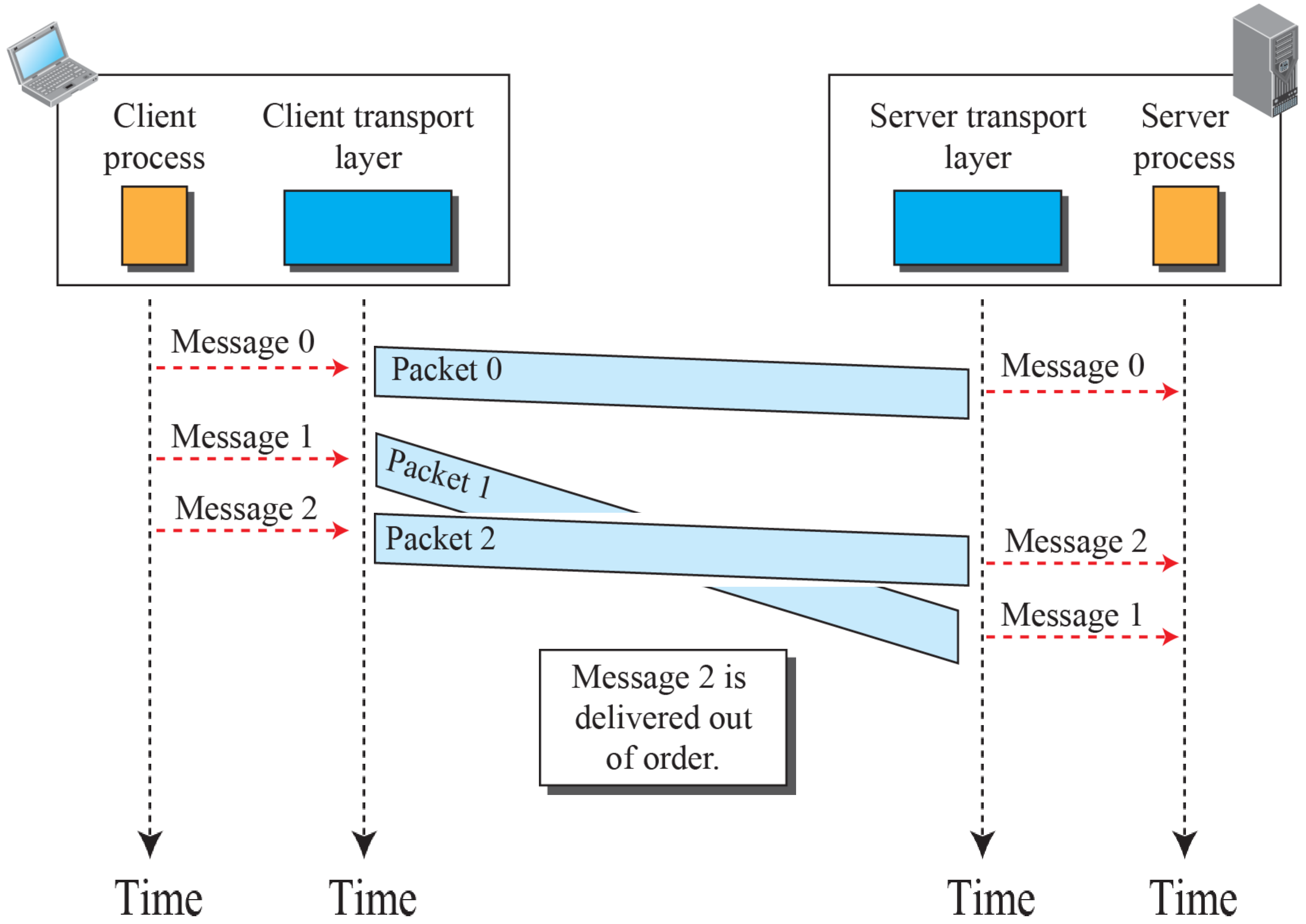
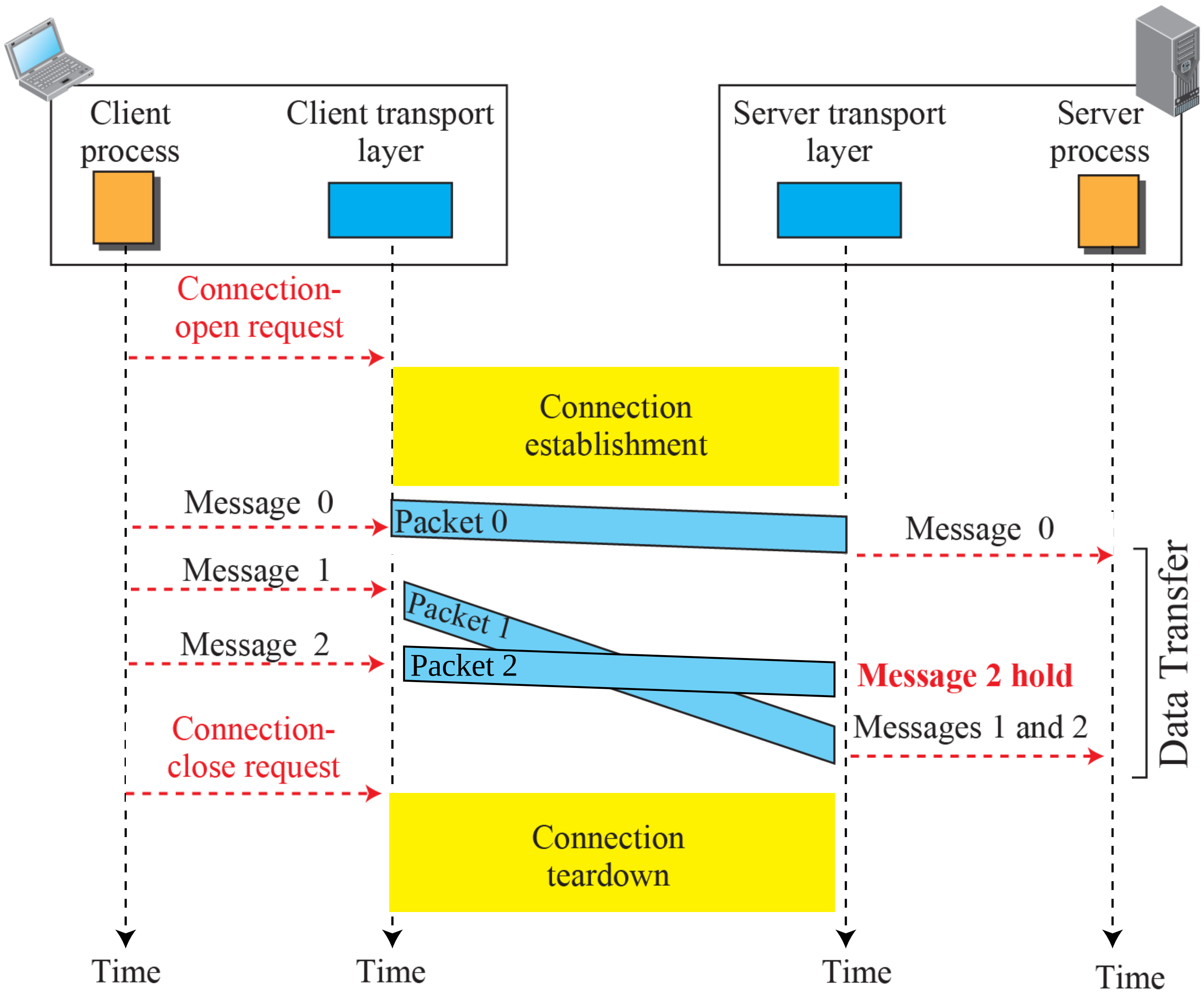


Figure 3.15: Connection-oriented service



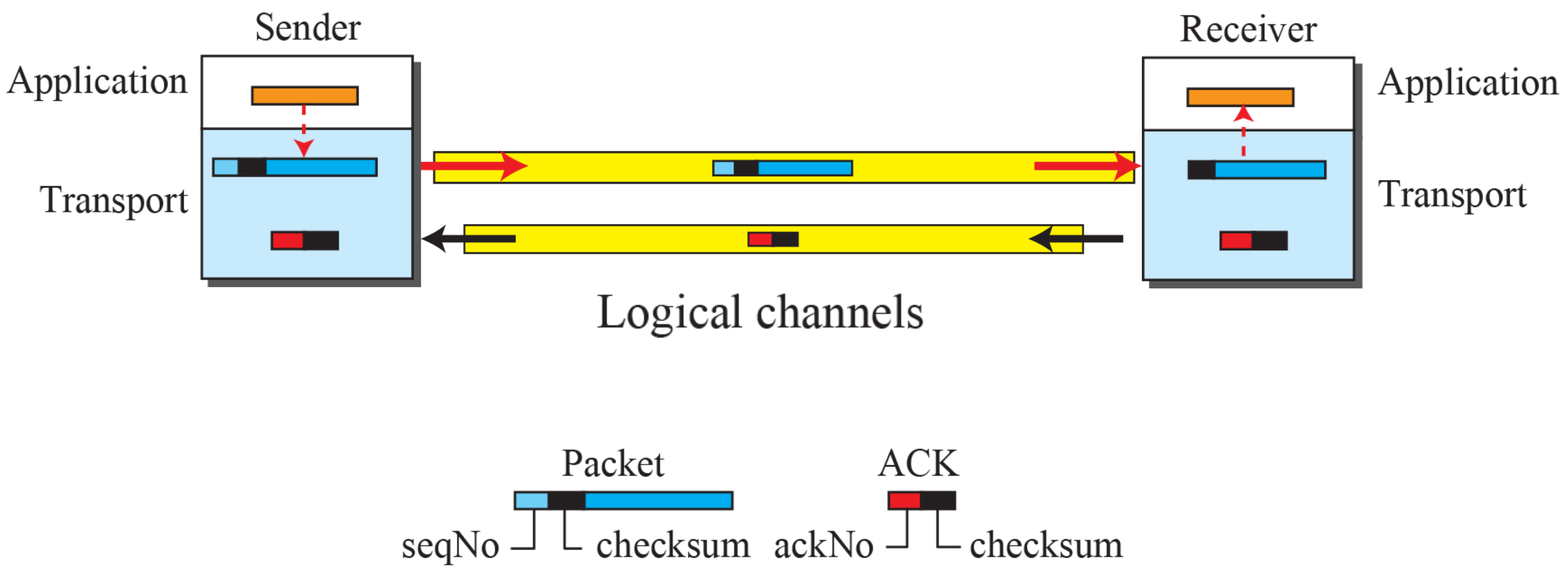
3.2.2 Stop-and-Wait Protocol

- ***uses both flow and error control***

- ***The sender sends one packet at a time and waits for an acknowledgment before sending the next one***

- ***To detect corrupted packets, we need to add a checksum to each data packet***

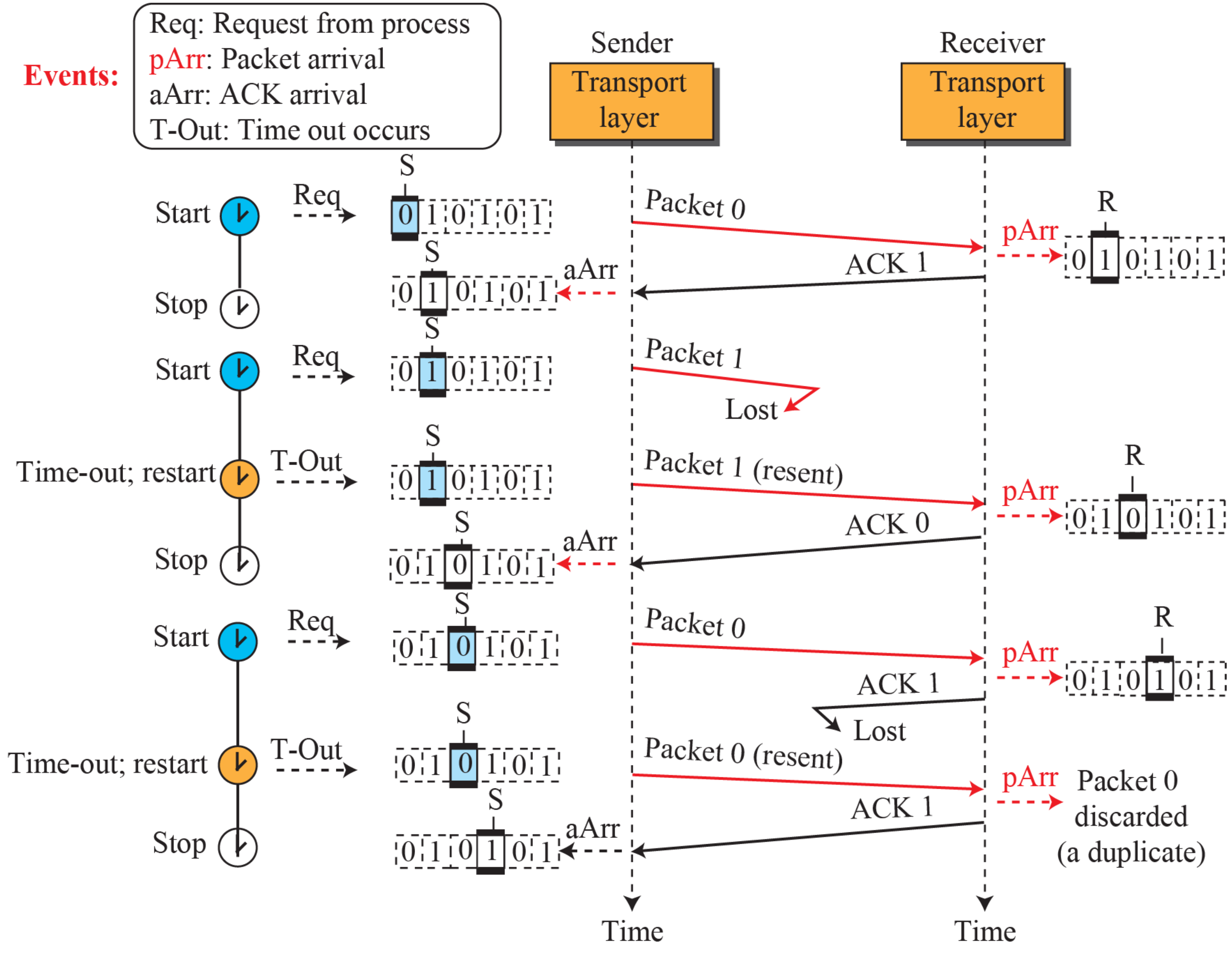
Figure 3.20: *Stop-and-Wait protocol*



Example 3.4

Figure 3.22 shows an example of the Stop-and-Wait protocol. Packet 0 is sent and acknowledged. Packet 1 is lost and resent after the time-out. The resent packet 1 is acknowledged and the timer stops. Packet 0 is sent and acknowledged, but the acknowledgment is lost. The sender has no idea if the packet or the acknowledgment is lost, so after the time-out, it resends packet 0, which is acknowledged.

Figure 3.22: Flow diagram for Example 3.4



Example 3.5

Assume that, in a Stop-and-Wait system, the bandwidth of the line is 1 Mbps, and 1 bit takes 20 milliseconds to make a round trip. What is the bandwidth-delay product? If the system data packets are 1,000 bits in length, what is the utilization percentage of the link?

Solution

The bandwidth-delay product is $(1 \times 10^6) \times (20 \times 10^{-3}) =$ 20,000 bits. The system can send 20,000 bits during the time it takes for the data to go from the sender to the receiver and the acknowledgment to come back. However, the system sends only 1,000 bits. The link utilization is only 1,000/20,000, or 5 percent.

Example 3.6

What is the utilization percentage of the link in Example 3.5 if we have a protocol that can send up to 15 packets before stopping and worrying about the acknowledgments?

Solution

The bandwidth-delay product is still 20,000 bits. The system can send up to 15 packets or 15,000 bits during a round trip. This means the utilization is 15,000/20,000, or 75 percent. Of course, if there are damaged packets, the utilization percentage is much less because packets have to be resent.

3.2.3 Go-Back-N Protocol

- ***To improve the efficiency of transmission multiple packets must be in transition while the sender is waiting for acknowledgment.***

- ***Go-Back-N (GBN)***

Figure 3.23: Go-Back-N protocol

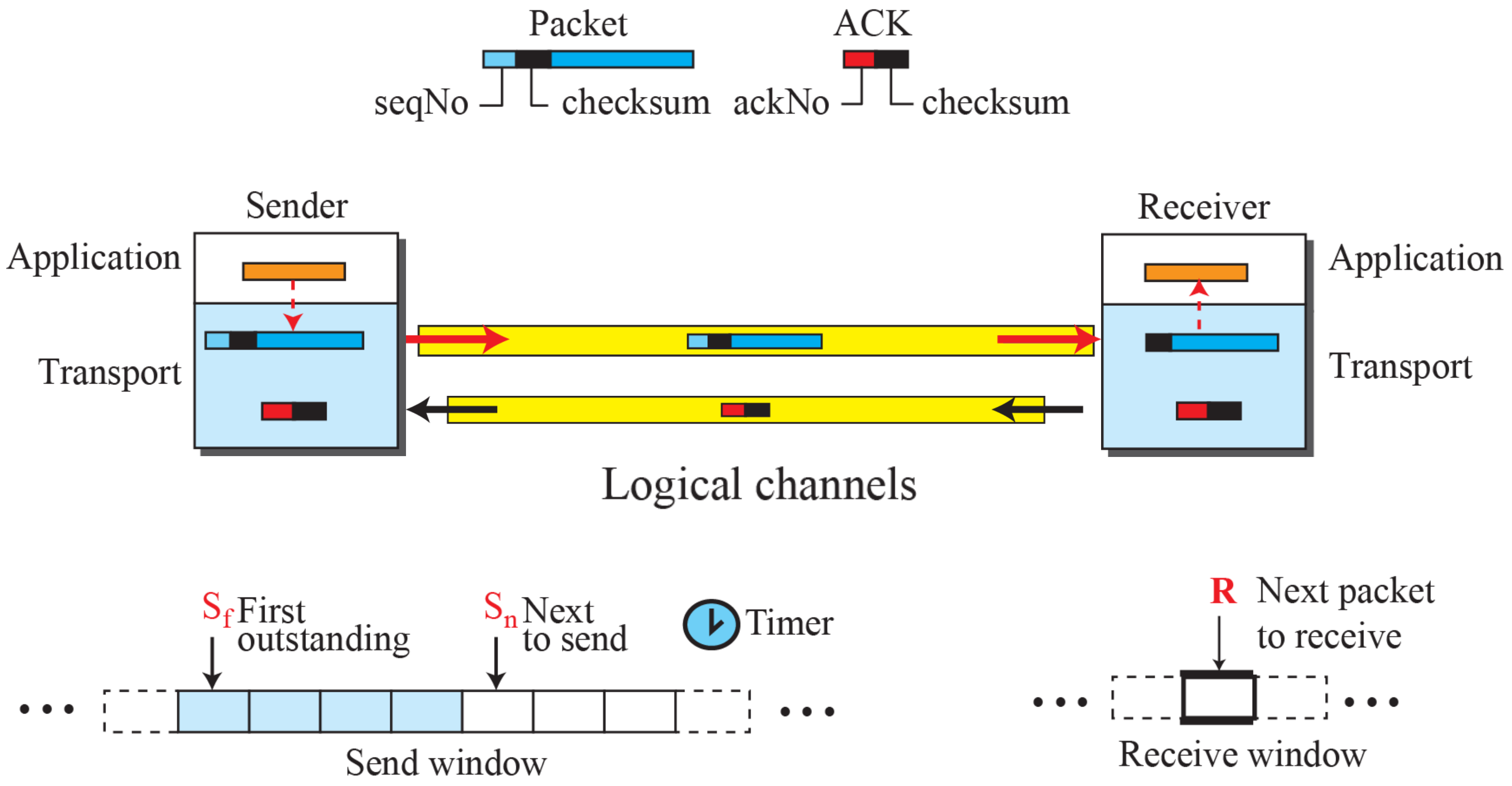
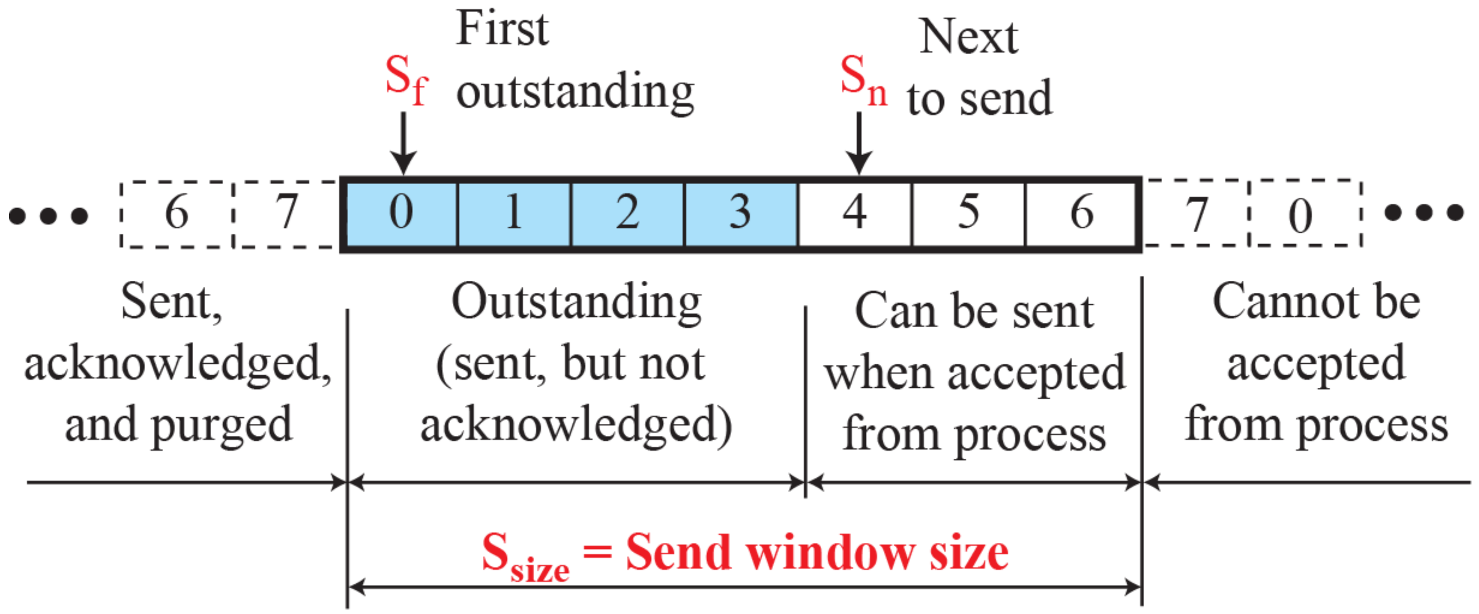
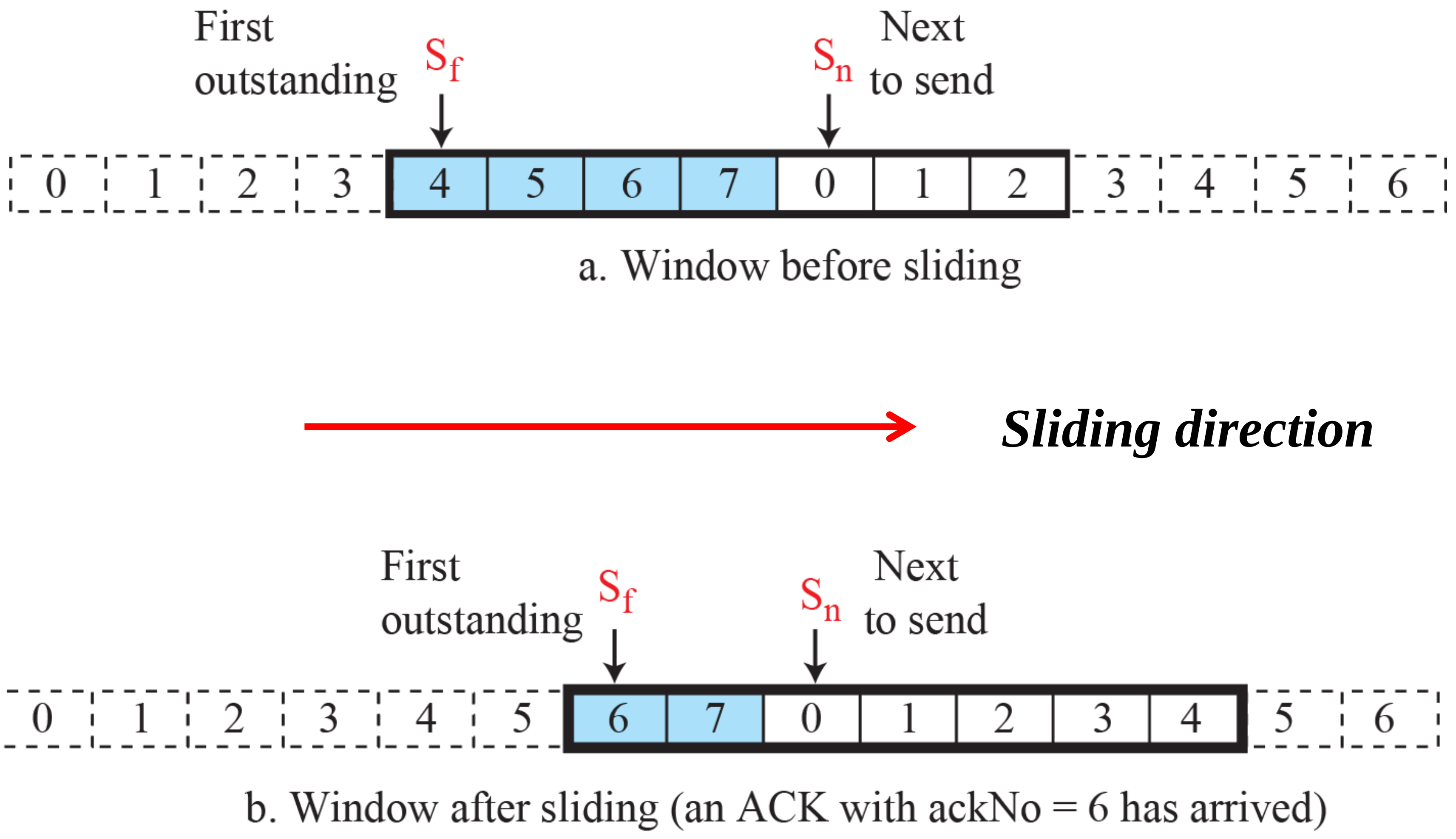


Figure 3.24: *Send window for Go-Back-N*



S_f First outstanding
S_n Next to send
• • • 6 7 0 1 2 3 4 5 6 7 0 • • •
Sent, acknowledged, and purged
Outstanding (sent, but not acknowledged)
Can be sent when accepted from process
Cannot be accepted from process
S_{size} = Send window size

Figure 3.25: Sliding the send window



a. Window before sliding

b. Window after sliding (an ACK with ackNo = 6 has arrived)

Figure 3.26: Receive window for Go-Back-N

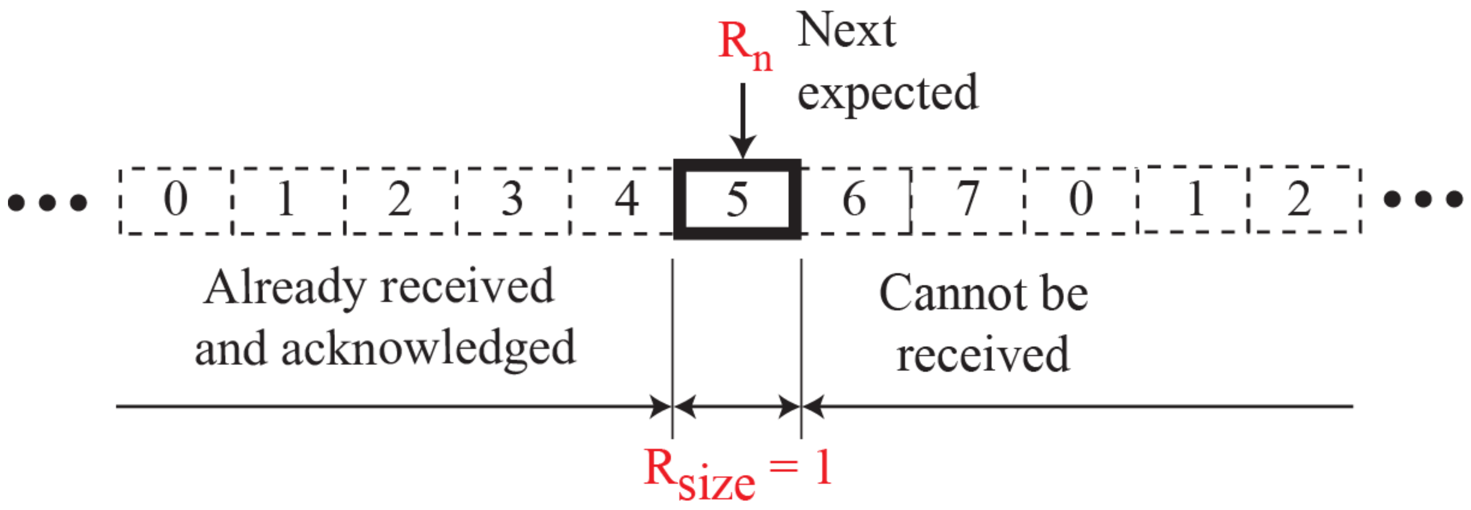
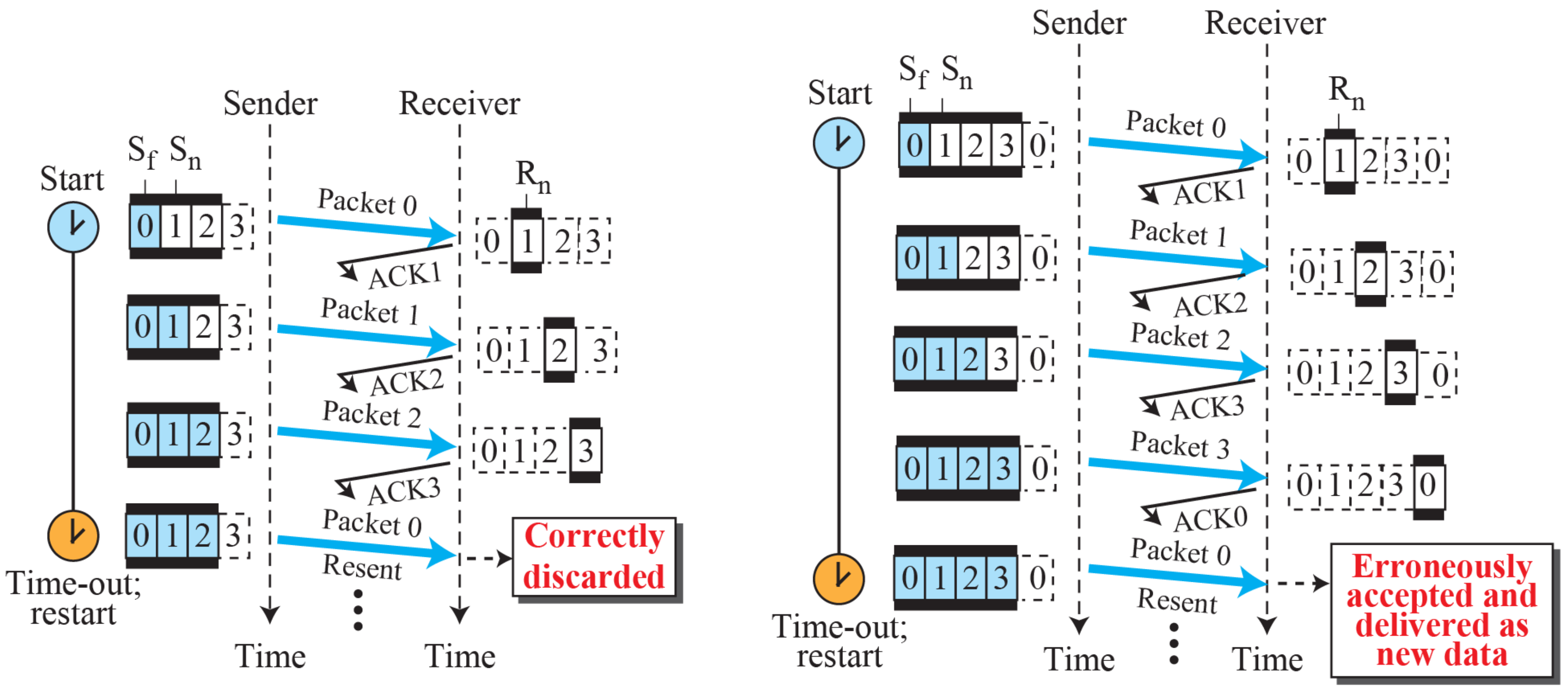


Figure 3.28: Send window size for Go-Back-N



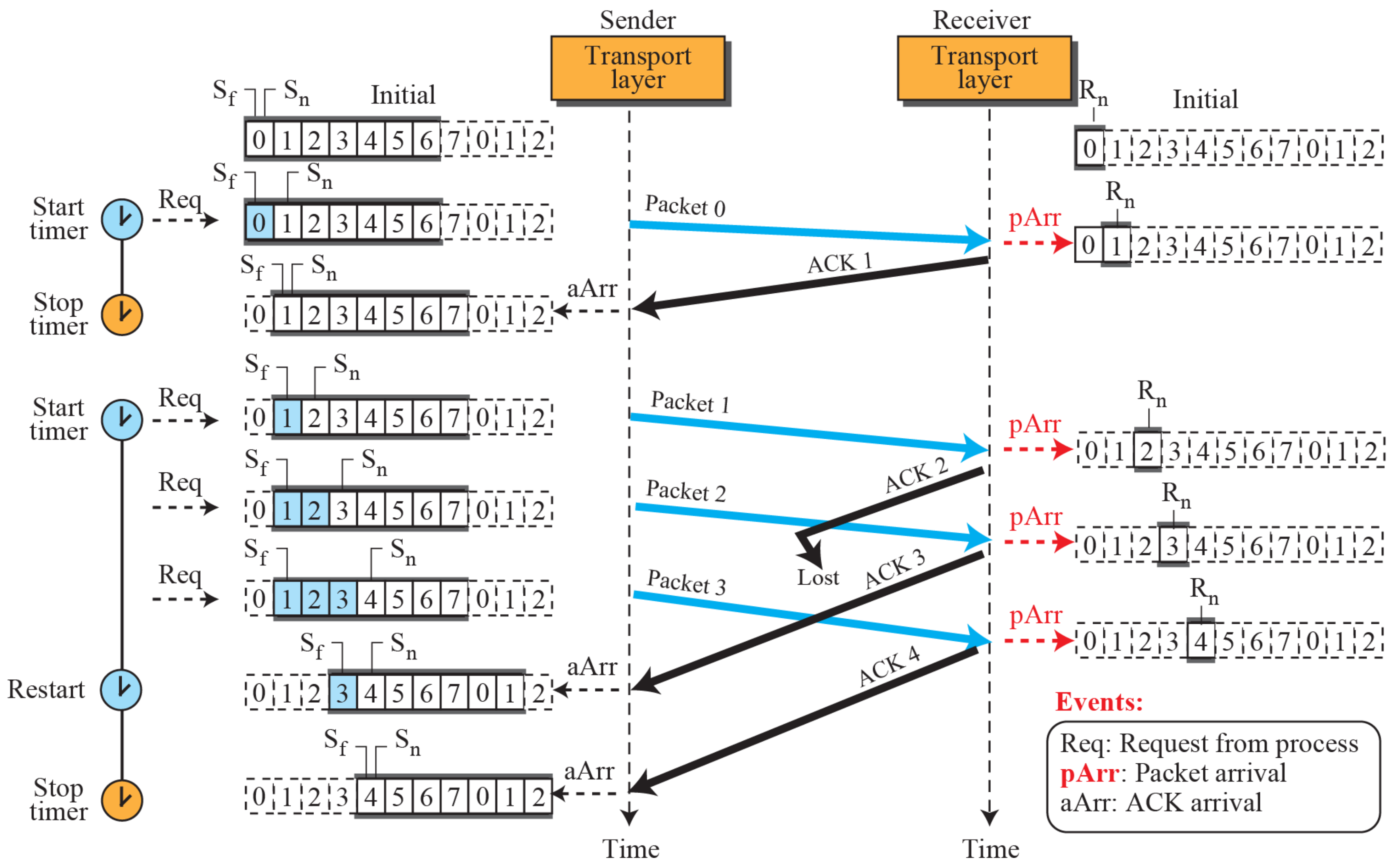
a. Send window of size $< 2^m$

b. Send window of size $= 2^m$

Example 3.7

Figure 3.29 shows an example of Go-Back-*N*. This is an example of a case where the forward channel is reliable, but the reverse is not. No data packets are lost, but some ACKs are delayed and one is lost. The example also shows how cumulative acknowledgments can help if acknowledgments are delayed or lost.

Figure 3.29: *Flow diagram for Example 3.7*



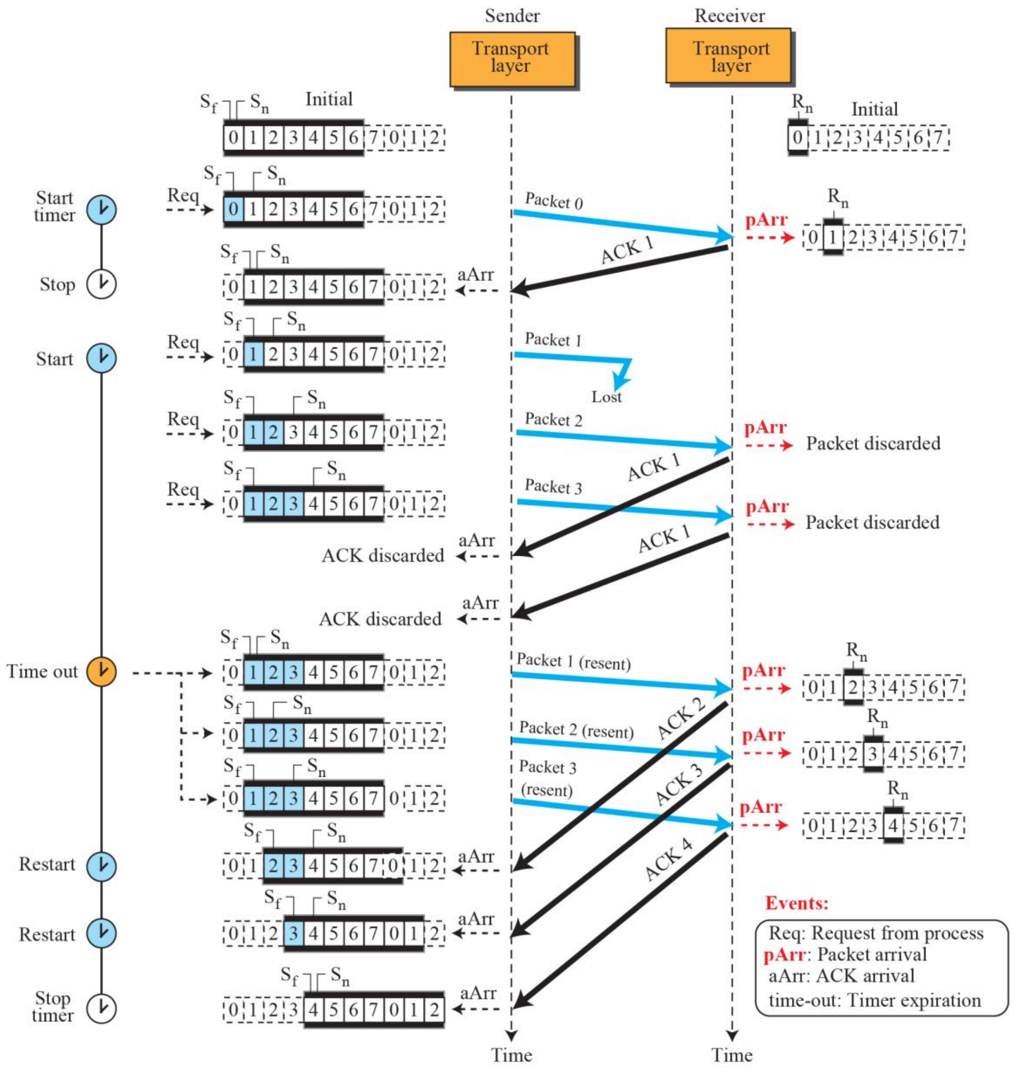
Events:

Req: Request from process
pArr: Packet arrival
aArr: ACK arrival

Example 3.8

Figure 3.30 shows what happens when a packet is lost. Packets 0, 1, 2, and 3 are sent. However, packet 1 is lost. The receiver receives packets 2 and 3, but they are discarded because they are received out of order (packet 1 is expected). When the receiver receives packets 2 and 3, it sends ACK1 to show that it expects to receive packet 1. However, these ACKs are not useful for the sender because the ackNo is equal to S_f, not greater that S_f . So the sender discards them. When the time-out occurs, the sender resends packets 1, 2, and 3, which are acknowledged.

Figure 3.30: *Flow diagram for Example 3.8*



3.2.4 Selective-Repeat Protocol

- resends only selective packets, those that are actually lost

Figure 3.31: Outline of Selective-Repeat

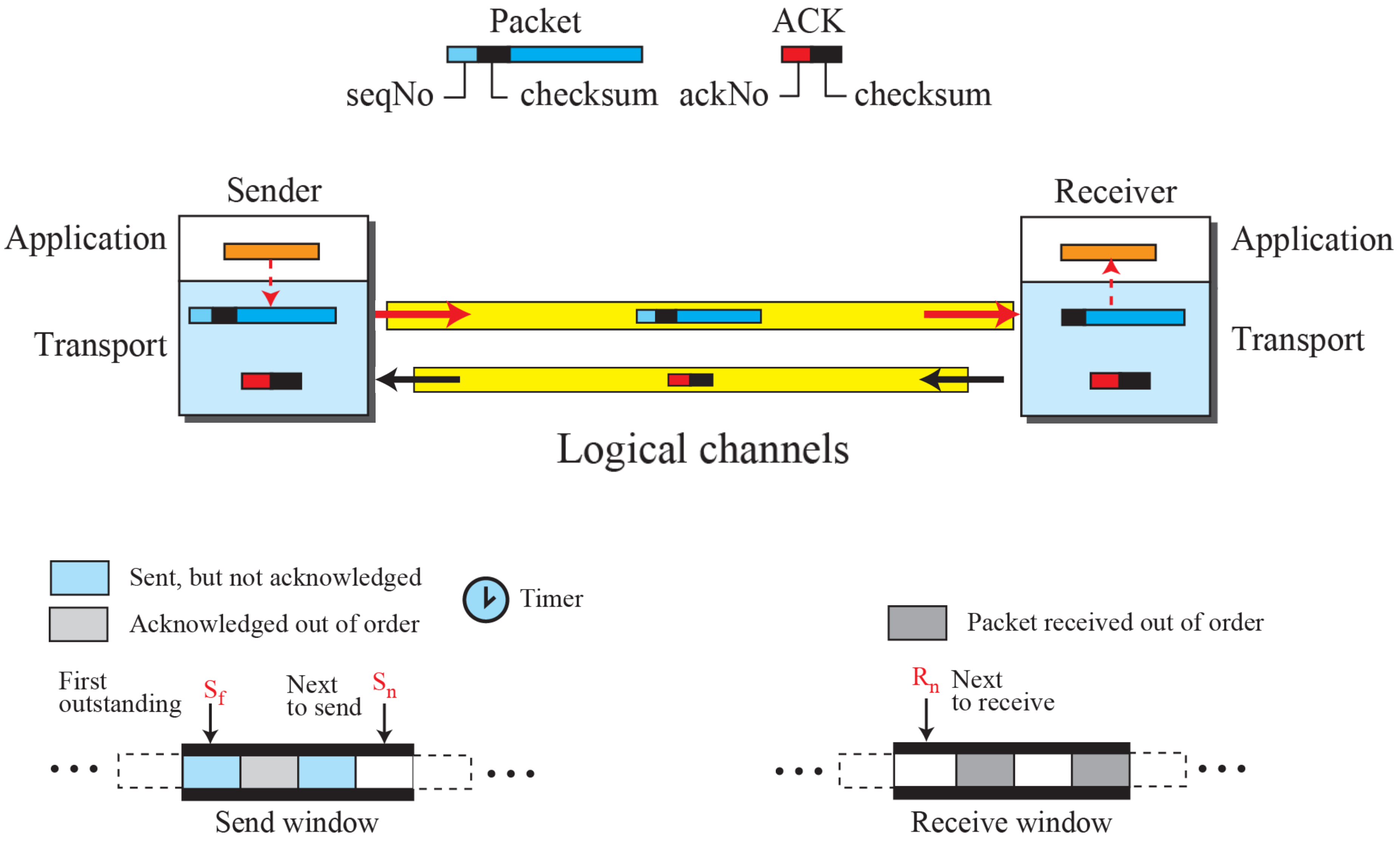
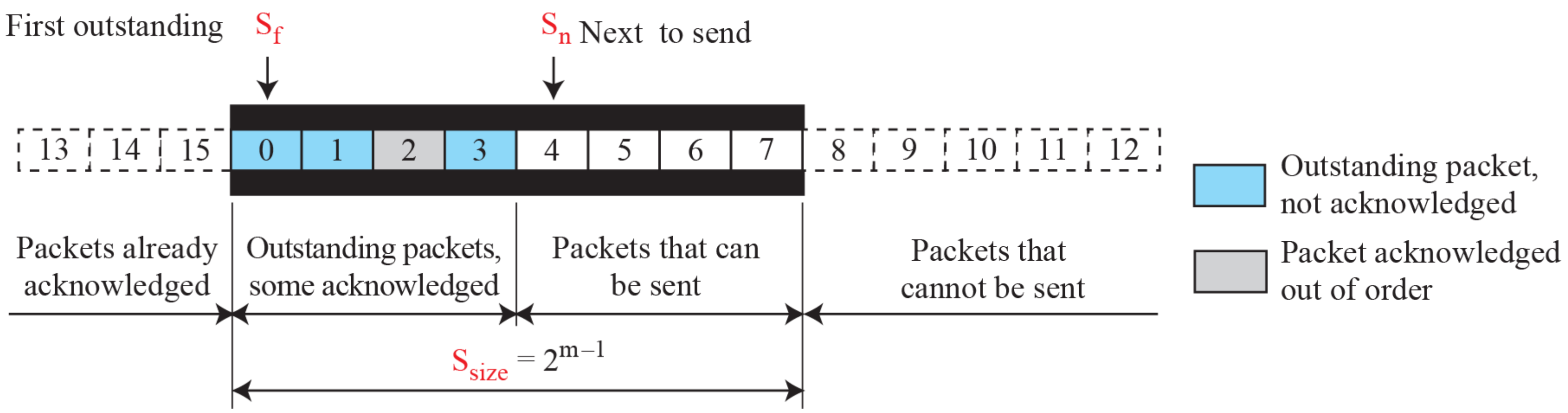
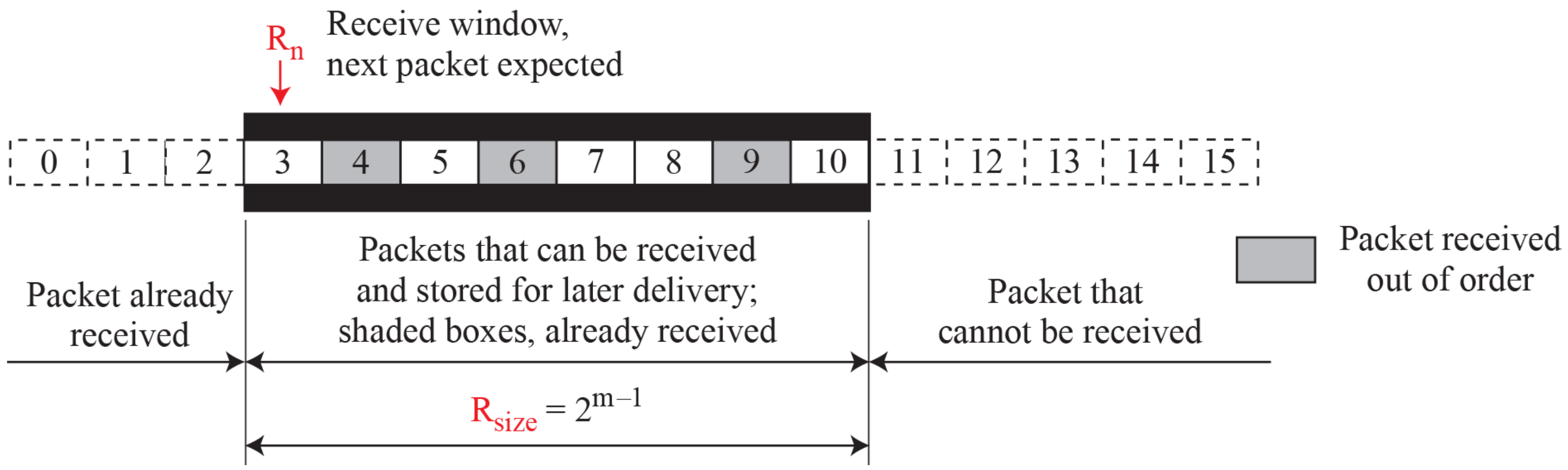


Figure 3.32: Send window for Selective-Repeat protocol



First outstanding
S_f
S_n Next to send
13
14
15
0
1
2
3
4
5
6
7
8
9
10
11
12
Packets already acknowledged
Outstanding packets, some acknowledged
Packets that can be sent
Packets that cannot be sent
$S_{size} = 2^{m-1}$
Outstanding packet, not acknowledged
Packet acknowledged out of order

Figure 3.33: Receive window for Selective-Repeat protocol


R_n
Receive window, next packet expected
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15
Packet already received
Packets that can be received and stored for later delivery; shaded boxes, already received
Packet that cannot be received
Packet received out of order
$R_{size} = 2^{m-1}$

Example 3.9

Assume a sender sends 6 packets: packets 0, 1, 2, 3, 4, and 5. The sender receives an ACK with ackNo = 3. What is the interpretation if the system is using GBN or SR?

Solution

If the system is using GBN, it means that packets 0, 1, and 2 have been received uncorrupted and the receiver is expecting packet 3. If the system is using SR, it means that packet 3 has been received uncorrupted; the ACK does not say anything about other packets.

Example 3.10

This example is similar to Example 3.8 (Figure 3.30) in which packet 1 is lost. We show how Selective-Repeat behaves in this case. Figure 3.35 shows the situation.

- At the sender, packet 0 is transmitted and acknowledged. Packet 1 is lost.
- Packets 2 and 3 arrive out of order and are acknowledged.
- When the timer times out, packet 1 (the only unacknowledged packet) is resent and is acknowledged.
- The send window then slides.

Example 3.10 (continued)

At the second arrival,

- packet 2 arrives and is stored and marked (shaded slot), but it cannot be delivered because packet 1 is missing.
- At the next arrival, packet 3 arrives and is marked and stored, but still none of the packets can be delivered.
- Only at the last arrival, when finally a copy of packet 1 arrives, can packets 1, 2, and 3 be delivered to the application layer.
- There are two conditions for the delivery of packets to the application layer: First, a set of consecutive packets must have arrived. Second, the set starts from the beginning of the window.

Figure 3.35: Flow diagram for Example 3.10

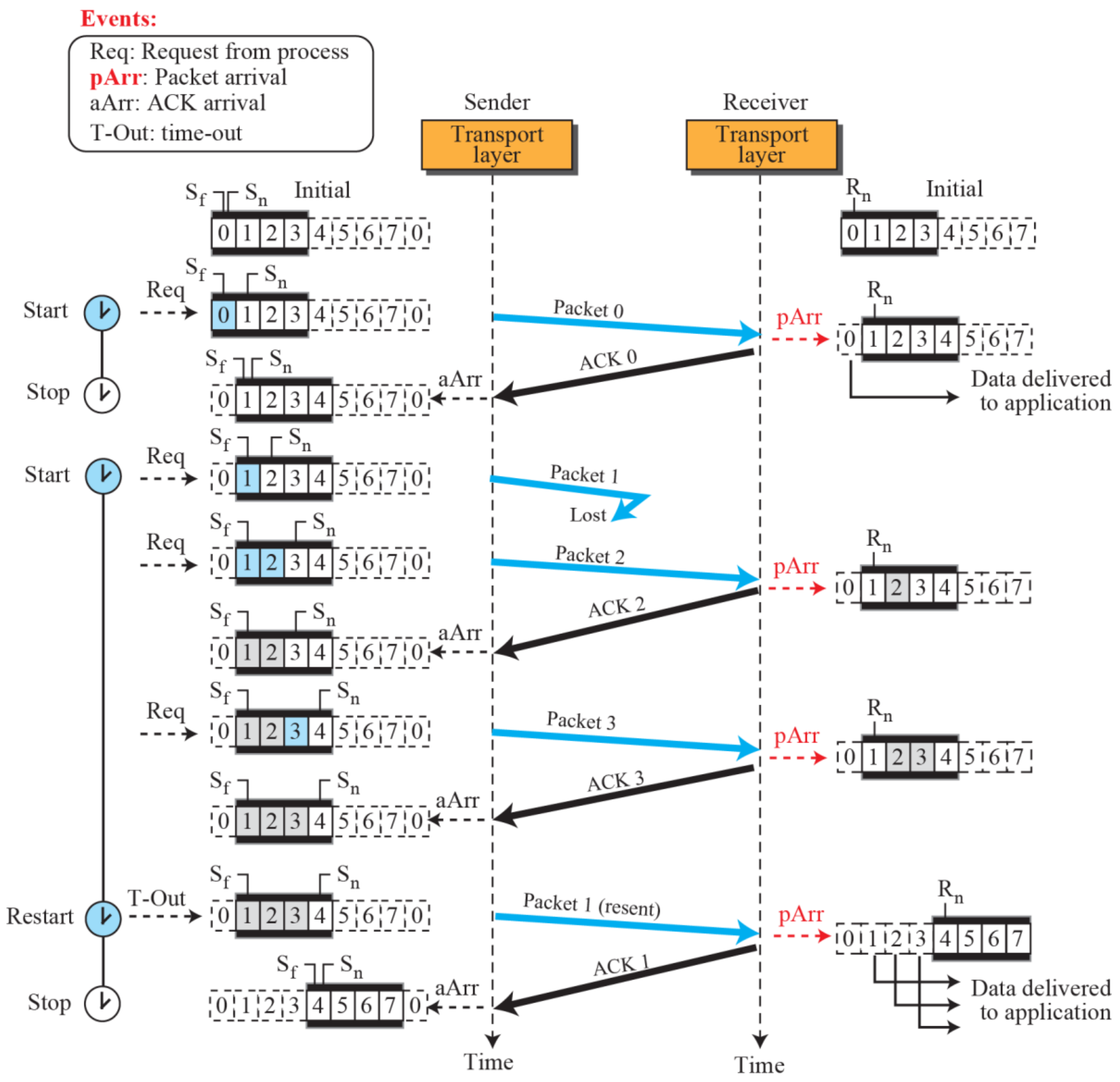
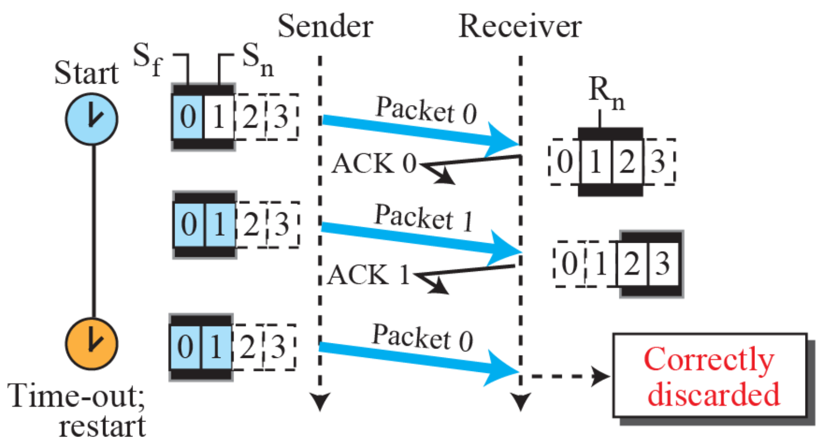
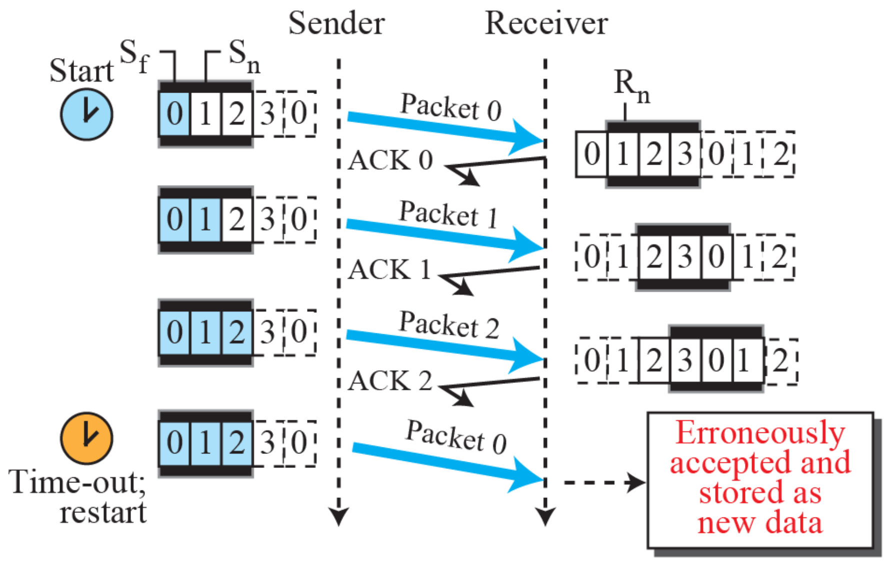


Figure 3.36: *Selective-Repeat, window size*



a. Send and receive windows of size = 2^{m-1}



b. Send and receive windows of size > 2^{m-1}

Figure 3.38: *Position of transport-layer protocols in the TCP/IP protocol suite*

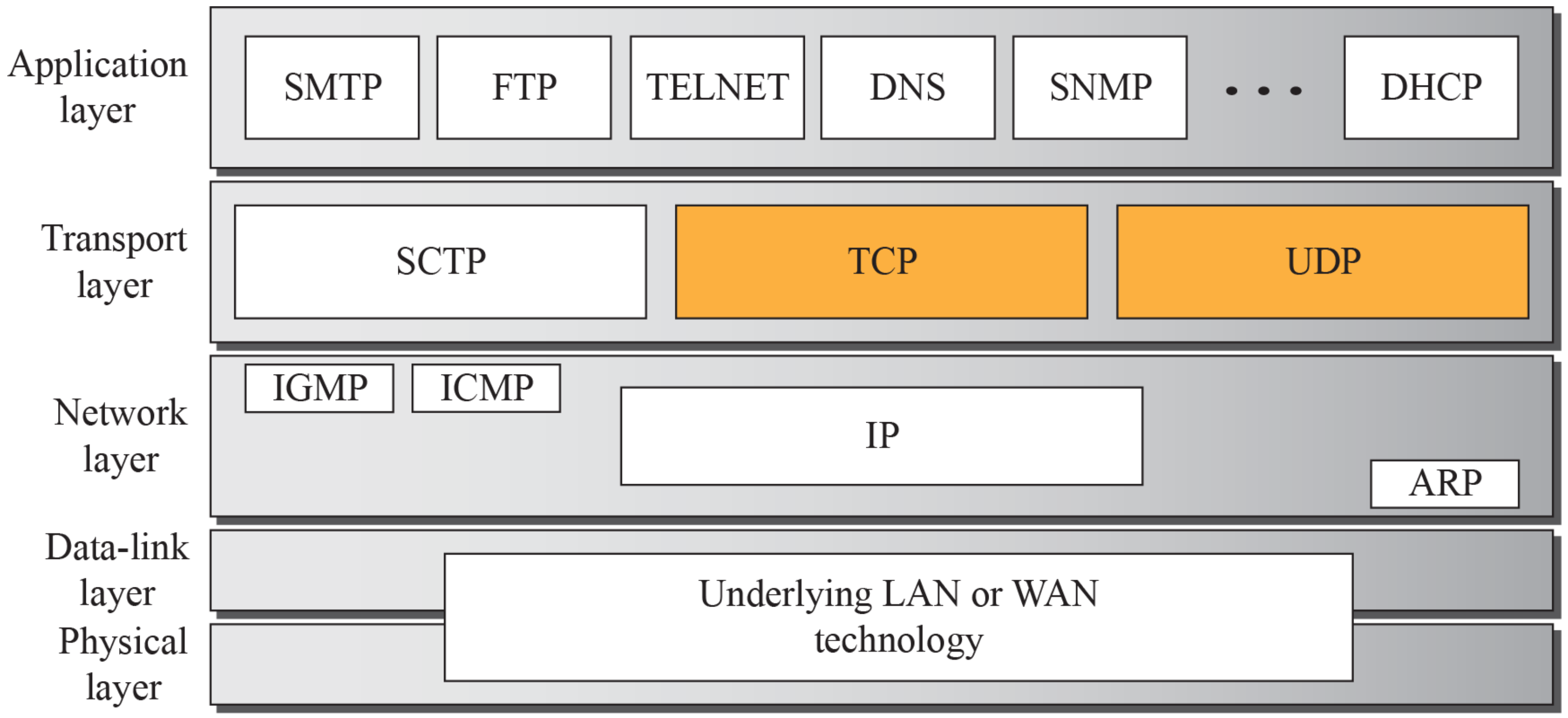


Table 3.1: Some well-known ports used with UDP and TCP

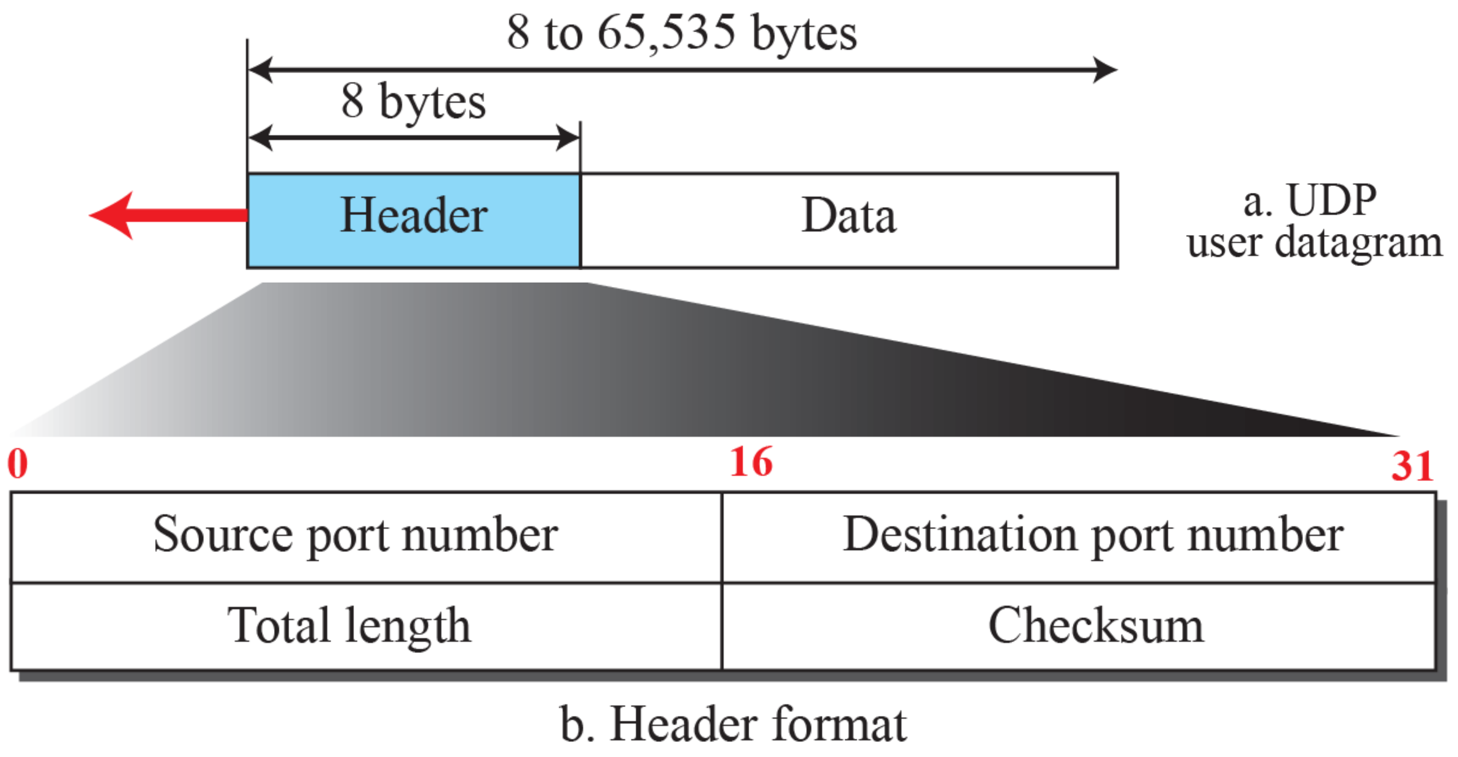
Port	*Protocol*	*UDP*	*TCP*	*Description*
7	Echo	√		Echoes back a received datagram
9	Discard	√		Discards any datagram that is received
11	Users	√	√	Active users
13	Daytime	√	√	Returns the date and the time
17	Quote	√	√	Returns a quote of the day
19	Chargen	√	√	Returns a string of characters
20, 21	FTP		√	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, 162	SNMP		√	Simple Network Management Protocol

3-3 USER DATAGRAM PROTOCOL (UDP)

- is a connectionless, unreliable transport protocol

- UDP is a very simple protocol using a minimum of overhead

Figure 3.39: *User datagram packet format*



Example 3.11

The following is the contents of a UDP header in hexadecimal format.

CB84000D001C001C

a. What is the source port number?
b. What is the destination port number?
c. What is the total length of the user datagram?
d. What is the length of the data?
e. Is the packet directed from a client to a server or vice versa?
f. What is the client process?

3-4 TRANSMISSION CONTROL PROTOCOL

- 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.

Figure 3.41: Stream delivery

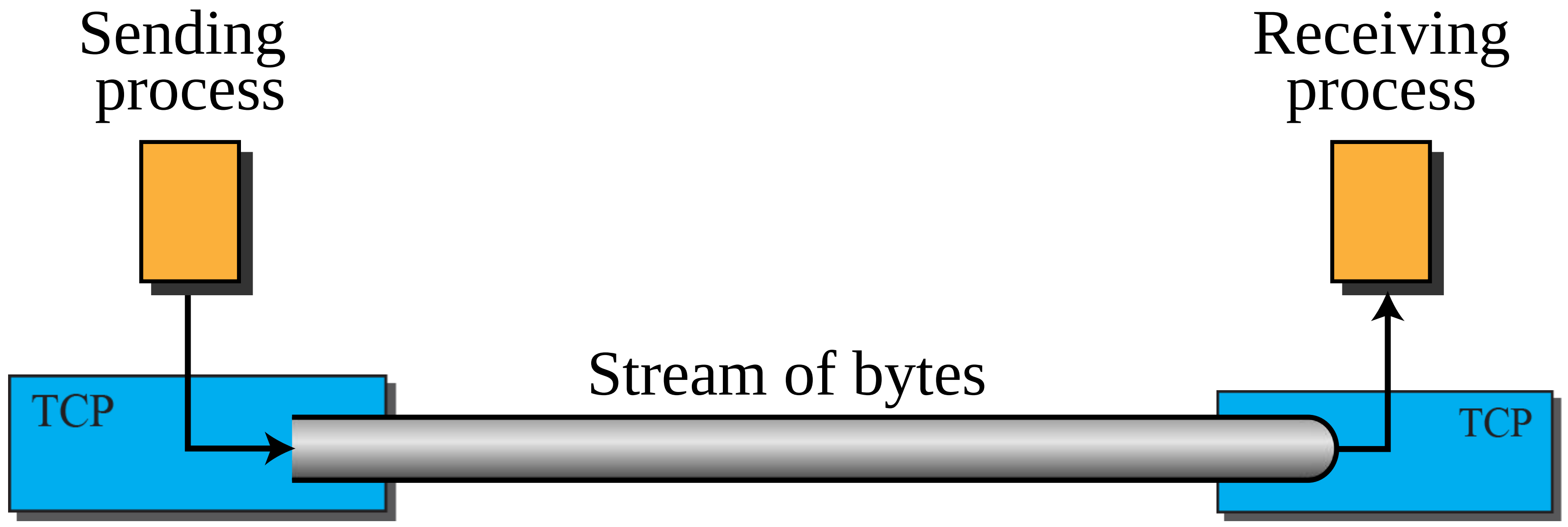


Figure 3.42: Sending and receiving buffers

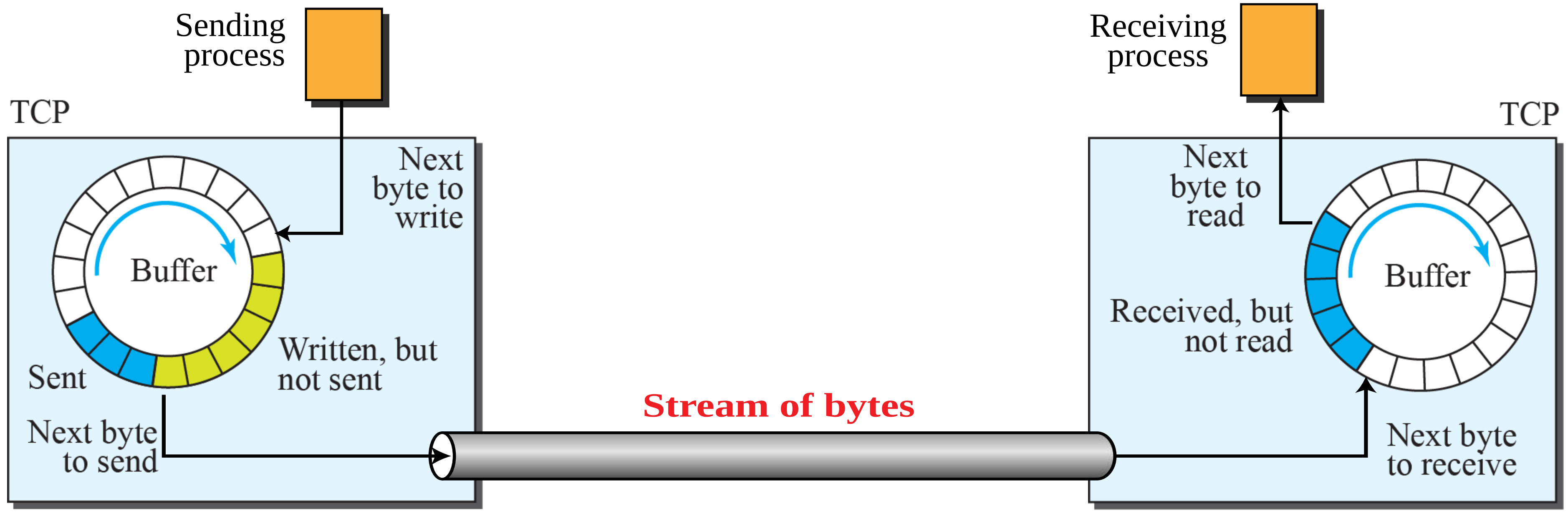
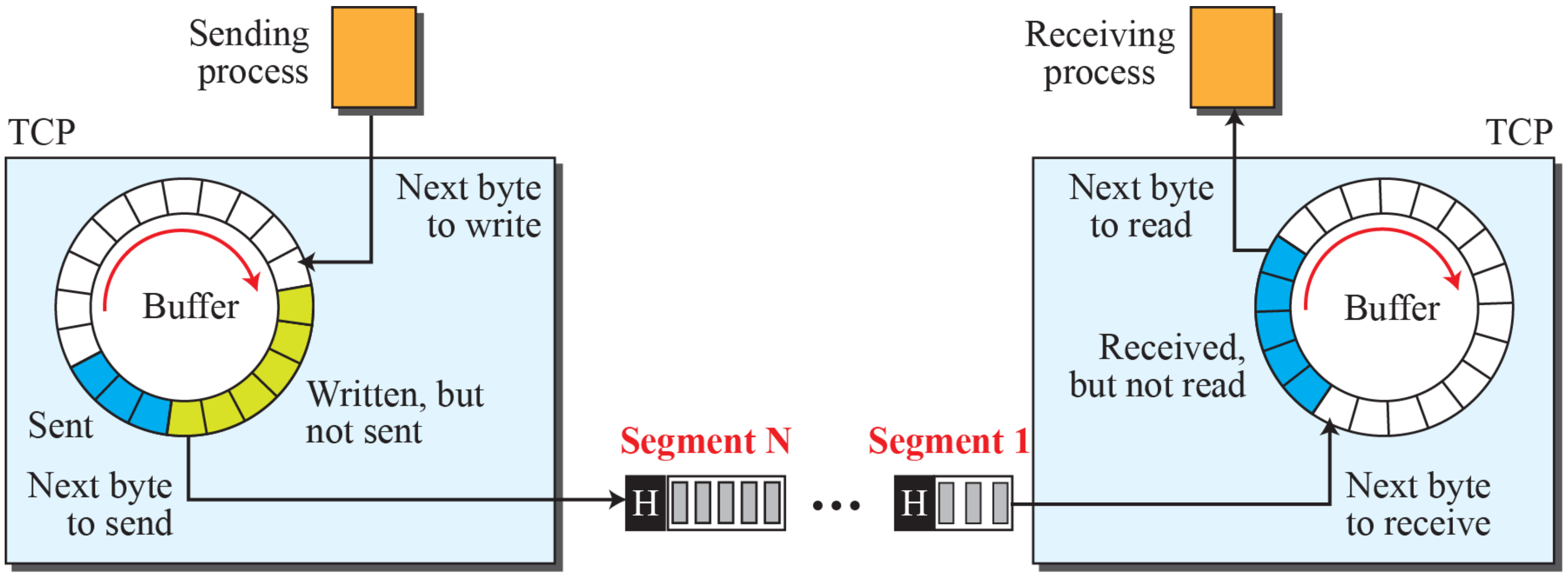


Figure 3.43: TCP segments

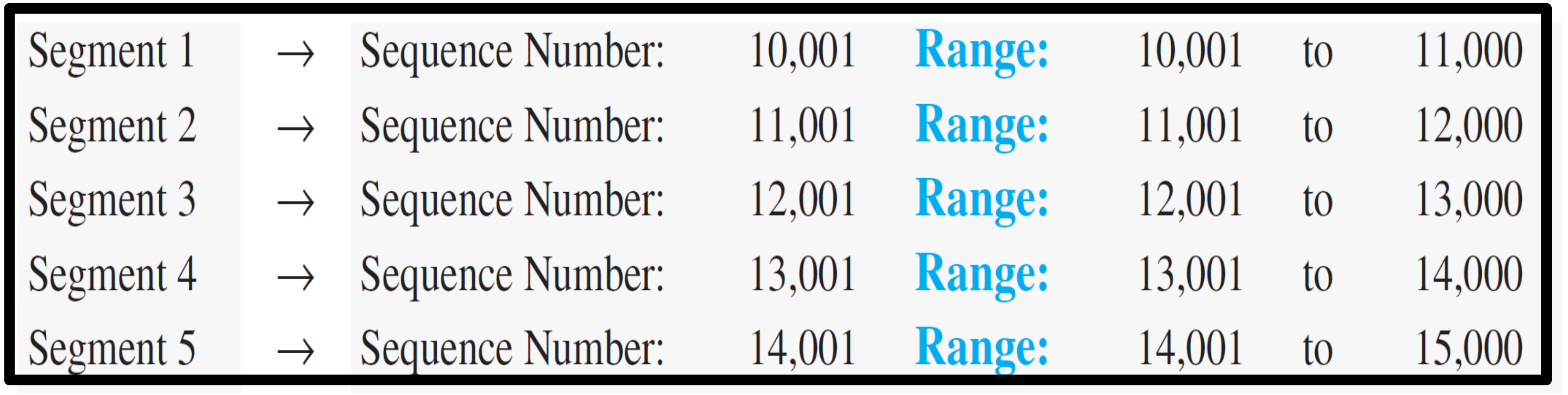


Example 3.17

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

Figure 3.44: *TCP segment format*

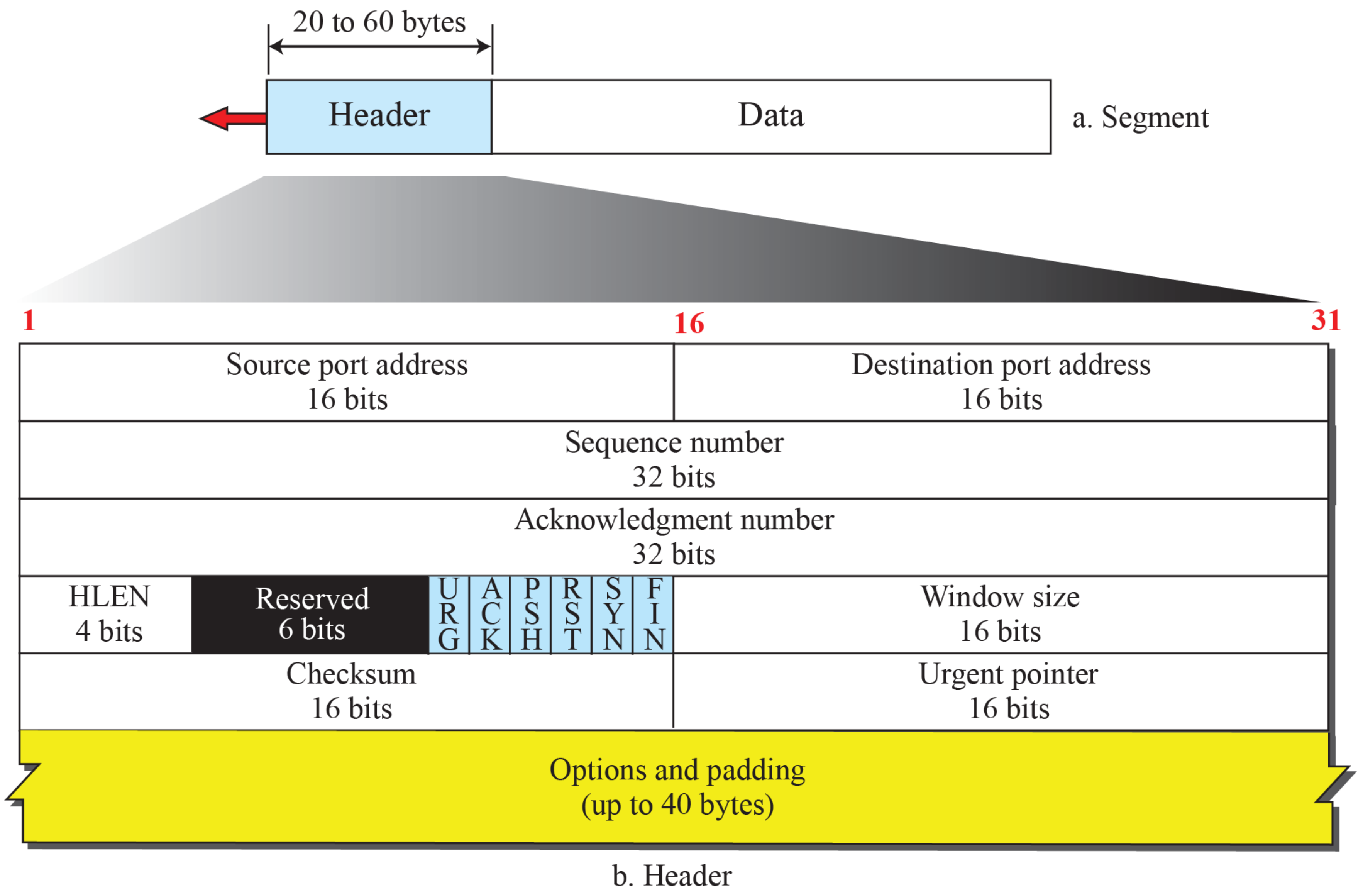
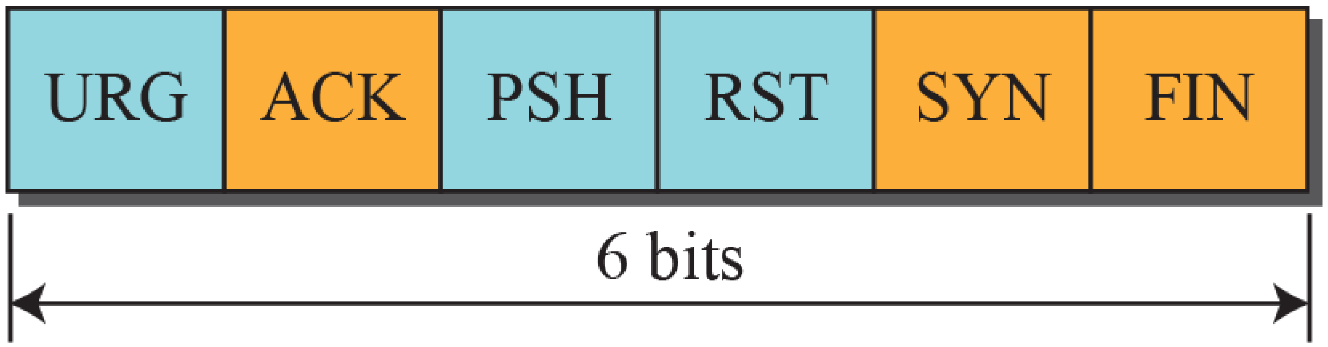


Figure 3.45: Control field



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

Figure 3.47: Connection establishment using three-way handshaking

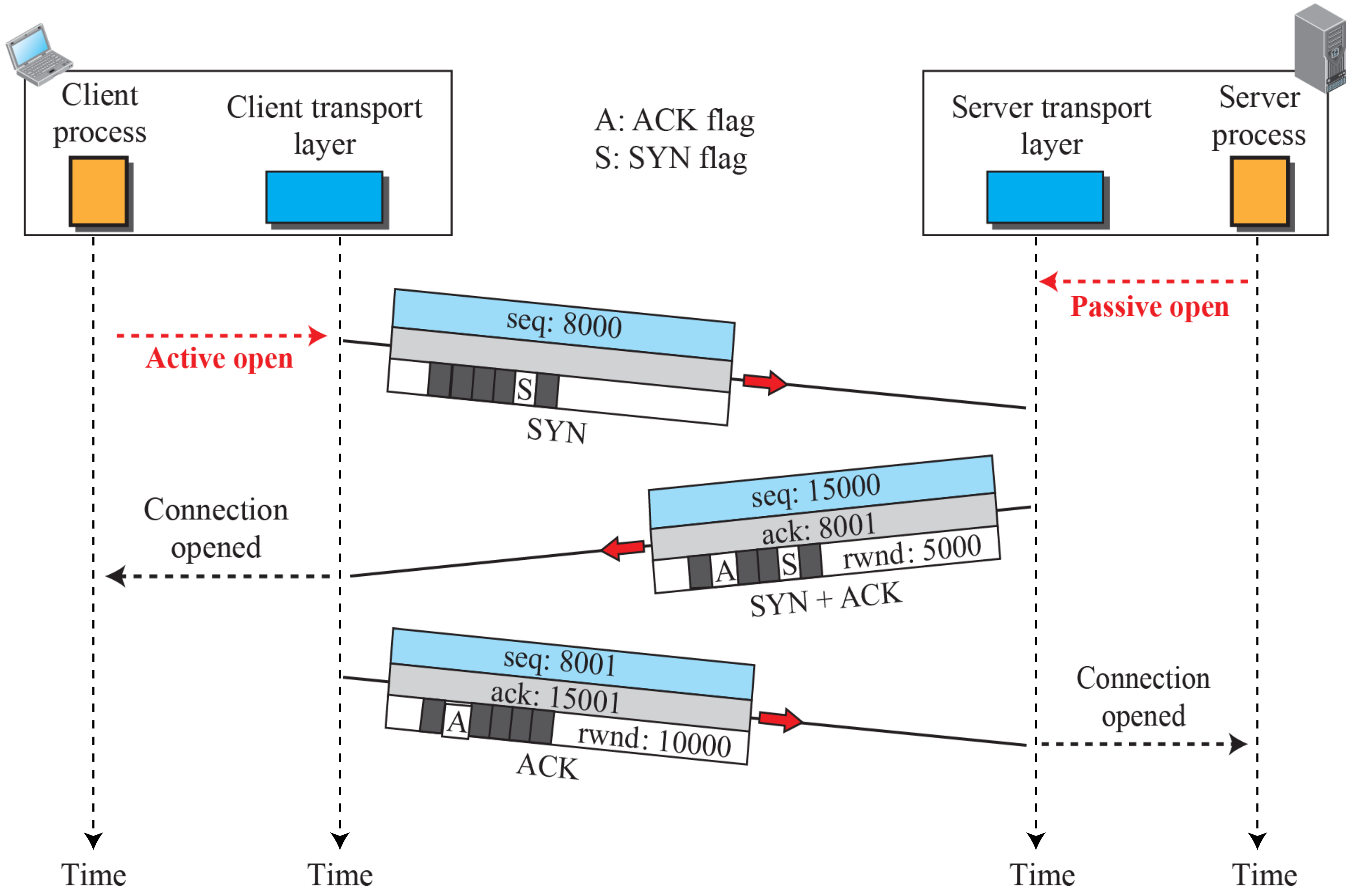


Figure 3.48: Data transfer

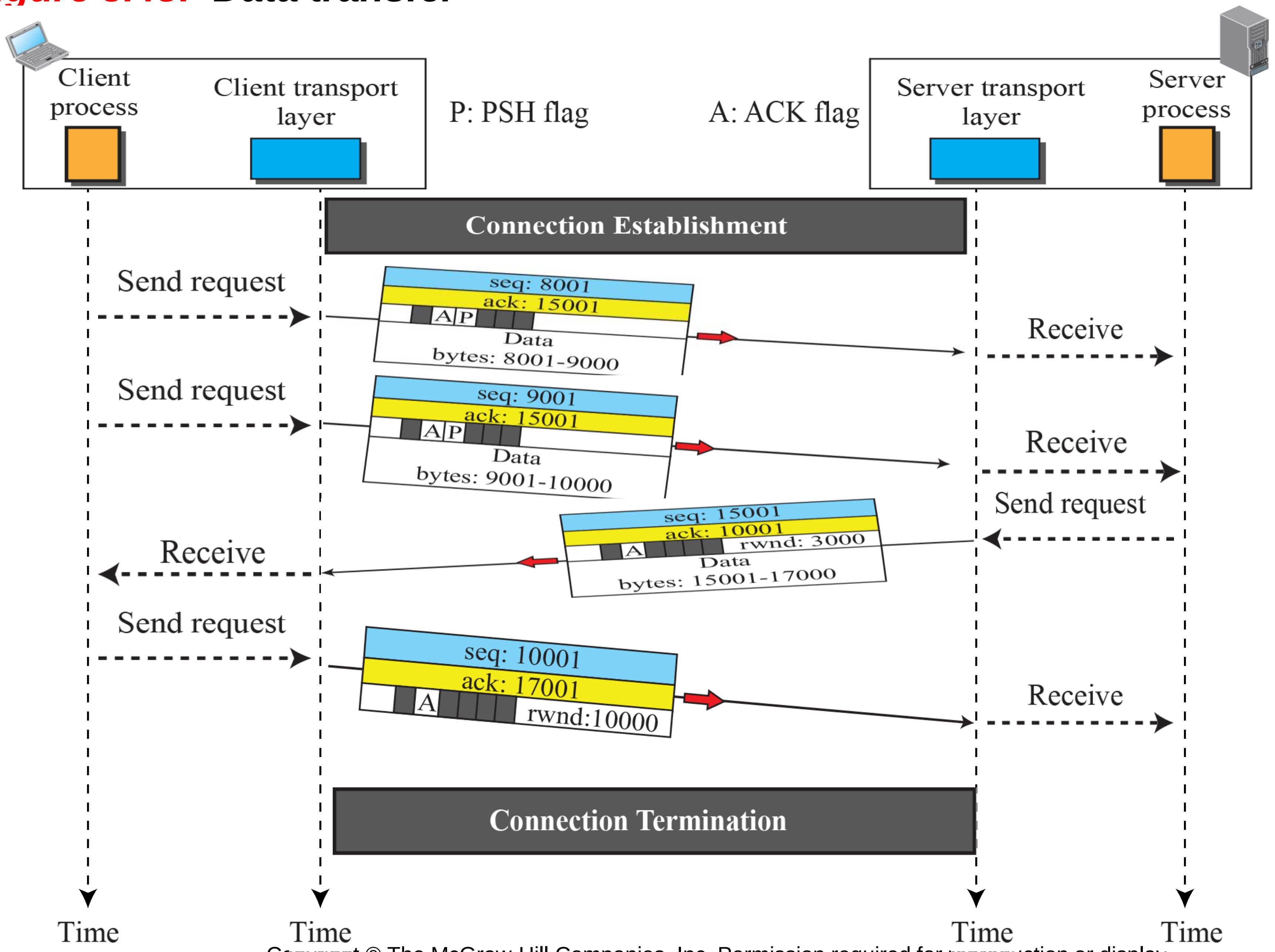


Figure 3.49: Connection termination using three-way handshaking

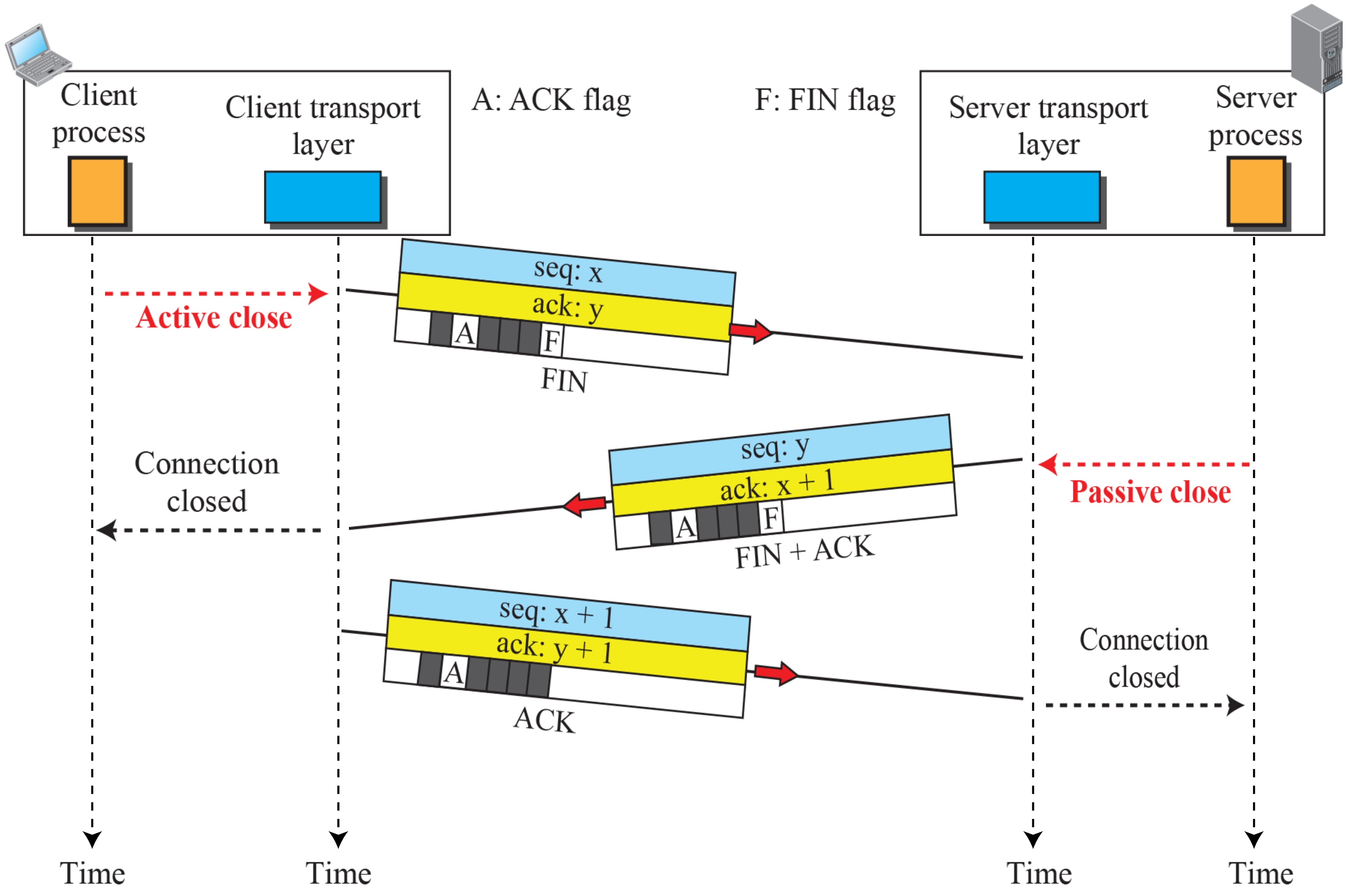
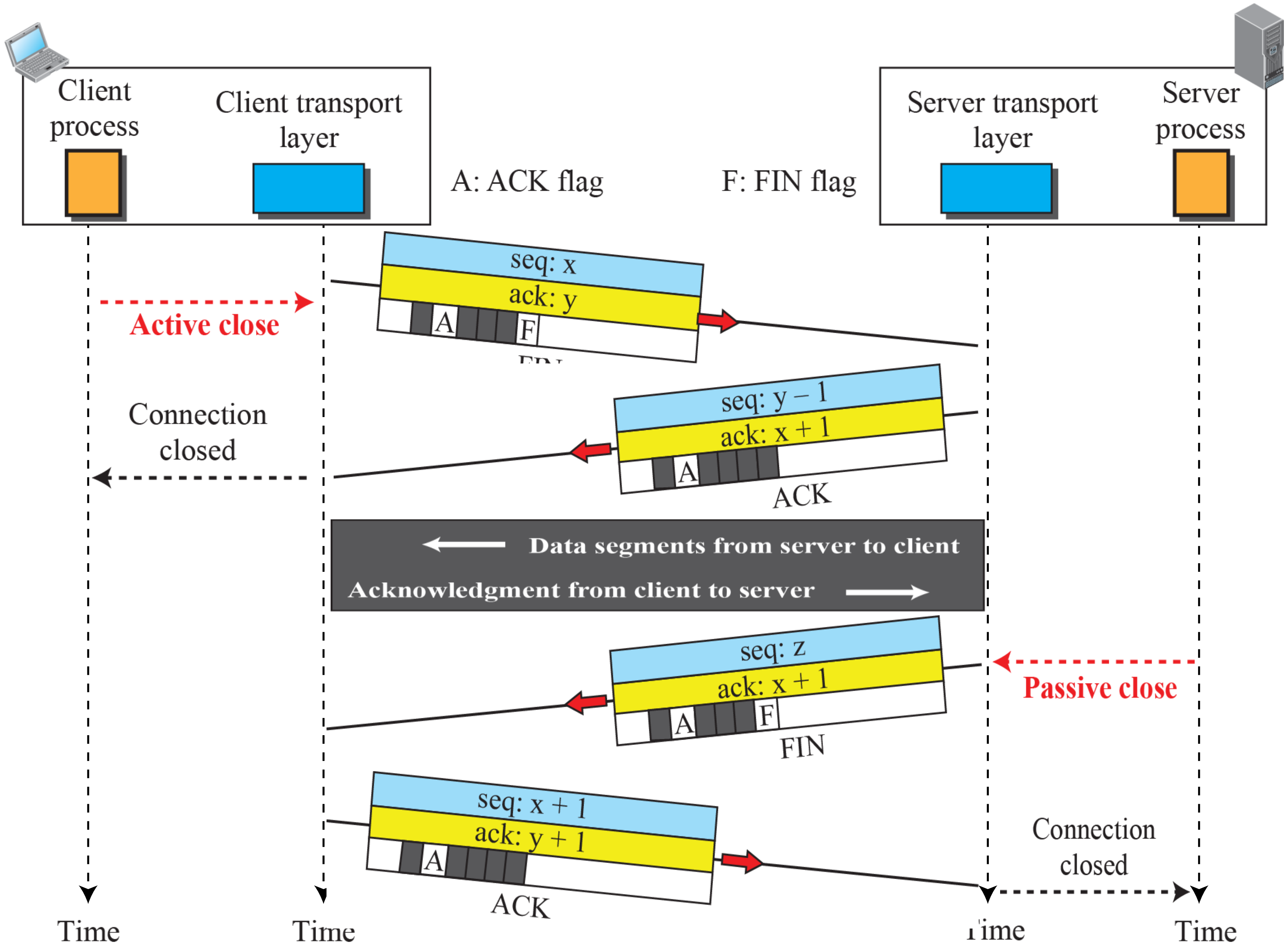


Figure 3.50: *Half-close*



3.4.7 Flow Control

Figure 3.56: Data flow and flow control feedbacks in TCP

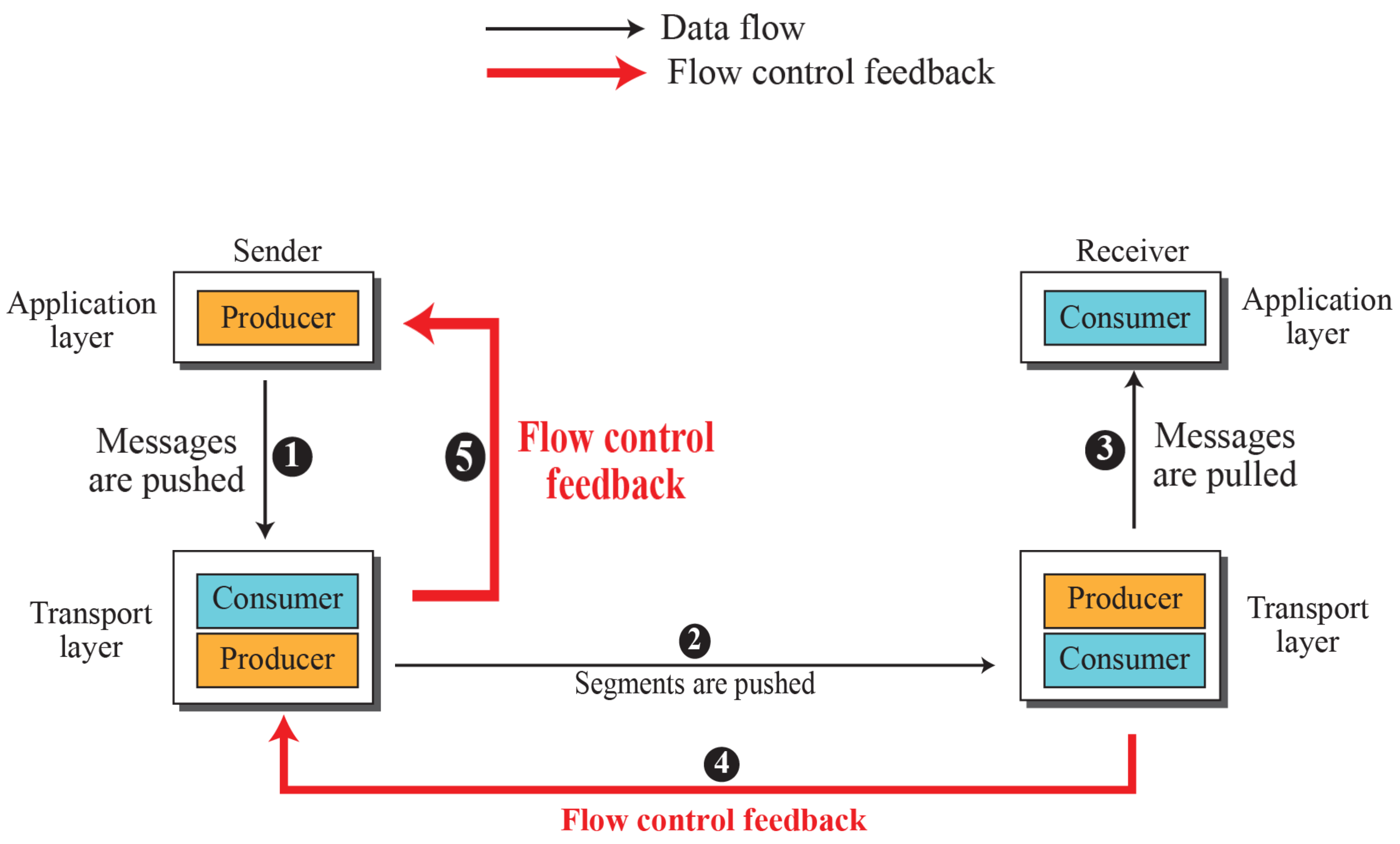
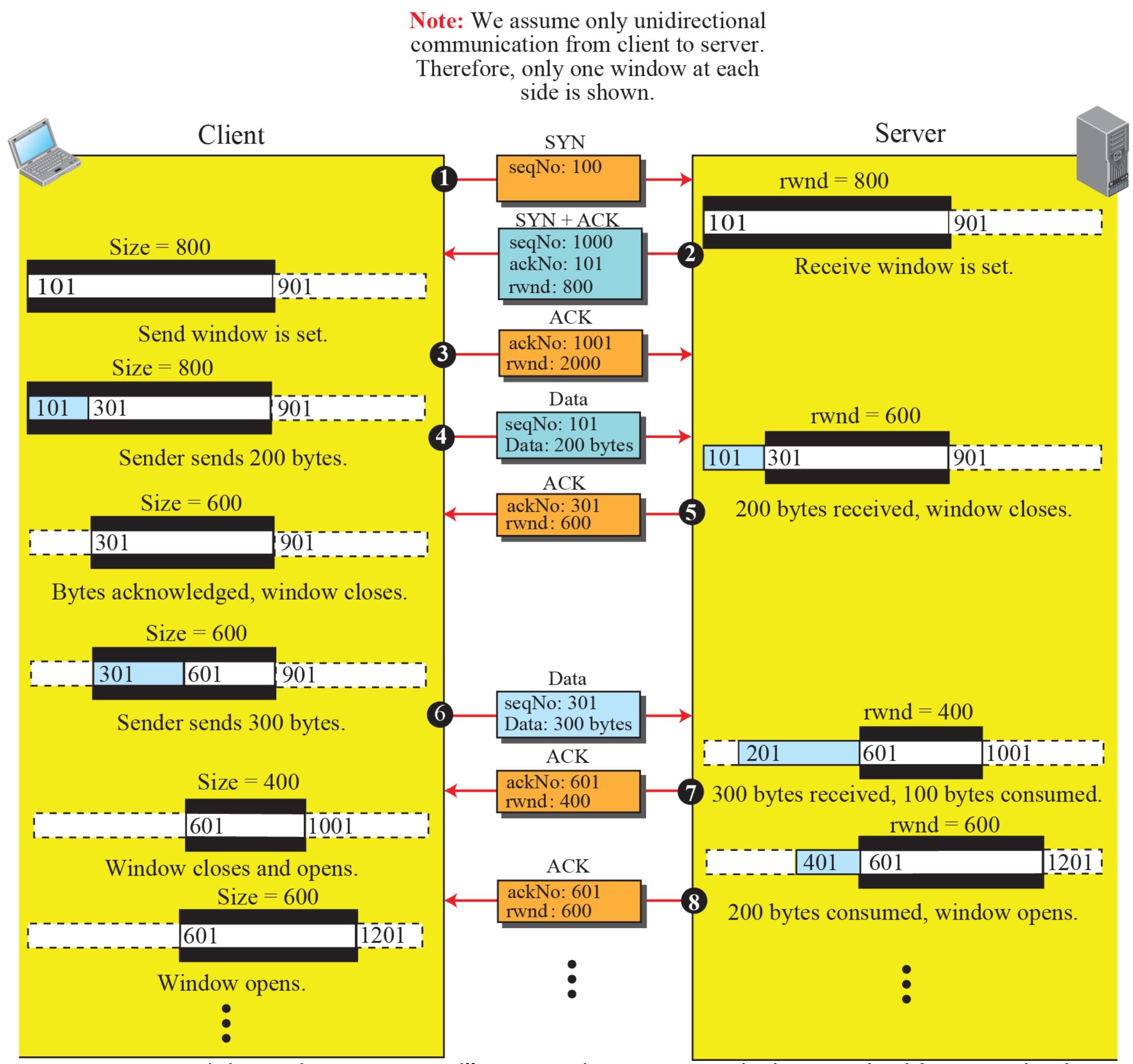


Figure 3.57: An example of flow control



3.4.8 Error Control

Figure 3.61: *Normal operation*

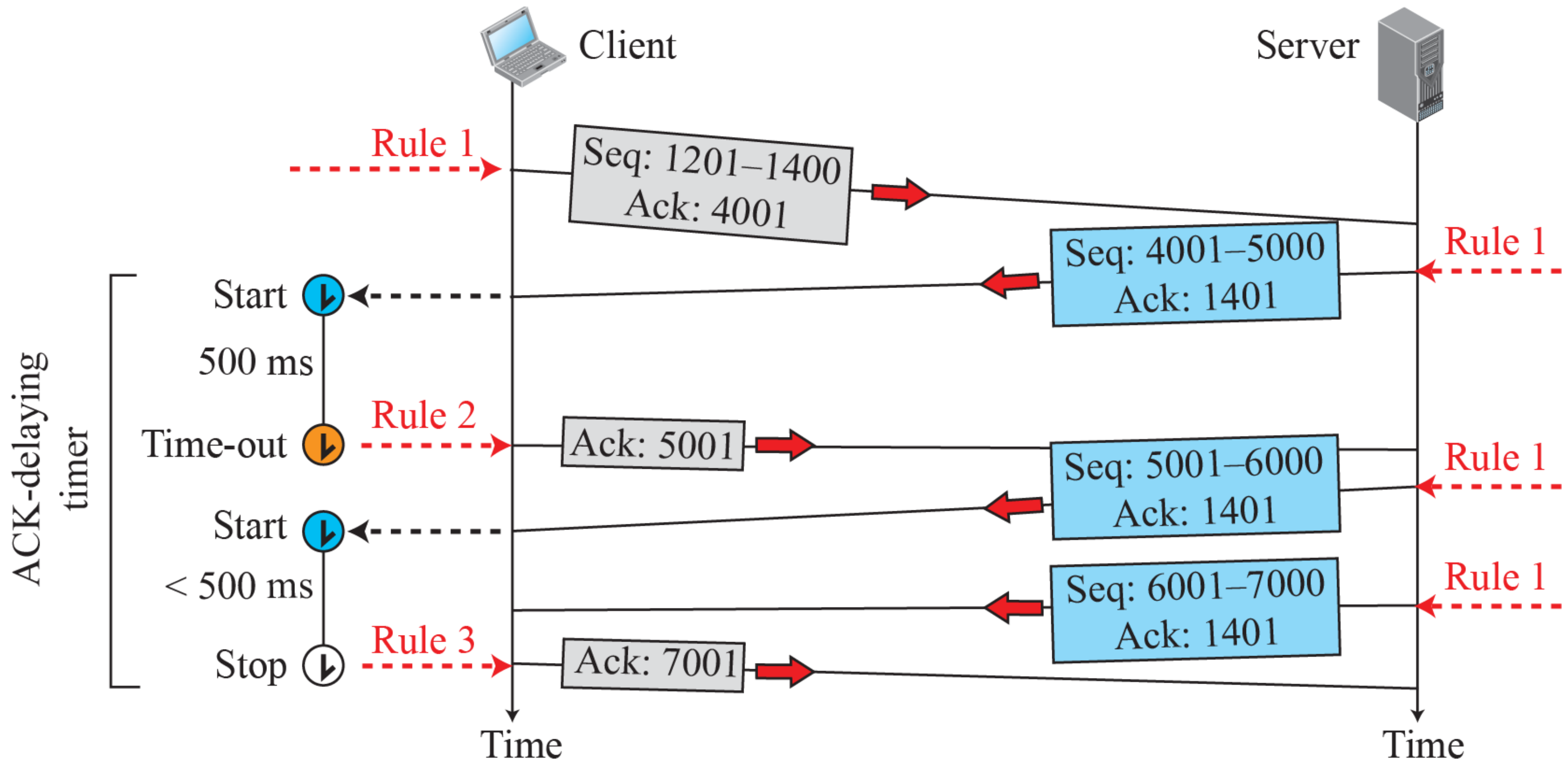


Figure 3.62: ***Lost segment***

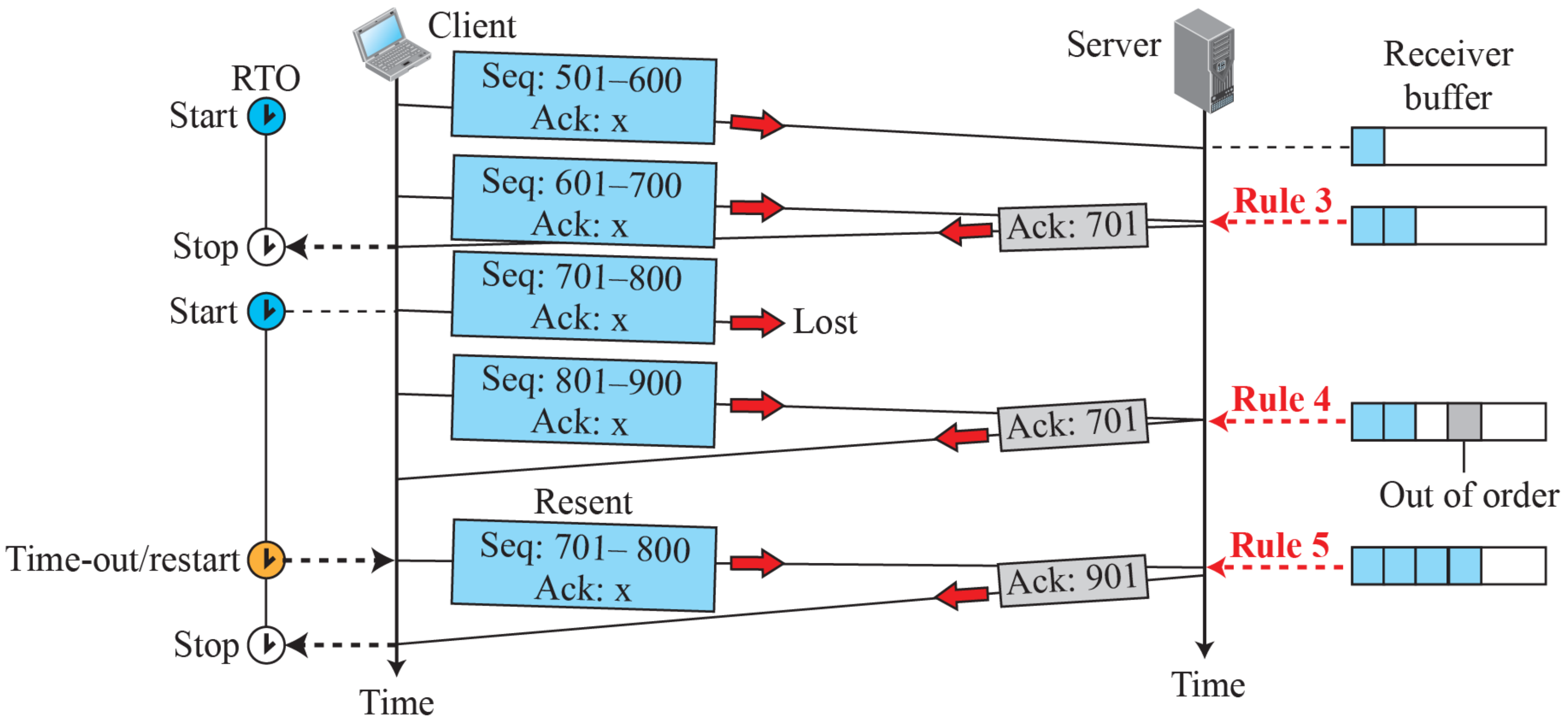


Figure 3.63: *Fast retransmission*

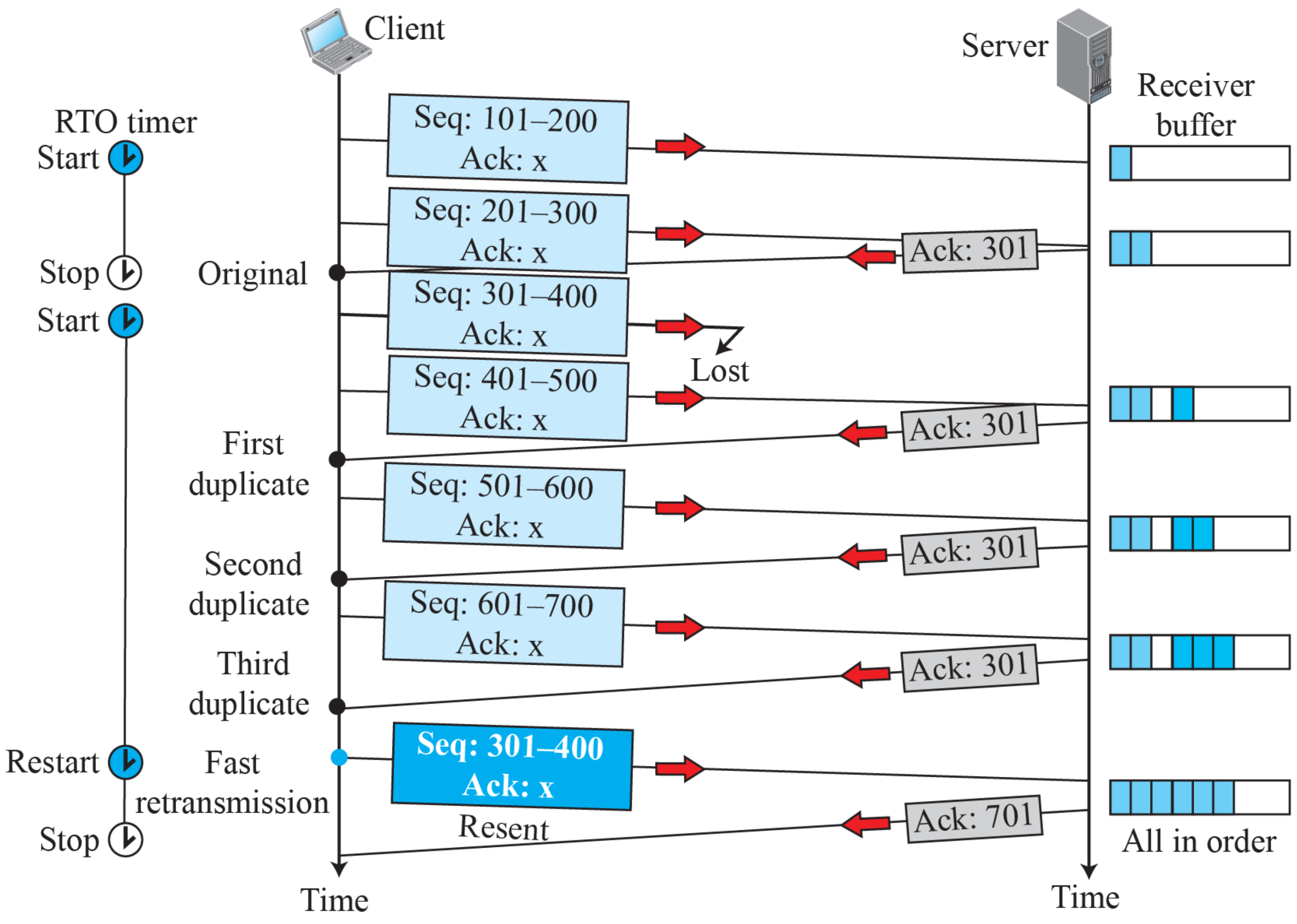


Figure 3.64: *Lost acknowledgment*

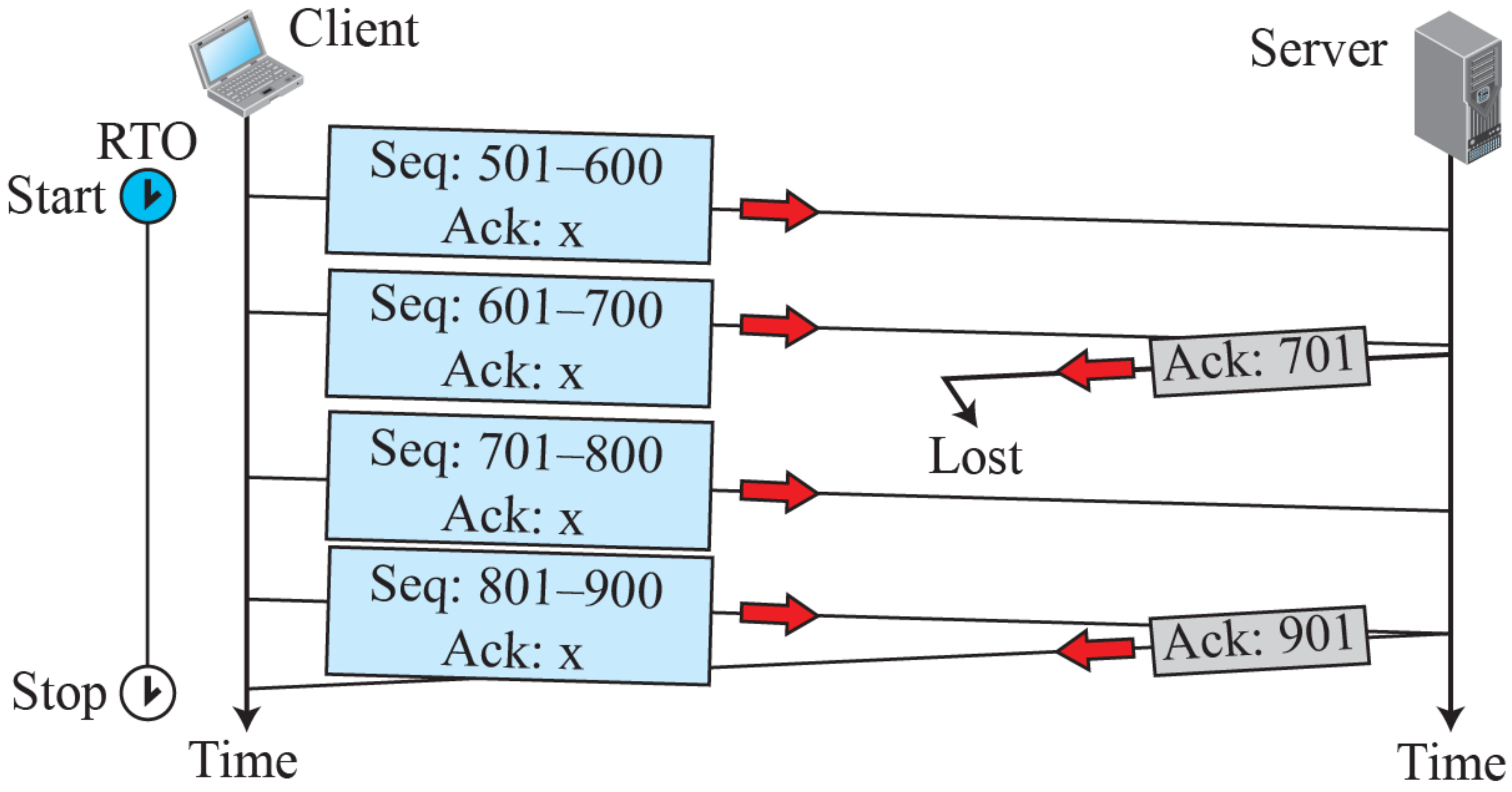
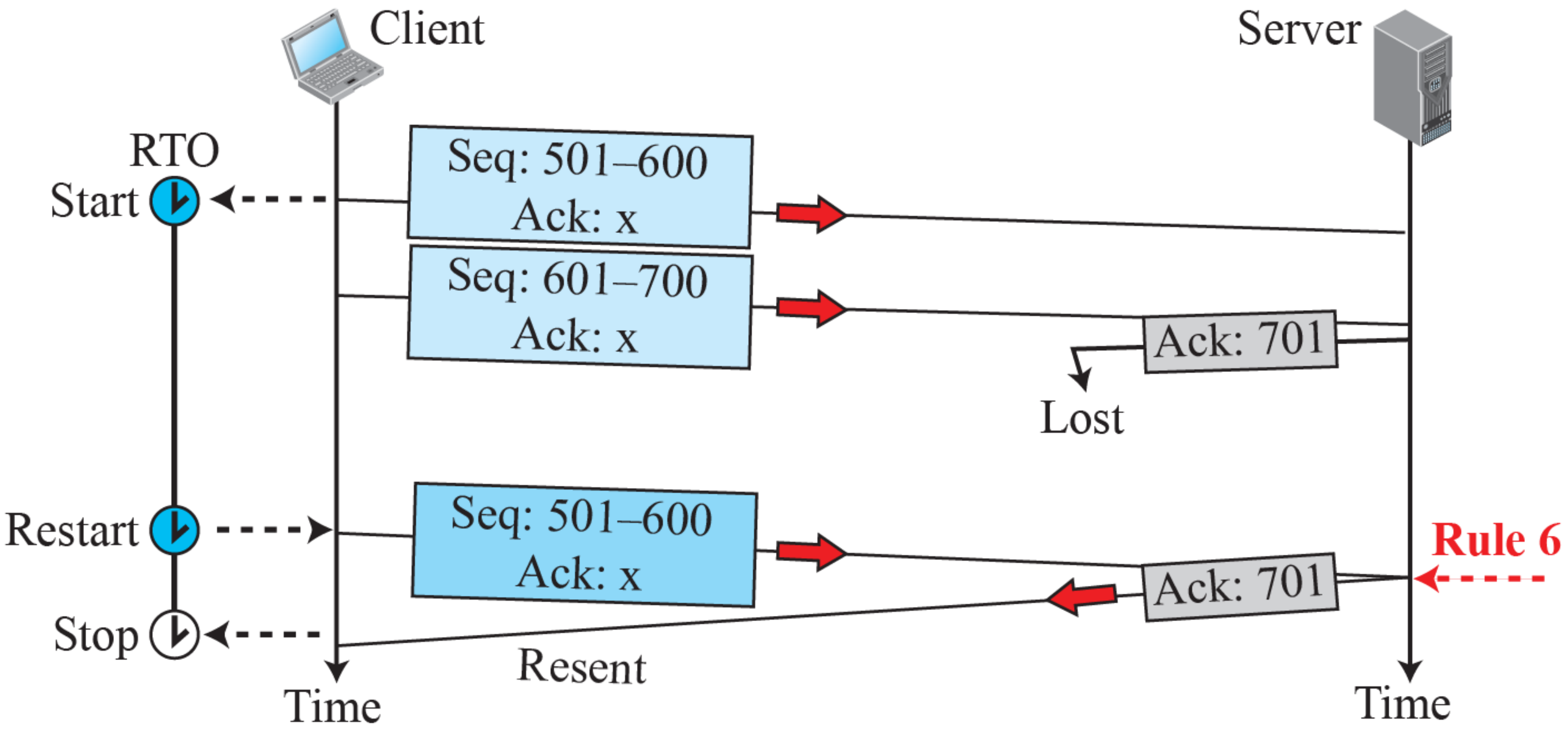


Figure 3.65: Lost acknowledgment corrected by resending a segment



3.4.9 TCP Congestion Control

- ***Slow Start: Exponential Increase***
- ***Congestion Avoidance: Additive Increase***

Figure 3.66: Slow start, exponential increase

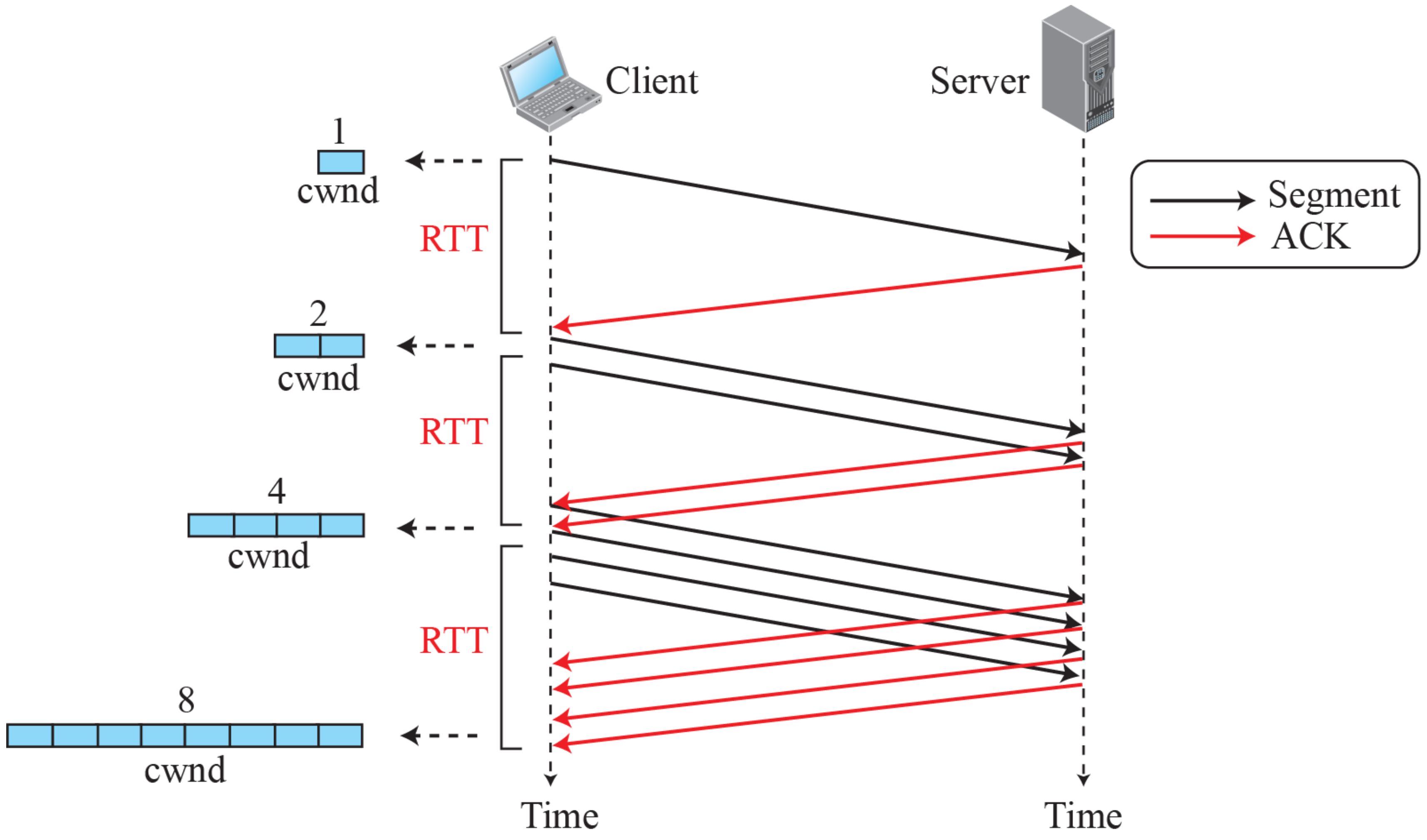


Figure 3.67: Congestion avoidance, additive increase

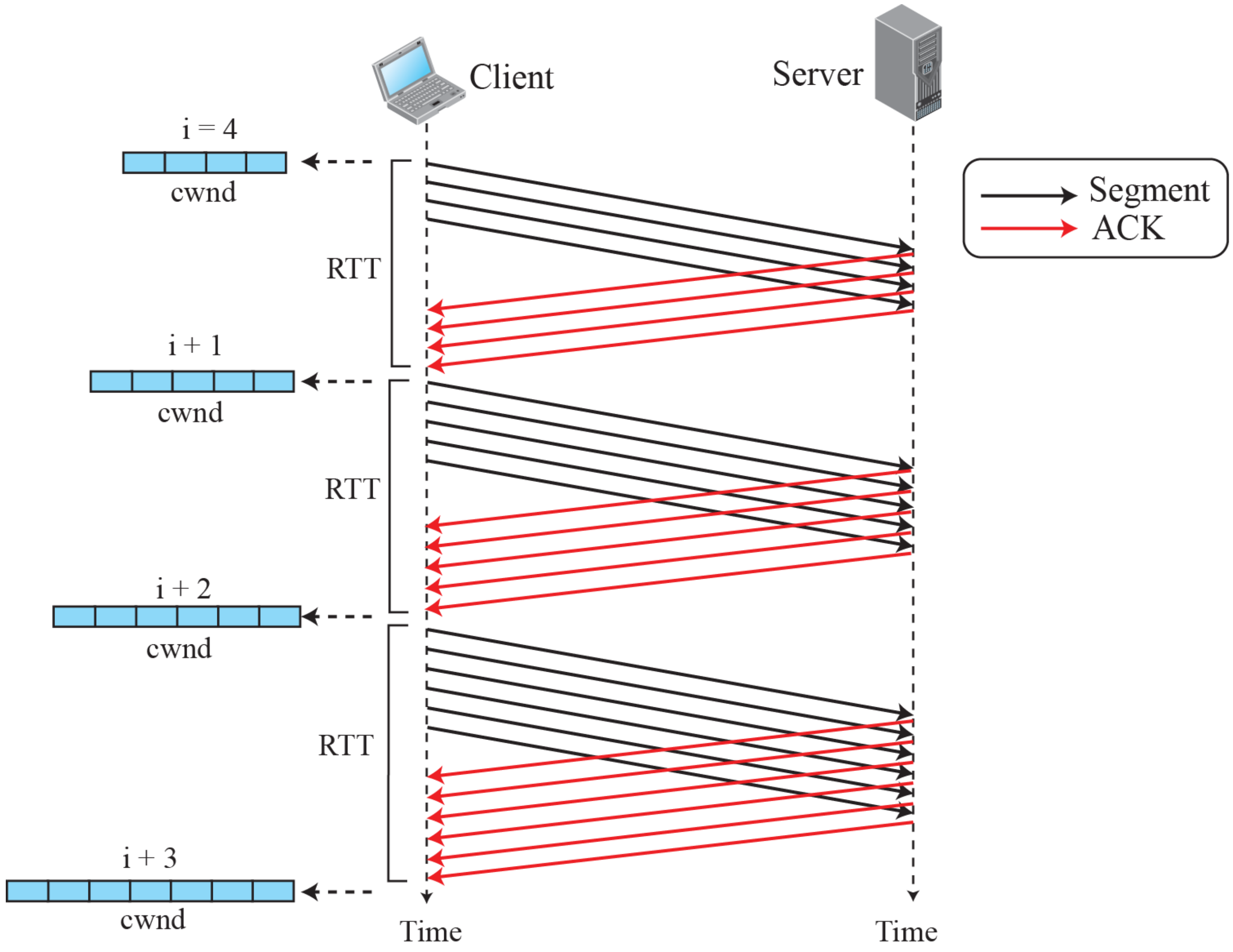


Figure 3.69: Example of Taho TCP

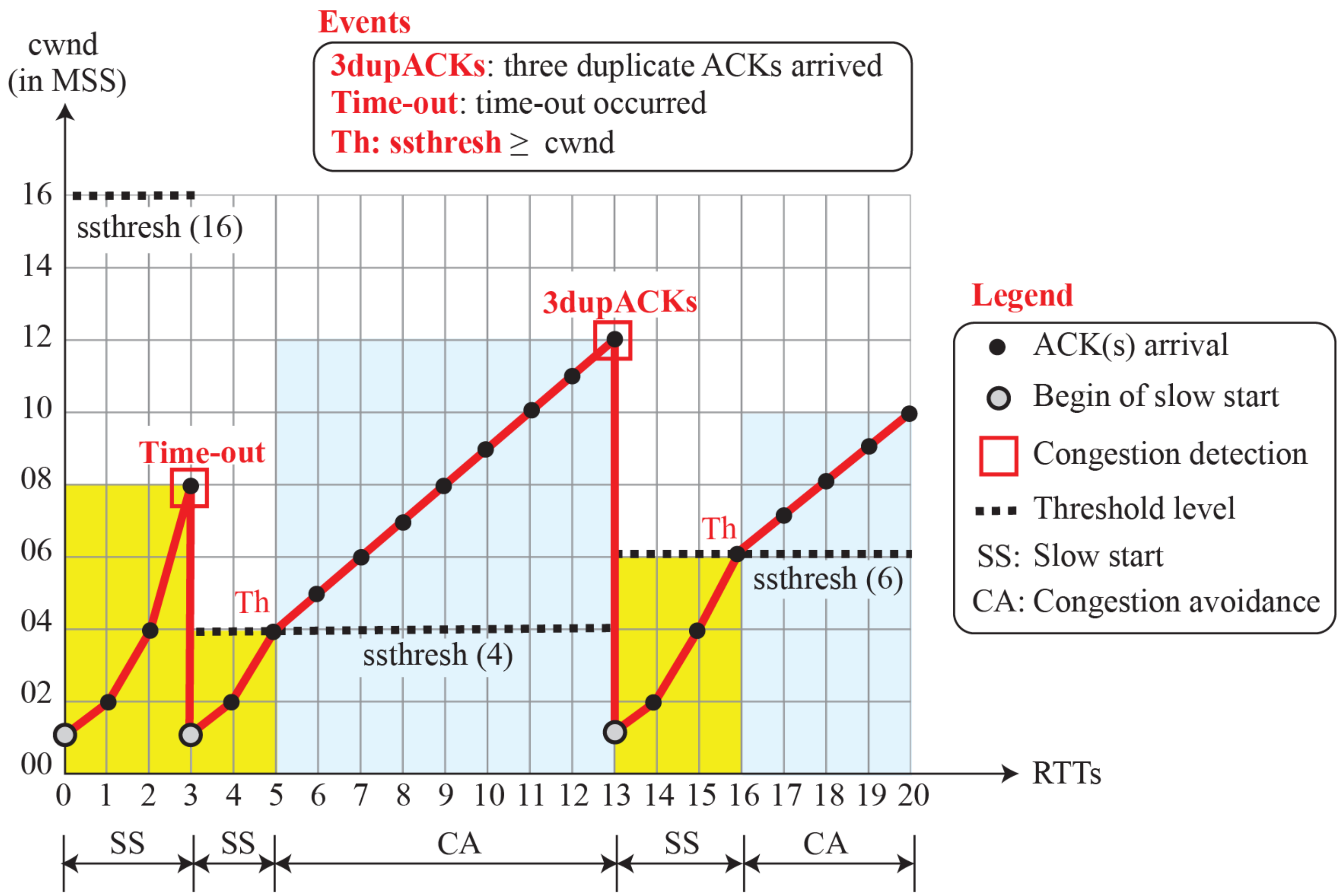


Figure 3.71: Example of a Reno TCP

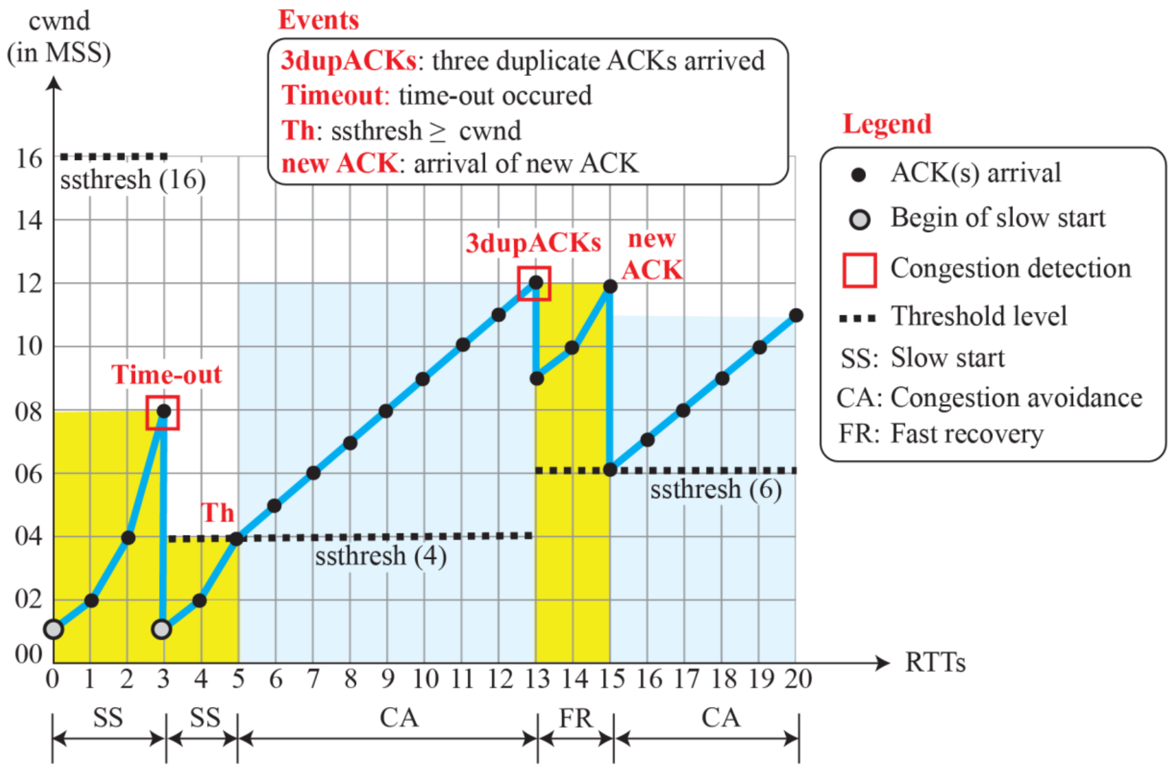
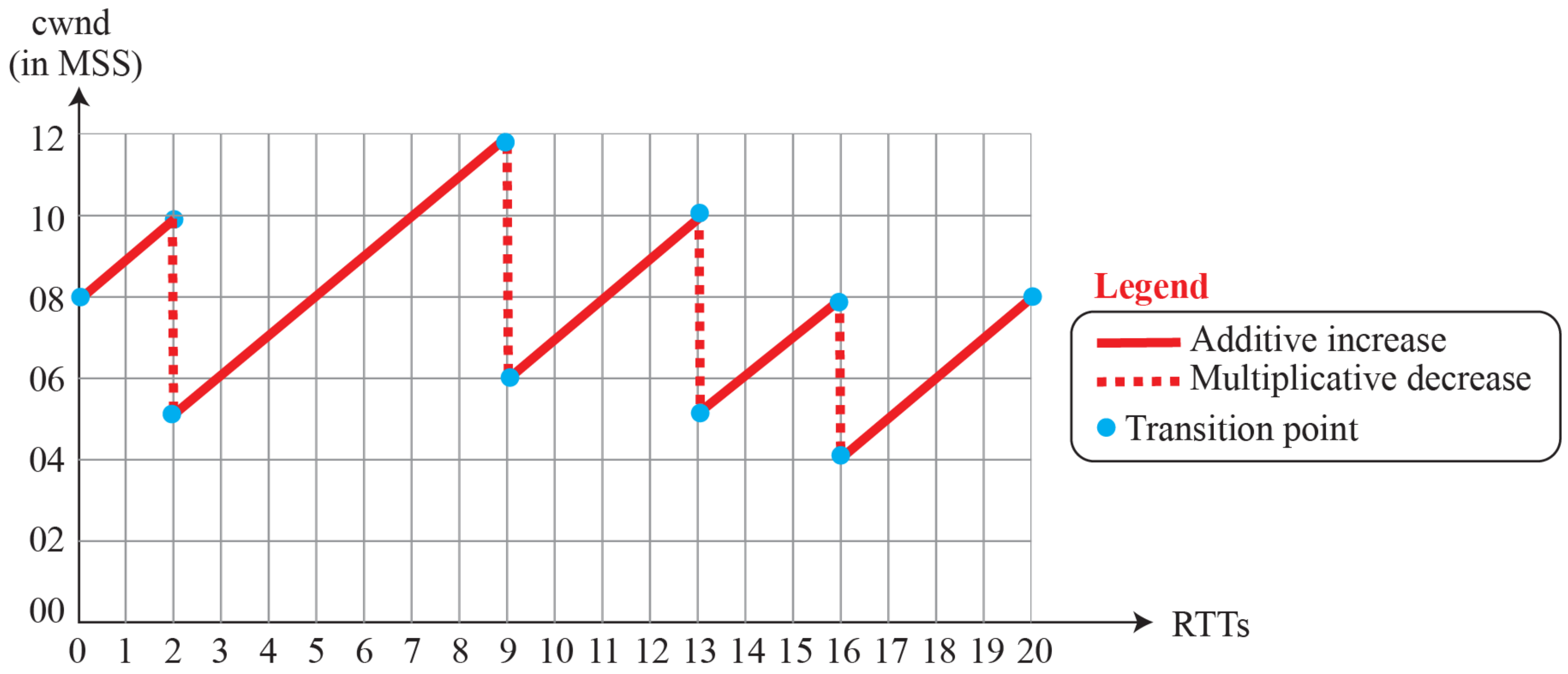


Figure 3.72: *Additive increase, multiplicative decrease (AIMD)*



Example 3.21

If MSS = 10 KB (kilobytes) and RTT = 100 ms in Figure 3.72, we can calculate the throughput as shown below.

$$W_{max} = (10 + 12 + 10 + 8 + 8) / 5 = 9.6 \text{ MSS}$$

$$\text{Throughput} = (0.75\ W_{max} / \text{RTT}) = 0.75 \times 960 \text{ kbps} / 100 \text{ ms} = 7.2 \text{ Mbps}$$