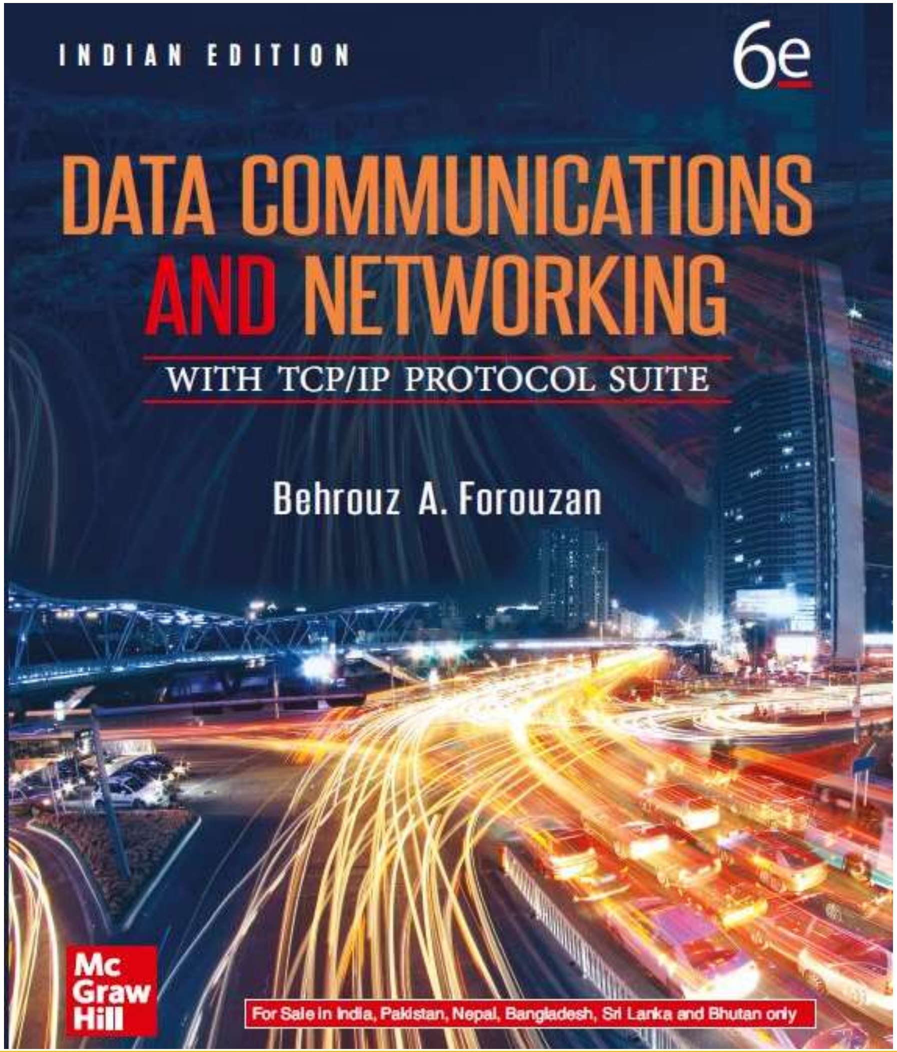
Because learning changes everything.®

Chapter 09

Transport Layer

Data Communications and Networking, With TCP/IP protocol suite
Sixth Edition
Behrouz A. Forouzan



Chapter 9: Outline

9.1 Transport-Layer Services

9.2 Transport-Layer Protocols

9.2 User-Datagram Protocol (UDP)

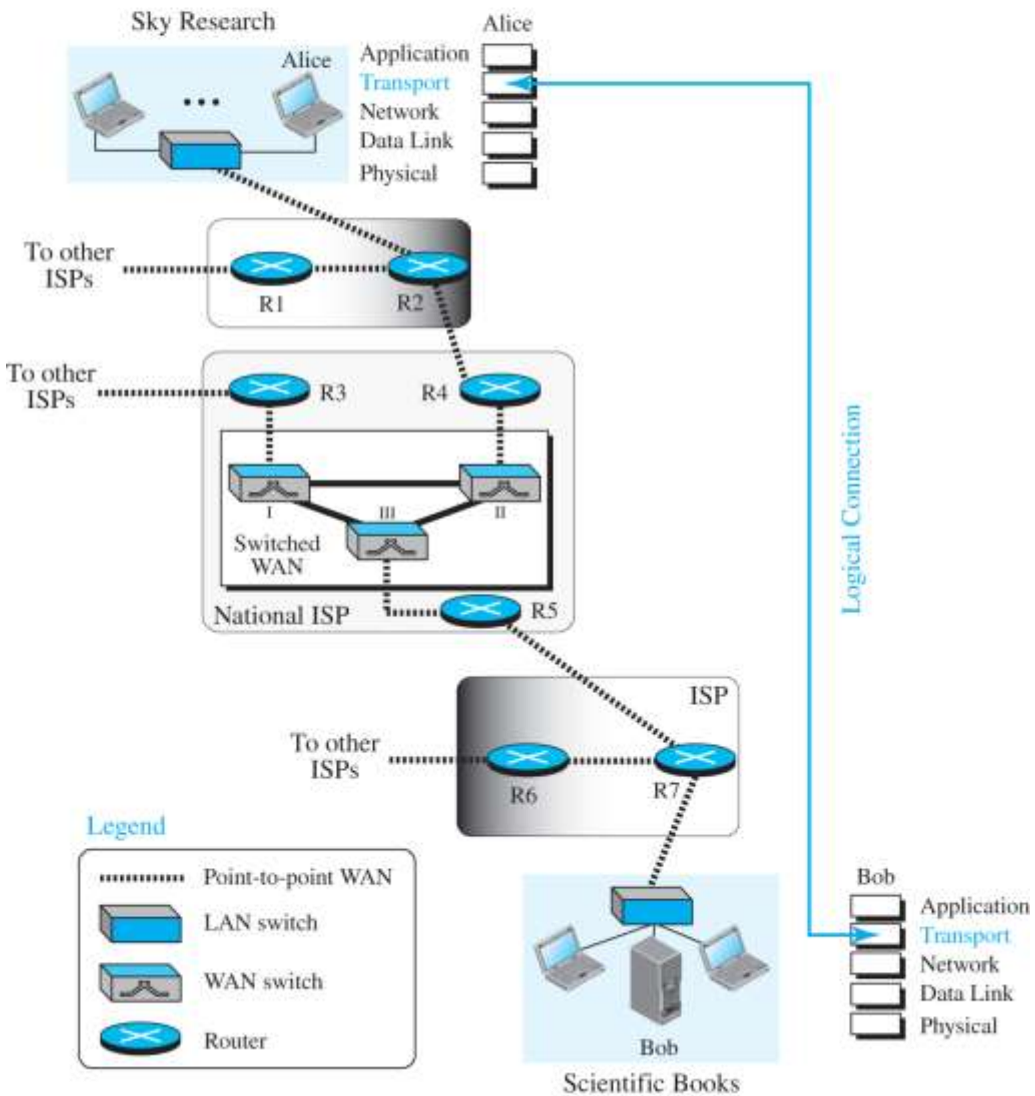
9.3 Transmission Control Protocol (TCP)

9.4 Stream Control Transmission Protocol (SCTP)

9-1 TRANSPORT LAYER SERVICES

The transport layer is located between the application layer and the network layer. It provides a process-to-process communication between two application layers, one at the local host and the other at the remote host. Communication is provided using a logical connection. Figure 9.1 shows the idea behind this logical connection.

Figure 9.1 Logical connection at the transport layer

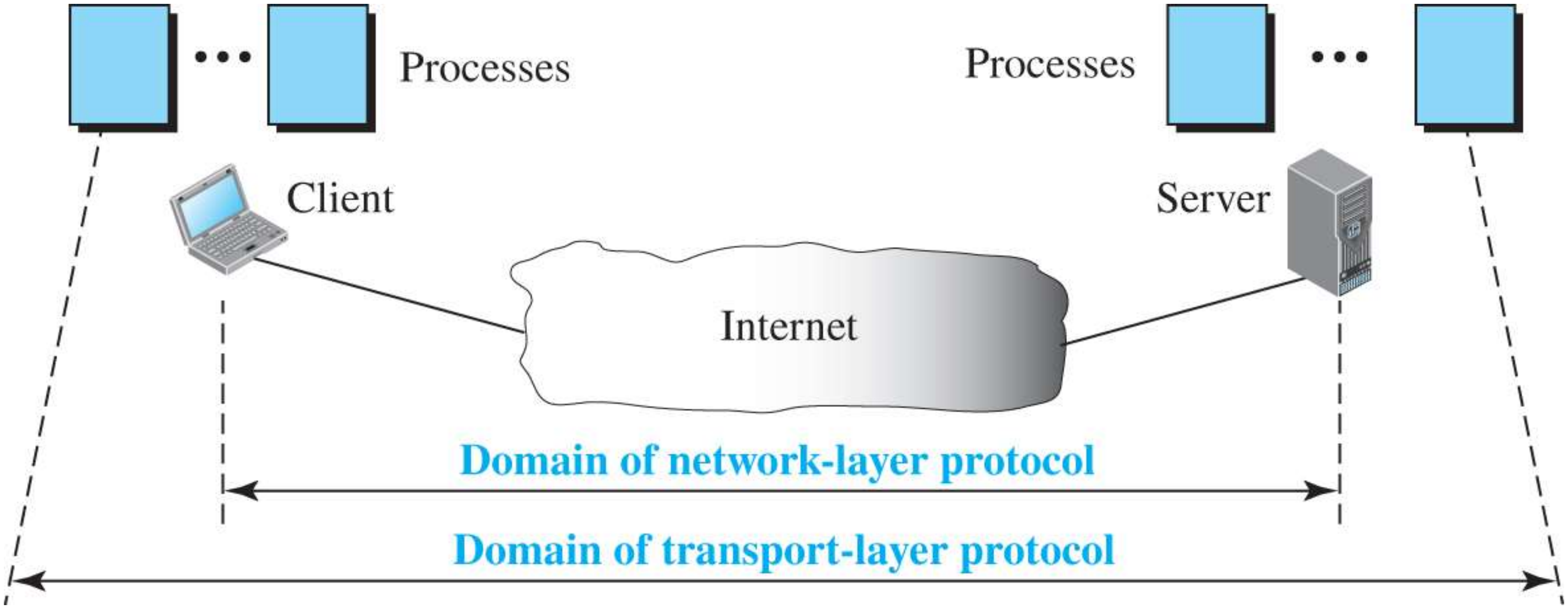


Access the text alternative for slide images.

9.1.1 Process-to-Process Communication

The first duty of a transport-layer protocol is to provide process-to-process communication. A process is an application-layer entity (running program) that uses the services of the transport layer. Before we discuss how process-to-process communication can be accomplished, we need to understand the difference between host-to-host communication and process-to-process communication.

Figure 9.2 Network layer versus transport layer

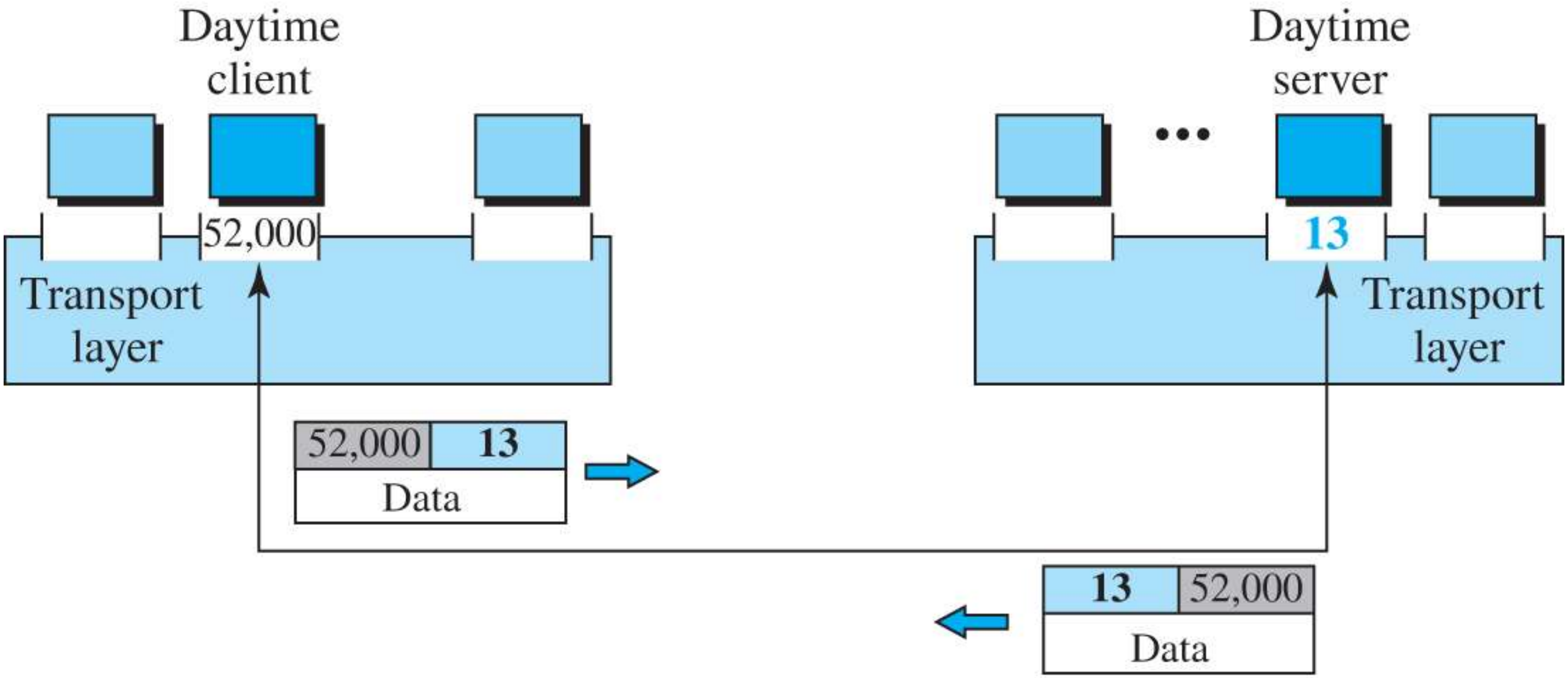


Access the text alternative for slide images.

9.1.2 Addressing: Port Numbers

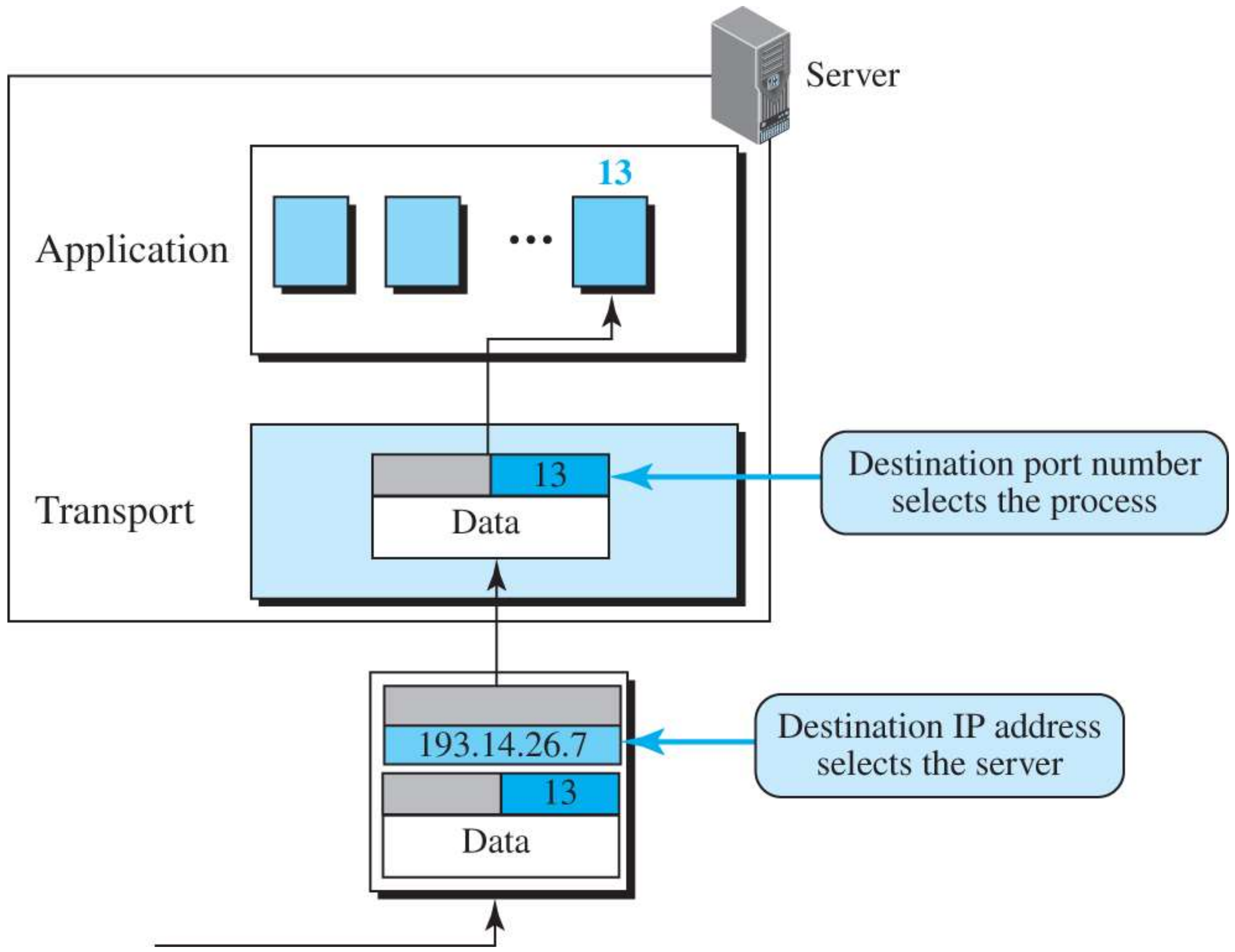
- *Although there are a few ways to achieve process-to-process communication, the most common is through the client-server paradigm. A process on the local host, called a client, needs services from a process usually on the remote host, called a server.*
- *However, operating systems today support both multiuser and multiprogramming environments. A remote computer can run several programs at the same time, just as several local computers can run one or more client programs at the same time.*

Figure 9.3 Port numbers



Access the text alternative for slide images.

Figure 9.4 IP addresses versus port numbers

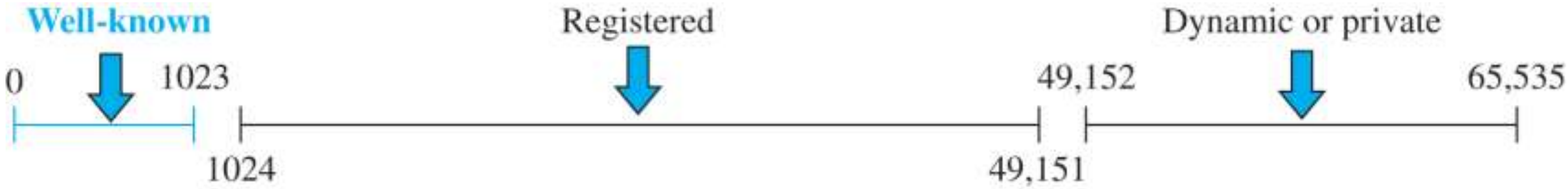


Access the text alternative for slide images.

ICANN Ranges

ICANN (Internet Corporation for Assigned Names and Numbers) has divided the port numbers into three ranges: well-known, registered, and dynamic (or private).

Figure 9.5 ICANN ranges



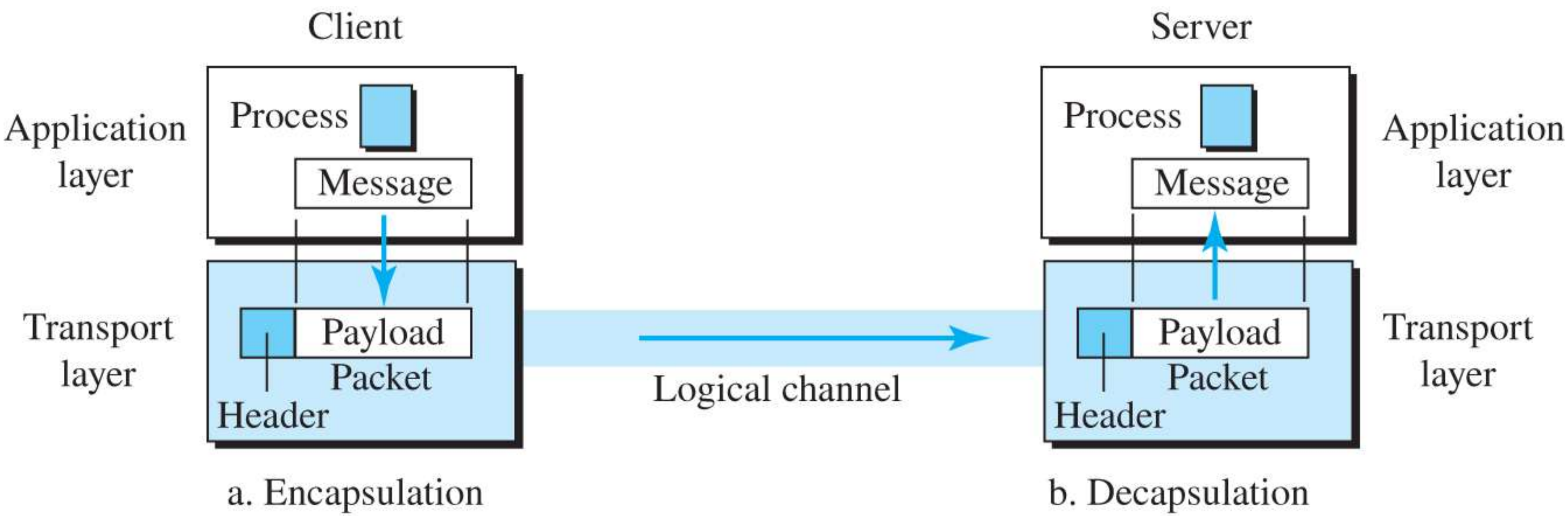
Access the text alternative for slide images.

Figure 9.6 Socket address

9.1.3 Encapsulation and Decapsulation

To send a message from one process to another, the transport-layer protocol encapsulates and decapsulates messages. Encapsulation happens at the sender site. When a process has a message to send, it passes the message to the transport layer along with a pair of socket addresses and some other pieces of information, which depend on the transport-layer protocol. The transport layer receives the data and adds the transport-layer header.

Figure 9.7 Encapsulation and decapsulation

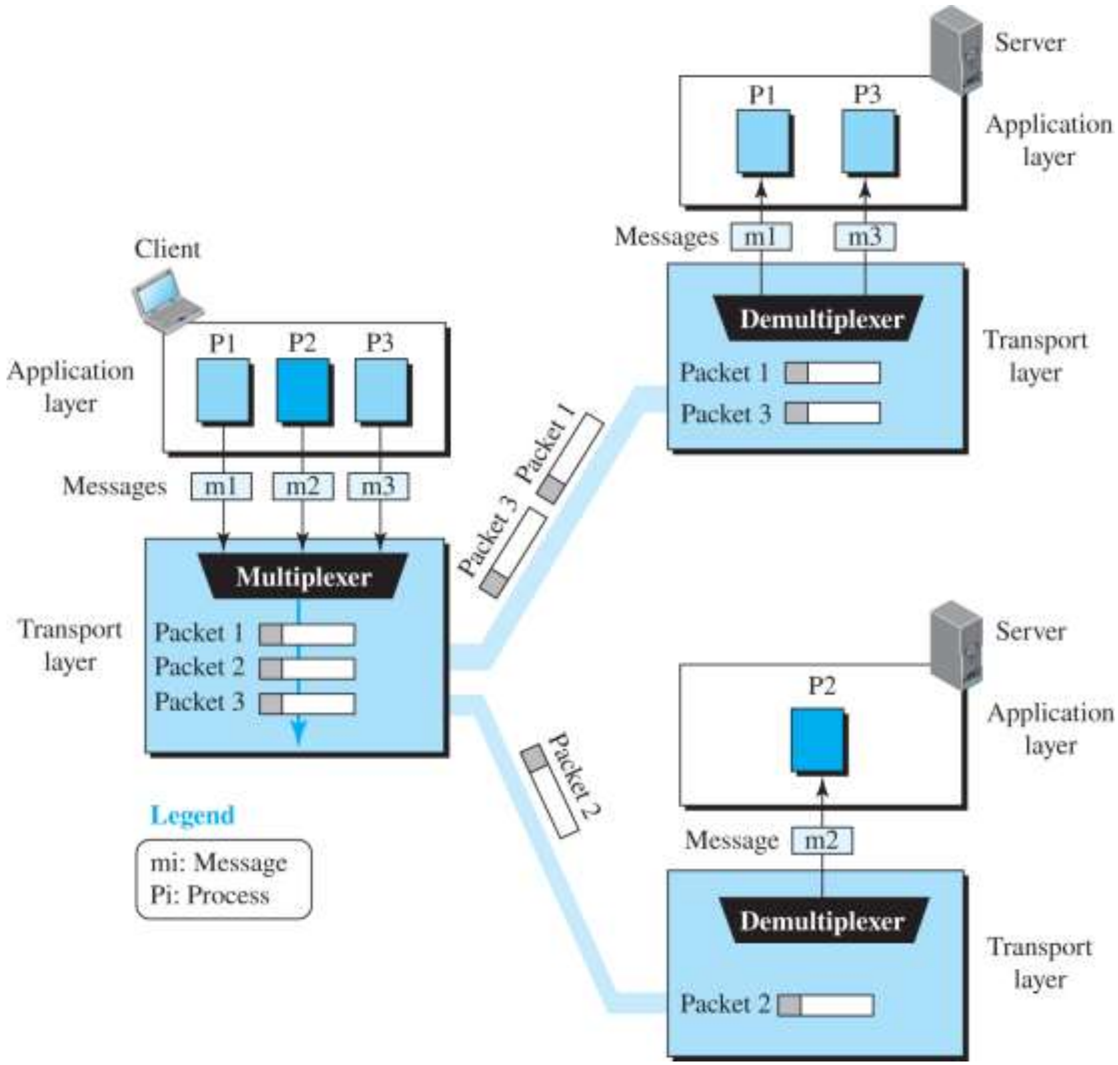


Access the text alternative for slide images.

9.1.4 Multiplexing and Demultiplexing

Whenever an entity accepts items from more than one source, this is referred to as multiplexing (many to one); whenever an entity delivers items to more than one source, this is referred to as demultiplexing (one to many). The transport layer at the source performs multiplexing; the transport layer at the destination performs demultiplexing.

Figure 9.8 Multiplexing and demultiplexing



Access the text alternative for slide images.

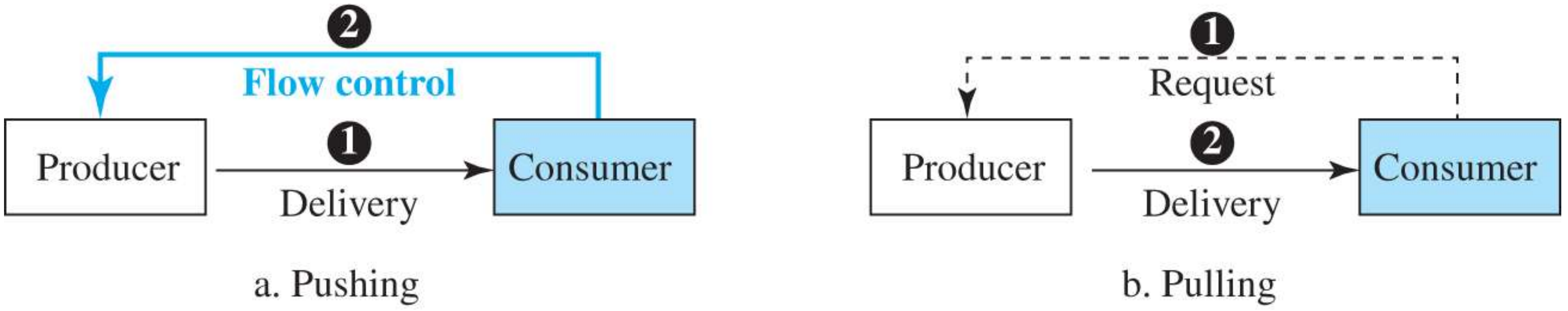
9.1.5 Flow Control

Whenever an entity produces items and another entity consumes them, there should be a balance between production and consumption rates. If the items are produced faster than they can be consumed, the consumer can be overwhelmed and may need to discard some items. If the items are produced more slowly than they can be consumed, the consumer must wait, and the system becomes less efficient. Flow control is related to the first issue. We need to prevent losing the data items at the consumer site.

Pushing and Pooling

Delivery of items from a producer to a consumer can occur in one of two ways: pushing or pulling. If the sender delivers items whenever they are produced without a prior request from the consumer, the delivery is referred to as pushing. If the producer delivers the items after the consumer has requested them, the delivery is referred to as pulling these two types of delivery.

Figure 9.9 Pushing or pulling

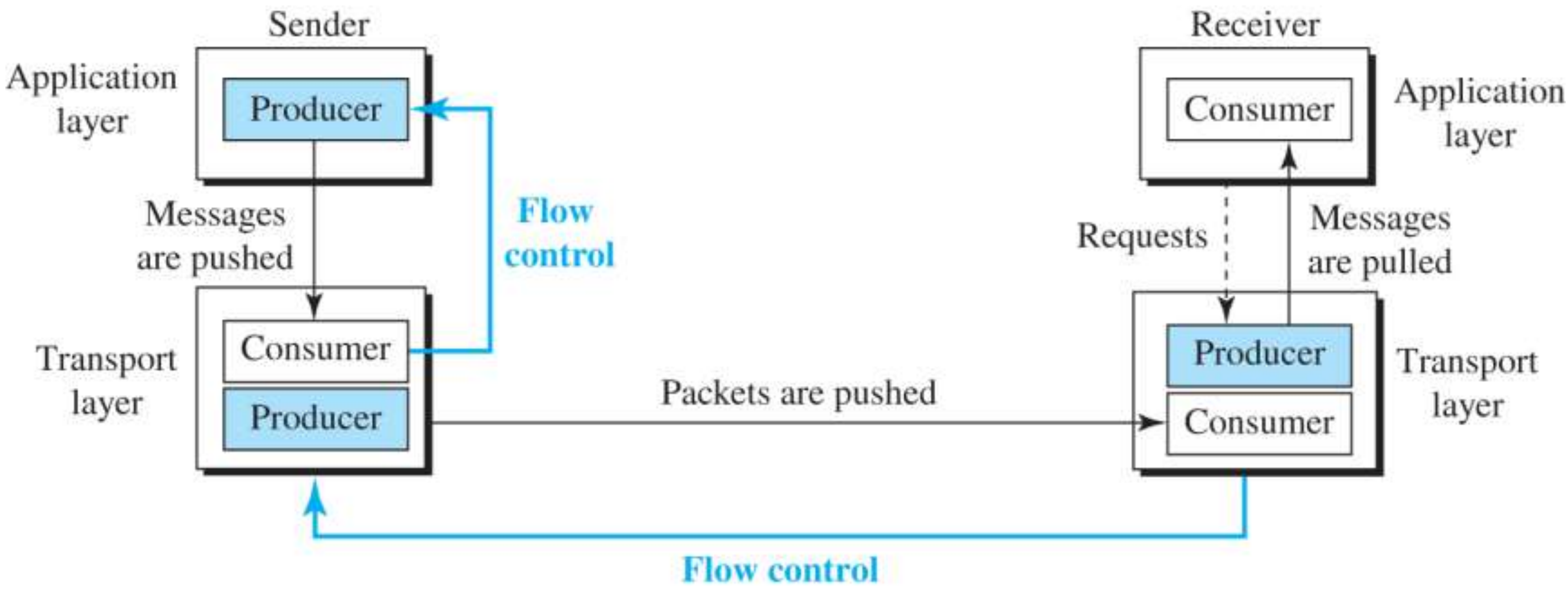


Access the text alternative for slide images.

Handling Error Control

In communication at the transport layer, we are dealing with four entities: sender process, sender transport layer, receiver transport layer, and receiver process. The sending process at the application layer is only a producer. It produces message chunks and pushes them to the transport layer. The sending transport layer has a double role: it is both a consumer and a producer. It consumes the messages pushed by the producer. It encapsulates the messages in packets and pushes them to the receiving transport layer. The receiving transport layer also has a double role, it is the consumer for the packets received from the sender and the producer that decapsulates the messages and delivers them to the application layer.

Figure 9.10 Flow control at the transport layer



[Access the text alternative for slide images.]

Example 9.1

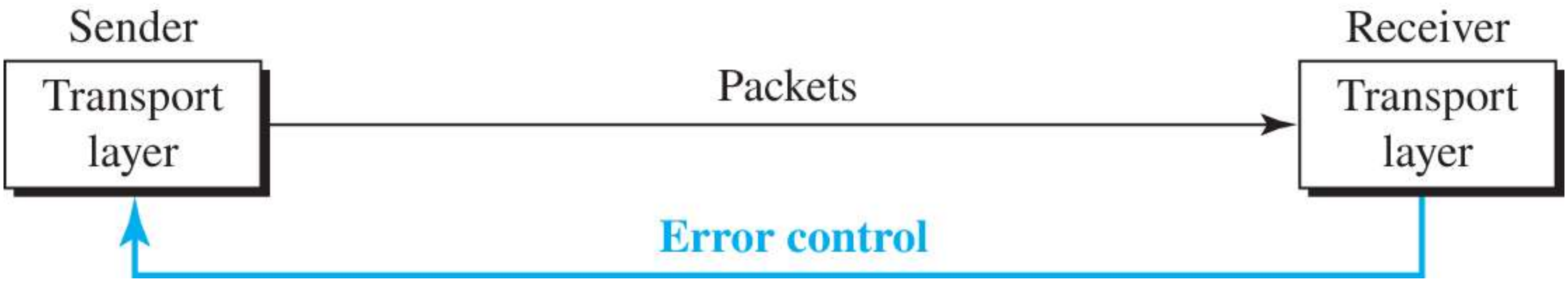
The above discussion requires that the consumers communicate with the producers on two occasions: when the buffer is full and when there are vacancies. If the two parties use a buffer with only one slot, the communication can be easier. Assume that each transport layer uses one single memory location to hold a packet. When this single slot in the sending transport layer is empty, the sending transport layer sends a note to the application layer to send its next chunk; when this single slot in the receiving transport layer is empty, it sends an acknowledgment to the sending transport layer to send its next packet. As we will see later, however, this type of flow control, using a single-slot buffer at the sender and the receiver, is inefficient.

9.1.6 Error Control

In the Internet, since the underlying network layer (IP) is unreliable, we need to make the transport layer reliable if the application requires reliability. Reliability can be achieved to add error control services to the transport layer. Error control at the transport layer is responsible for:

1. *Detecting and discarding corrupted packets.*
2. *Keeping track of lost and discarded packets and resending them.*
3. *Recognizing duplicate packets and discarding them.*
4. *Buffering out-of-order packets until the missing packets arrive.*

Figure 9.11 Error control at the transport layer

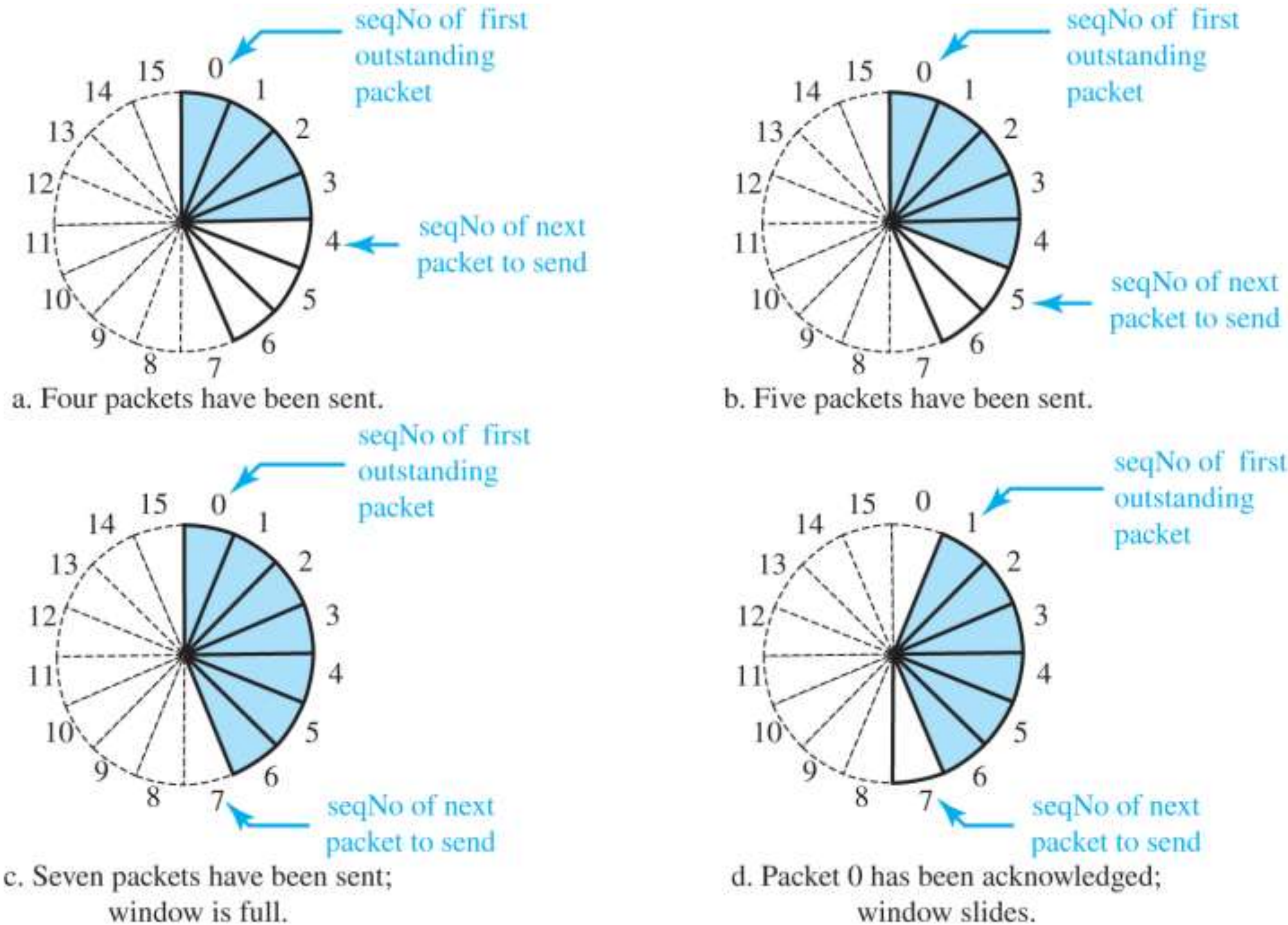


Access the text alternative for slide images.

9.1.7 Combination of Flow and Error Control

We have discussed that flow control requires the use of two buffers, one at the sender site and the other at the receiver site. We have also discussed that error control requires the use of sequence and acknowledgment numbers by both sides. These two requirements can be combined if we use two numbered buffers, one at the sender, one at the receiver.

Figure 9.12 Sliding window in circular format



a. Four packets have been sent.

b. Five packets have been sent.

c. Seven packets have been sent; window is full.

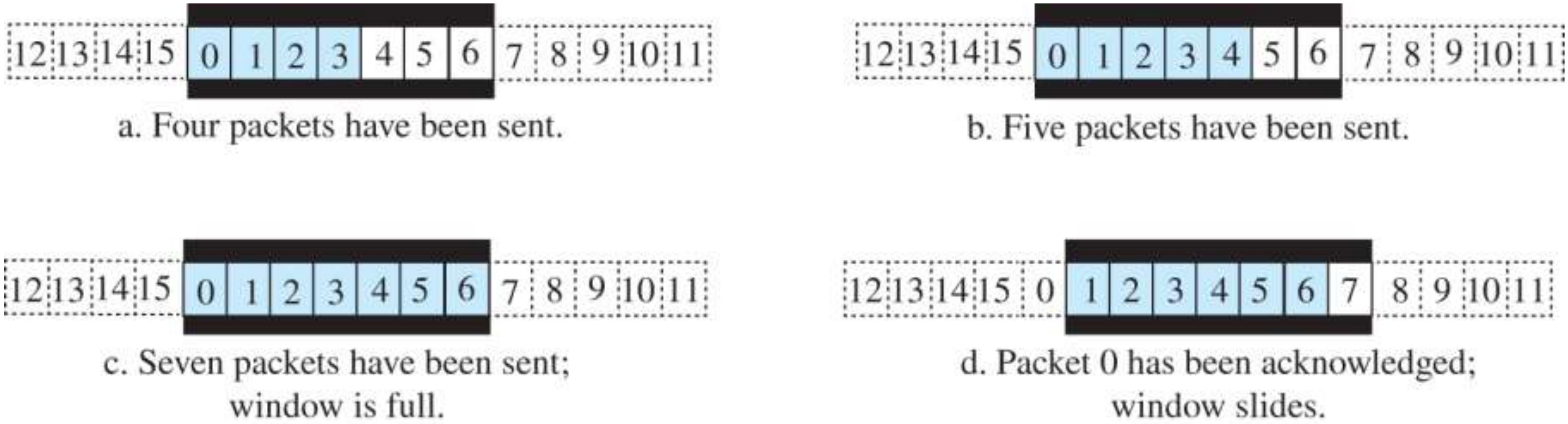
d. Packet 0 has been acknowledged; window slides.

[Access the text alternative for slide images.]

9.1.8 Congestion Control

An important issue in a packet-switched network, such as the Internet, is congestion. Congestion in a network may occur if the load on the network—the number of packets sent to the network—is greater than the capacity of the network—the number of packets a network can handle. Congestion control refers to the mechanisms and techniques that control the congestion and keep the load below the capacity.

Figure 9.13 Sliding window in linear format



Access the text alternative for slide images.

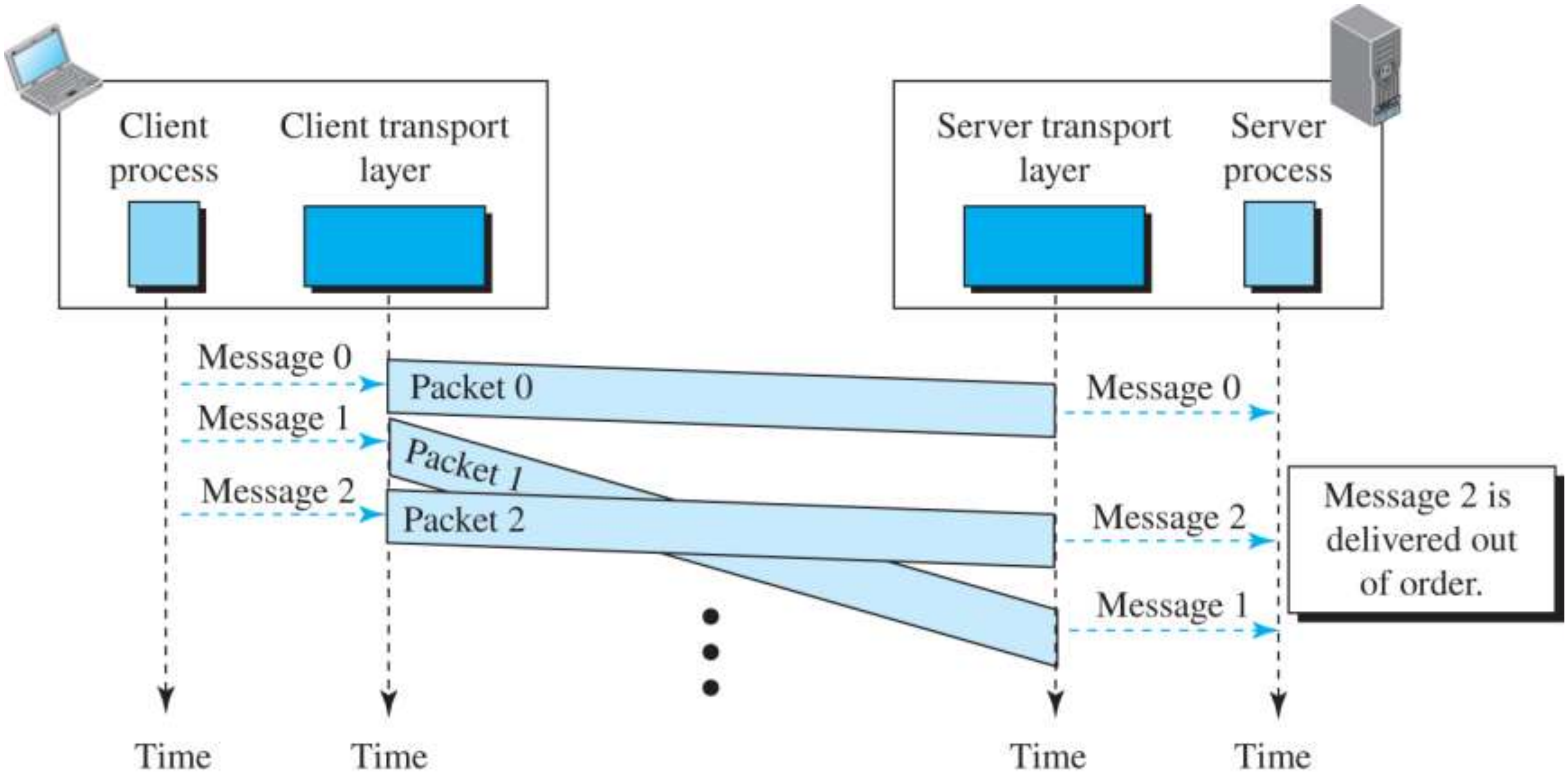
9.1.9 Connectionless and Connection-Oriented Protocol

A transport-layer protocol, like a network-layer protocol, can provide two types of services: connectionless and connection-oriented. The nature of these services at the transport layer, however, is different from the ones at the network layer. At the network layer, a connectionless service may mean different paths for different datagrams belonging to the same message. Connectionless service at the transport layer means independency between packets; connection-oriented means dependency. Let us elaborate on these two services.

Connectionless Service

In a connectionless service, the source process needs to divide its message into chunks of data of the size acceptable by the transport layer and deliver them to the transport layer one by one. The transport layer treats each chunk as a single unit without any relation between the chunks. When a chunk arrives from the application layer, the transport layer encapsulates it in a packet and sends it. (Figure 9.14).

Figure 9.14 Connectionless service

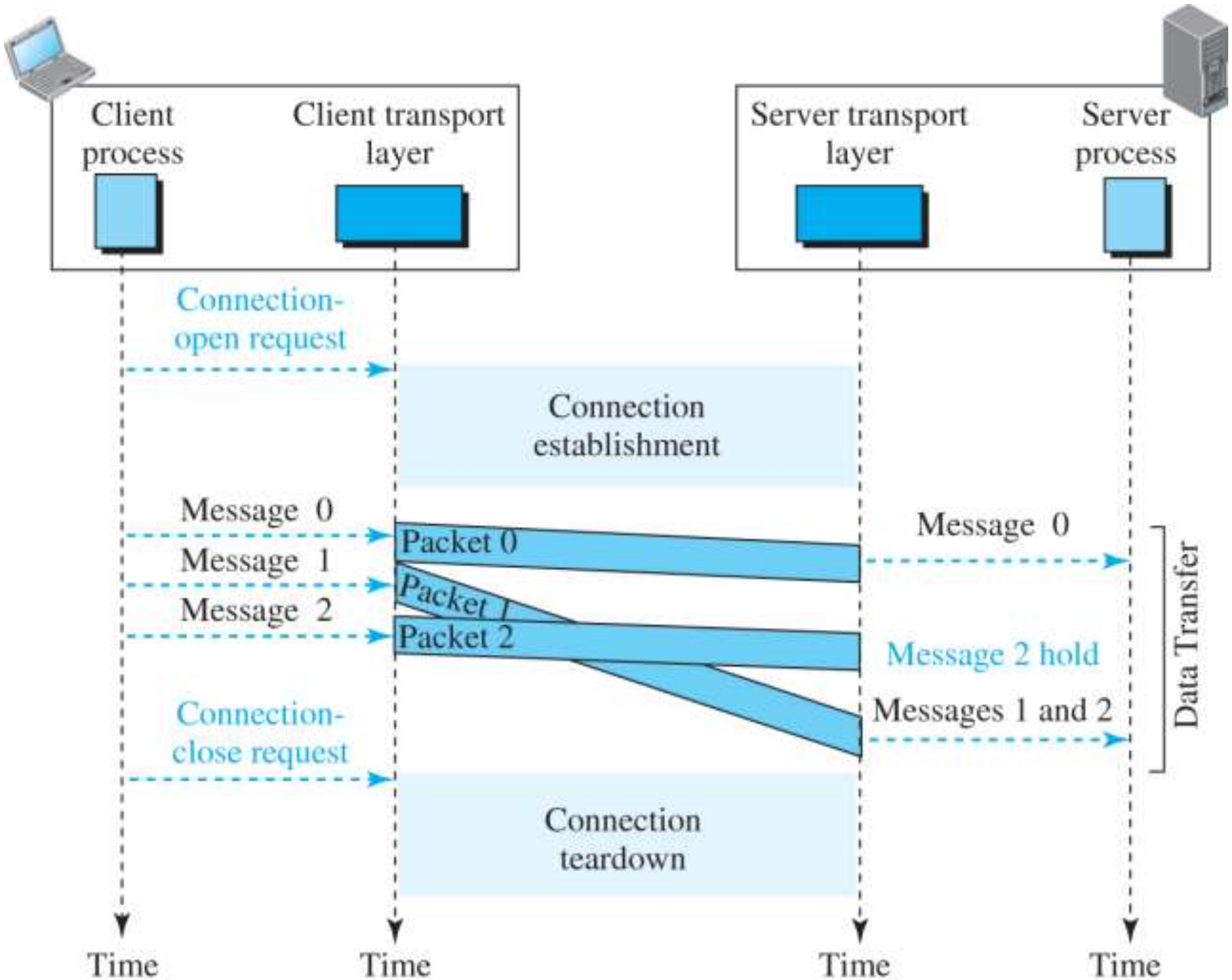


Access the text alternative for slide images.

Connection-Oriented Service 1

In a connection-oriented service, the client and the server first need to establish a logical connection between themselves. The data exchange can only happen after the connection establishment. After data exchange, the connection needs to be torn down (Figure 9.15).

Figure 9.15 Connection-oriented service

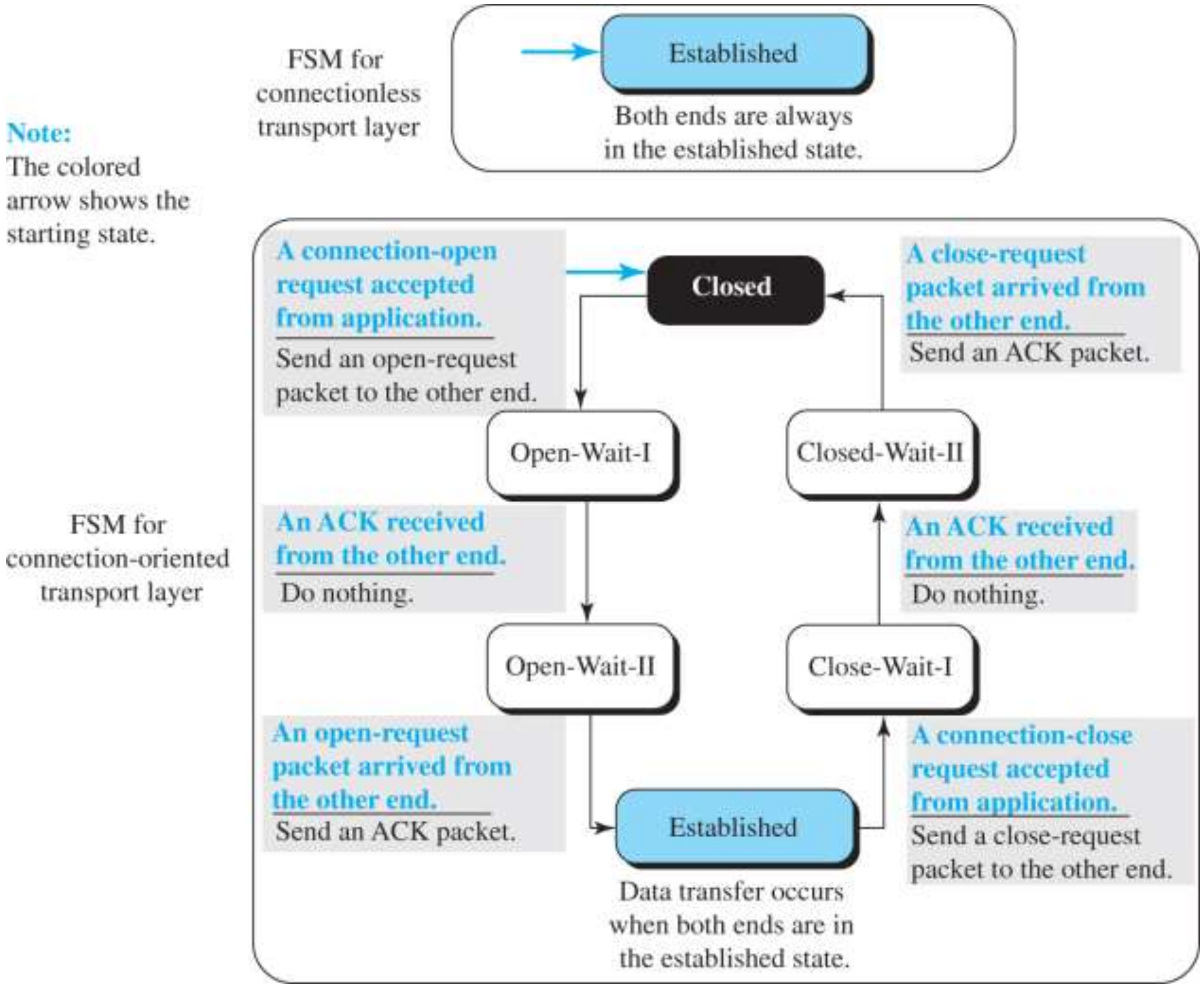


[Access the text alternative for slide images.]

Finite State Machine

The behavior of a transport-layer protocol, both when it provides a connectionless and when it provides a connection-oriented protocol, can be better shown as a finite state machine (FSM). Figure 9.16 shows a representation of a transport layer using an FSM. Using this tool, each transport layer (sender or receiver) is taught as a machine with a finite number of states. The machine is always in one of the states until an event occurs.

Figure 9.16 Connectionless and connection-oriented service represented as FSMs

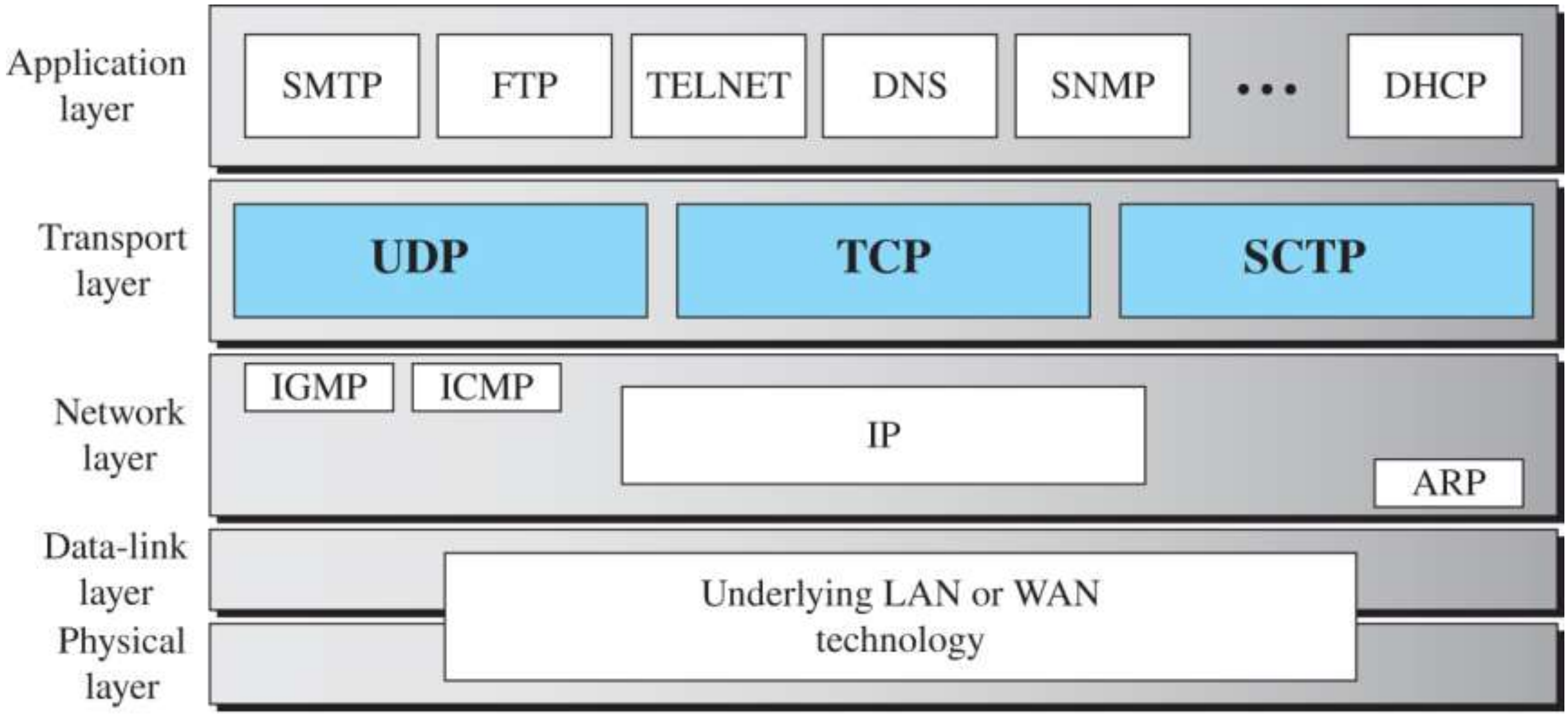


Access the text alternative for slide images.

9-2 TRANSPORT-LAYER PROTOCOLS

After discussing the general principle behind the transport layer in the previous section, we concentrate on the transport protocols in the Internet in this section. shows the position of these three protocols in the TCP/IP protocol suite: UDP, TCP, and SCTP.

Figure 9.17 Position of transport-layer protocols in the TCP/IP protocol suite



Access the text alternative for slide images.

9.2.1 Services

Each protocol provides a different type of service and should be used appropriately.

UDP

UDP is an unreliable connectionless transport-layer protocol used for its simplicity and efficiency in applications where error control can be provided by the application-layer process.

TCP

TCP is a reliable connection-oriented protocol that can be used in any application where reliability is important.

SCTP

SCTP is a new transport-layer protocol that combines the features of UDP and TCP.

9.2.2 Port Numbers

A transport-layer protocol usually has several responsibilities. One is to create a process-to-process communication; these protocols use port numbers to accomplish this. Port numbers provide end-to-end addresses at the transport layer and allow multiplexing and de-multiplexing at this layer, just as IP addresses do at the network layer. Table 9.1 gives some common port numbers for all three protocols we discuss in this chapter.

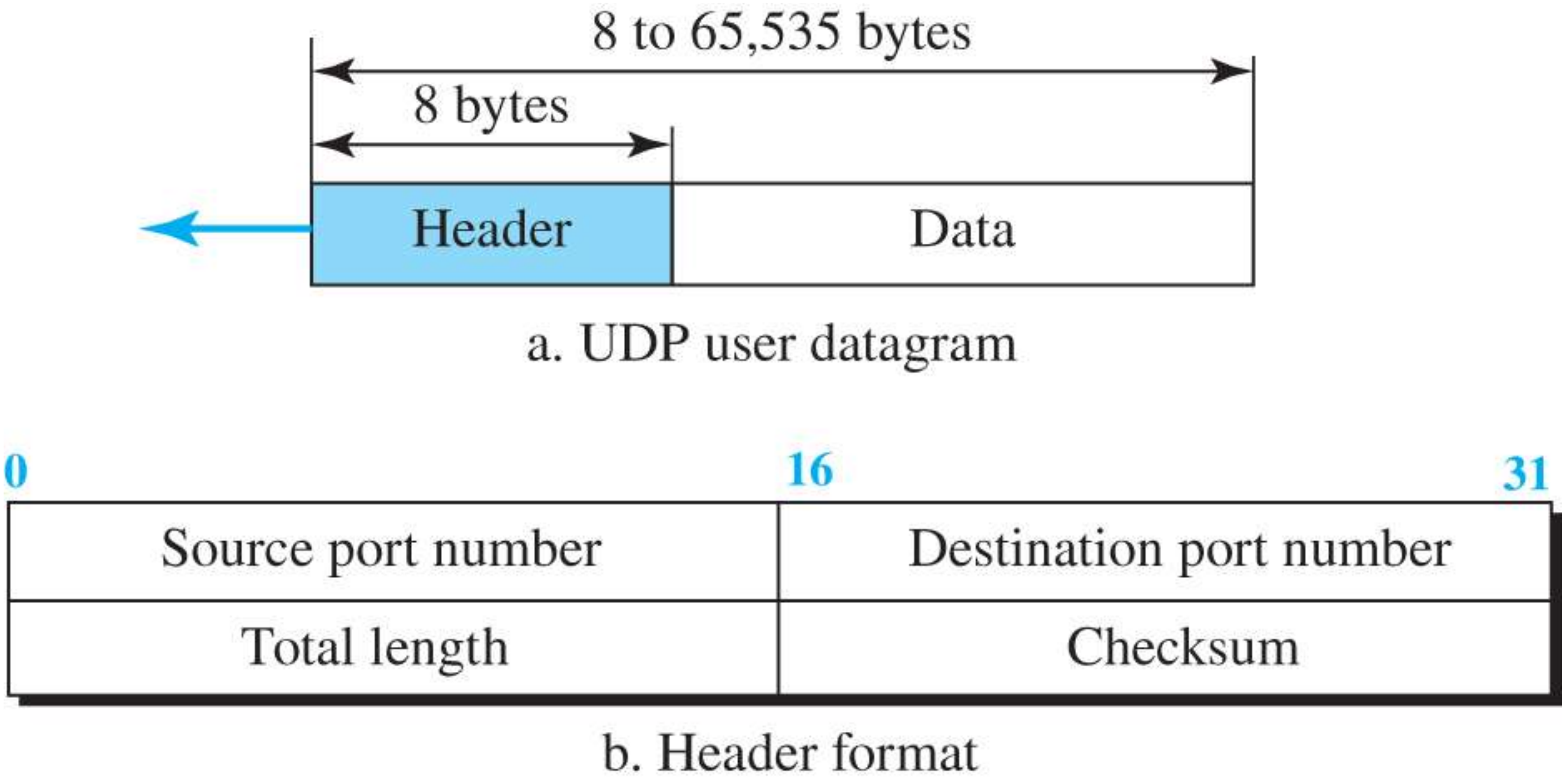
Table 9.1 Some well-known ports used with UDP and TCP

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 System
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

9-3 USER DADAGRAM PROTOCOL (UDP)

The User Datagram Protocol (UDP) is a connectionless, unreliable transport protocol. If UDP is so powerless, why would a process want to use it? With the disadvantages come some advantages. UDP is a very simple protocol using a minimum of overhead.

Figure 9.18 User datagram packet format



[Access the text alternative for slide images.]

Example 9.2 (1)

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?

Example 9.2 (2)

Solution

a. The source port number is the first four hexadecimal digits $(CB84)_{16}$ or 52100
b. The destination port number is the second four hexadecimal digits $(000D)_{16}$ or 13.
c. The third four hexadecimal digits $(001C)_{16}$ define the length of the whole UDP packet as 28 bytes.
d. The length of the data is the length of the whole packet minus the length of the header, or $28 - 8 = 20$ bytes.
e. Since the destination port number is 13 (well-known port), the packet is from the client to the server.
f. The client process is the Daytime.

9.3.1 UDP Services

Earlier we discussed the general services provided by a transport-layer protocol. In this section, we discuss what portions of those general services are provided by UDP.

Process-to-Process Communication 1

UDP provides process-to-process communication using socket addresses, a combination of IP addresses and port numbers.

Connectionless Services

As mentioned previously, UDP provides a connectionless service. This means that each user datagram sent by UDP is an independent datagram. There is no relationship between the different user datagrams even if they are coming from the same source process and going to the same destination program. The user datagrams are not numbered.

Flow Control

UDP is a very simple protocol. There is no flow control, and hence no window mechanism. The receiver may overflow with incoming messages. The lack of flow control means that the process using UDP should provide for this service, if needed.

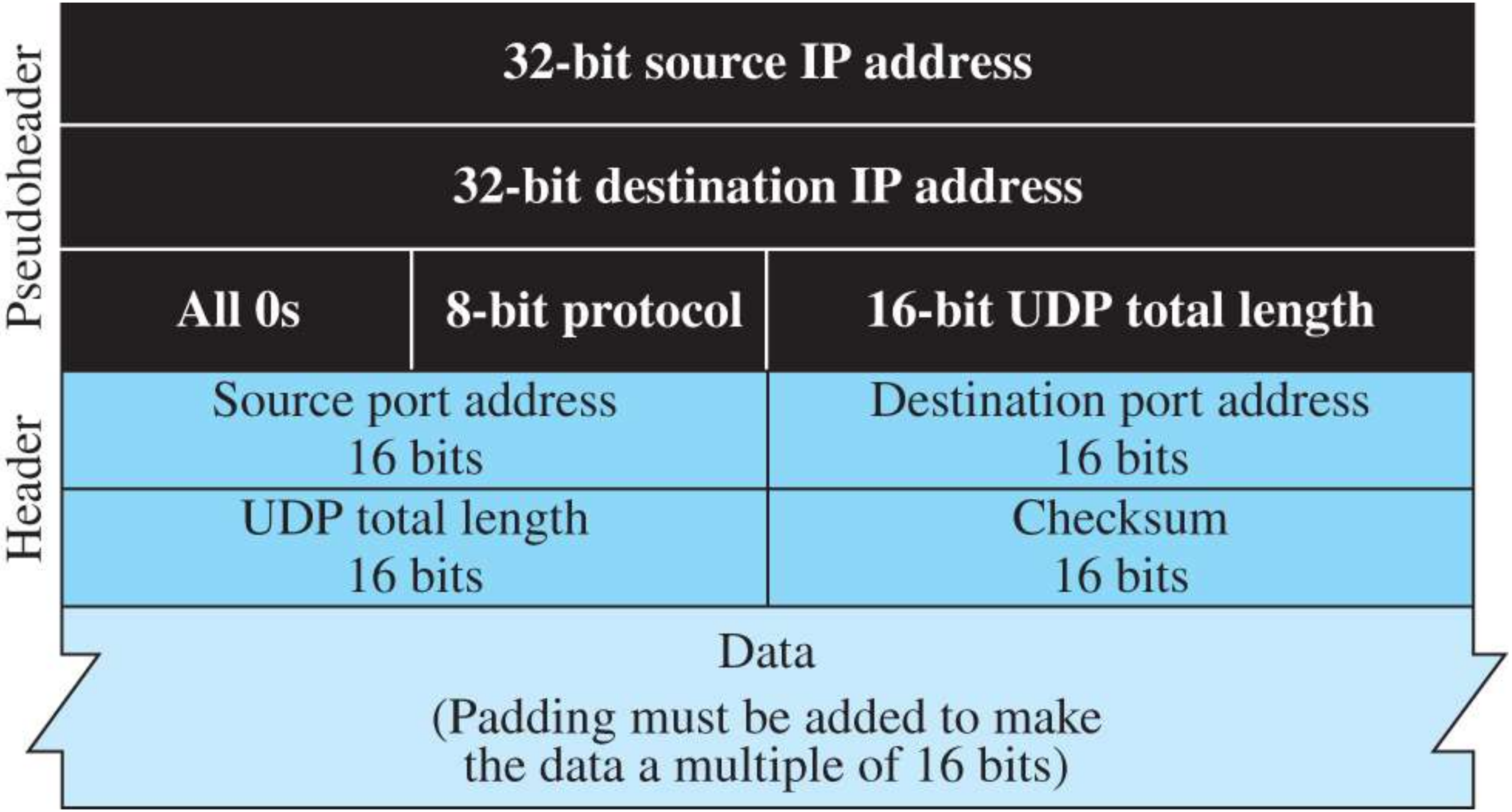
Error Control

There is no error control mechanism in UDP except for the checksum. This means that the sender does not know if a message has been lost or duplicated. When the receiver detects an error through the checksum, the user datagram is silently discarded. The lack of error control means that the process using UDP should provide for this service, if needed.

Checksum 1

UDP checksum calculation includes three sections: a pseudo-header, the UDP header, and the data coming from the application layer. The pseudo-header is the part of the header of the IP packet in which the user datagram is to be encapsulated with some fields filled with 0s.

Figure 9.19 Pseudoheader for checksum calculation



Access the text alternative for slide images.

Example 9.3

What value is sent for the checksum in one of the following hypothetical situations?

a. The sender decides not to include the checks

b. The sender decides to include the checksum, but the value of the sum is all 1s.

c. The sender decides to include the checksum, but the value of the sum is all 0s.

Example 9.3 Solution

a. The value sent for the checksum field is all 0s to show that the checksum is not -calculated.

b. When the sender complements the sum, the result is all 0s; the sender complements the result again before sending. The value sent for the checksum is all 1s. The second -complement operation is needed to avoid confusion with the case in part a.

c. This situation never happens because it implies that the value of every term included in the calculation of the sum is all 0s, which is impossible; some fields in the pseudo-header have nonzero values.

Congestion Control 1

Since UDP is a connectionless protocol, it does not provide congestion control. UDP assumes that the packets sent are small and sporadic and cannot create congestion in the network. This assumption may or may not be true today, when UDP is used for interactive real-time transfer of audio and video.

Encapsulation and Decapsulation

To send a message from one process to another, the UDP protocol encapsulates and decapsulates messages.

Queuing

- *We have talked about ports without discussing the actual implementation of them. In UDP, queues are associated with ports.*
- *At the client site, when a process starts, it requests a port number from the operating system. Some implementations create both an incoming and an outgoing queue associated with each process. Other implementations create only an incoming queue associated with each process.*

Multiplexing and Demultiplexing [1]

In a host running a TCP/IP protocol suite, there is only one UDP but possibly several processes that may want to use the services of UDP. To handle this situation, UDP multiplexes and demultiplexes.

Comparison

We can compare UDP with the connectionless simple protocol we discussed earlier. The only difference is that UDP provides an optional checksum to detect corrupted packets at the receiver site. If the checksum is added to the packet, the receiving UDP can check the packet and discard the packet if it is corrupted. No feedback, however, is sent to the sender.

9.3.2 UDP Applications

Although UDP meets almost none of the criteria we mentioned earlier for a reliable transport-layer protocol, UDP is preferable for some applications. The reason is that some services may have some side effects that are either unacceptable or not preferable. An application designer sometimes needs to compromise to get the optimum. For example, in our daily life, we all know that a one-day delivery of a package by a carrier is more expensive than a three-day delivery. Although high speed and low cost are both desirable features in delivery of a parcel, they are in conflict with each other.

UDP Features

We briefly discuss some features of UDP and their advantages and disadvantages.

Example 9.4

A client-server application such as DNS uses the services of UDP because a client needs to send a short request to a server and to receive a quick response from it. The request and response can each fit in one user datagram. Since only one message is exchanged in each direction, the connectionless feature is not an issue; the client or server does not worry that messages are delivered out of order.

Example 9.5

A client-server application such as SMTP, which is used in electronic mail, cannot use the services of UDP because a user can send a long e-mail message, which may include multimedia (images, audio, or video). If the application uses UDP and the message does not fit in one single user datagram, the message must be split by the application into different user datagrams. Here the connectionless service may create problems. The user datagrams may arrive and be delivered to the receiver application out of order. The receiver application may not be able to reorder the pieces. This means the connectionless service has a disadvantage for an application program that sends long messages.

Example 9.6

Assume we are downloading a very large text file from the Internet. We definitely need to use a transport layer that provides reliable service. We don’t want part of the file to be missing or corrupted when we open the file. The delay created between the deliveries of the parts is not an overriding concern for us; we wait until the whole file is composed before looking at it. In this case, UDP is not a suitable transport layer.

Example 9.7

Assume we are using a real-time interactive application, such as Skype. Audio and video are divided into frames and sent one after another. If the transport layer is supposed to resend a corrupted or lost frame, the synchronizing of the whole transmission may be lost. The viewer suddenly sees a blank screen and needs to wait until the second transmission arrives. This is not tolerable. However, if each small part of the screen is sent using one single user datagram, the receiving UDP can easily ignore the corrupted or lost packet and deliver the rest to the application program. That part of the screen is blank for a very short period of time, which most viewers do not even notice.

Typical Applications

- *UDP is suitable for a process that requires simple request-response communication with little concern for flow and error control.*
- *UDP is suitable for a process with internal flow- and error-control mechanisms.*
- *UDP is a suitable transport protocol for multicasting.*
- *UDP is used for management processes such as SNMP.*
- *UDP is used for some route updating protocols such as Routing Information Protocol (RIP).*
- *UDP is normally used for interactive real-time applications that cannot tolerate uneven delay between sections of a received message.*

9-4 TRANSMISSION CONTROL PROTOCOL

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.

9.4.1 TCP Services

Before discussing TCP in detail, let us explain the services offered by TCP to the processes at the application layer.

Process-to-Process Communication 2

Before discussing TCP in detail, let us explain the services offered by TCP to the processes at the application layer.

Stream Delivery Service

TCP allows the sending process to deliver data as a stream of bytes and allows the receiving process to obtain data as a stream of bytes. TCP creates an environment in which the two processes seem to be connected by an imaginary "tube" that carries their bytes across the Internet. This imaginary environment is depicted in Figure 9.20. The sending process produces (writes to) the stream and the receiving process consumes (reads from) it.

Figure 9.20 Stream delivery

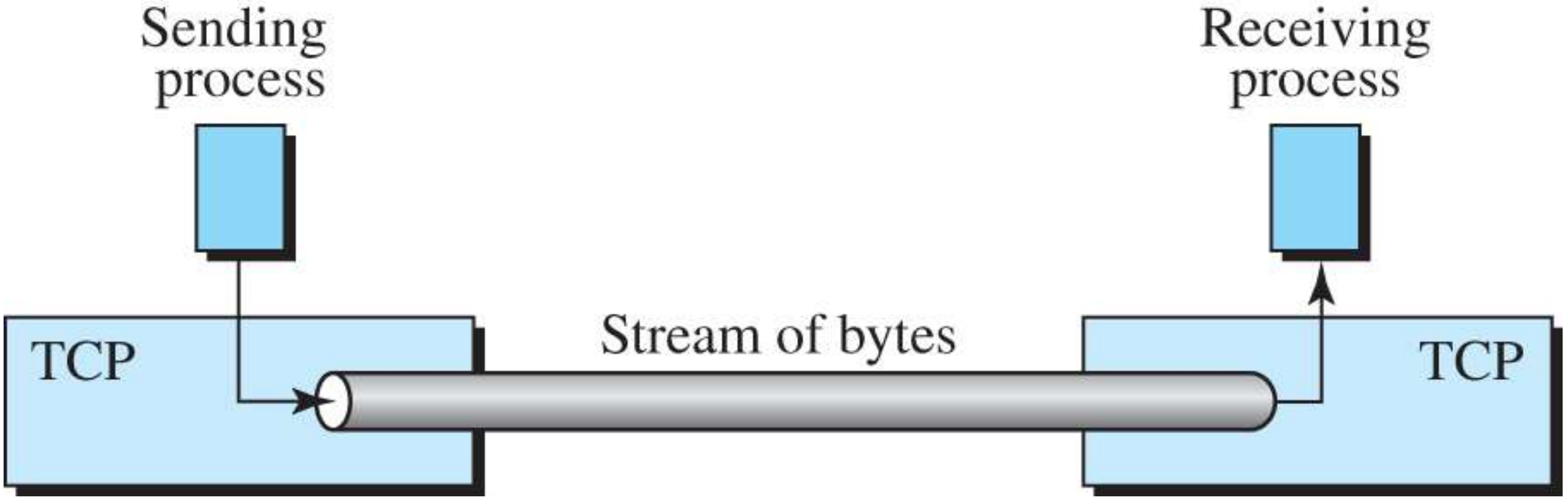
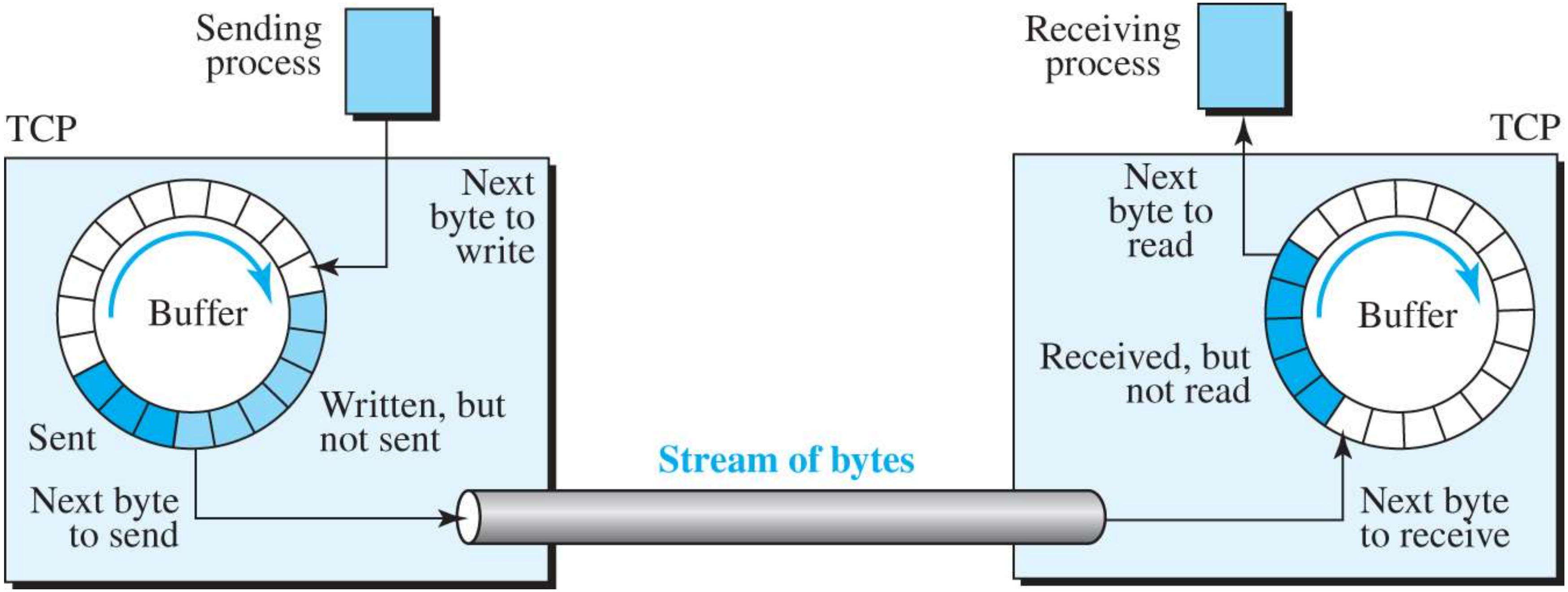
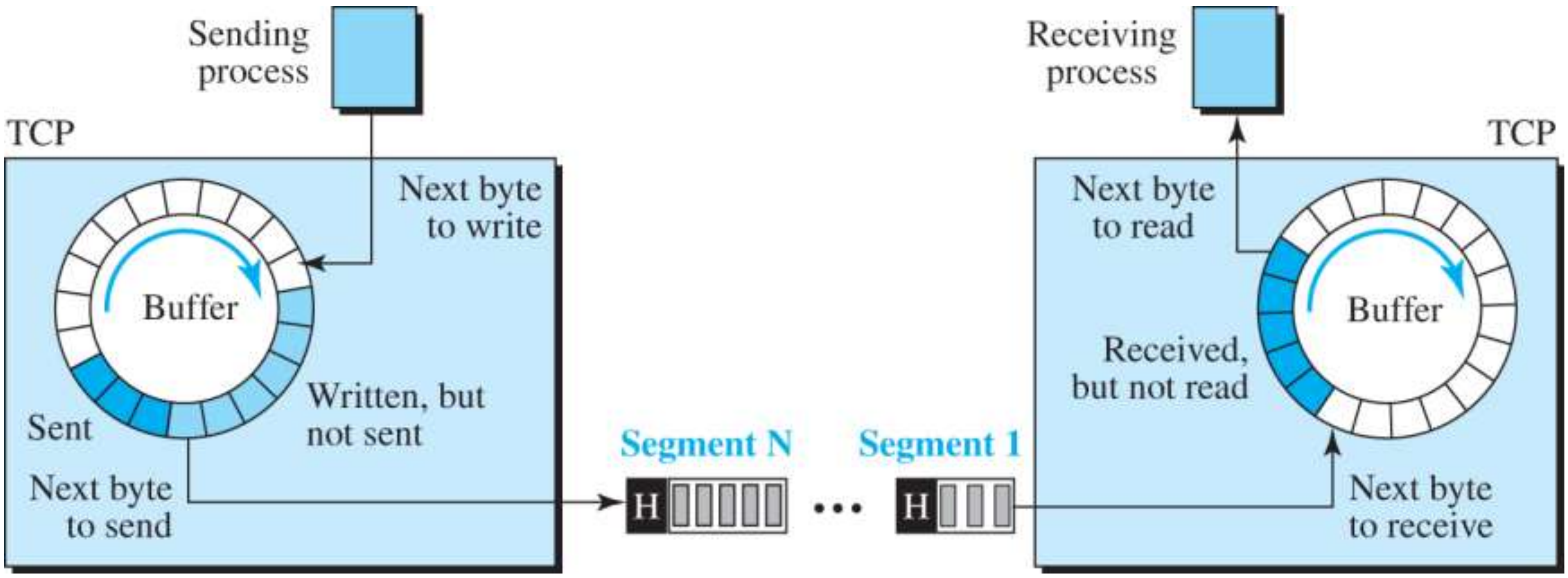


Figure 9.21 Sending and receiving buffers



[Access the text alternative for slide images.]

Figure 9.22 TCP segments



Access the text alternative for slide images.

Full-Duplex Communication [1]

TCP offers full-duplex service, where data can flow in both directions at the same time. Each TCP endpoint then has its own sending and receiving buffer, and segments move in both directions.

Multiplexing and Demultiplexing 2

Like UDP, TCP performs multiplexing at the sender and demultiplexing at the receiver. However, since TCP is a connection-oriented protocol, a connection needs to be established for each pair of processes.

Connection-Oriented Service *2*

TCP, unlike UDP, is a connection-oriented protocol. When a process at site A wants to send to and receive data from another process at site B, the following three phases occur:

1. *The two TCP's establish a logical connection.*
2. *Data are exchanged in both directions.*
3. *The connection is terminated.*

Reliable Service 1

TCP is a reliable transport protocol. It uses an acknowledgment mechanism to check the safe and sound arrival of data. We will discuss this feature further in the section on error control.

9.4.2 TCP Features

To provide the services mentioned in the previous section, TCP has several features that are briefly summarized in this section and discussed later in detail.

Numbering System

Although the TCP software keeps track of the segments being transmitted or received, there is no field for a segment number value in the segment header. Instead, there are two fields, called the sequence number and the acknowledgment number. These two fields refer to a byte number and not a segment number.

Example 9.8

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

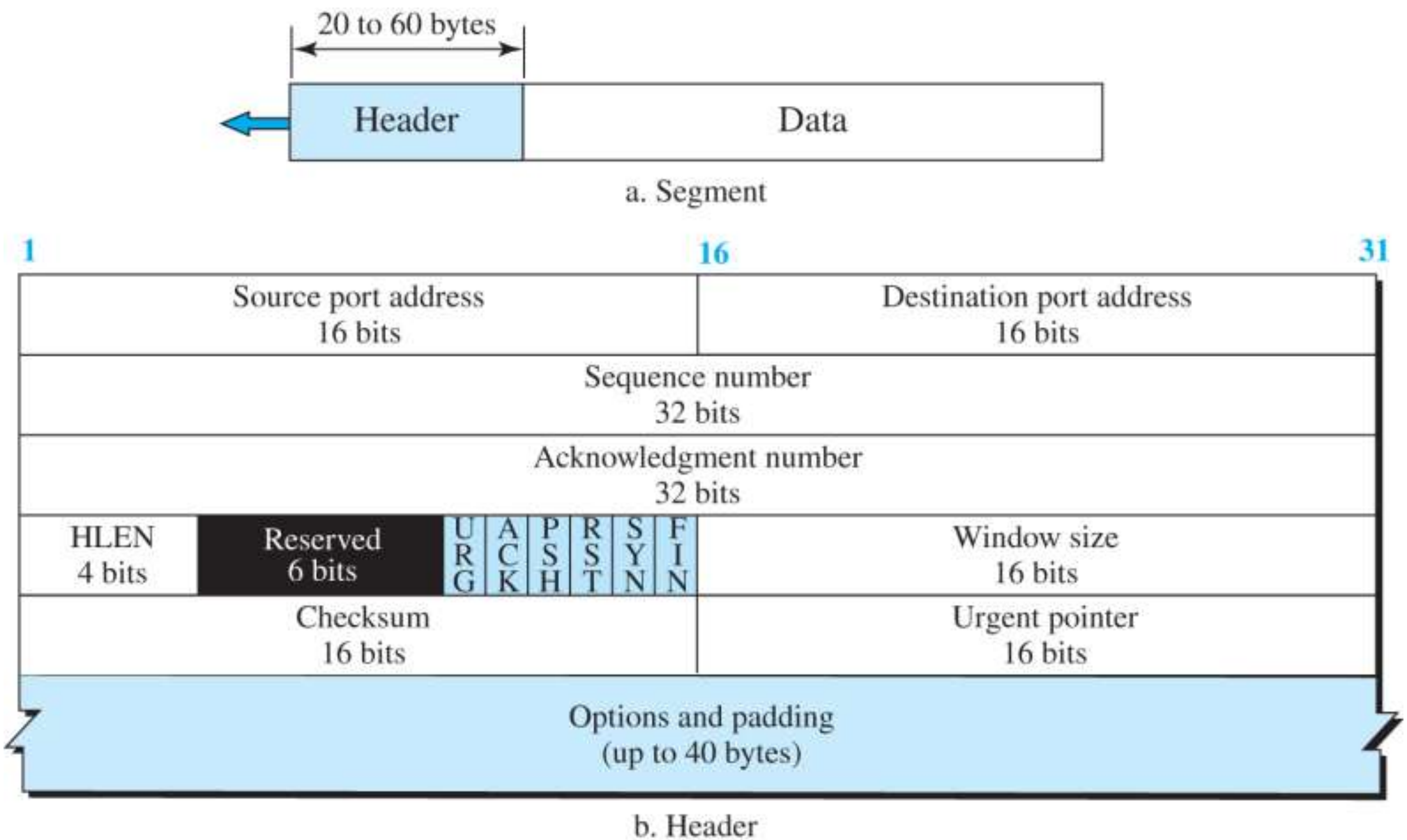
9.4.3 Segment

Before discussing TCP in more detail, let us discuss the TCP packets themselves. A packet in TCP is called a segment.

Format

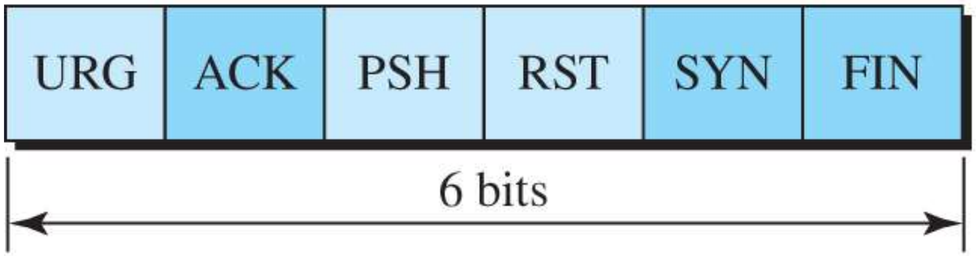
The format of a segment is shown in Figure 9.23. The segment consists of a header of 20 to 60 bytes, followed by data from the application program. The header is 20 bytes if there are no options and up to 60 bytes if it contains options. We will discuss some of the header fields in this section. The meaning and purpose of these will become clearer as we proceed through the section.

Figure 9.23 TCP segment format



Access the text alternative for slide images.

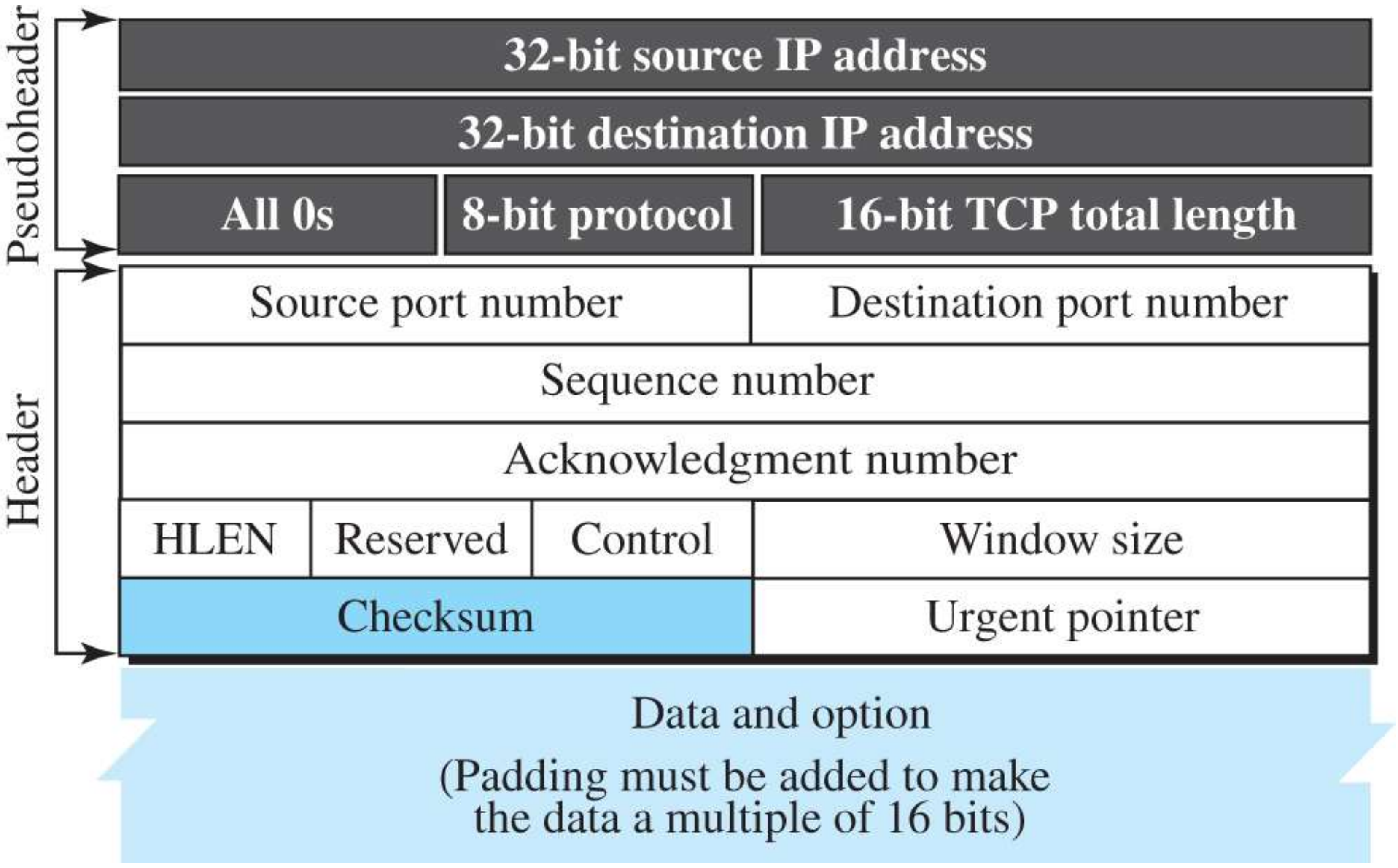
Figure 9.24 Control field



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

[Access the text alternative for slide images.](#)

Figure 9.25 Pseudoheader added to the TCP datagram



Access the text alternative for slide images.

Encapsulation

A TCP segment encapsulates the data received from the application layer. The TCP segment is encapsulated in an IP datagram, which in turn is encapsulated in a frame at the data-link layer.

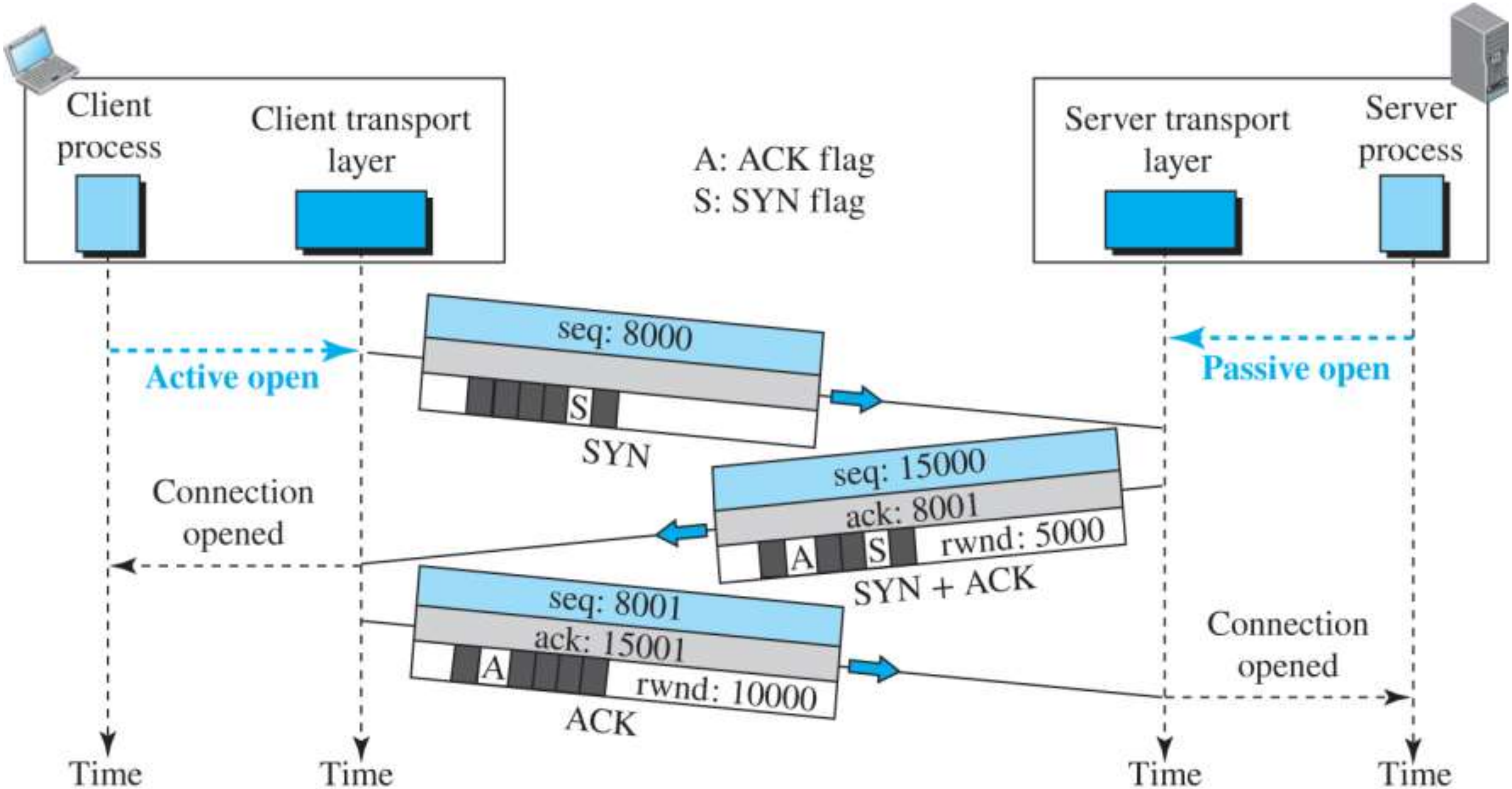
9.4.4 A TCP Connection

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. You may wonder 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

TCP transmits data in full-duplex mode. When two TCPs in two machines are connected, they are able to send segments to each other simultaneously. This implies that each party must initialize communication and get approval from the other party before any data are transferred.

Figure 9.26 Connection establishment using three-way handshaking

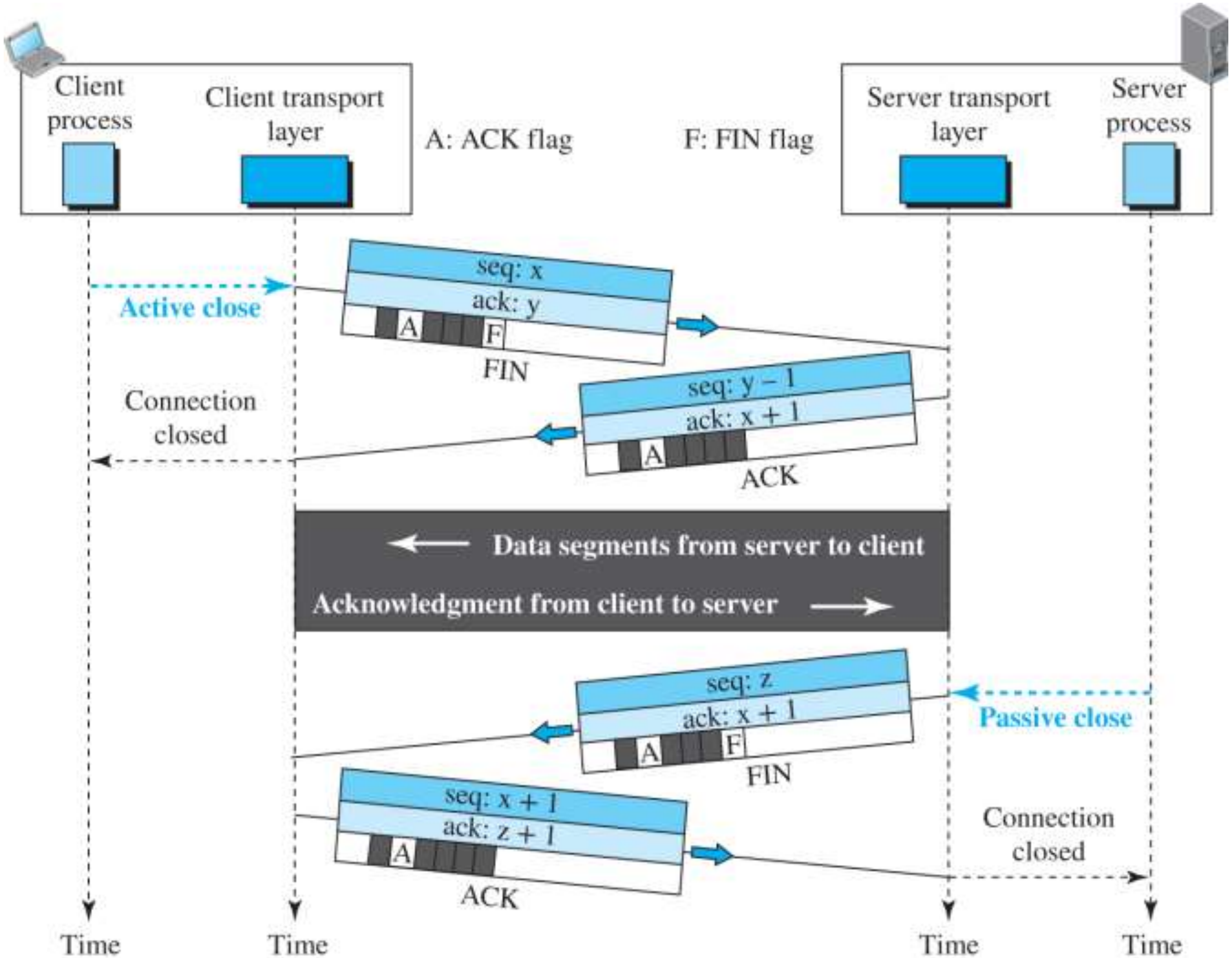


Access the text alternative for slide images.

Data Transfer 1

After connection is established, bidirectional data transfer can take place. The client and server can send data and acknowledgments in both directions. We will study the rules of acknowledgment later in the chapter; for the moment, it is enough to know that data traveling in the same direction as an acknowledgment are carried on the same segment. The acknowledgment is piggybacked with the data. Figure 9.27 shows an example.

Figure 9.27 Data transfer

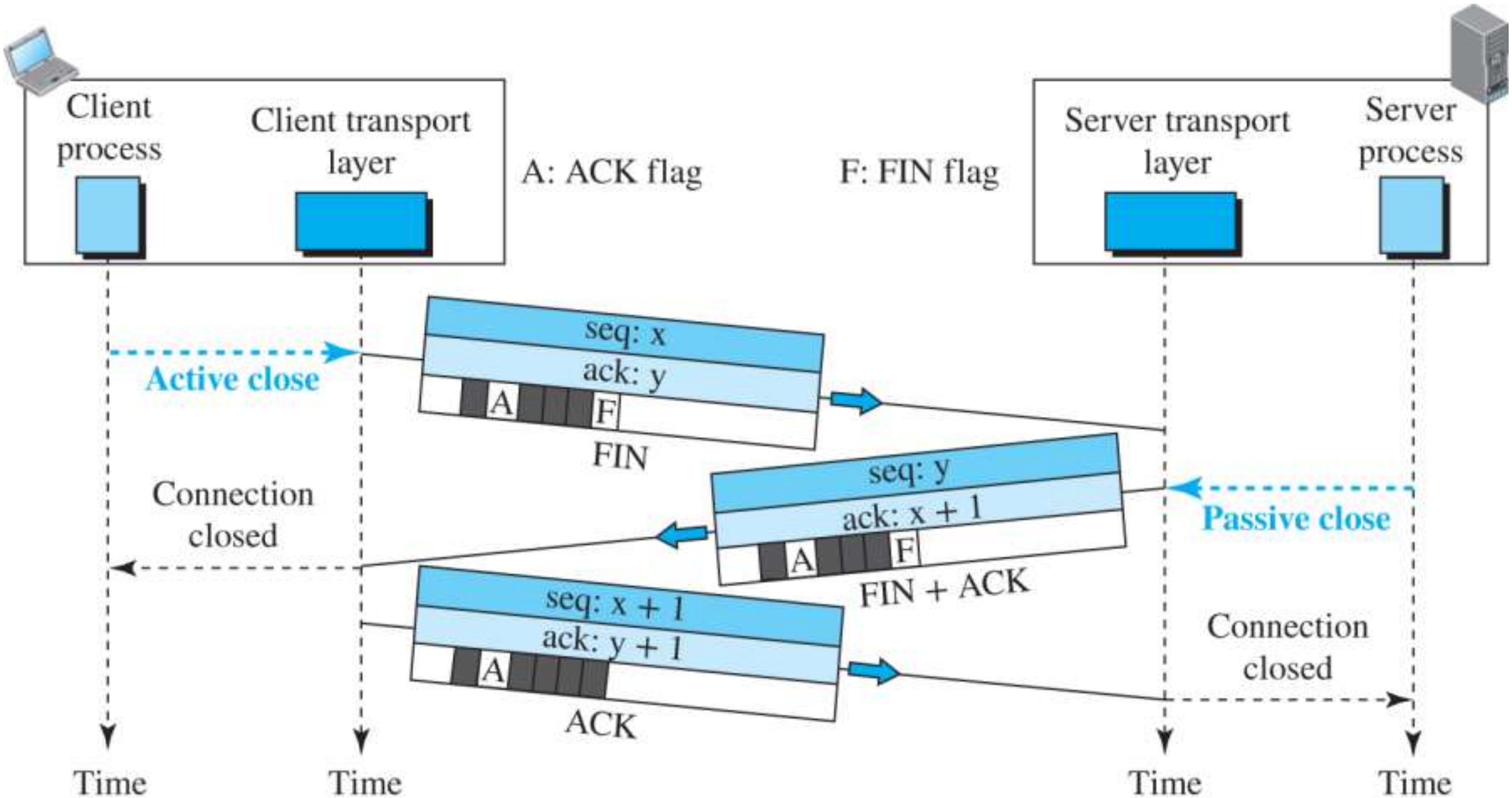


Access the text alternative for slide images.

Connection Termination

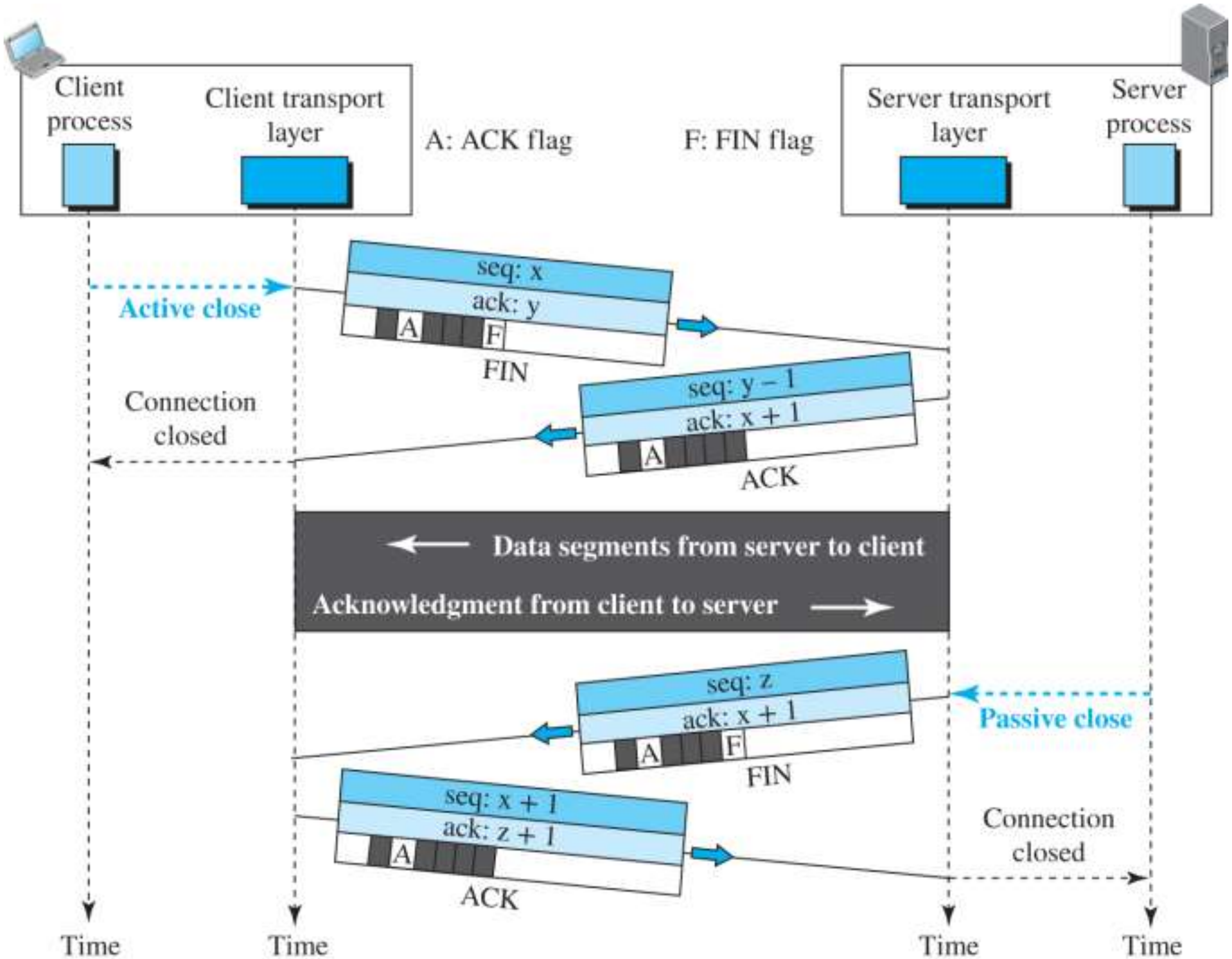
Either of the two parties involved in exchanging data (client or server) can close the connection, although it is usually initiated by the client. Most implementations today allow two options for connection termination: three-way handshaking and four-way handshaking with a half-close option.

Figure 9.28 Connection termination using three-way handshaking



Access the text alternative for slide images.

Figure 9.29 Half-close



Access the text alternative for slide images.

Connection Reset

TCP at one end may deny a connection request, may abort an existing connection, or may terminate an idle connection. All of these are done with the RST (reset) flag.

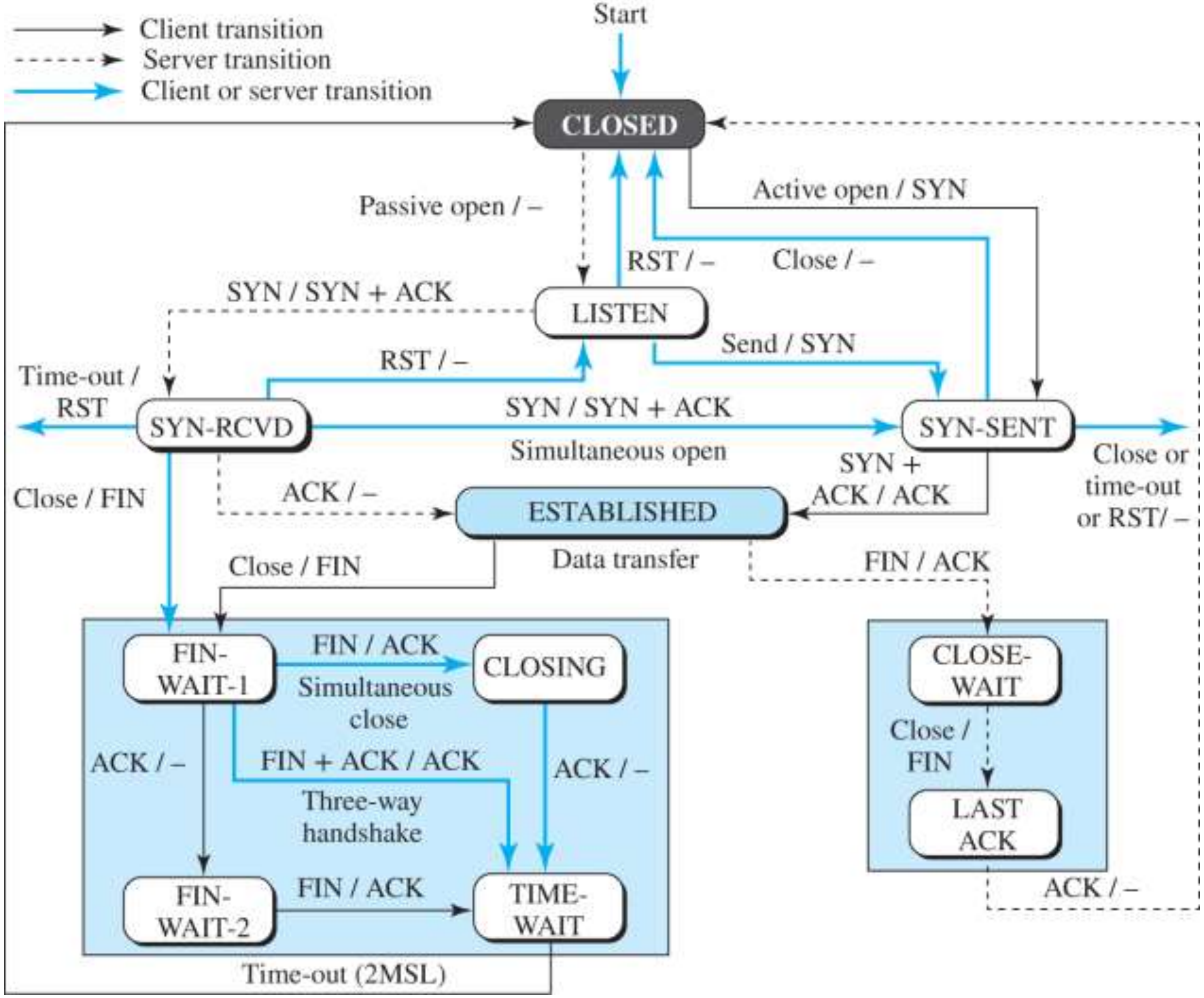
9.4.5 State Transition Diagram

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) as shown in Figure 9.30.

Scenarios

To understand the TCP state machines and the transition diagrams, we go through one scenario in this section.

Figure 9.30 State transition diagram

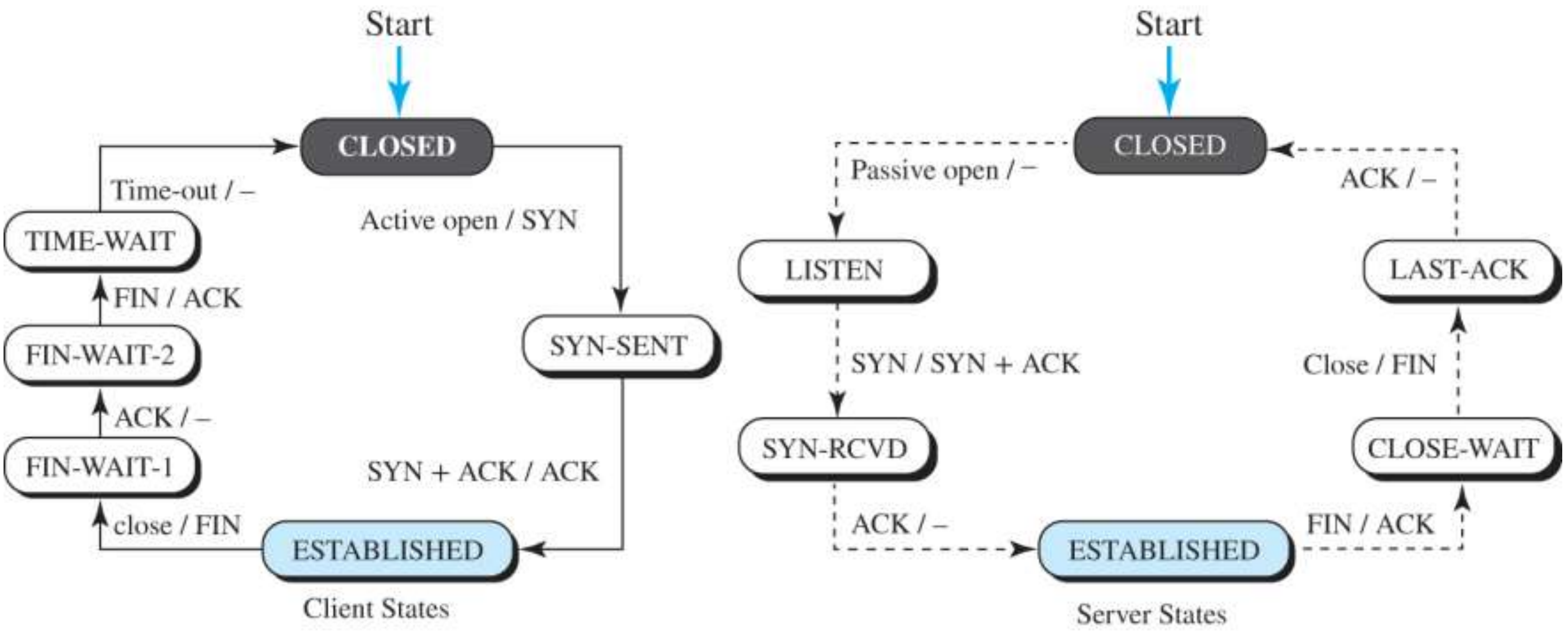


Access the text alternative for slide images.

Table 9.2 States for TCP

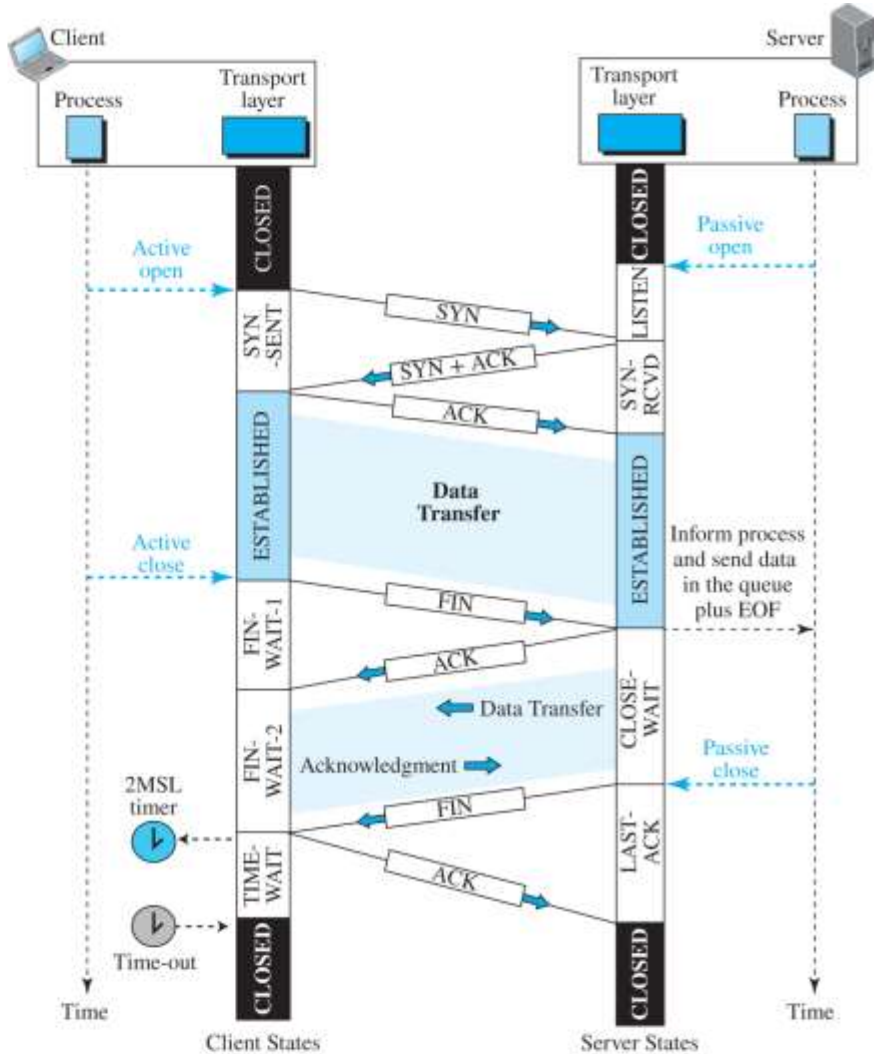
State	*Description*
CLOSED	No connection exists
LISTEN	Passive open received; waiting for SYN
SYN-SENT	SYN sent; waiting for ACK
SYN-RCVD	SYN + ACK sent; waiting for ACK
ESTABLISHED	Connection established; data transfer in progress
FIN-WAIT-1	First FIN sent; waiting for ACK
FIN-WAIT-2	ACK to first FIN 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

Figure 9.31 Transition diagram with half-close connection termination



Access the text alternative for slide images.

Figure 9.32 Time-line diagram for a common scenario



[Access the text alternative for slide images.]

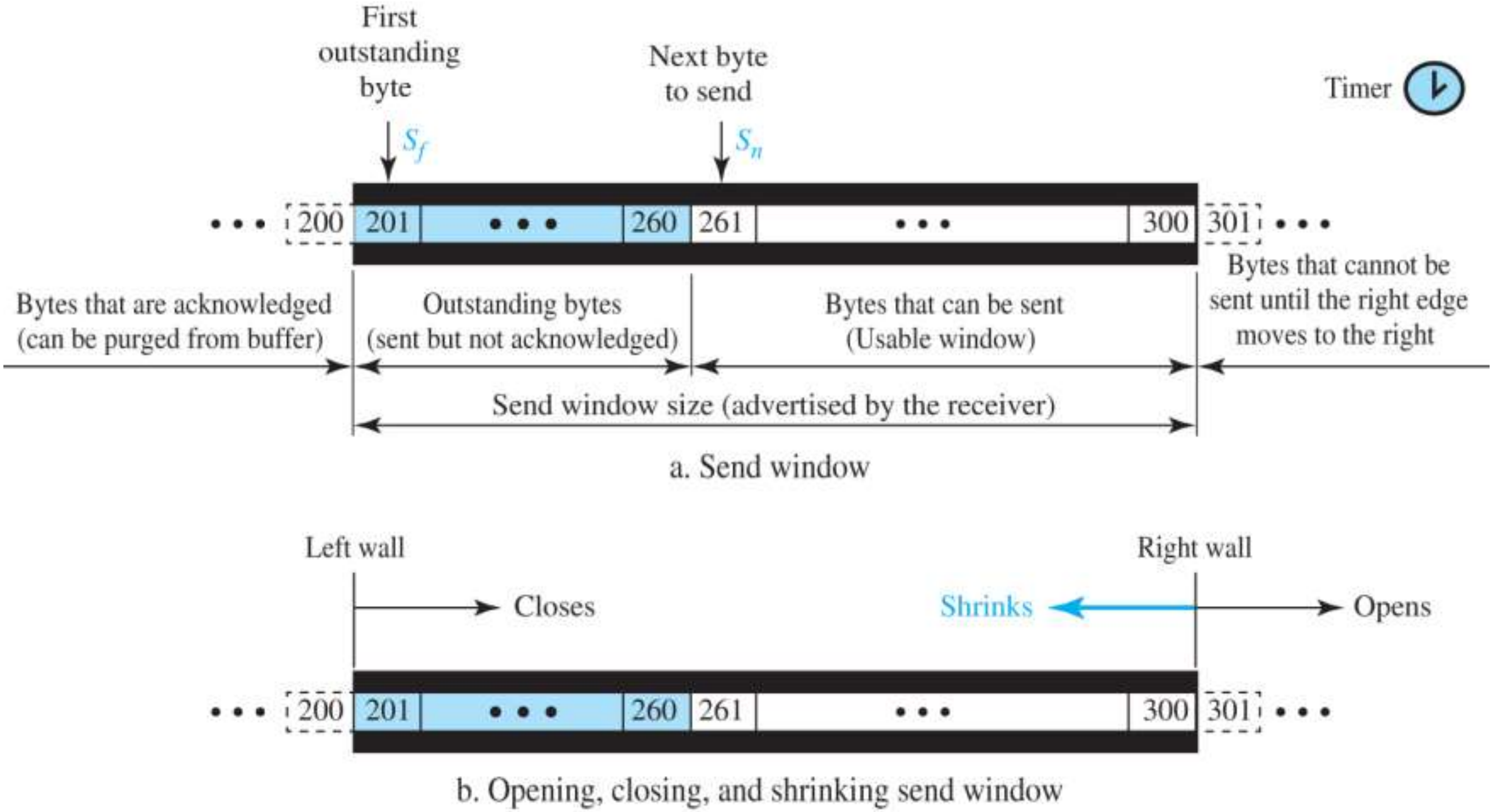
9.4.6 Windows in TCP

Before discussing data transfer in TCP and the issues such as flow, error, and congestion control, we describe the windows used 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

Figure 9.33 shows an example of a send window. The window size is 100 bytes but later we see that the send window size is dictated by the receiver (flow control) and the congestion in the underlying network (congestion control). The figure shows how a send window opens, closes, or shrinks.

Figure 9.33 Send window in TCP

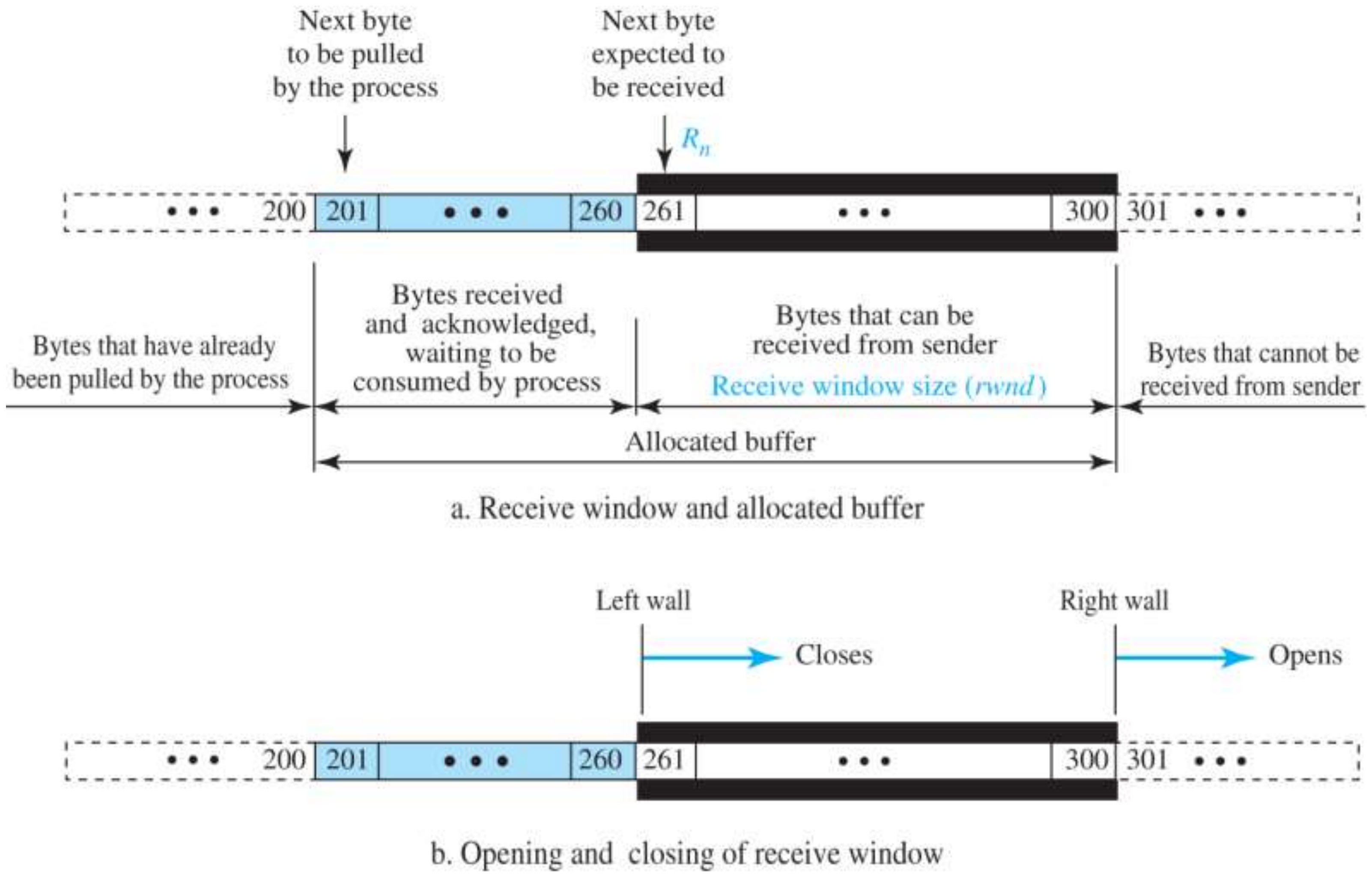


[Access the text alternative for slide images.]

Receive Window

Figure 9.34 shows an example of a receive window. The window size is 100 bytes. The figure also shows how the receive window opens and closes; in practice, the window should never shrink.

Figure 9.34 Receive window in TCP

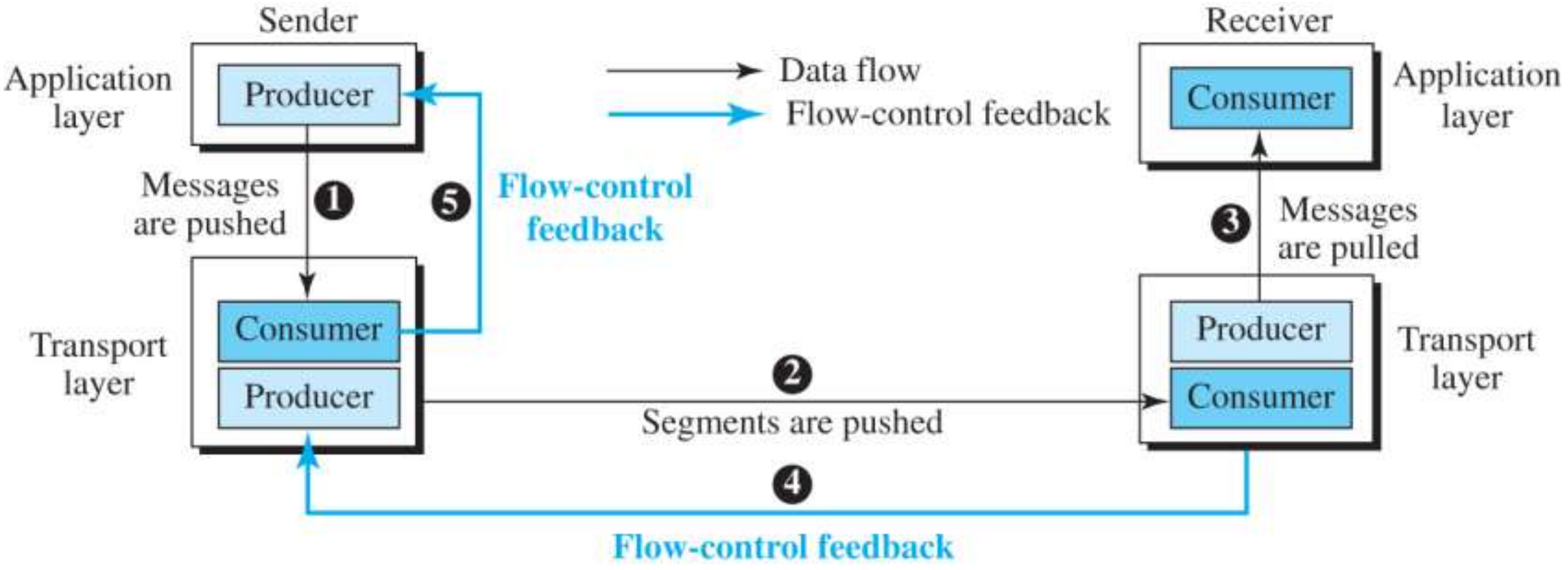


[Access the text alternative for slide images.]

9.4.7 Flow Control

As discussed before, 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. In this section we discuss flow control, ignoring error control. We assume that the logical channel between the sending and receiving TCP is error-free.

Figure 9.35 Data flow and flow control feedbacks in TCP

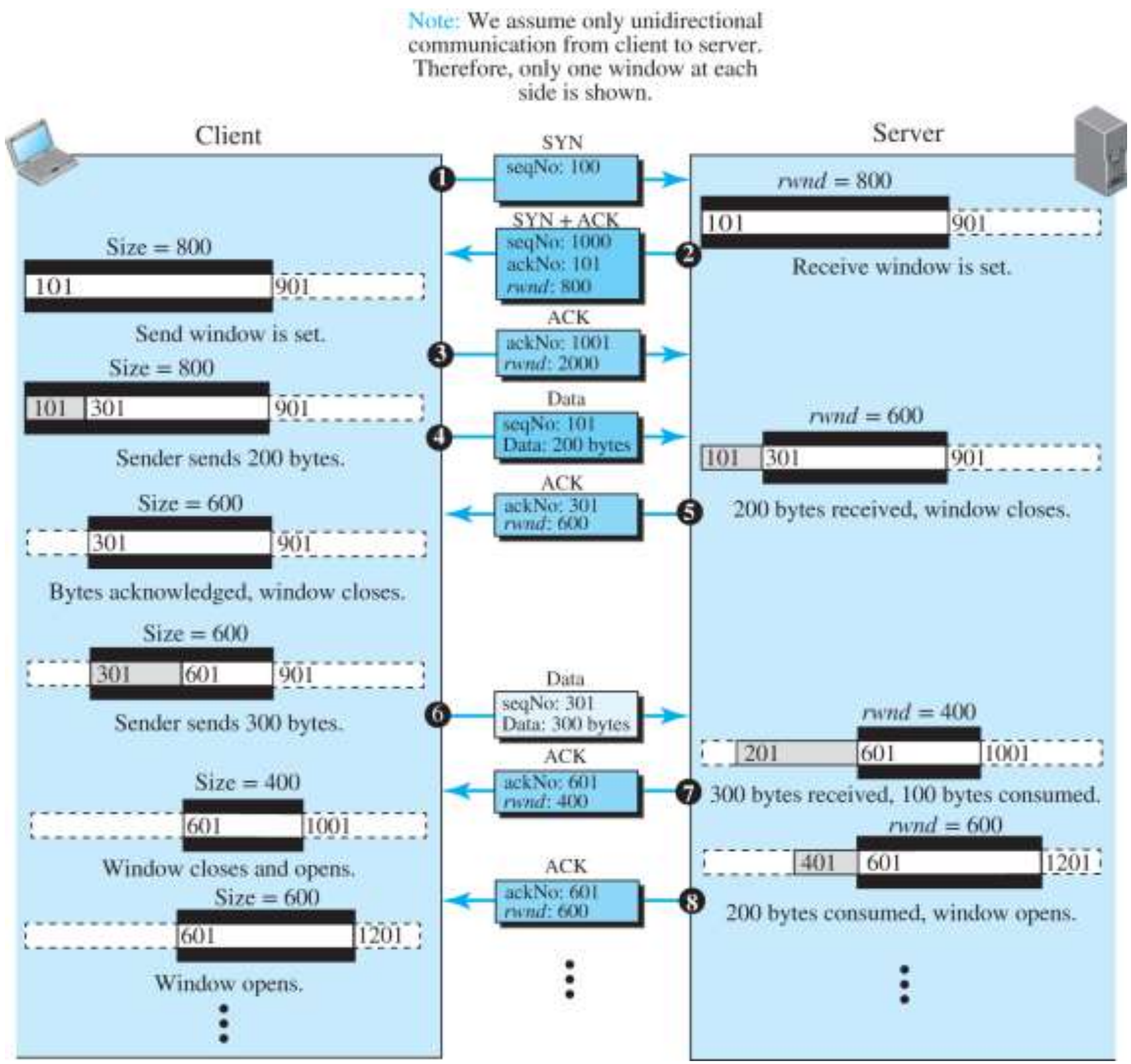


Access the text alternative for slide images.

Opening and Closing Windows

To achieve flow control, TCP forces the sender and the receiver to adjust their window sizes, although the size of the buffer for both parties is fixed when the connection is established. The receive window closes (moves its left wall to the right) when more bytes arrive from the sender; it opens (moves its right wall to the right) when more bytes are pulled by the process. We assume that it does not shrink (the right wall does not move to the left).

Figure 9.36 An example of flow control



Access the text alternative for slide images.

Shrinking of Windows

As we said before, the receive window cannot shrink. The send window, on the other hand, can shrink if the receiver defines a value for rwnd that results in shrinking the window. However, some implementations do not allow shrinking of the send window. The limitation does not allow the right wall of the send window to move to the left. In other words, 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.

$$\text{new ackNo} + \text{new rwnd} >= \text{last ackNo} + \text{last rwnd}$$

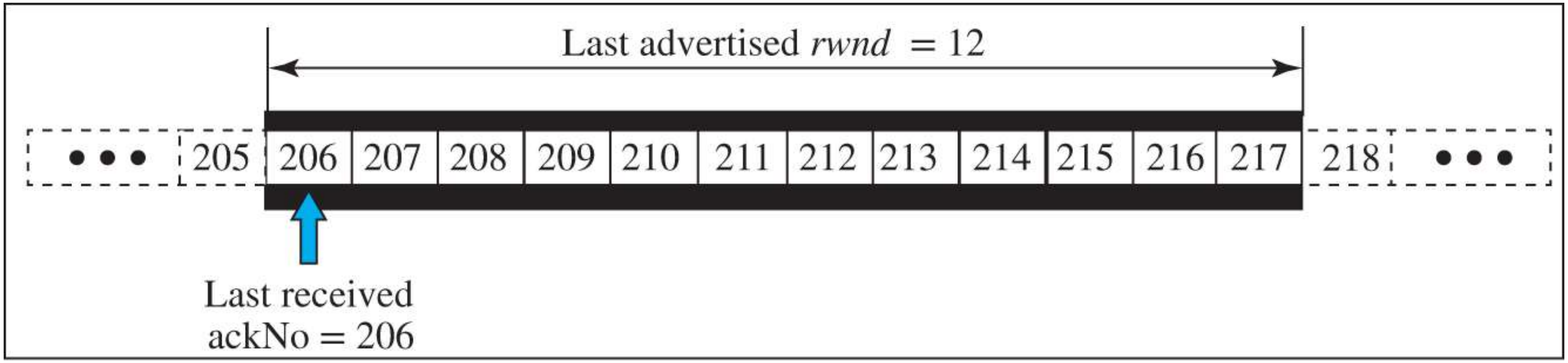
Example 9.9

Figure 9.37 shows the reason for this mandate.

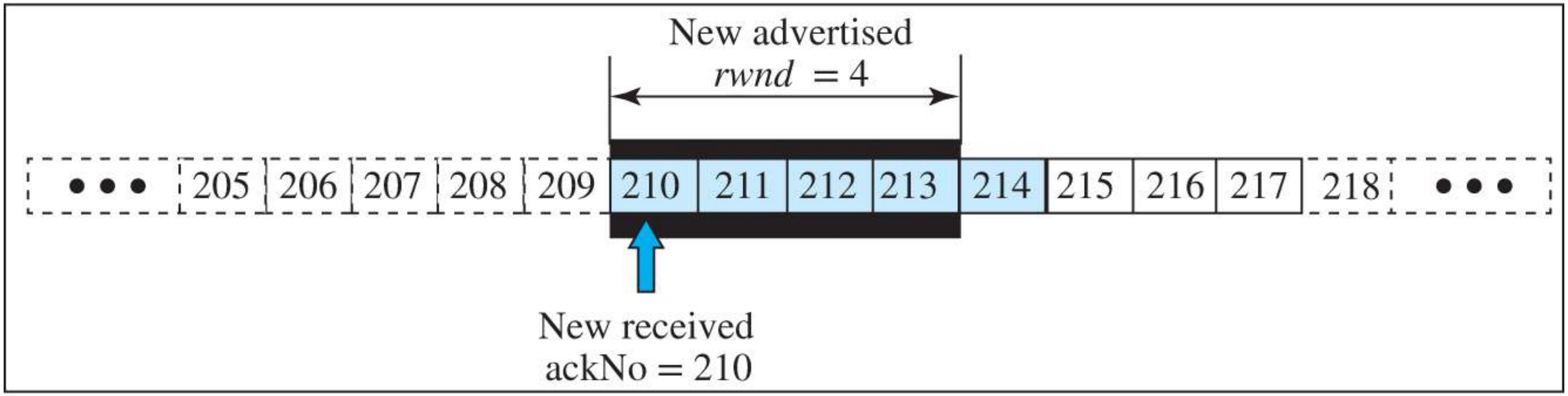
Part ***a*** of the figure shows the values of the last acknowledgment and *rwnd*. Part ***b*** shows the situation in which the sender has sent bytes 206 to 214. Bytes 206 to 209 are acknowledged and purged. The new advertisement, however, defines the new value of *rwnd* as 4, in which

$$210 + 4 < 206 + 12.$$

Figure 9.37 Example 9.9



a. The window after the last advertisement



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

[Access the text alternative for slide images.]

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. This problem is called the silly window syndrome. For each site, we first describe how the problem is created and then give a proposed solution.

9.4.8 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.

Checksum *2*

Each segment includes a checksum field, which is used to check for a corrupted segment. If a segment is corrupted, as detected by an invalid checksum, the segment is discarded by the destination TCP and is considered as lost. TCP uses a 16-bit checksum that is mandatory in every segment.

Acknowledgment

TCP uses acknowledgments to confirm the receipt of data segments. Control segments that carry no data, but consume a sequence number, are also acknowledged. ACK segments are never acknowledged.

Retransmission

The heart of the error control mechanism is the retransmission of segments. When a segment is sent, it is stored in a queue until it is acknowledged. When the retransmission timer expires or when the sender receives three duplicate ACKs for the first segment in the queue, that segment is retransmitted.

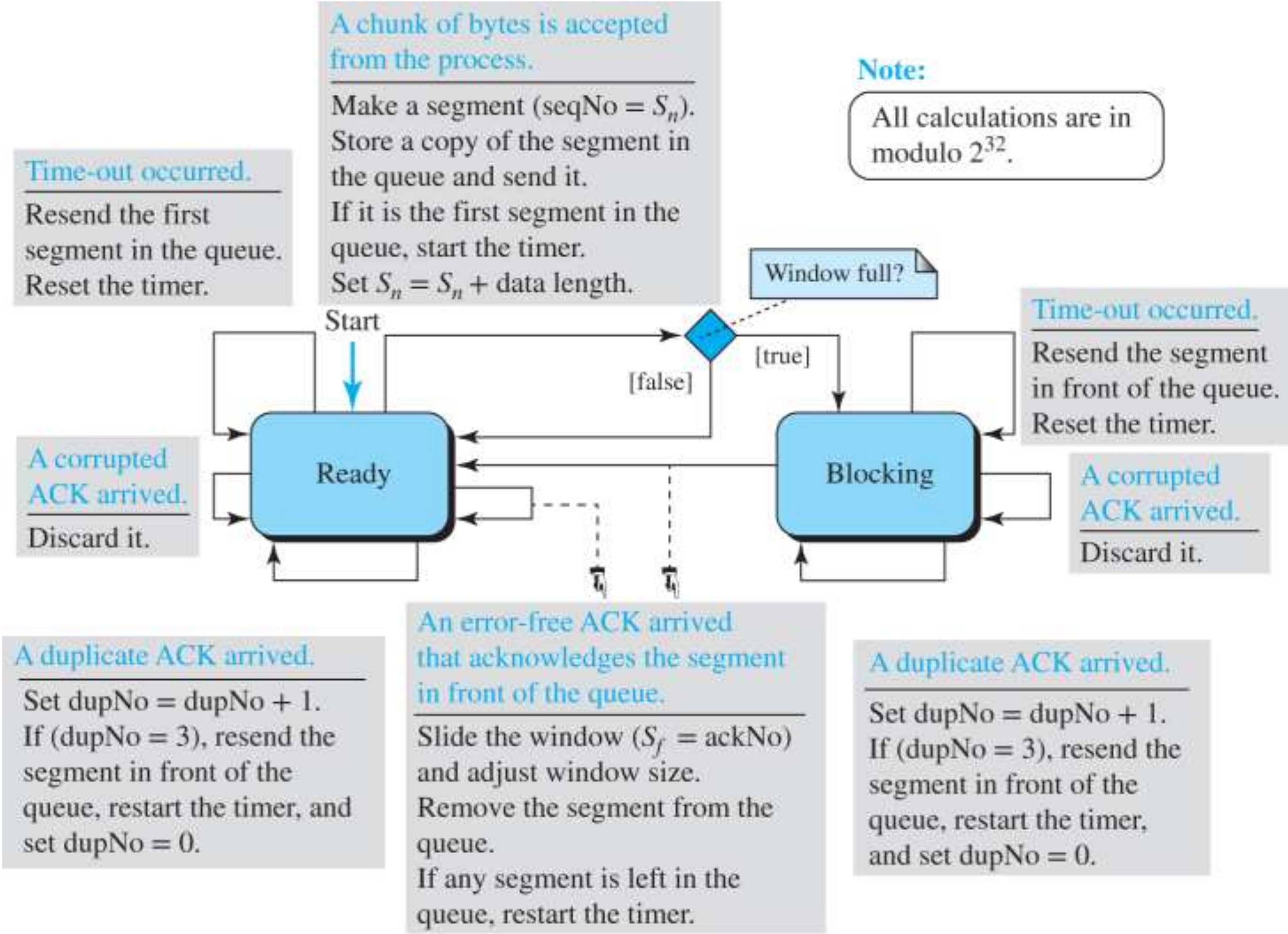
Out-of-Order Segments

TCP implementations today do not discard out-of-order segments. They store them temporarily and flag them as out-of-order segments until the missing segments arrive. Note, however, that out-of-order segments are never delivered to the process. TCP guarantees that data are delivered to the process in order.

FSMs for Data Transfer in TCP

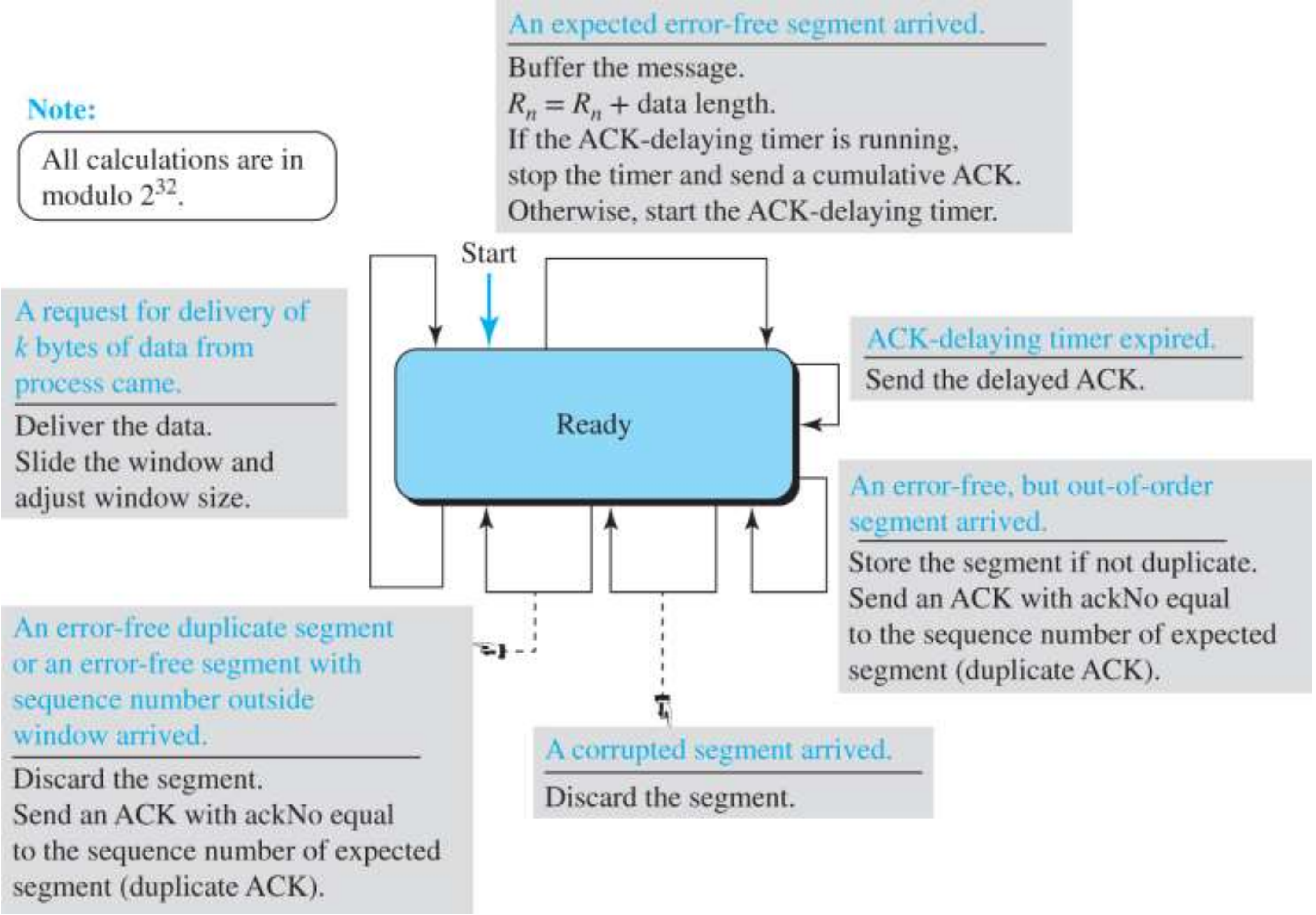
Data transfer in TCP is close to the Selective-Repeat protocol with a slight similarity to GBN. Since TCP accepts out-of-order segments, TCP can be thought of as behaving more like the SR protocol, but since the original acknowledgments are cumulative, it looks like GBN. However, if the TCP implementation uses SACKs, then TCP is closest to SR.

Figure 9.38 Simplified FSM for the TCP sender side



Access the text alternative for slide images.

Figure 9.39 Simplified FSM for the TCP receiver side

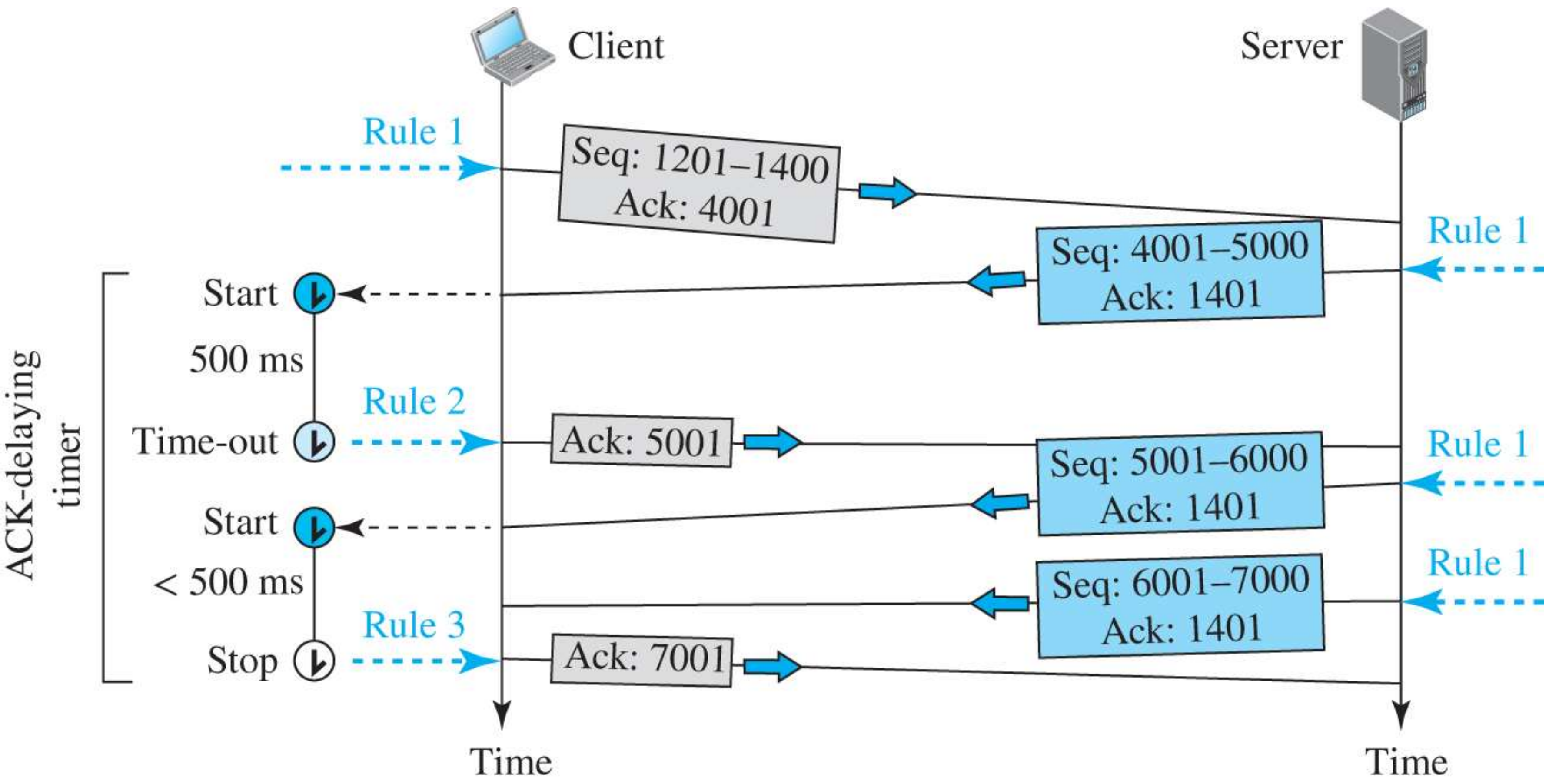


Access the text alternative for slide images.

Some Scenarios

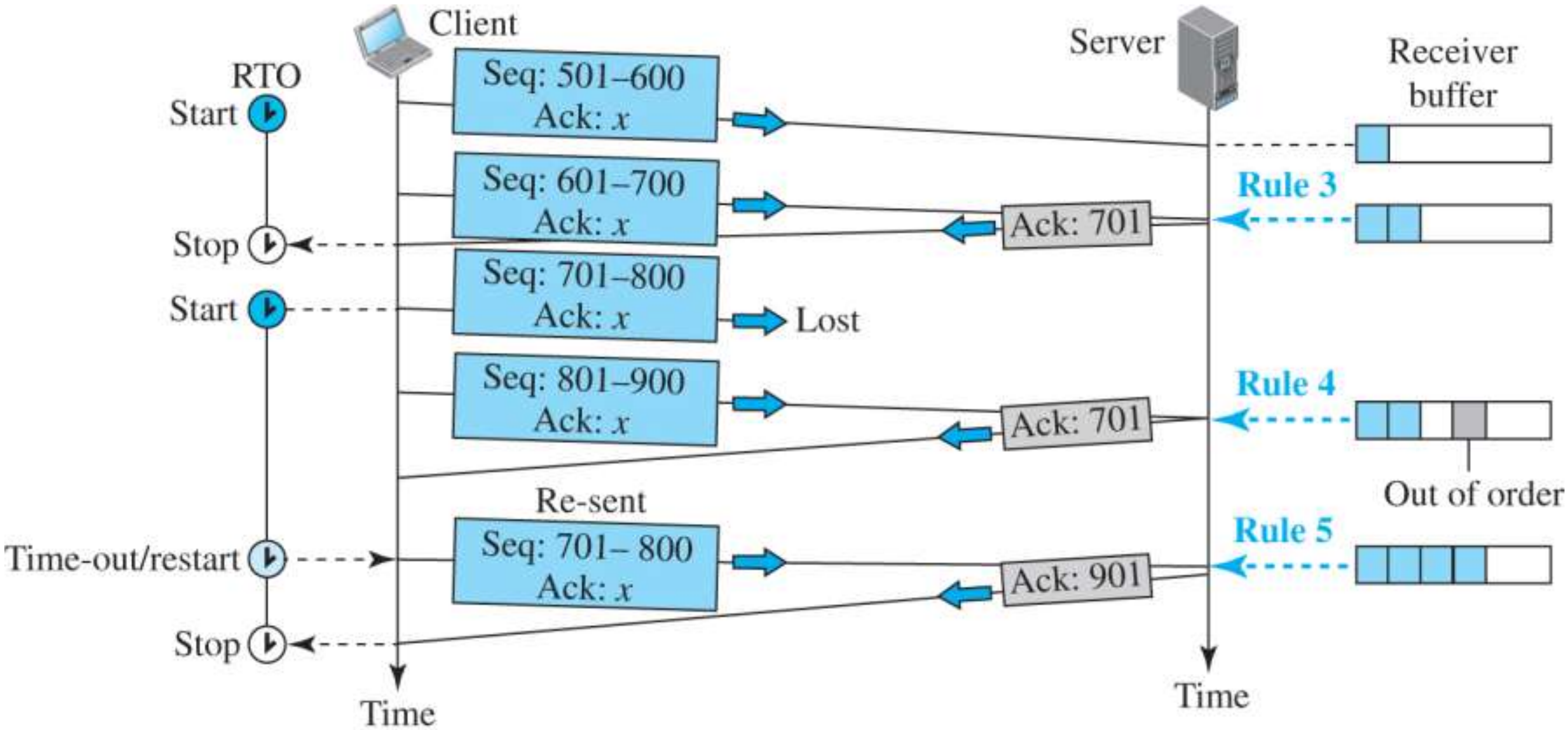
In this section we give some examples of scenarios that occur during the operation of TCP, considering only error control issues. In these scenarios, we show a segment by a rectangle. If the segment carries data, we show the range of byte numbers and the value of the acknowledgment field. If it carries only an acknowledgment, we show only the acknowledgment number in a smaller box.

Figure 9.40 Normal operation



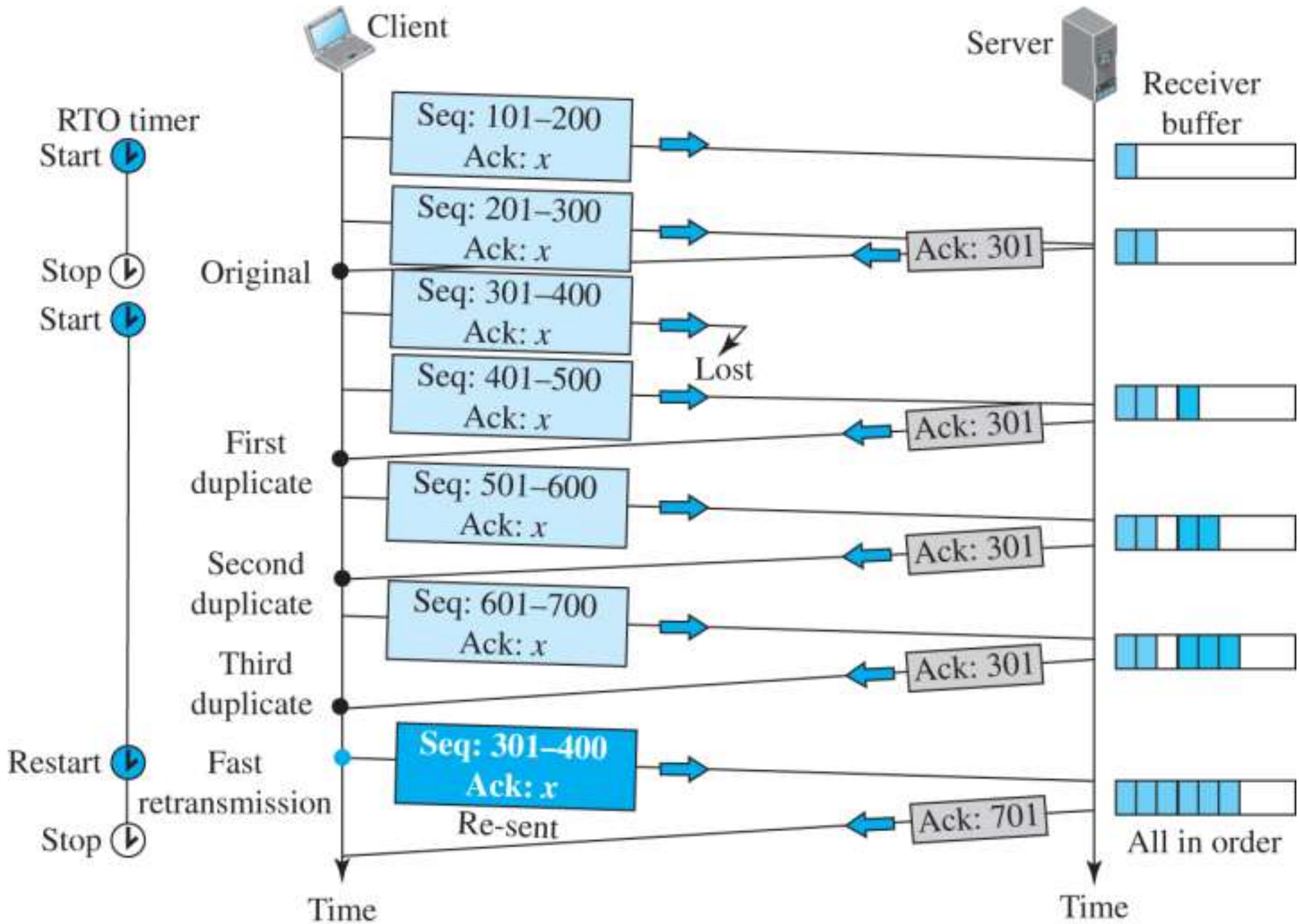
Access the text alternative for slide images.

Figure 9.41 Lost segment



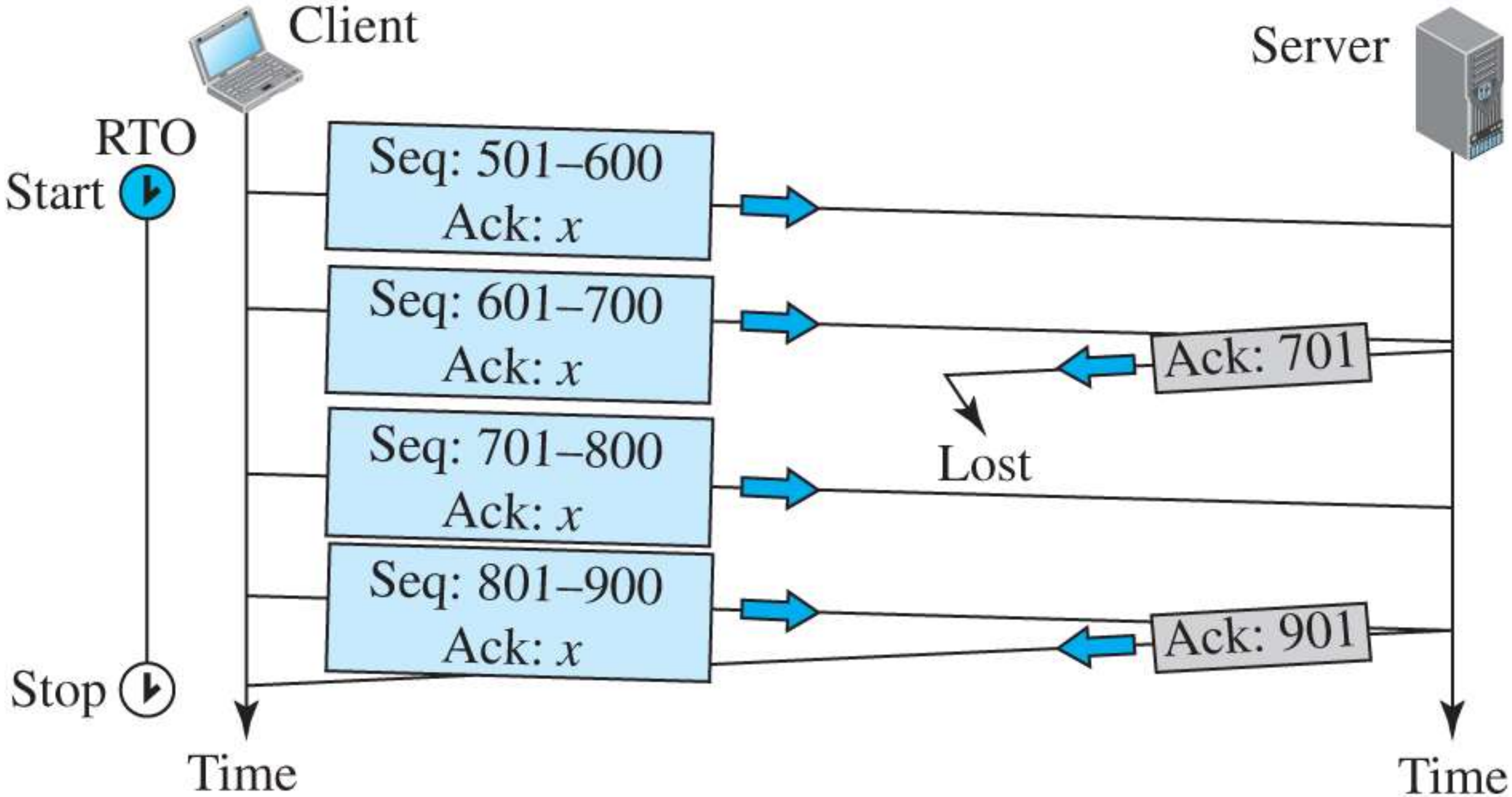
Access the text alternative for slide images.

Figure 9.42 Fast retransmission



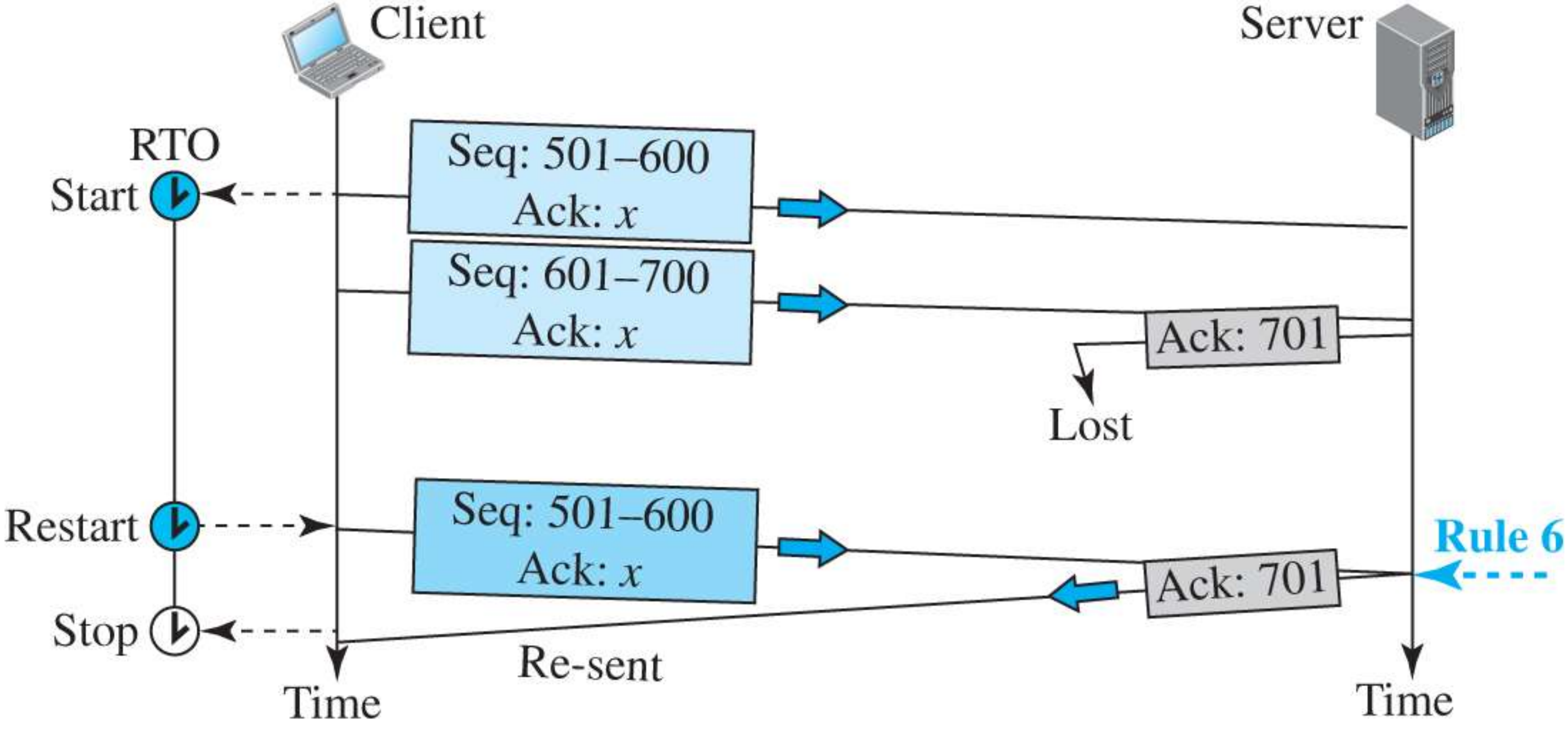
Access the text alternative for slide images.

Figure 9.43 Lost acknowledgment



Access the text alternative for slide images.

Figure 9.44 Lost acknowledgment corrected by resending a segment



[Access the text alternative for slide images.]

9.4.9 TCP Congestion Control

TCP uses different policies to handle the congestion in the network. We describe these policies in this section.

Congestion Window

When we discussed flow control in TCP, we mentioned that 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). This, however, does not mean that the intermediate buffers, buffers in the routers, do not become congested. A router may receive data from more than one sender.

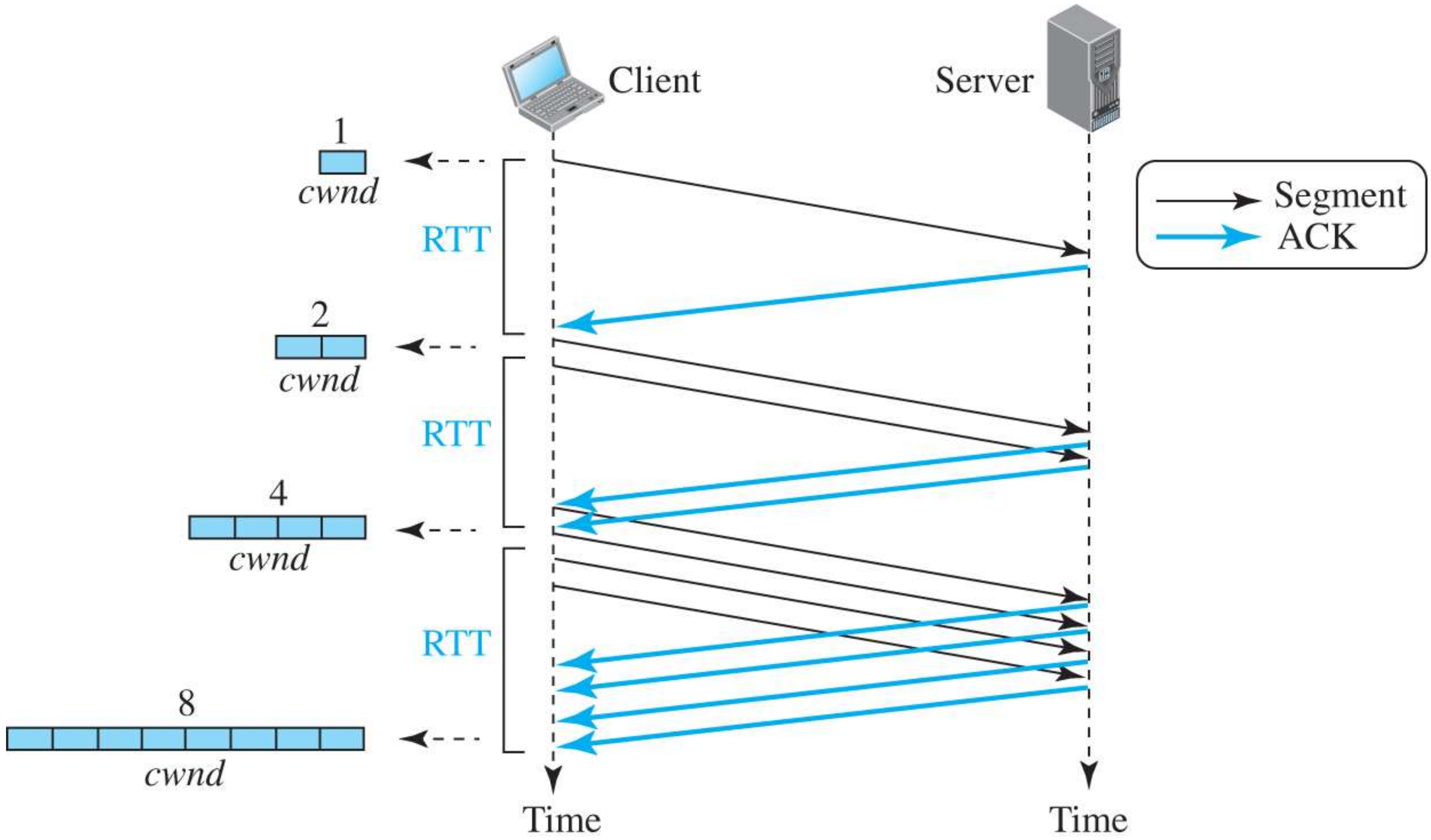
Congestion Detection

- *Before discussing how the value of cwnd should be set and changed, we need to describe how a 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.*
- *The first is the 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.*

Congestion Policies

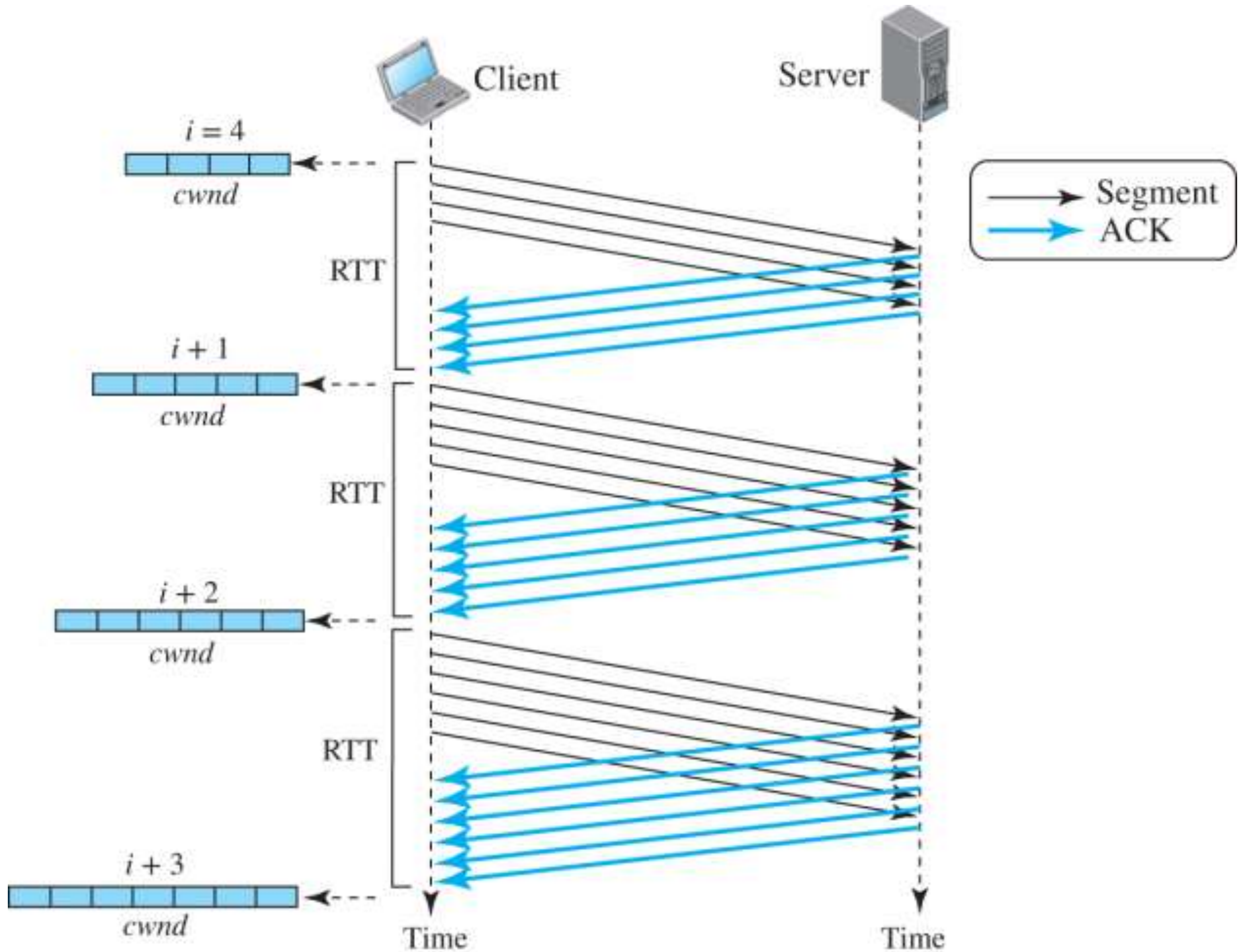
TCP's general policy for handling congestion is based on three algorithms: slow start, congestion avoidance, and fast recovery.

Figure 9.45 Slow start, exponential increase



[Access the text alternative for slide images.]

Figure 9.46 Congestion avoidance, additive increase

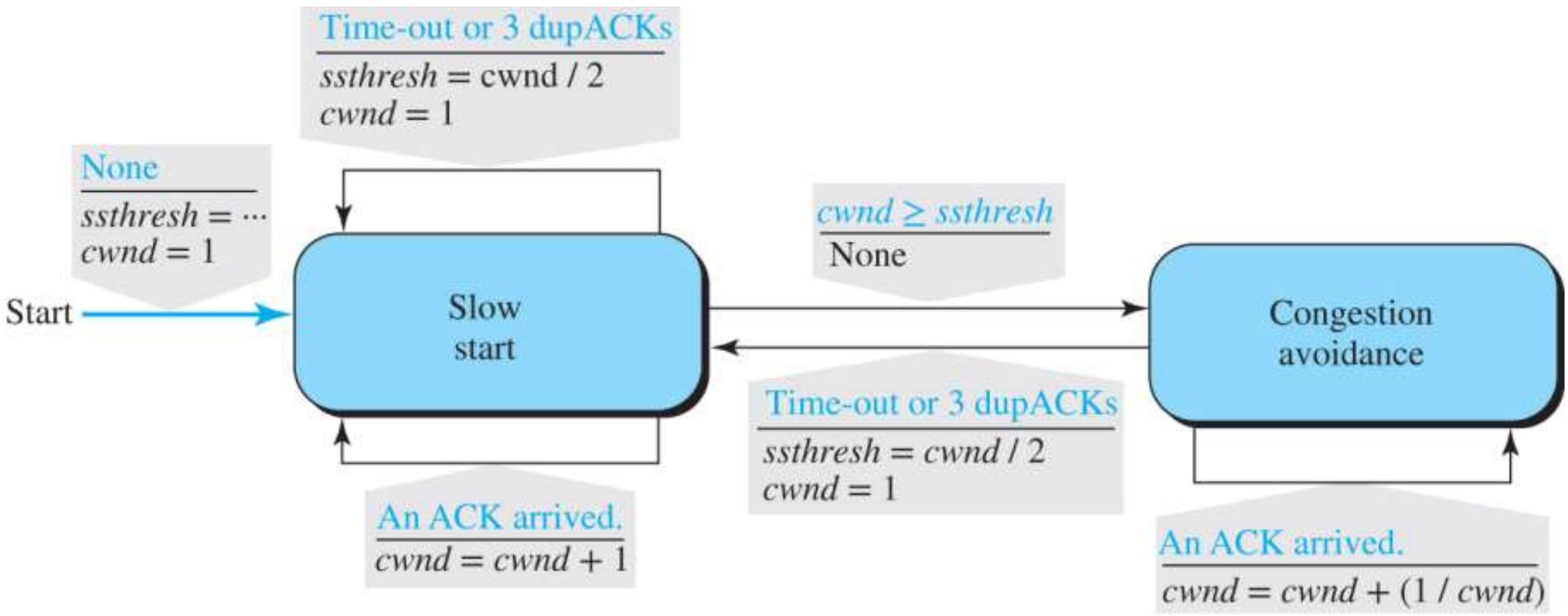


Access the text alternative for slide images.

Policy Transition

We discussed three congestion policies in TCP. Now the question is when each of these policies are used and when TCP moves from one policy to another. To answer these questions, we need to refer to three versions of TCP: Taho TCP, Reno TCP, and New Reno TCP.

Figure 9.47 FSM for Taho TCP



Access the text alternative for slide images.

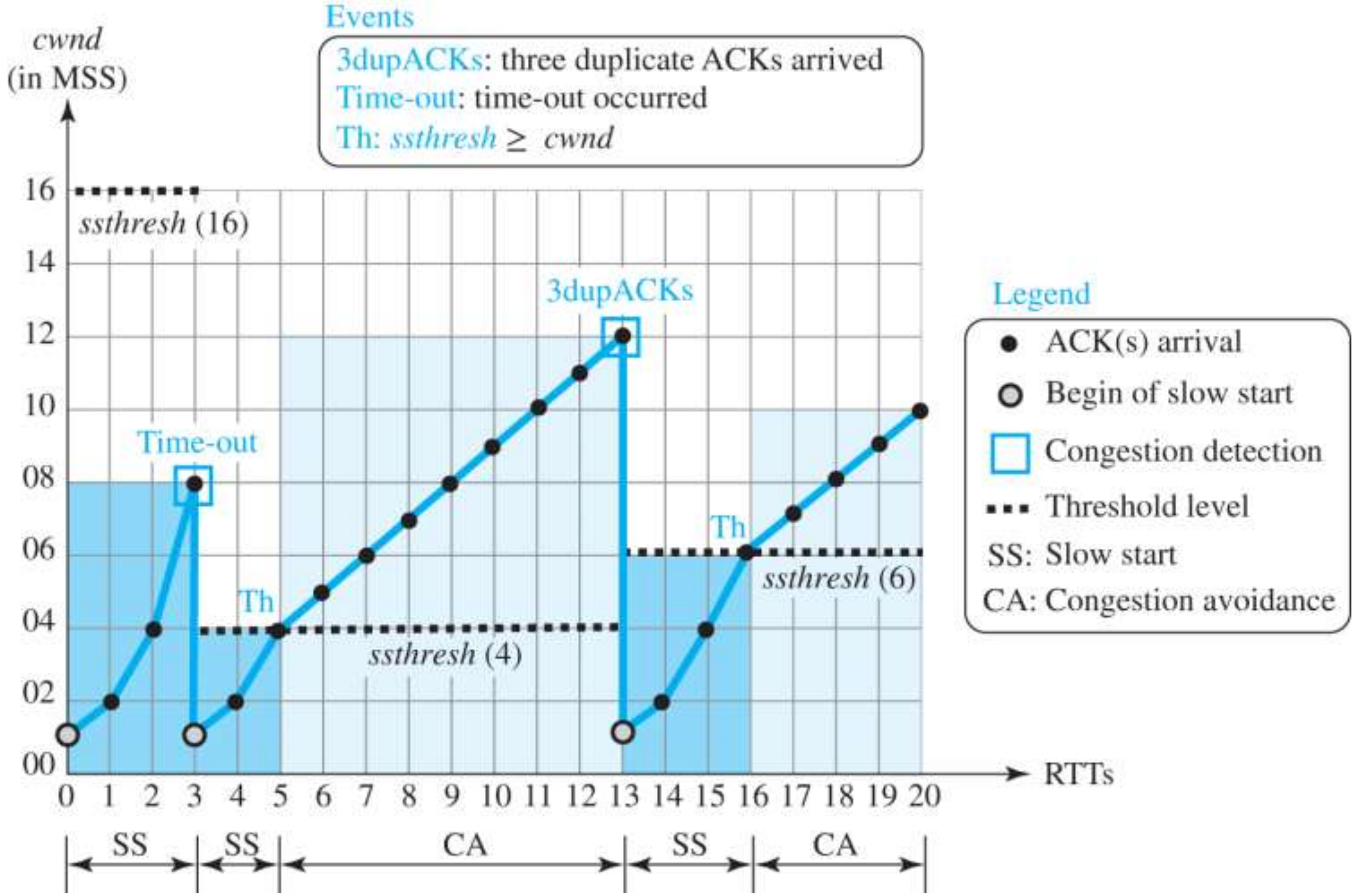
Example 9.10 *(1)*

Figure 9.48 shows an example of congestion control in a Taho TCP. 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 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.

Example 9.10 *(2)*

At this moment, three duplicate ACKs arrive, another indication of the 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 the 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.

Figure 9.48 Example of Taho TCP

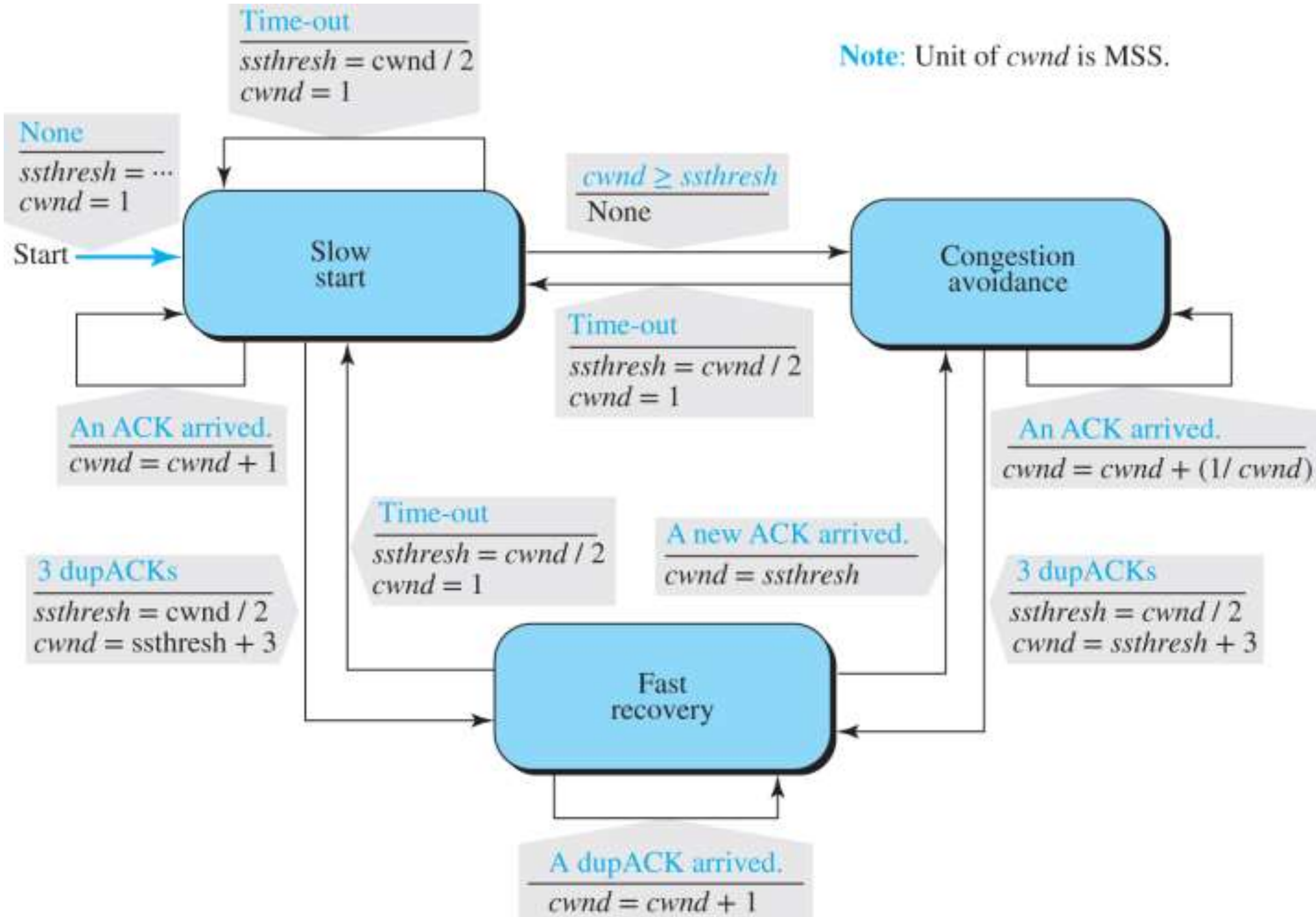


[Access the text alternative for slide images.]

Additive Increase, Multiplicative Decrease

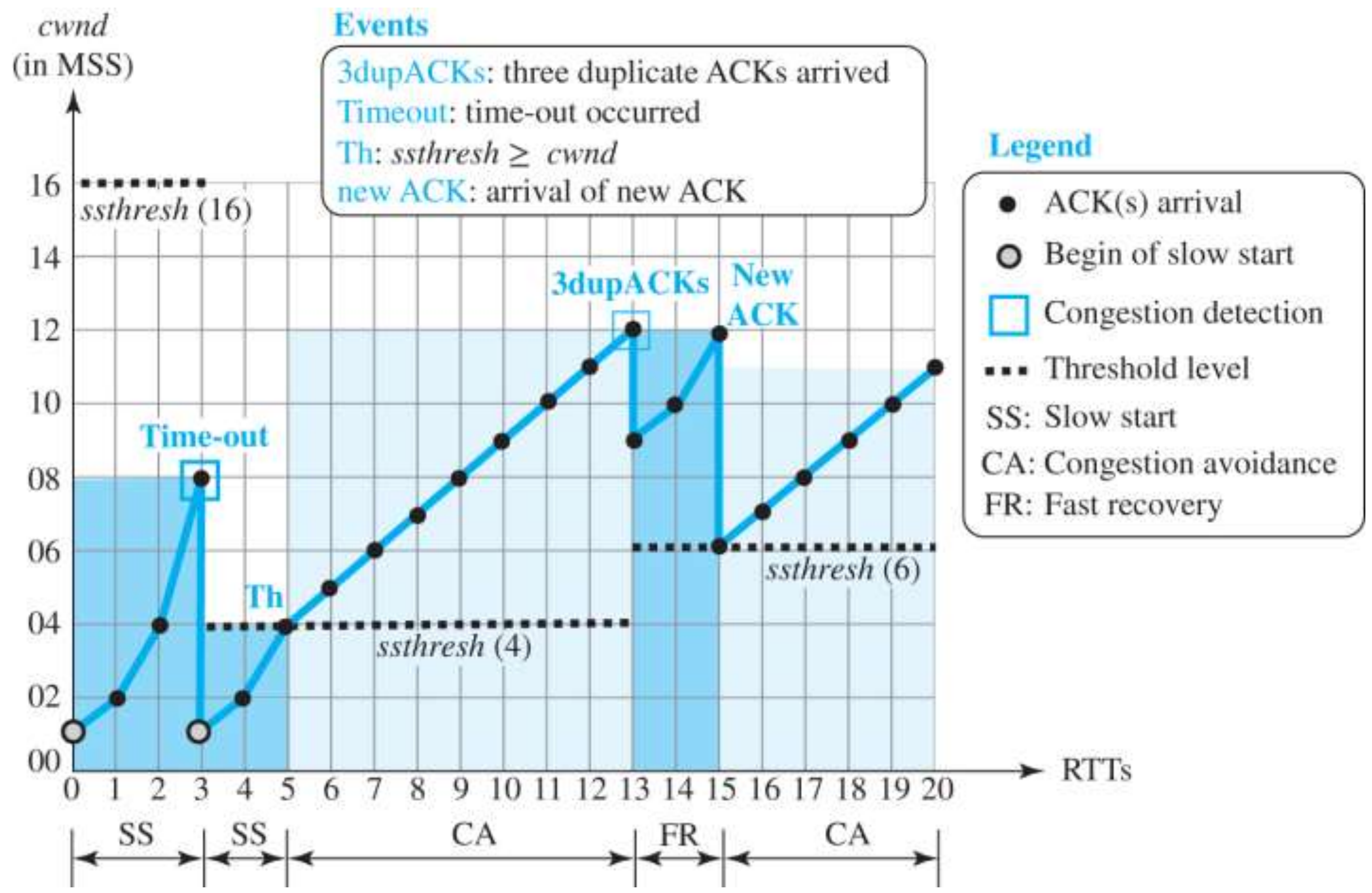
Out of the three versions of TCP, the Reno version is most common today. It has been observed that, in this version, most of the time the congestion is detected and taken care of by observing the three duplicate ACKs. Even if there are some time-out events, TCP recovers from them by aggressive exponential growth.

Figure 9.49 FSM for Reno TCP



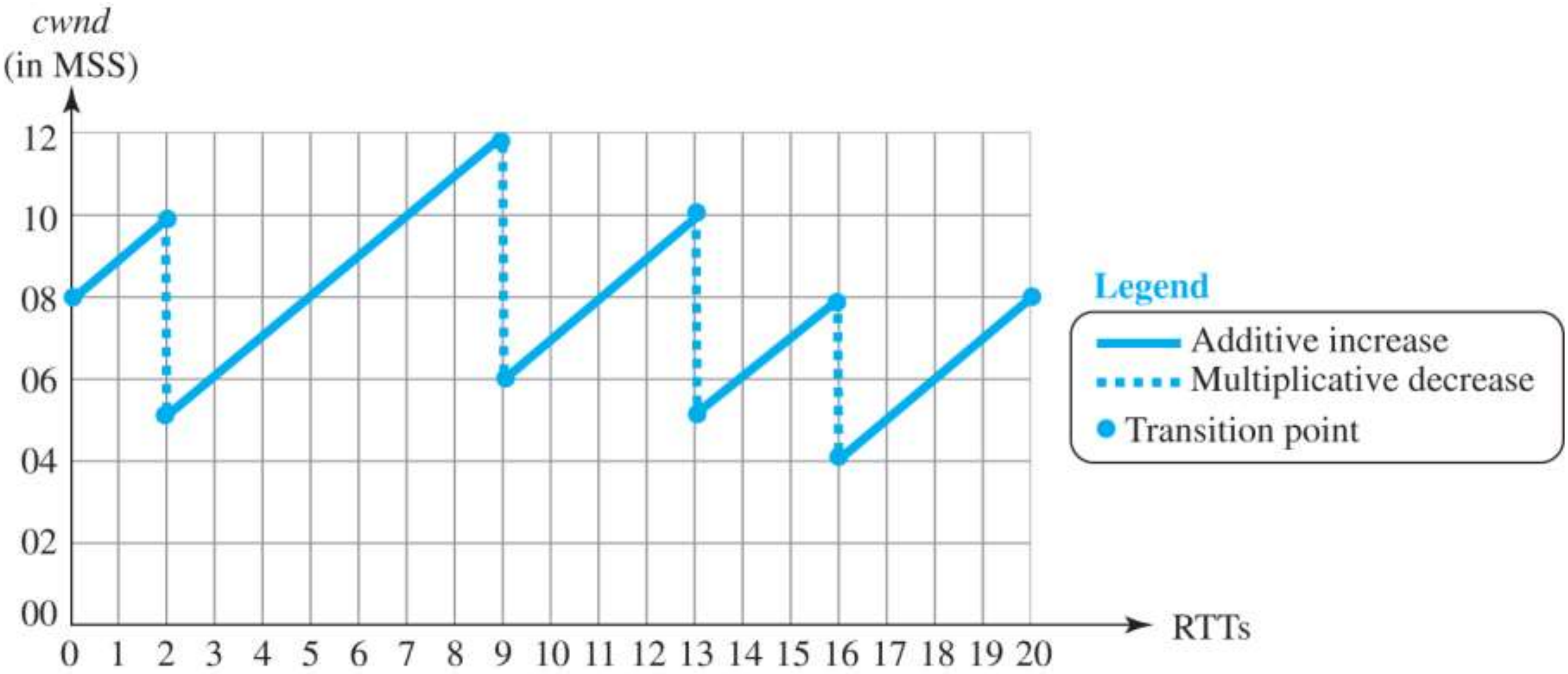
[Access the text alternative for slide images.]

Figure 9.50 Example of a Reno TCP



[Access the text alternative for slide images.]

Figure 9.51 Additive increase, multiplicative decrease (AIMD)



[Access the text alternative for slide images.]

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.

Throughput = (0.75) Wmax / RTT

Example 9.12

If MSS = 10 KB (kilobytes) and RTT = 100 ms in Figure 9.51, we can calculate the throughput as shown below.

Wmax = (10 + 12 + 10 + 8 + 8) / 5 = 9.6 MSS

Throughput = (0.75 Wmax / RTT) = 0.75 * 960 kbps / 100 ms = 7.2 Mbps

9.4.10 TCP Timers

To perform their operations smoothly, most TCP implementations use at least four timers: retransmission, persistence, keepalive, and TIME-WAIT

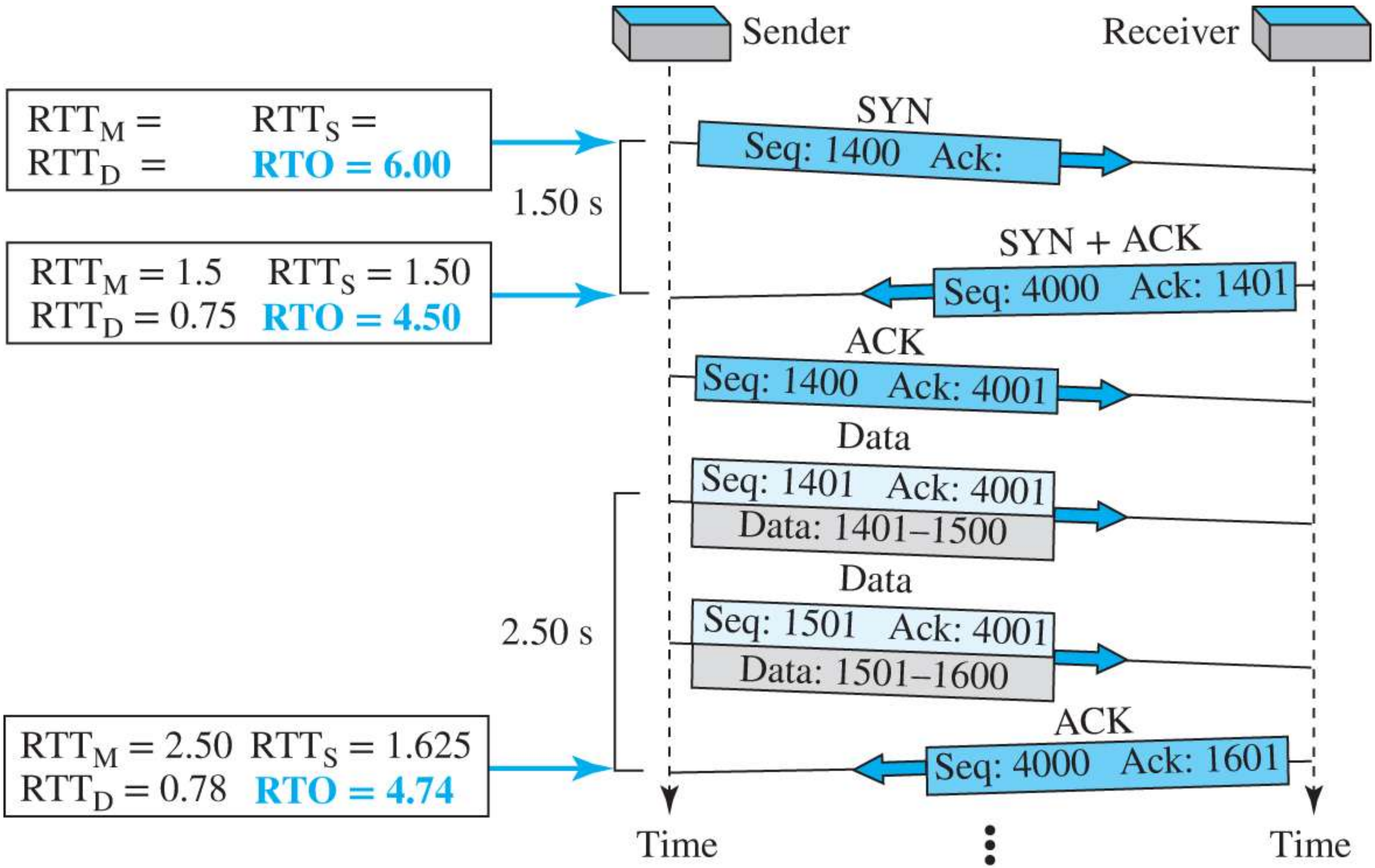
Retransmission Timer

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.

Example 9.13

Let us give a hypothetical example. Figure 9.52 shows part of a connection. The figure shows the connection establishment and part of the data transfer phases.

Figure 9.52 Example 9.13

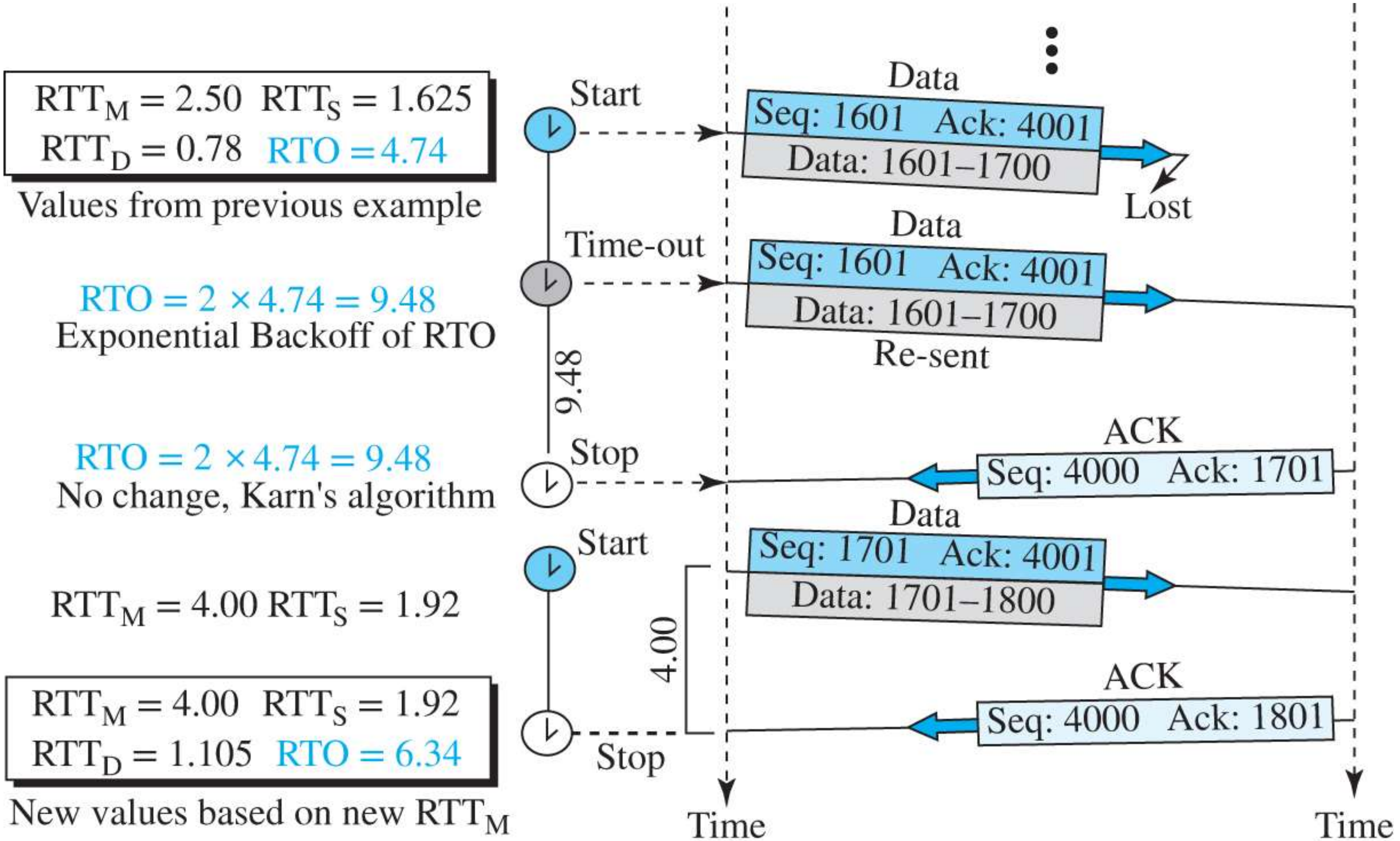


Access the text alternative for slide images.

Example 9.14

Figure 9.53 is a continuation of the previous example. There is retransmission and Karn’s algorithm is applied. The first segment in the figure is sent, but lost. The RTO timer expires after 4.74 seconds. The segment is re-transmitted and the timer is set to 9.48, twice the previous value of RTO. This time an ACK is received before the time-out. We wait until we send a new segment and receive the ACK for it before recalculating the RTO (Karn’s algorithm).

Figure 9.53 Example 9.14



Access the text alternative for slide images.

Keepalive Timer

A keepalive timer is used in some implementations to prevent a long idle connection between two TCP's. 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.

TIME-WAIT Timer

The TIME-WAIT (2MSL) timer is used during connection termination. The maximum segment life time (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 for 2 MSL amount of time to allow TCP 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.

9.4.11 Options

The TCP header can have up to 40 bytes of optional information. Options convey additional information to the destination or align other options. These options are included on the book website for further reference.

9-5 STREAM CONTROL TRANSMISSION PROTOCOL

Stream Control Transmission Protocol (SCTP) is a new transport-layer protocol designed to combine some features of UDP and TCP in an effort to create a protocol for multimedia communication.

9.5.1 SCTP Services

Before discussing the operation of SCTP, let us explain the services offered by SCTP to the application-layer processes.

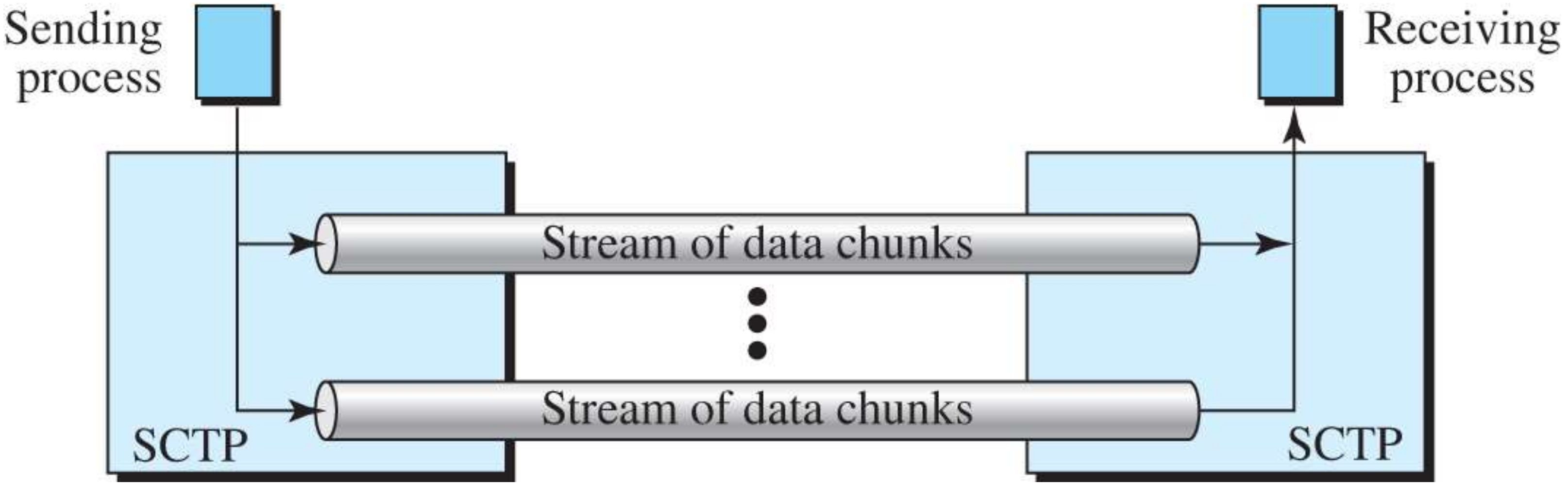
Process-to-Process Communication 3

SCTP, like UDP or TCP, provides process-to-process communication.

Multiple Streams

We learned that TCP is a stream-oriented protocol. Each connection between a TCP client and a TCP server involves one single stream. The problem with this approach is that a loss at any point in the stream blocks the delivery of the rest of the data. This can be acceptable when we are transferring text; it is not when we are sending real-time data such as audio or video. SCTP allows multistream service in each connection, which is called association in SCTP terminology. If one of the streams is blocked, the other streams can still deliver their data. Figure 9.54 shows the idea of multiple-stream delivery.

Figure 9.54 Multiple-stream concept

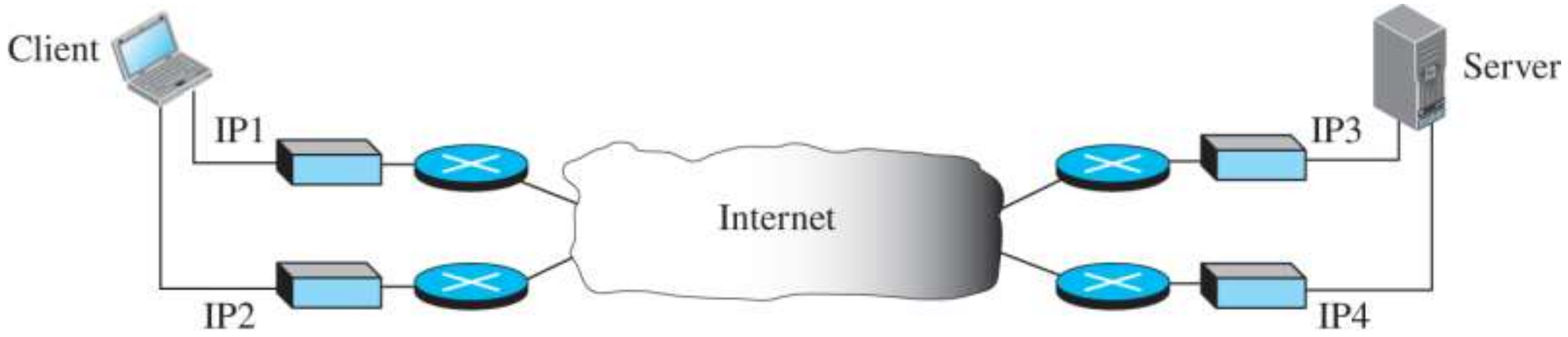


[Access the text alternative for slide images.]

Multihoming

A TCP connection involves one source and one destination IP address. This means that even if the sender or receiver is a multihomed host (connected to more than one physical address with multiple IP addresses), only one of these IP addresses per end can be used during the connection. An SCTP association, on the other hand, supports multihoming service. The sending and receiving host can define multiple IP addresses in each end for an association. In this fault-tolerant approach, when one path fails, another interface can be used for data delivery without interruption. This fault-tolerant feature is very helpful when we are sending and receiving a real-time payload such as Internet telephony.

Figure 9.55 Multihoming concept



[Access the text alternative for slide images.]

Full-Duplex Communication 2

Like TCP, SCTP offers full-duplex service, where data can flow in both directions at the same time. Each SCTP then has a sending and receiving buffer and packets are sent in both directions.

Connection-Oriented Service

Like TCP, SCTP is a connection-oriented protocol. However, in SCTP, a connection is called an association.

Reliable Service 2

SCTP, like TCP, is a reliable transport protocol. It uses an acknowledgment mechanism to check the safe and sound arrival of data. We will discuss this feature further in the section on error control.

9.5.2 SCTP Features

The following shows the general features of SCTP.

Transmission Sequence Number (TSN)

The unit of data in SCTP is a data chunk, which may or may not have a one-to-one relationship with the message coming from the process because of fragmentation (discussed later). Data transfer in SCTP is controlled by numbering the data chunks. SCTP uses a transmission sequence number (TSN) to number the data chunks. In other words, the TSN in SCTP plays the analogous role as the sequence number in TCP. TSN's are 32 bits long and randomly initialized between 0 and 232 - 1. Each data chunk must carry the corresponding TSN in its header.

Stream Identifier (SI)

In SCTP, there may be several streams in each association. Each stream in SCTP needs to be identified using a stream identifier (SI). Each data chunk must carry the SI in its header so that when it arrives at the destination, it can be properly placed in its stream. The SI is a 16-bit number starting from 0.

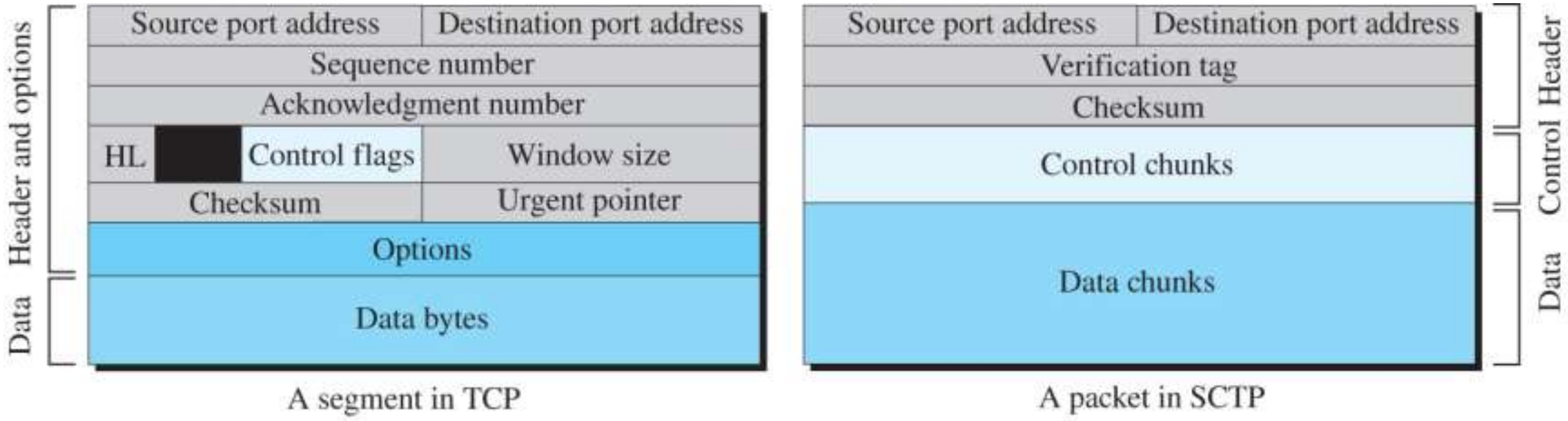
Stream Sequence Number (SSN)

When a data chunk arrives at the destination SCTP, it is delivered to the appropriate stream and in the proper order. This means that, in addition to an SI, SCTP defines each data chunk in each stream with a stream sequence number (SSN).

Packets

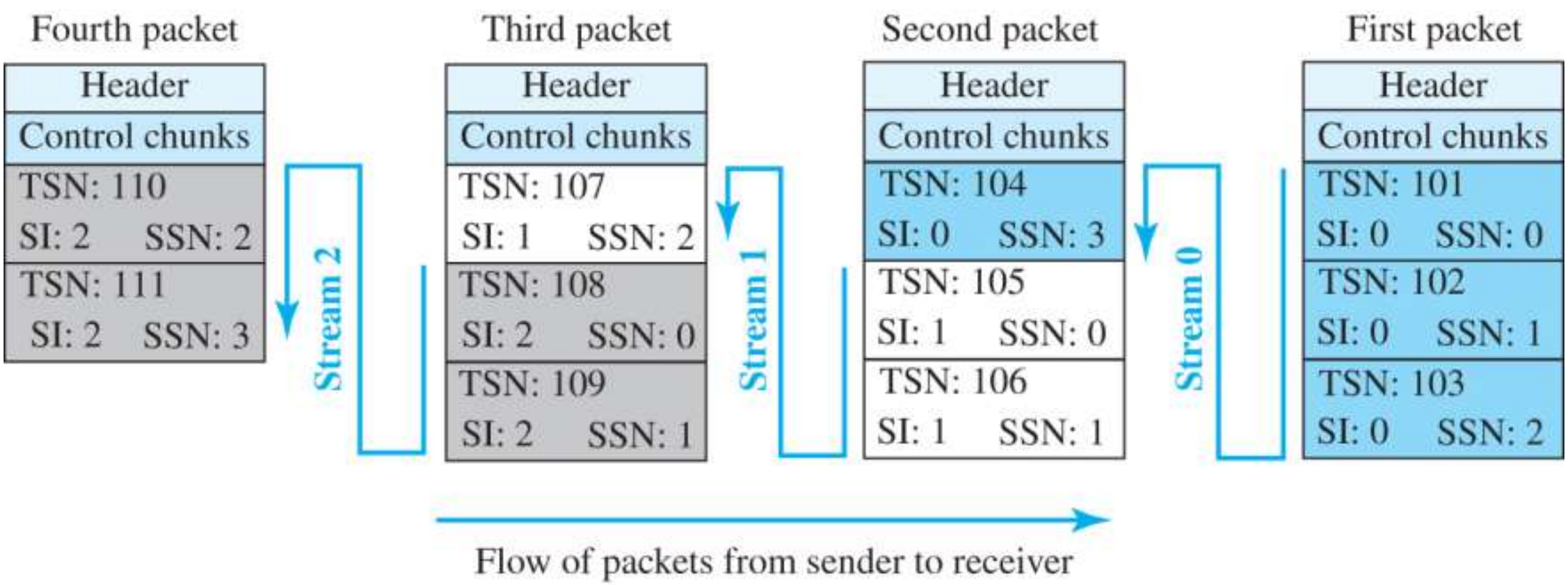
In TCP, a segment carries data and control information. Data are carried as a collection of bytes; control information is defined by six control flags in the header. The design of SCTP is totally different: data are carried as data chunks, control information as control chunks. Several control chunks and data chunks can be packed together in a packet. A packet in SCTP plays the same role as a segment in TCP.

Figure 9.56 Comparison between a TCP segment and an SCTP packet



Access the text alternative for slide images.

Figure 9.57 Packets, data chunks, and streams



Access the text alternative for slide images.

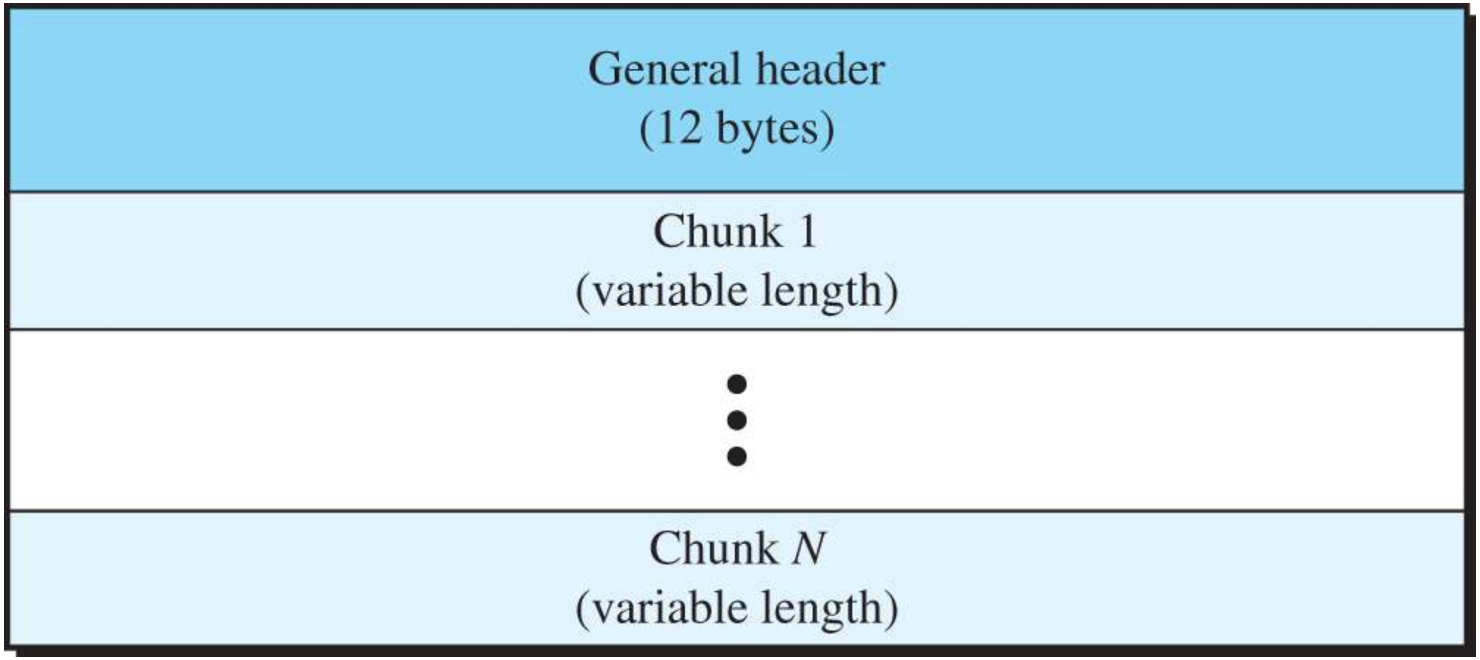
Acknowledgment Number

SCTP acknowledgment numbers are chunk-oriented. They refer to the TSN. In SCTP, the control information is carried by control chunks, which do not need a TSN. These control chunks are acknowledged by another control chunk of the appropriate type (some need no acknowledgment). For example, an INIT control chunk is acknowledged by an INIT-ACK chunk. There is no need for a sequence number or an acknowledgment number.

9.5.3 Packet Format

An SCTP packet has a mandatory general header and a set of blocks called chunks. There are two types of chunks: control chunks and data chunks. A control chunk controls and maintains the association; a data chunk carries user data. In a packet, the control chunks come before the data chunks. Figure 9.58 shows the general format of an SCTP packet.

Figure 9.58 SCTP packet format



Access the text alternative for slide images.

General Header

The general header (packet header) defines the end points of each association to which the packet belongs, guarantees that the packet belongs to a particular association, and preserves the integrity of the contents of the packet including the header itself. The format of the general header is shown in Figure 9.59.

Figure 9.59 General header

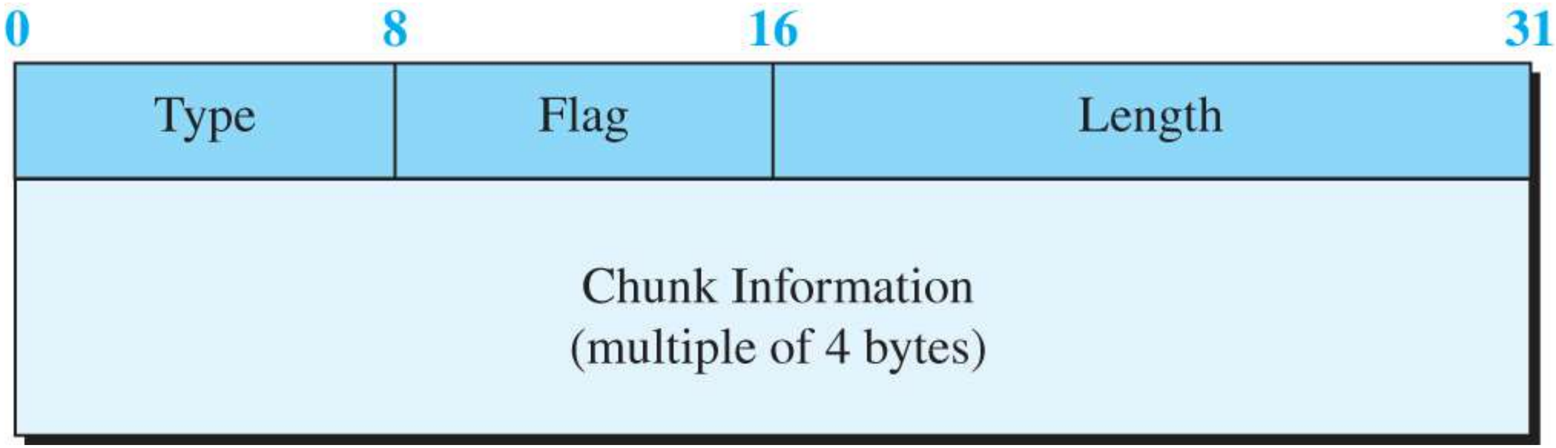
Source port address 16 bits	Destination port address 16 bits
Verification tag 32 bits	
Checksum 32 bits	

Access the text alternative for slide images.

Chunks

Control information or user data are carried in chunks. Chunks have a common layout, as shown in Figure 9.60. The first three fields are common to all chunks; the information field depends on the type of chunk. The type field can define up to 256 types of chunks. Only a few have been defined so far; the rest are reserved for future use.

Figure 9.60 Common layout of a chunk



Access the text alternative for slide images.

Table 9.3 Chunks

Type	*Chunk*	*Description*
0	DATA	User data
1	INIT	Sets up an association
2	INIT ACK	Acknowledges INIT chunk
3	SACK	Selective acknowledgment
4	HEARTBEAT	Probes the peer for liveliness
5	HEARTBEAT ACK	Acknowledges HEARTBEAT chunk
6	ABORT	Aborts an association
7	SHUTDOWN	Terminates an association
8	SHUTDOWN ACK	Acknowledges SHUTDOWN chunk
9	ERROR	Reports errors without shutting down
10	COOKIE ECHO	Third packet in association establishment
11	COOKIE ACK	Acknowledges COOKIE ECHO chunk
14	SHUTDOWN COMPLETE	Third packet in association termination
192	FORWARD TSN	For adjusting cumulating TSN

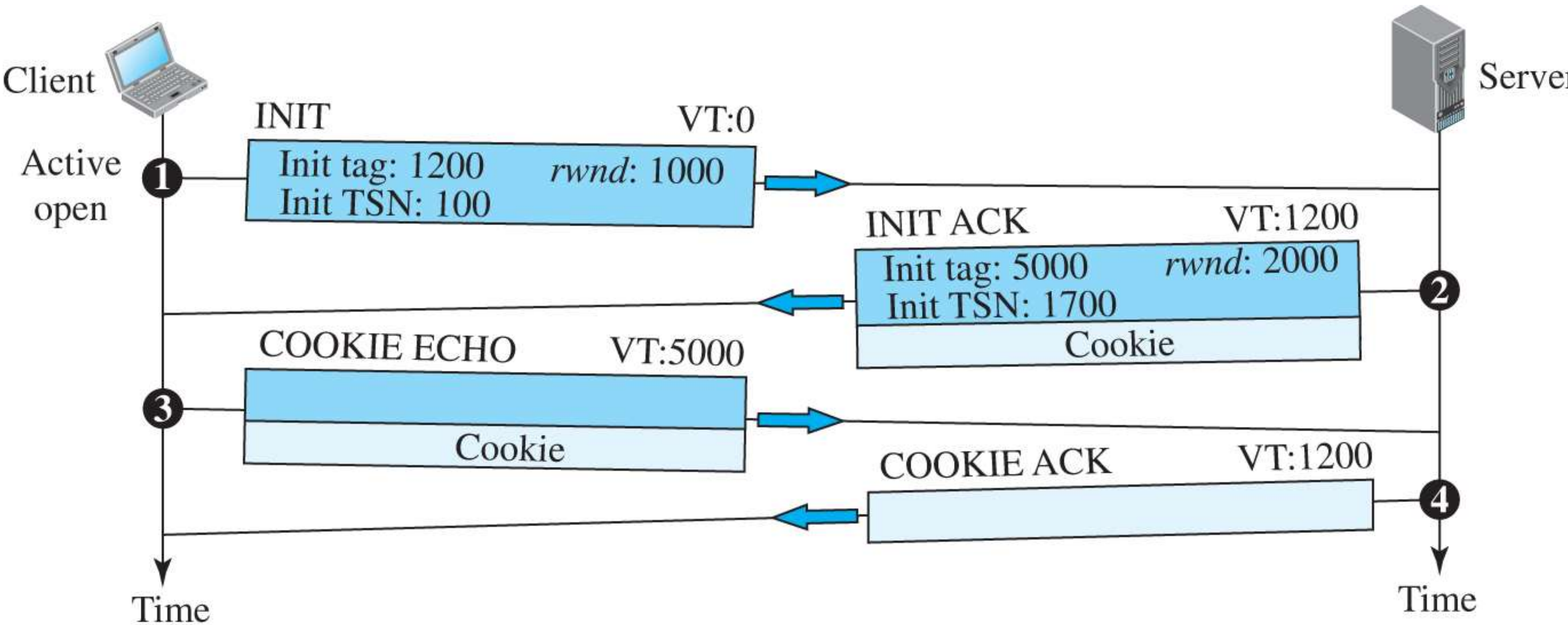
9.5.4 An SCTP Association

SCTP, like TCP, is a connection-oriented protocol. However, a connection in SCTP is called an association to emphasize multihoming.

Association Establishment

Association establishment in SCTP requires a four-way handshake. In this procedure, a process, normally a client, wants to establish an association with another process, normally a server, using SCTP as the transport-layer protocol. Similar to TCP, the SCTP server needs to be prepared to receive any association (passive open). Association establishment, however, is initiated by the client (active open). SCTP association establishment is shown in Figure 9.61.

Figure 9.61 Four-way handshaking



Access the text alternative for slide images.

Data Transfer 2

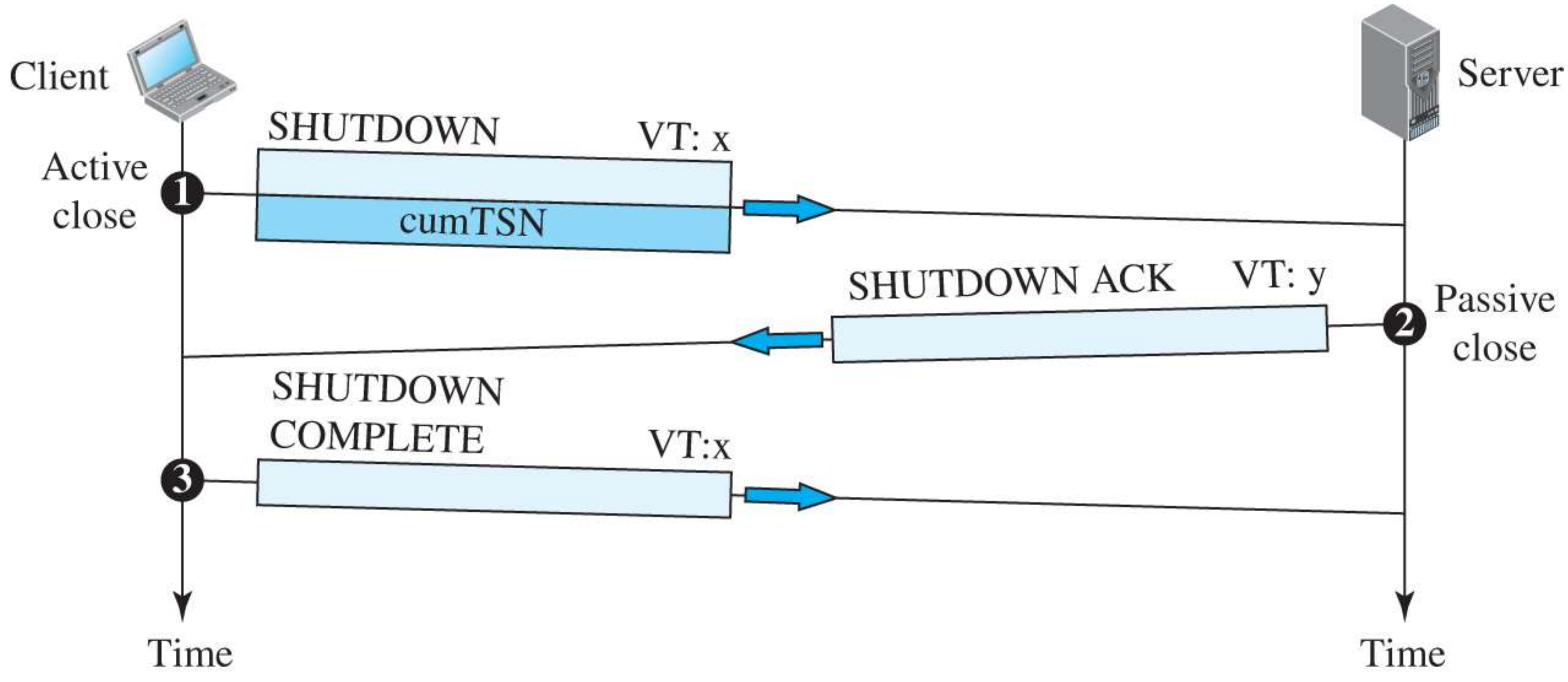
- *The whole purpose of an association is to transfer data between two ends. After the association is established, bidirectional data transfer can take place. The client and the server can both send data. Like TCP, SCTP supports piggybacking.*

- *There is a major difference, however, between data transfer in TCP and SCTP. TCP receives messages from a process as a stream of bytes without recognizing any boundary between them. The only ordering system imposed by TCP is the byte numbers.*

Association Termination

In SCTP, like TCP, either of the two parties involved in exchanging data (client or server) can close the connection. However, unlike TCP, SCTP does not allow a "half-closed" association. If one end closes the association, the other end must stop sending new data. If any data are left over in the queue of the recipient of the termination request, they are sent and the association is closed. Association termination uses three packets, as shown in Figure 9.62. Note that although the figure shows the case in which termination is initiated by the client, it can also be initiated by the server.

Figure 9.62 Association termination



[Access the text alternative for slide images.]

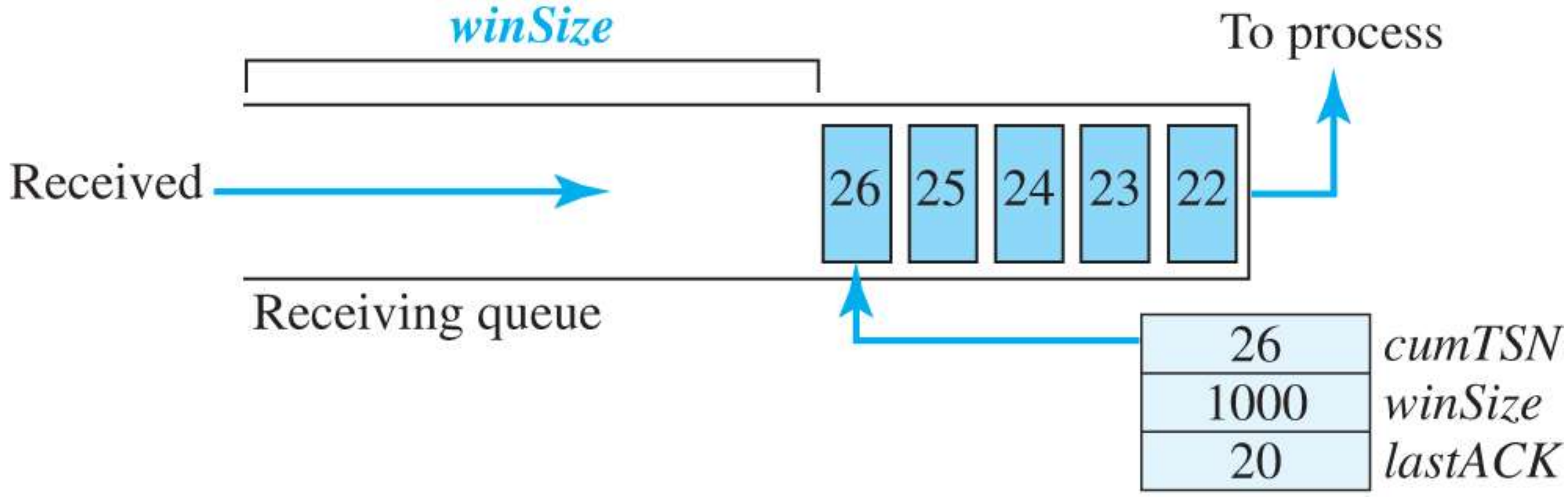
9.5.5 Flow Control

Flow control in SCTP is similar to that in TCP. In SCTP, we need to handle two units of data, the byte and the chunk. The values of rwnd and cwnd are expressed in bytes; the values of TSN and acknowledgments are expressed in chunks. To show the concept, we make some unrealistic assumptions. We assume that there is never congestion in the network and that the network is error free.

***Receiver Site* 1**

The receiver has one buffer (queue) and three variables. The queue holds the received data chunks that have not yet been read by the process. The first variable holds the last TSN received, cumTSN. The second variable holds the available buffer size, winSize. The third variable holds the last cumulative acknowledgment, lastACK. Figure 9.63 shows the queue and variables at the receiver site.

Figure 9.63 Flow control, receiver site

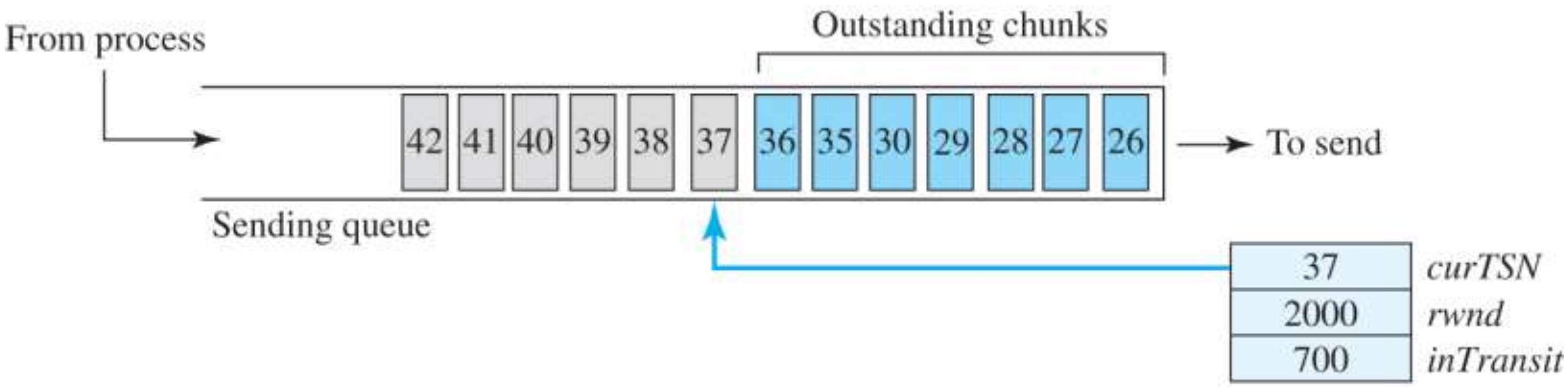


Access the text alternative for slide images.

Sender Site 1

The sender has one buffer (queue) and three variables: curTSN, rwnd, and inTransit, as shown in Figure 9.64. We assume each chunk is 100 bytes long.

Figure 9.64 Flow control, sender site



Access the text alternative for slide images.

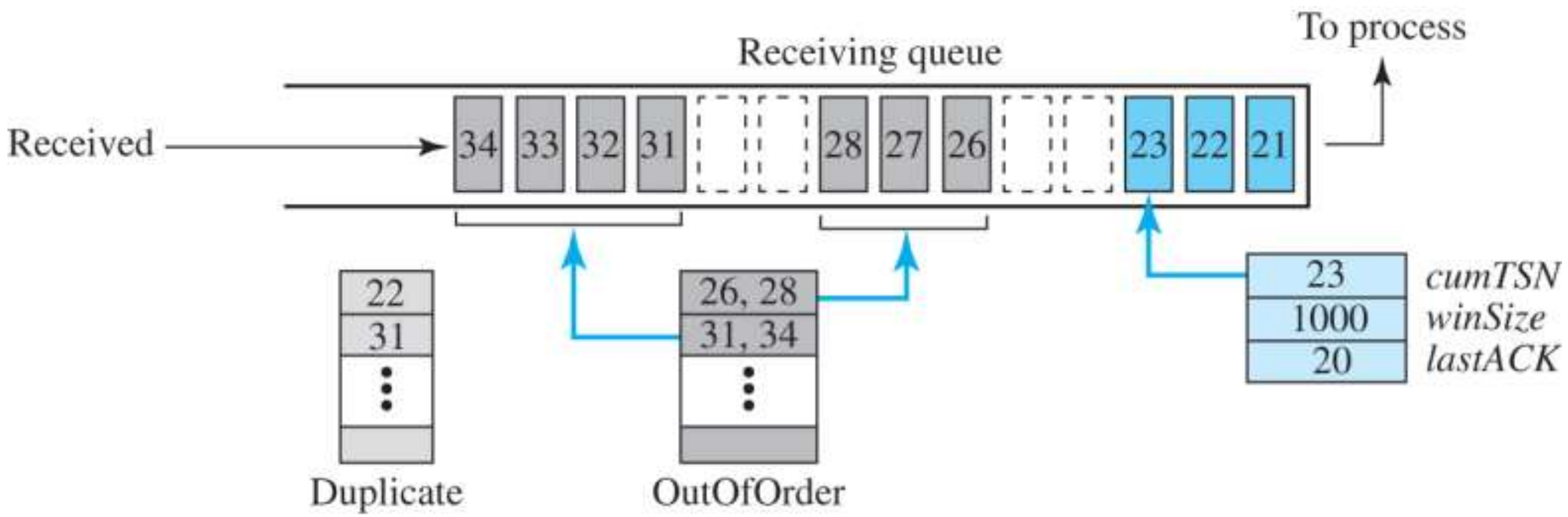
9.5.6 Error Control

SCTP, like TCP, is a reliable transport-layer protocol. It uses a SACK chunk to report the state of the receiver buffer to the sender. Each implementation uses a different set of entities and timers for the receiver and sender sites. We use a very simple design to convey the concept to the reader.

Receiver Site 2

In our design, the receiver stores all chunks that have arrived in its queue including the out-of-order ones. However, it leaves spaces for any missing chunks. It discards duplicate messages, but keeps track of them for reports to the sender. Figure 9.65 shows a typical design for the receiver site and the state of the receiving queue at a particular point in time.

Figure 9.65 Error control, receiver site



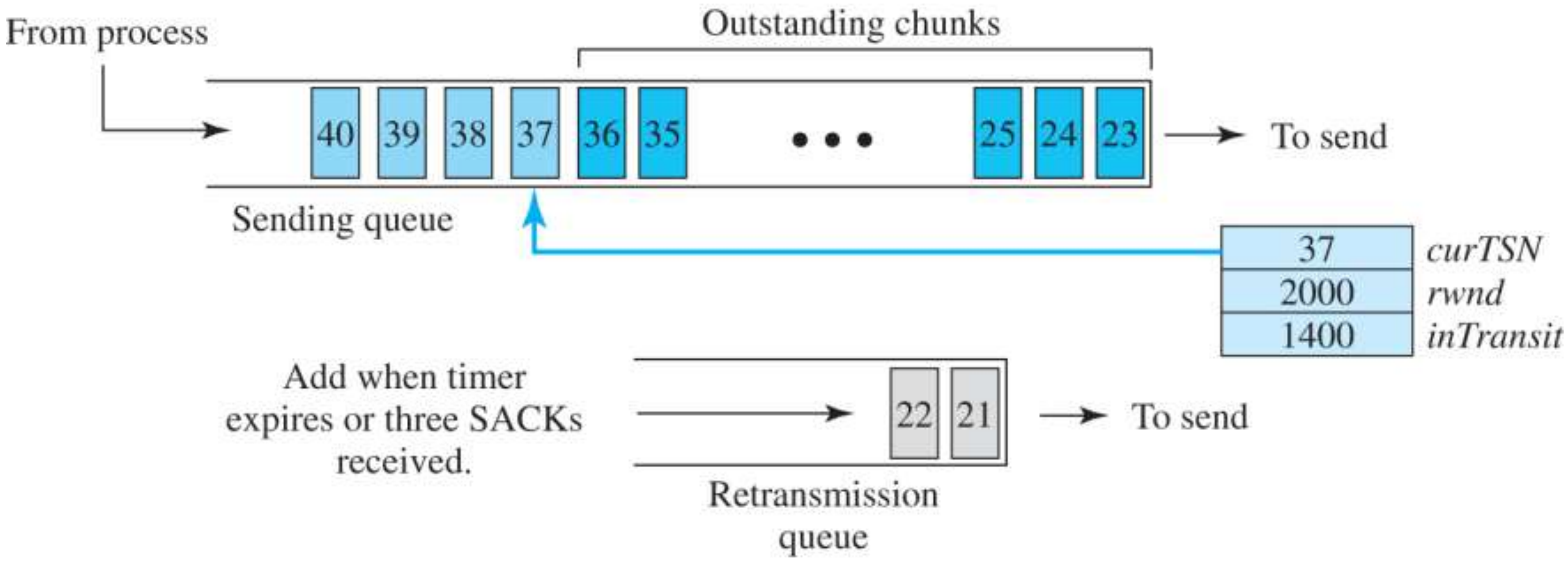
Type: 3 / Flag: 0	Length: 32
Cumulative TSN: 23	
Advertised receiver window credit: 1000	
Number of gap ACK blocks: 2	Number of duplicates: 2
Gap ACK block #1 start: 3	Gap ACK block #1 end: 5
Gap ACK block #2 start: 8	Gap ACK block #2 end: 11
Duplicate TSN: 22	
Duplicate TSN: 31	

Access the text alternative for slide images.

Sender Site 2

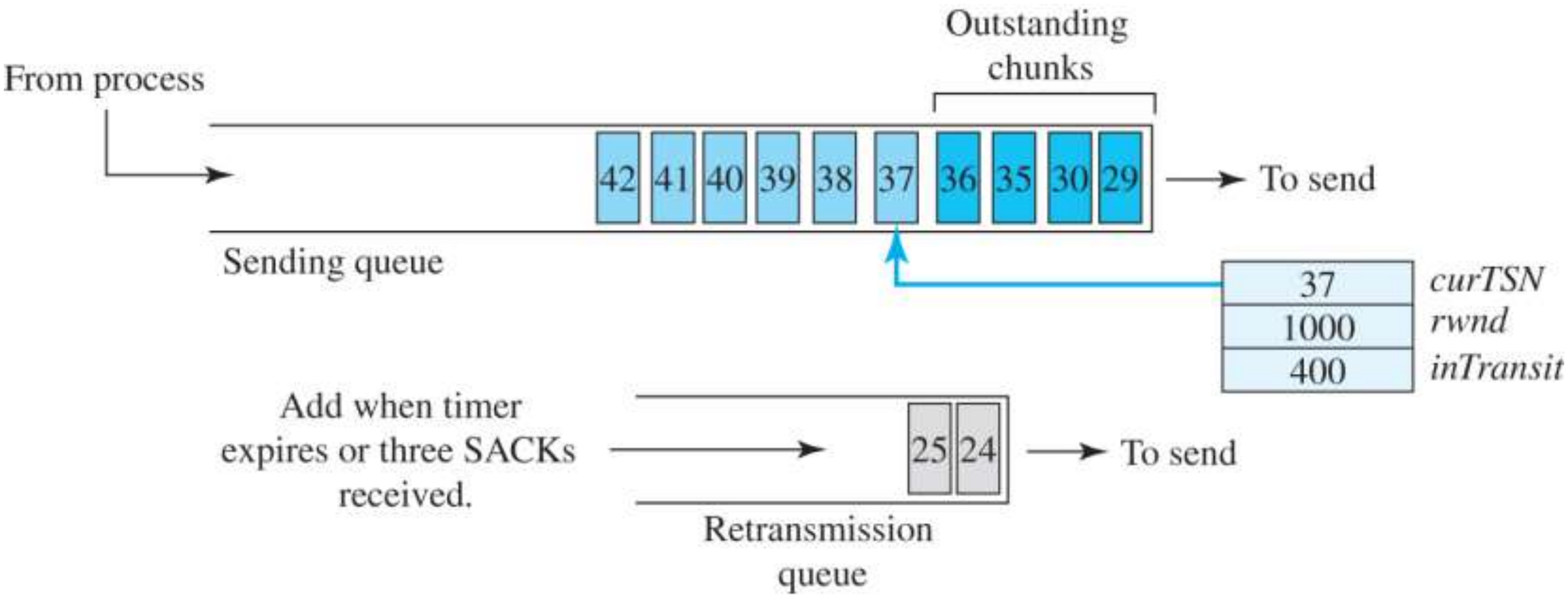
At the sender site, our design demands two buffers (queues): a sending queue and a retransmission queue. We also use three variables: rwnd, inTransit, and curTSN, as described in the previous section. Figure 9.66 shows a typical design.

Figure 9.66 Error control, sender site



Access the text alternative for slide images.

Figure 9.67 New state at the sender site after receiving a SACK chunk



Access the text alternative for slide images.

Sending Data Chunks

An end can send a data packet whenever there are data chunks in the sending queue with a TSN greater than or equal to curTSN or if there are data chunks in the retransmission queue. The retransmission queue has priority. However, the total size of the data chunk or chunks included in the packet must not exceed the (rwnd - inTransit) value and the total size of the frame must not exceed the MTU size, as we discussed in previous sections.

Generating SACK Chunks

Another issue in error control is the generation of SACK chunks. The rules for generating SCTP SACK chunks are similar to the rules used for acknowledgment with the TCP ACK flag.

Congestion Control 2

SCTP, like TCP, is a transport-layer protocol with packets subject to congestion in the network. The SCTP designers have used the same strategies for congestion control as those used in TCP.

Because learning changes everything.®

www.mheducation.com