UNIT 4- TRANSPORT LAYER
UDP and TCP

Prepared By

Haseeba Yaseen

23-1 PROCESS-TO-PROCESS DELIVERY

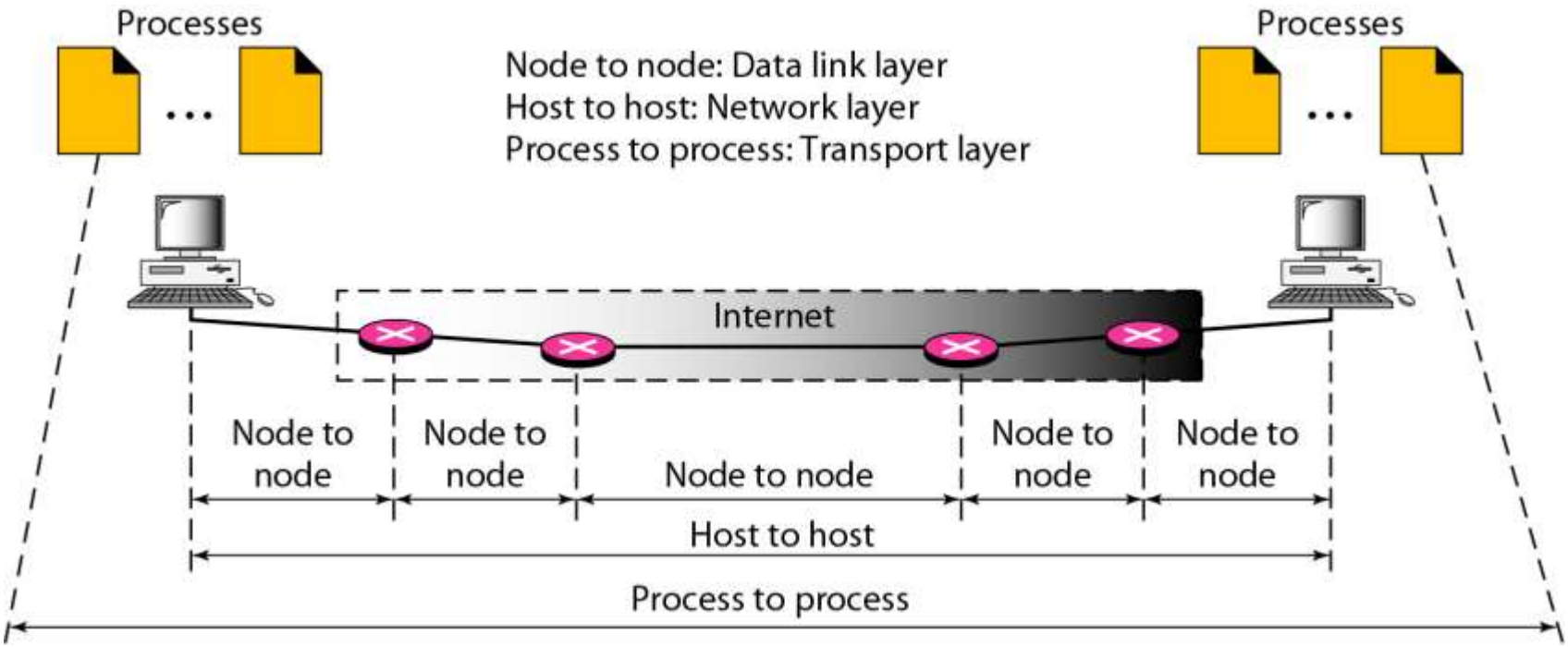
***The transport layer is responsible for process-to-process delivery*—the delivery of a packet, part of a message, from one process to another. Two processes communicate in a client/server relationship.**

Topics discussed in this section:

Client/Server Paradigm
Multiplexing and Demultiplexing
Connectionless Versus Connection-Oriented Service
Reliable Versus Unreliable
Three Protocols

- Real communication takes place between two processes (application programs).
- At any moment, several processes may be running on the source host and several on the destination host.
- **To complete the delivery, we need a mechanism to deliver data from one of these processes running on the source host** to the **corresponding process running on the destination host**.
- The transport layer is responsible for process-to-process delivery-the delivery of a packet, from one process to another.
- **Two processes communicate in a client/server relationship,**
- Figure 23.1 shows these three types of deliveries and their domains.

Figure 23.1 *Types of data deliveries*



1. Client/Server Paradigm:

- There are several ways to achieve process-to-process communication, the **most common one 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.
- Both processes (client and server) have the same name.
- **For example**, to get the day and time from a remote machine,

  we need a Daytime client process running on the local host

  and a Daytime server process running on a remote machine.

For communication, we must define the following:

1. Local host
2. Local process
3. Remote host
4. Remote process

2.Addressing:

- Whenever we need to deliver something to one specific destination among many, we need an address.
- At the transport layer, we need a transport layer address, called a port number, to choose among multiple processes running on the destination host.
- The destination port number is needed for delivery; the source port number is needed for the reply
- In the Internet model, the port numbers are 16-bit integers between 0 and 65,535.
- The client program defines itself with a port number, chosen randomly by the transport layer software running on the client host. This is the ephemeral port number.

- The server process must also define itself with a port number. This port number, however, cannot be chosen randomly.
- If the computer at the server site runs a server process and assigns a random number as the port number,
- the process at the client site that wants to access that server and use its services **will not know the port number.**
- The Internet has decided to use universal port numbers for servers; **these are called well known port numbers.**
- There are some exceptions to this rule; for example, there are clients that are assigned well-known port numbers.
- Every client process knows the well-known port number of the corresponding server process

Figure 23.2 *Port numbers*

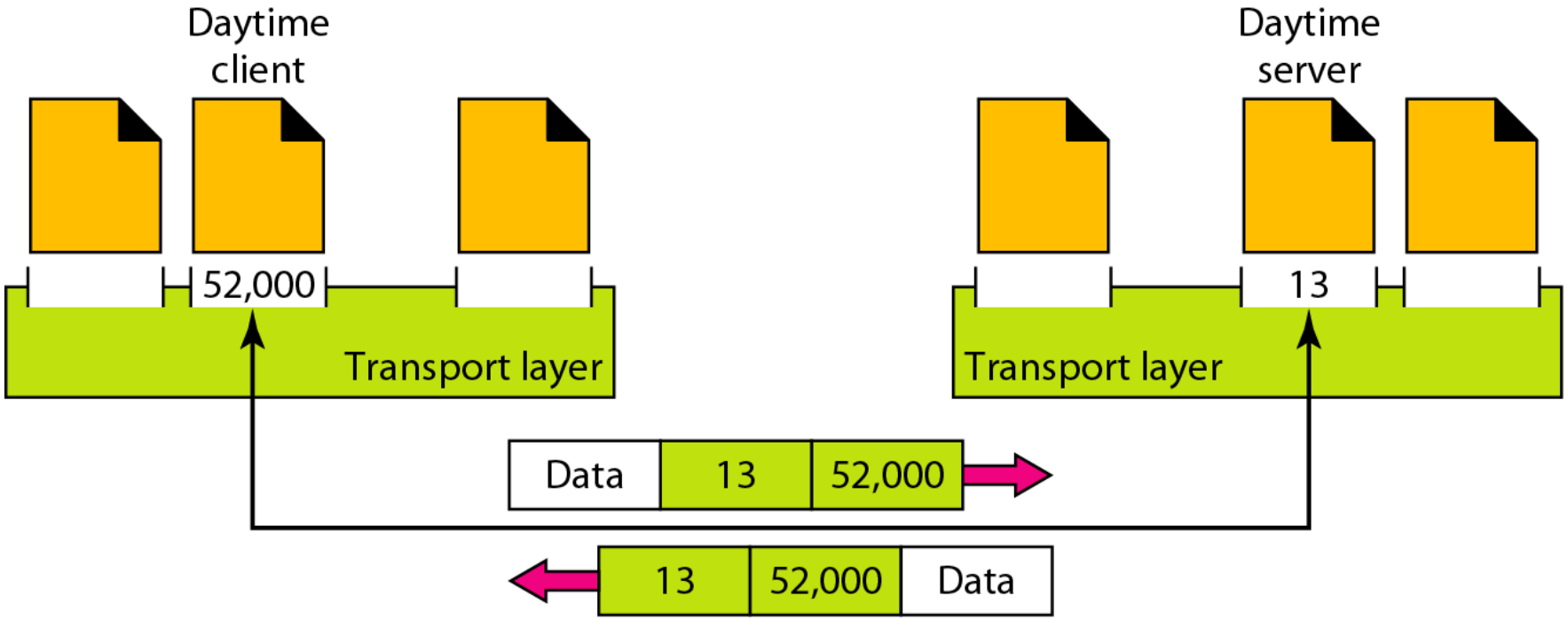
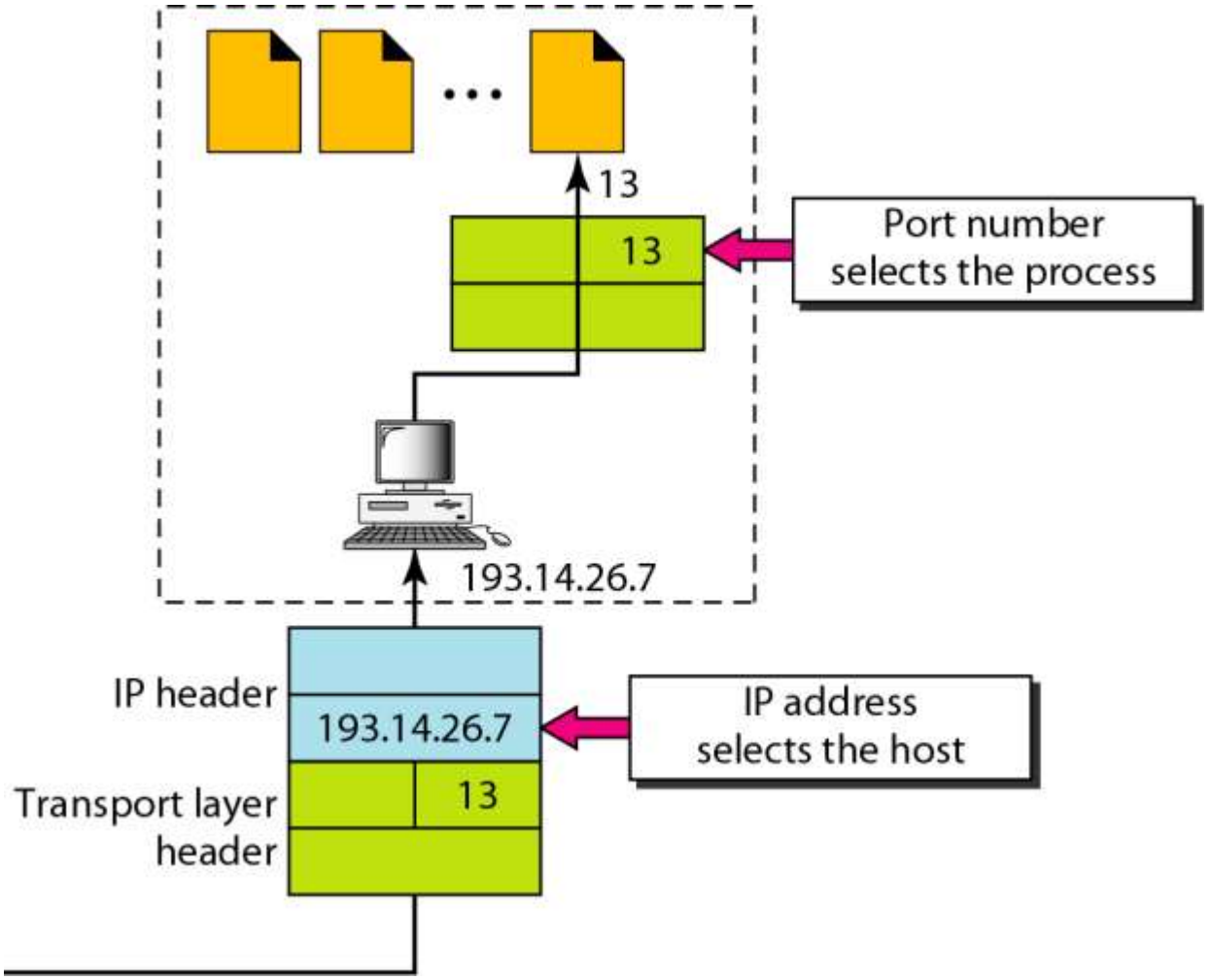


Figure 23.3 *IP addresses versus port numbers*



3. lANA Ranges:

- The lANA (Internet Assigned Number Authority) has divided the port numbers into three ranges:

1.Well-known ports.

- The ports ranging from 0 to 1023 are assigned and controlled by lANA. These are the well-known ports.

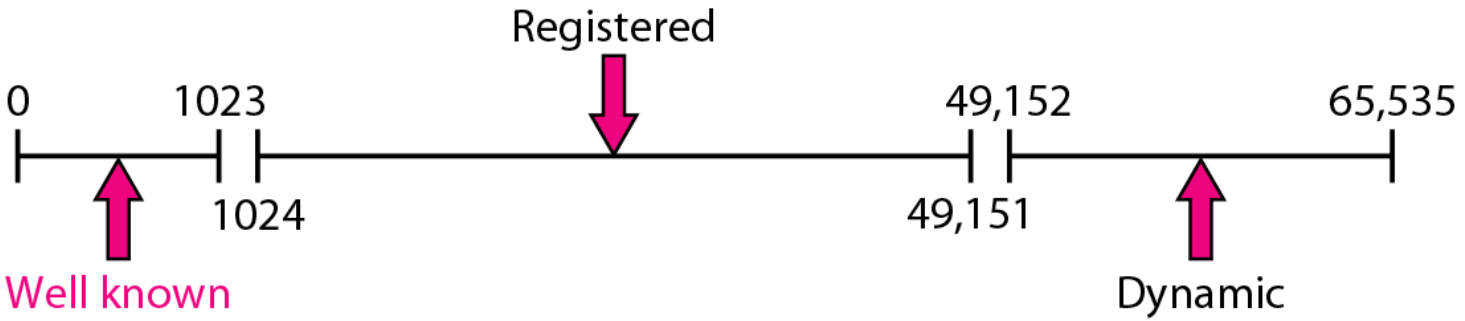
2.Registered ports.

- The ports ranging from 1024 to 49,151 are not assigned or controlled by lANA.
- They can only be registered with lANA to prevent duplication.

3.**Dynamic ports**. The ports ranging from 49,152 to 65,535 are neither controlled nor registered.

- They can be used by any process. These are the ephemeral ports

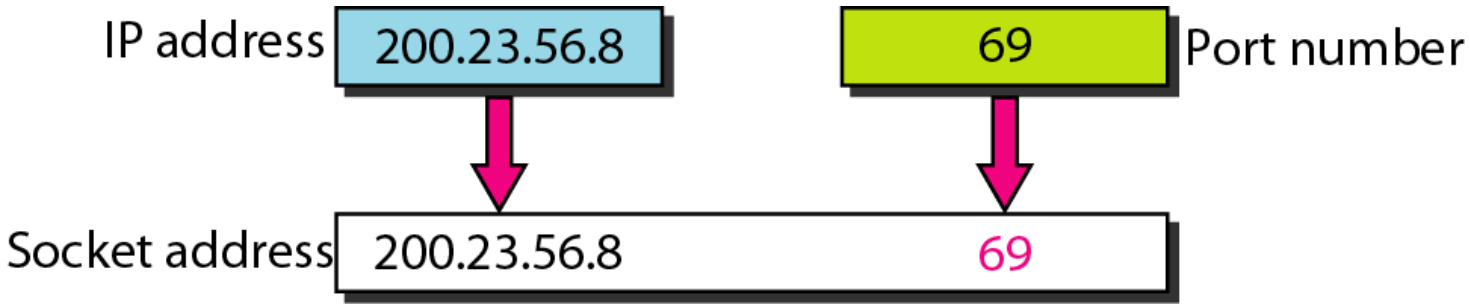
Figure 23.4 *IANA ranges*



4. Socket Addresses:

- Process-to-process delivery needs two identifiers, IP address and the port number, at each end to make a connection.
- The combination of an IP address and a port number is called a socket address.
- The client socket address defines the client process uniquely just as the server socket address defines the server process uniquely (see Figure 23.5).
- A transport layer protocol needs a pair of socket addresses: the client socket address and the server socket address.
- These four pieces of information are part of the IP header and the transport layer protocol header.
- The IP header contains the IP addresses; **the UDP or TCP header contains the port numbers.**

Figure 23.5 *Socket address*



5. Multiplexing and Demultiplexing :

- The addressing mechanism allows multiplexing and demultiplexing by the transport layer, as shown in Figure 23.6.

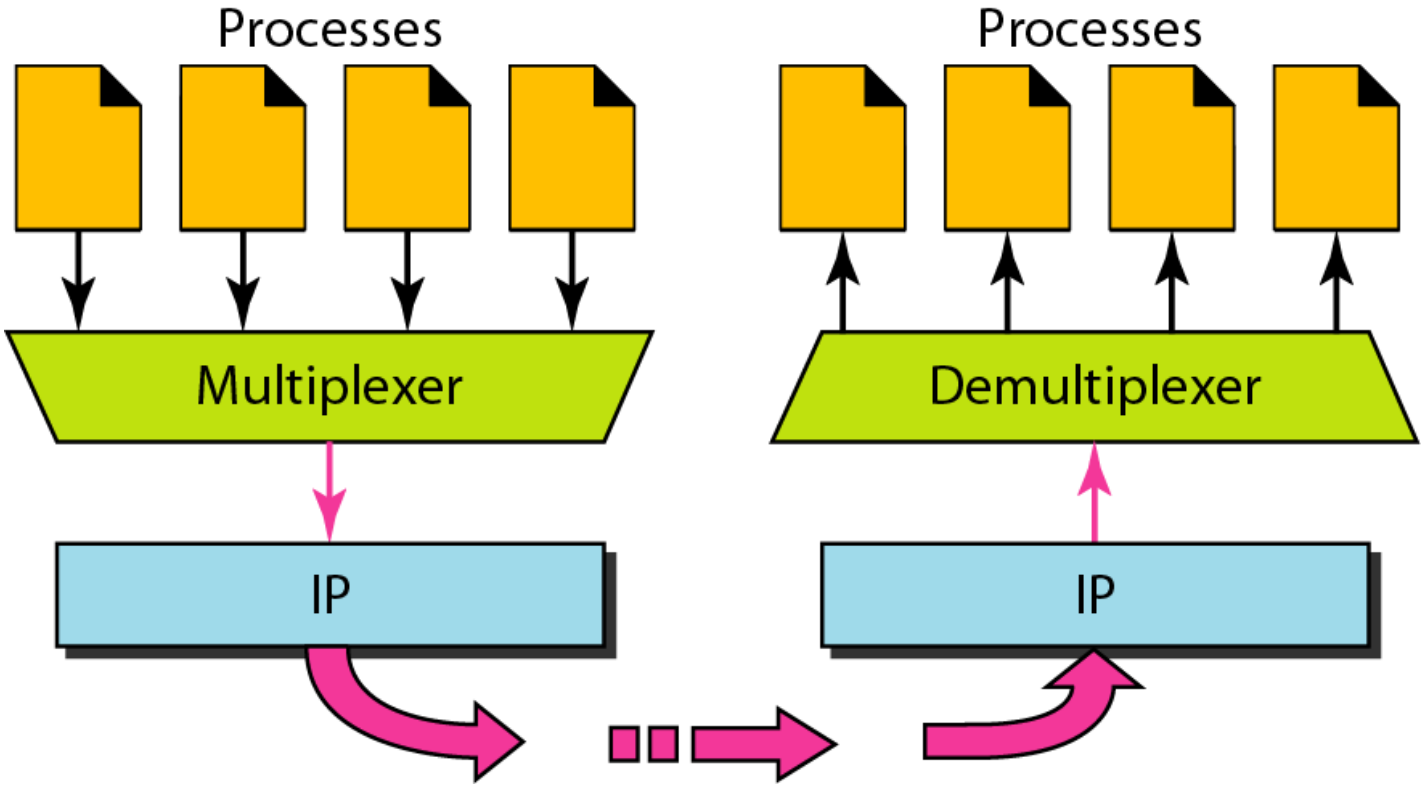
Multiplexing:

- At the sender site, there may be several processes that need to send packets.
- However, there is only one transport layer protocol at any time.
- This is a many-to-one relationship **and requires multiplexing**.
- The protocol accepts messages from different processes, differentiated by their assigned port numbers.
- After adding the header, the transport layer passes the packet to the network layer.

Demultiplexing:

- At the receiver site, the relationship is one-to-many and requires demultiplexing.
- The transport layer receives datagrams from the network layer. After error checking and dropping of the header,
- the transport layer delivers each message to the appropriate process based on the port number.

Figure 23.6 *Multiplexing and demultiplexing*



6. Connectionless Versus Connection-Oriented Service:

- A transport layer protocol can either be connectionless or connection-oriented.

Connectionless Service:

- In a connectionless service, the packets are sent from one party to another with no need for connection establishment or connection release.
  - they may be delayed or lost or may arrive out of sequence.
  - There is no acknowledgment either.
  - one of the transport layer protocols in the Internet model, UDP, is connectionless.

Connection~Oriented Service:

- In a connection-oriented service, a connection is first established between the sender and the receiver.
- Data are transferred.
- At the end, the connection is released.
- TCP and SCTP are connection-oriented protocols.

Reliable Versus Unreliable:

- The transport layer service can be reliable or unreliable.
- If the application layer program needs reliability, we use a reliable transport layer protocol by implementing flow and error control at the transport layer.
- This is for slower and more complex service.
- On the other hand, if the application program does not need reliability
- it needs fast service or the nature of the service does not demand flow and error control (real-time applications), then an unreliable protocol can be used.
- UDP is connectionless and unreliable;
- TCP and SCTP are connection oriented and reliable.
- These three can respond to the demands of the application layer programs.

Figure 23.7 *Error control*

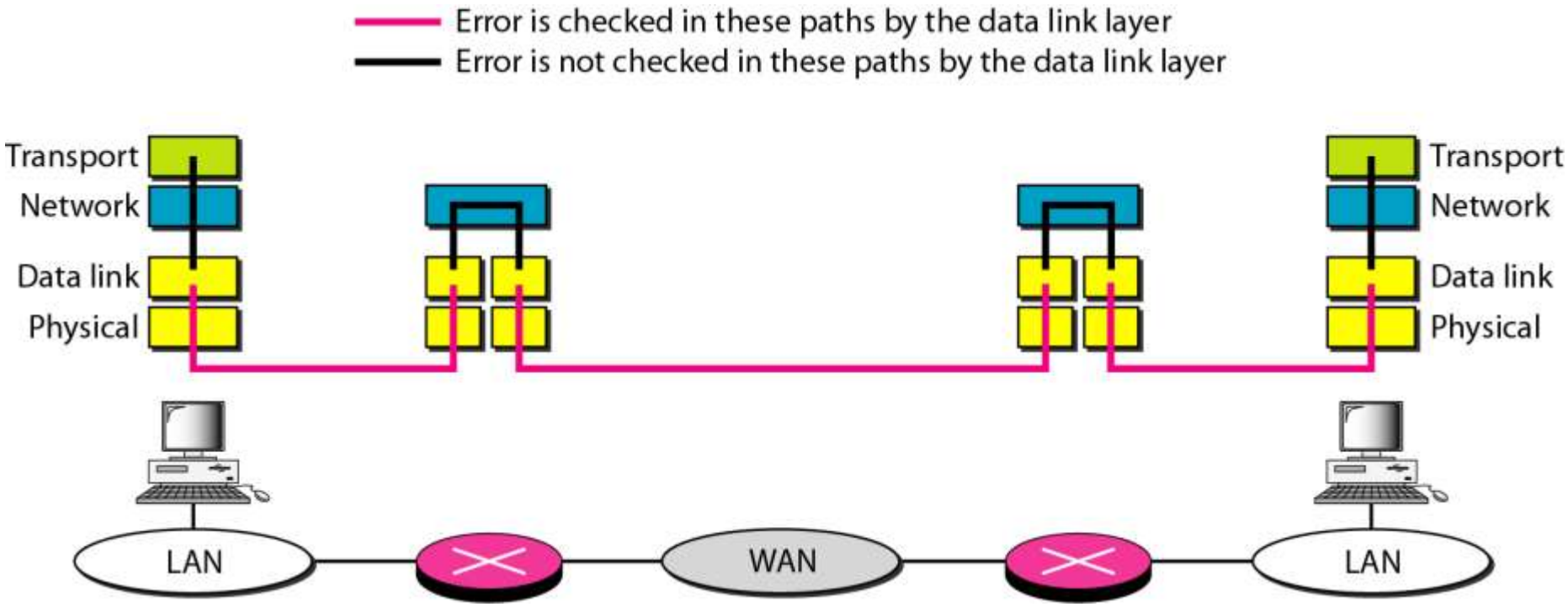
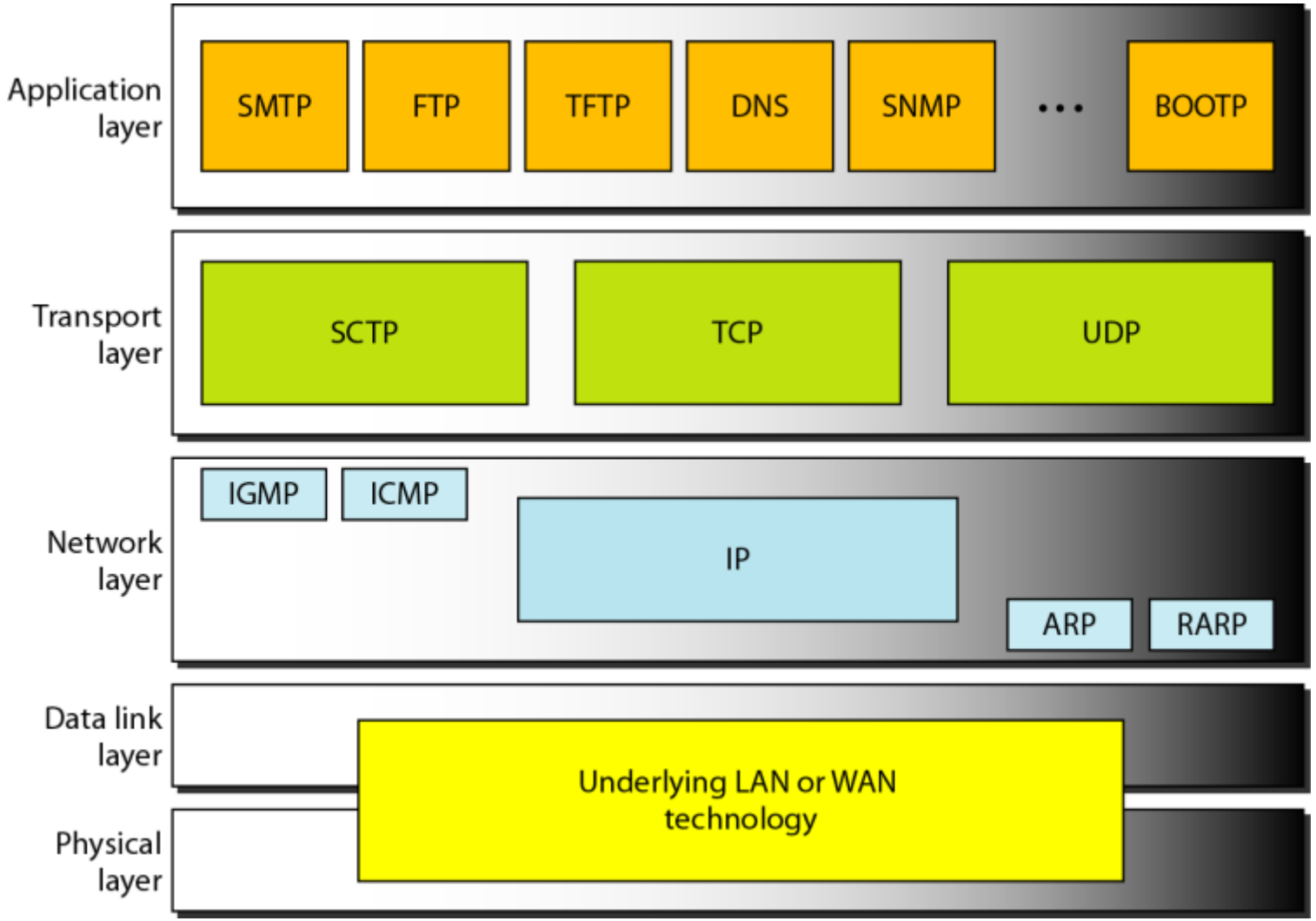


Figure 23.8 *Position of UDP, TCP, and SCTP in TCP/IP suite*



23-2 USER DATAGRAM PROTOCOL (UDP)

The User Datagram Protocol (UDP) is called a connectionless, unreliable transport protocol.

It does not add anything to the services of IP except to provide process-to-process communication instead of host-to-host communication.

Topics discussed in this section:

Well-Known Ports for UDP
User Datagram
Checksum
UDP Operation
Use of UDP

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

- If a process wants to send a small message and **does not care much about reliability, it can use UDP.**
- Sending a small message by using UDP takes much less interaction between the sender and receiver than using TCP or SCTP.

Well-Known Ports for UDP:

- Table 23.1 shows some well-known port numbers used by UDP. Some port numbers can be used by both UDP and TCP

Table 23.1 *Well-known ports used with UDP*

Port	*Protocol*	*Description*
7	Echo	Echoes a received datagram back to the sender
9	Discard	Discards any datagram that is received
11	Users	Active users

Example 23.1

In UNIX, the well-known ports are stored in a file called /etc/services. Each line in this file gives the name of the server and the well-known port number. We can use the grep utility to extract the line corresponding to the desired application. The following shows the port for FTP. Note that FTP can use port 21 with either UDP or TCP.

```
$ grep    ftp   /etc/services
ftp           21/tcp
ftp           21/udp
```

Example 23.1 (continued)

SNMP uses two port numbers (161 and 162), each for a different purpose.

```
$ grep      snmp /etc/services
snmp            161/tcp         #Simple Net  Mgmt Proto
snmp            161/udp         #Simple Net  Mgmt Proto
snmptrap        162/udp         #Traps for SNMP
```

1. User Datagram UDP packets, called user datagrams, have a fixed-size header of 8 bytes.

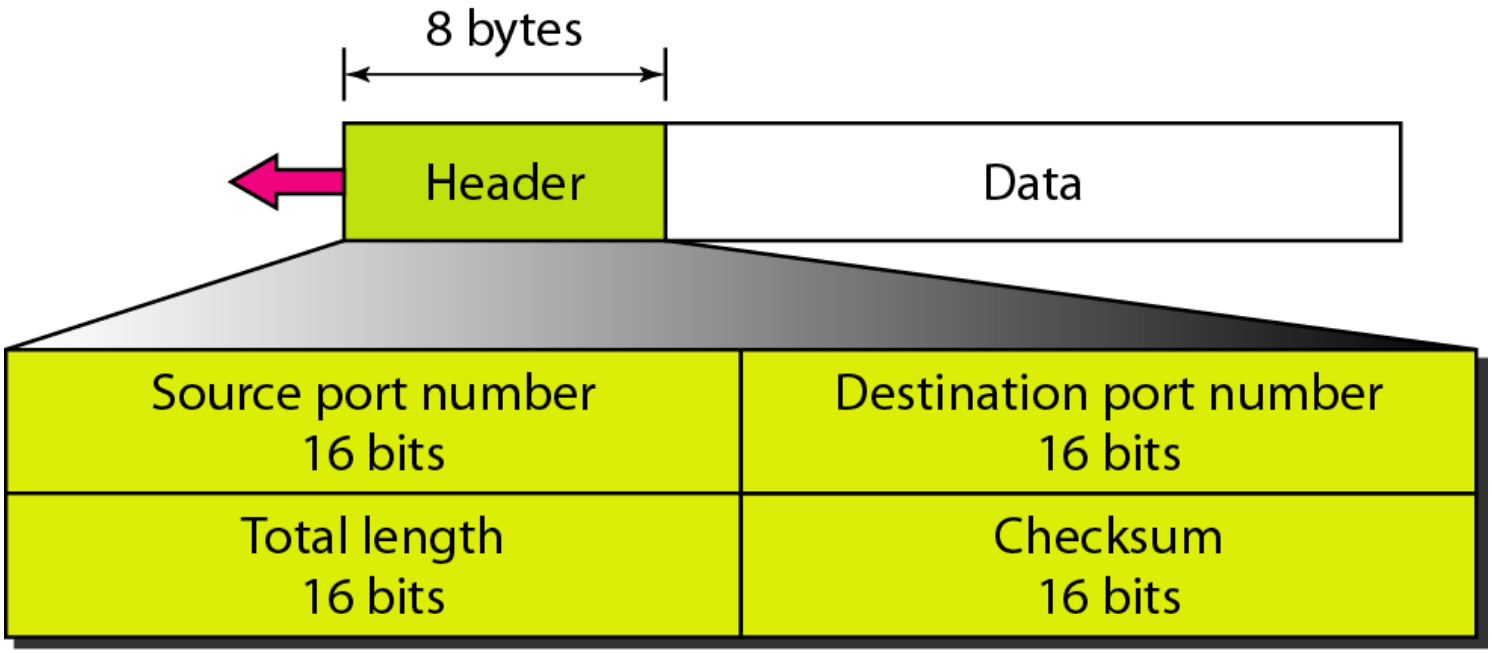
Figure 23.9 shows the format of a user datagram.

The fields are as follows:

Source port number.

- ✓ This is the port number used by the process running on the source host. It is 16 bits long, which means that the port number can range from 0 to 65,535.
- ✓ If the source host is the client (a client sending a request), the port number, in most cases, is an ephemeral port number
- ✓ requested by the process and chosen by the UDP software running on the source host.
- ✓ If the source host is the server (a server sending a response), the port number, in most cases, is a well-known port number

Figure 23.9 *User datagram format*



Destination port number.

- This is the port number used by the process running on the destination host.
- It is also 16 bits long.
- If the destination host is the server (a client sending a request), the port number, in most cases, is a well-known port number.
- If the destination host is the client (a server sending a response), the port number, in most cases, is an ephemeral port number.
- In this case, the server copies the ephemeral port number it has received in the request packet.

- **Length.**
  - This is a 16-bit field that defines the total length of the user datagram, header plus data.
  - The 16 bits can define a total length of 0 to 65,535 bytes.
  - However, the total length needs to be much less because a UDP user datagram is stored in an IP datagram with a total length of 65,535 bytes.
  - UDP Length= IP length- IP Header's Length
  - **Checksum**. This field is used to detect errors over the entire user datagram (header plus data).

Checksum:

- It 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 O's

- The protocol field is added to ensure that the packet belongs to UDP, and not to other transport-layer protocols.
- The value of the protocol field for UDP is 17.
- If this value is changed during transmission, the checksum calculation at the receiver will detect it and UDP drops the packet

Figure 23.10 *Pseudoheader for checksum calculation*

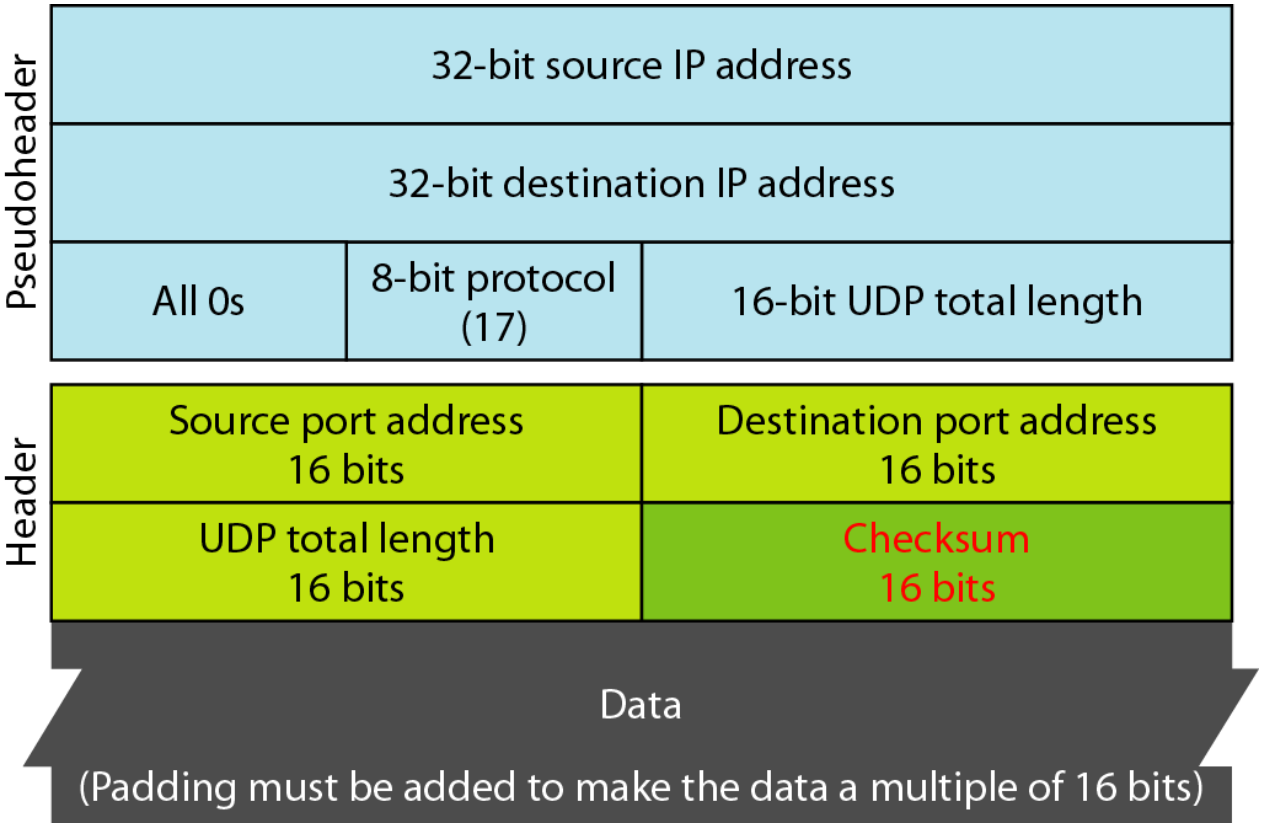
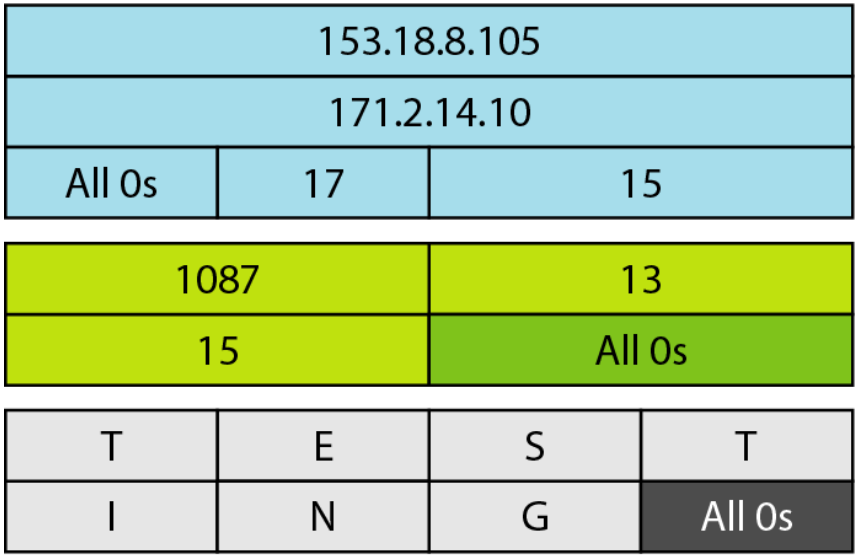
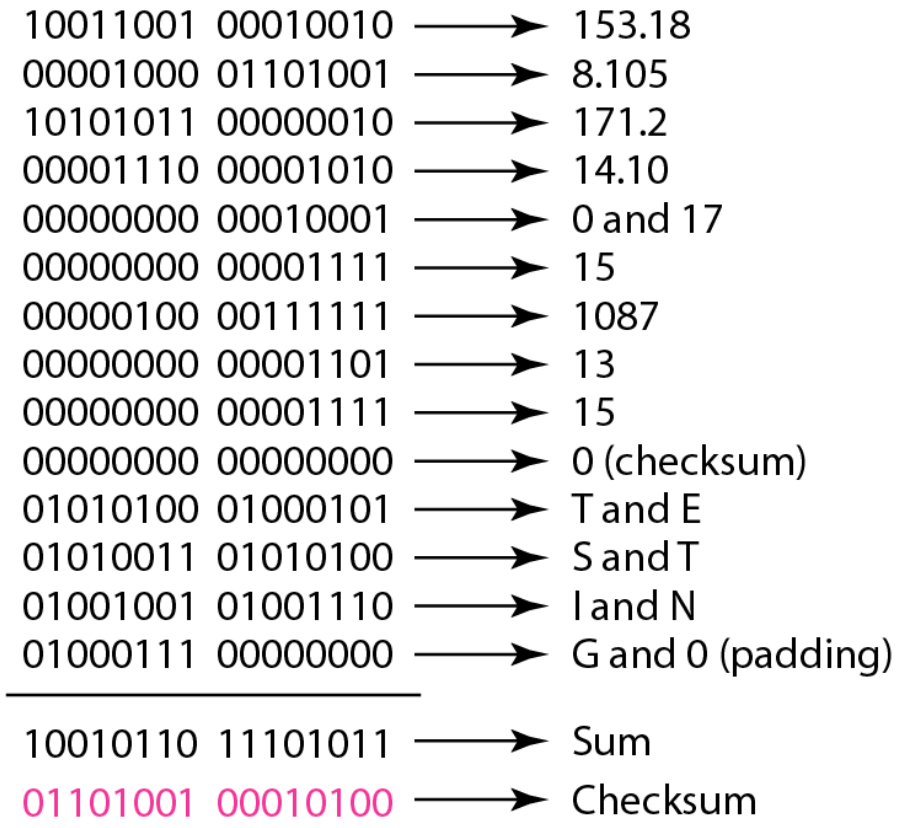


Figure 23.11 *Checksum calculation of a simple UDP user datagram*





2. UDP Operation :

1.Connectionless Services :

- 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.
- each user datagram can travel on a different path.
- process that uses UDP cannot send a stream of data to UDP
- Instead each request must be small enough to fit into one user datagram.
- Only **those processes sending short messages should use UDP.**
- **No Connection Establishment and Termination.**

2.Flow and Error Control UDP

- is a very simple, unreliable transport protocol.
- There is no flow control and hence no window mechanism.
- The receiver may overflow with incoming messages.
- 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 flow control and error control means that the process using UDP should provide these mechanisms.

3.Encapsulation and Decapsulation:

- To send a message from one process to another, the UDP protocol encapsulates and decapsulates messages in an IP datagram.

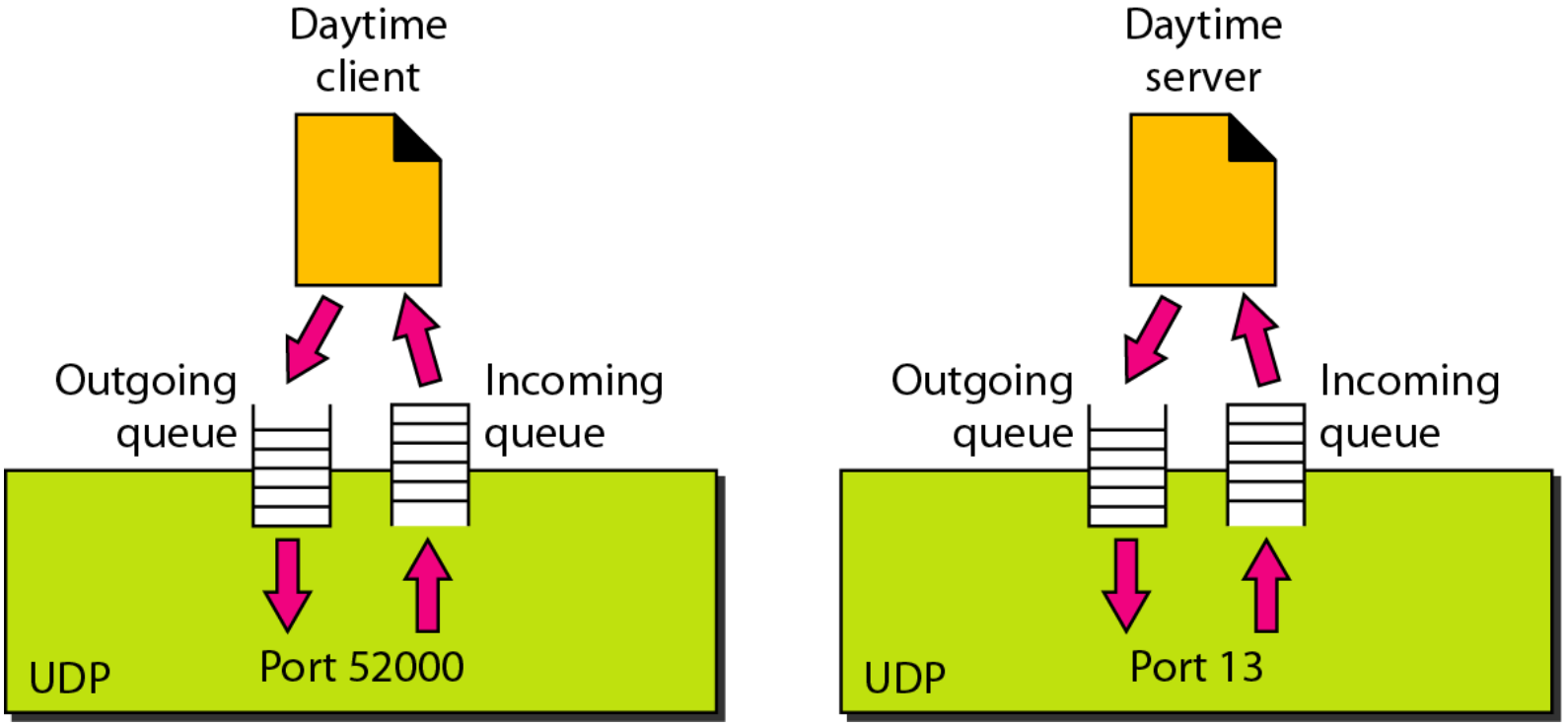
4.Queuing:

- 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
- even if a process wants to communicate with multiple processes, it obtains only one port number **and eventually one outgoing and one incoming queue.**

- The queues opened by the client are, in most cases, identified by ephemeral port numbers.
- The queues function as long as the process is running. When the process terminates, the queues are destroyed.
- The client process can send messages to the outgoing queue by using the source port number specified in the request.
- UDP removes the messages one by one and, after adding the UDP header, delivers them to IP.
- An outgoing queue can overflow.
- If this happens, the operating system can ask the client process to wait before sending any more messages.

- When a message arrives for a client, UDP checks to see if an incoming queue has been created for the port number **specified in the destination port number field of the user datagram.**
- If there is such a queue, UDP sends the received user datagram to the end of the queue.
- If there is no such queue, UDP discards the user datagram and asks the ICMP protocol to send a port unreachable message to the server
- All the incoming messages for one particular client program, whether coming from the same or a different server, are sent to the same queue.
- An incoming queue can overflow.
- If this happens, UDP drops the user datagram and asks for a port unreachable message to be sent to the server

Figure 23.12 *Queues in UDP*



- At the server site, the mechanism of creating queues is different.
- In its simplest form, a server asks for incoming and outgoing queues, **using its well-known port, when it starts running.**
- The queues remain open as long as the server is running.
- When a message arrives for a server, UDP checks to see if an incoming queue has been created for the port number specified in the destination port number field of the user datagram.
- If there is such a queue, UDP sends the received user datagram to the end of the queue.
- If there is no such queue, UDP discards the user datagram and asks the ICMP protocol to send a port unreachable message to the client.
- All the incoming messages for one particular server, whether coming from the same or a different client, are sent to the same queue.

- An incoming queue can overflow.
- If this happens, UDP drops the user datagram and asks for a port unreachable message to be sent to the client.
- When a server wants to respond to a client, it sends messages to the outgoing queue, using the source port number specified in the request.
- UDP removes the messages one by one and, after adding the UDP header, delivers them to IP.
- An outgoing queue can overflow.
- If this happens, **the operating system asks the server to wait before sending any more messages.**

Use of UDP:

- UDP is suitable for a process that requires simple request-response communication with **little concern for flow and error control.**
- It is not usually **used for a process such as FTP** that needs to send bulk data
- UDP is suitable for a process with internal flow and error control mechanisms.
- For example, the **Trivial File Transfer Protocol (TFTP)** process includes flow and error control. It can easily use UDP.
- **UDP is a suitable transport protocol for multicasting**.
- **Multicasting capability is embedded in the UDP software but not in the TCP software**.
- **UDP is used for management processes such as SNMP**
- UDP is used for some route updating protocols such as Routing Information Protocol (RIP)

23-3 TCP

TCP is a connection-oriented protocol; it creates a virtual connection between two TCPs to send data. In addition, TCP uses flow and error control mechanisms at the transport level.

Topics discussed in this section:

TCP Services
TCP Features
Segment
A TCP Connection
Flow Control
Error Control

- TCP is called a connection-oriented, reliable transport protocol.
- It adds connection-oriented and reliability features to the services of IP.

TCP Services: services offered by TCP to the processes at the application layer.

1.Process-to-Process Communication:

- TCP provides process-to-process communication using port numbers.
- Table 23.2 lists some well-known port numbers used by TCP

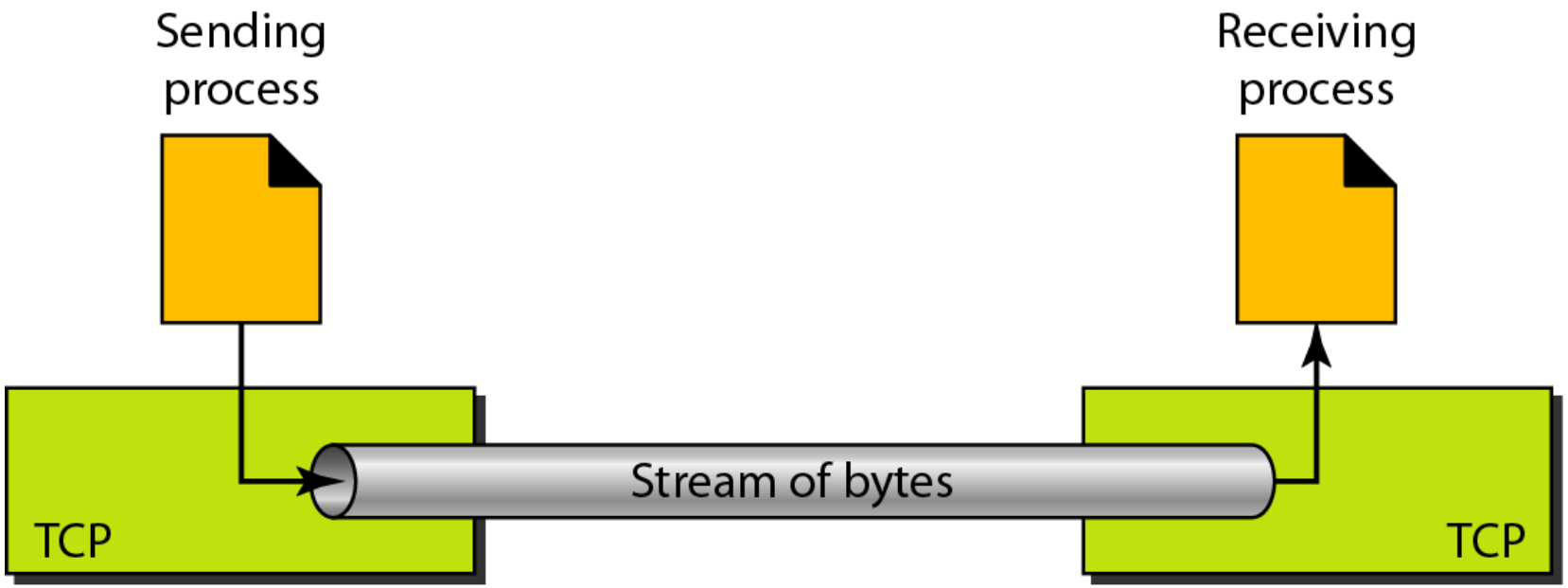
Table 23.2 *Well-known ports used by TCP*

Port	*Protocol*	*Description*
7	Echo	Echoes a received datagram back to the sender
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 (data connection)
21	FTP, Control	File Transfer Protocol (control connection)
23	TELNET	Terminal Network
25	SMTP	Simple Mail Transfer Protocol
53	DNS	Domain Name Server
67	BOOTP	Bootstrap Protocol
79	Finger	Finger
80	HTTP	Hypertext Transfer Protocol
111	RPC	Remote Procedure Call

2.Stream Delivery Service

- TCP, unlike UDP, is a stream-oriented protocol.
- 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 data across the Internet
- Figure 23.13. The sending process produces (writes to) the stream of bytes, and the receiving process consumes (reads from) them.

Figure 23.13 *Stream delivery*

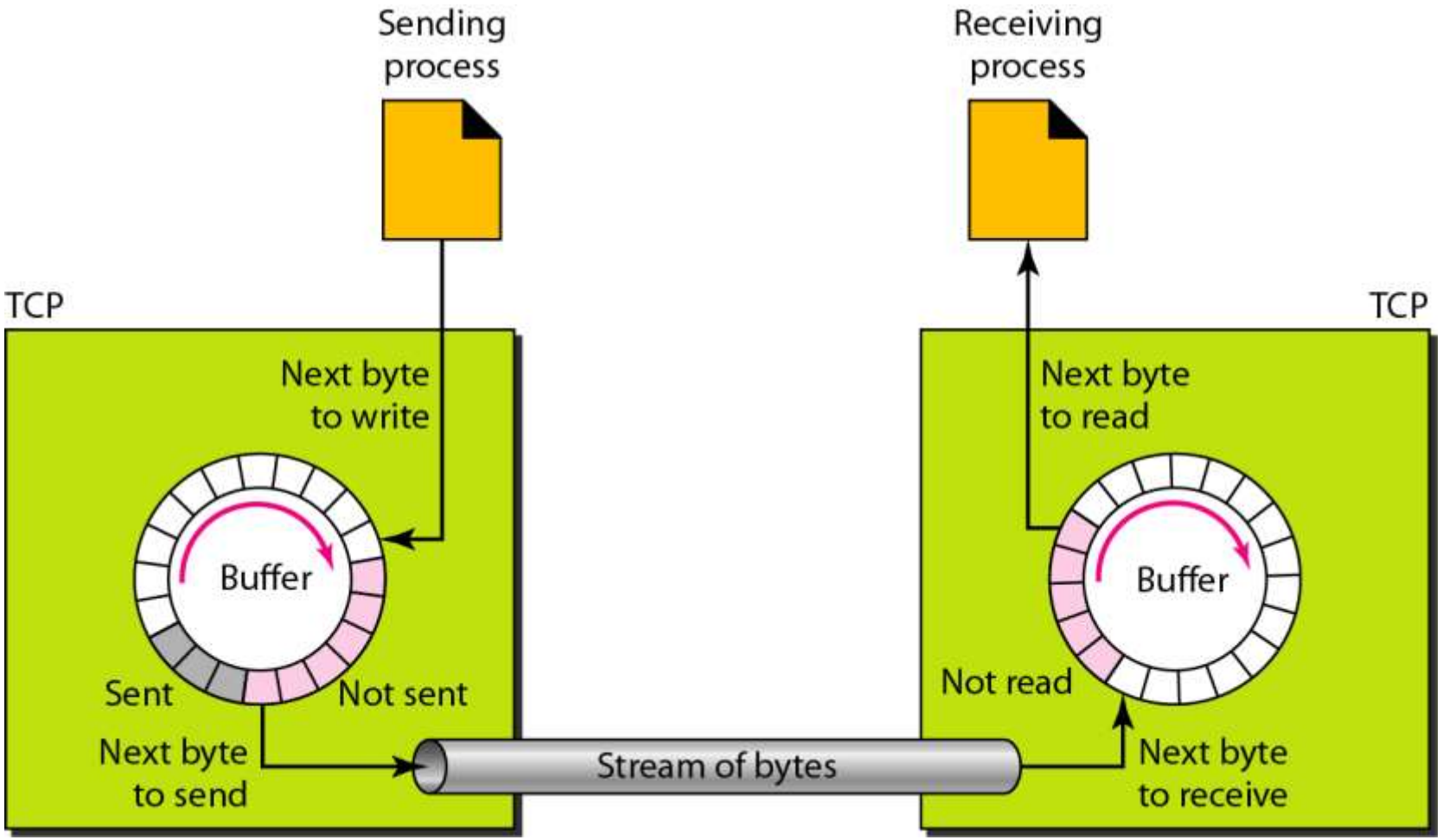


3.Sending and Receiving Buffers:

- the sending and the receiving processes may not write or read data at the same speed, TCP needs buffers for storage.
- There are two buffers, the sending buffer and the receiving buffer, one for each direction.
- One way to implement a buffer is to use a circular array of 1-byte locations as shown in Figure 23.14.

- Figure 23.14 shows the movement of the data in one direction.
- At the sending site, the buffer has three types of chambers.
- The white section contains empty chambers that can be filled by the sending process (producer).
- The gray area holds bytes that have been sent but not yet acknowledged.
- TCP keeps these bytes in the buffer until it receives an acknowledgment.
- The colored area contains bytes to be sent by the sending TCP.
- after the bytes in the gray chambers are acknowledged, the chambers are recycled and available for use by the sending process. Thus it is a circular buffer.

Figure 23.14 *Sending and receiving buffers*

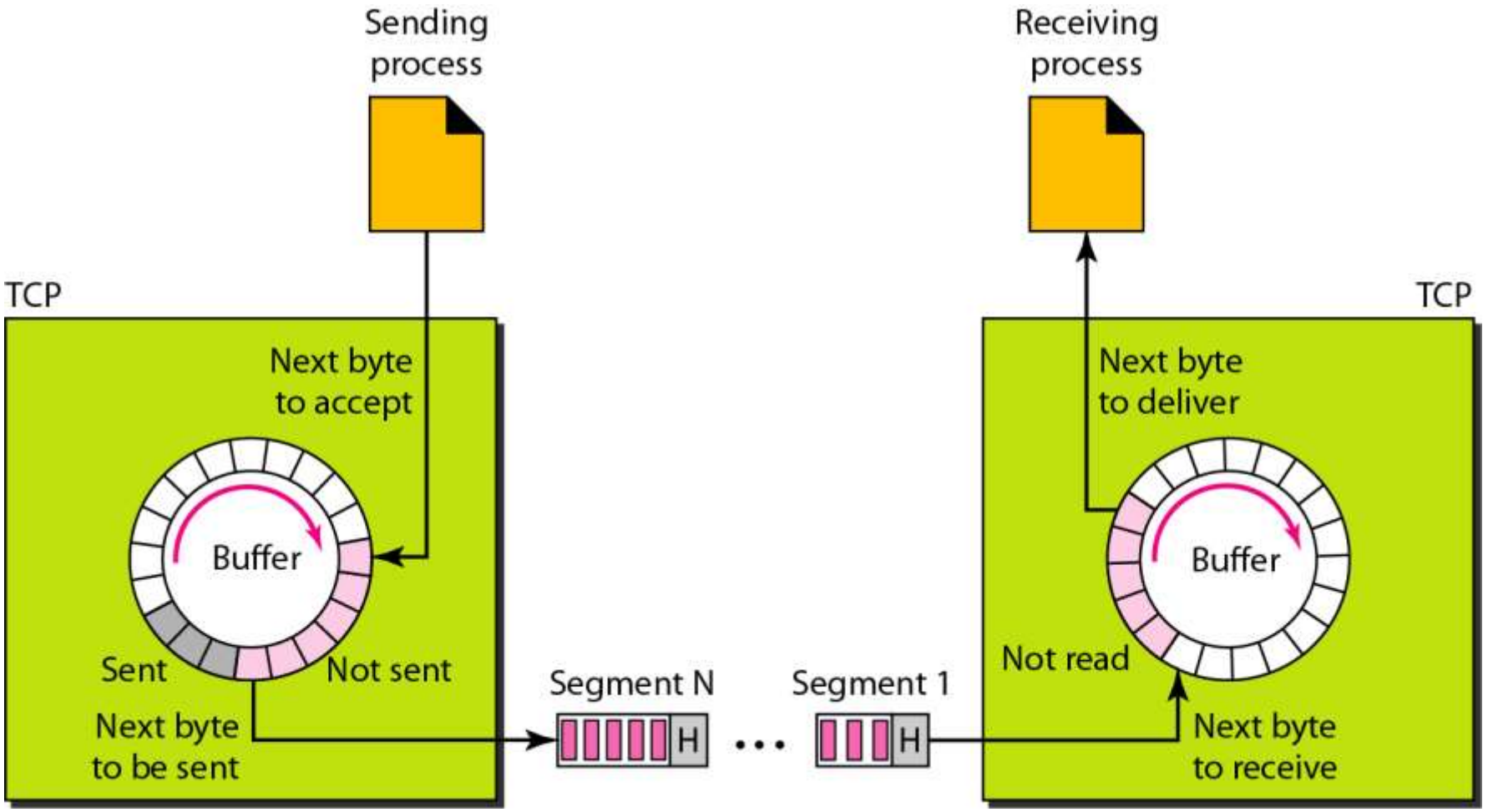


- Buffer at the receiver:-
- The circular buffer is divided into two areas (shown as white and colored).
- The white area contains empty chambers to be filled by bytes received from the network.
- The colored sections contain received bytes that can be read by the receiving process.
- When a byte is read by the receiving process, the chamber is recycled and added to the pool of empty chambers.

4. Segments:

- At the transport layer, TCP groups a number of bytes together **into a packet called a segment**.
- TCP adds a header to each segment (for control purposes) and delivers the segment to the IP layer for transmission.
- The segments are encapsulated in IP datagrams and transmitted.
- This entire operation is transparent to the receiving process.
- All these are handled by TCP with the receiving process unaware of any activities
- Segments are not necessarily the same size

Figure 23.15 *TCP segments*



5.Full-Duplex Communication TCP offers full-duplex service, in which data can flow in both directions at the same time.

Each TCP then has a sending and receiving buffer, and segments move in both directions.

6.Connection-Oriented Service

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

1. The two TCPs establish a connection between them.
2. Data are exchanged in both directions.
3. The connection is terminated.

- ✓ this is a virtual connection, not a physical connection.
- TCP creates a stream-oriented environment in which it accepts the responsibility of TCP delivering the bytes in order to the other site.

7. Reliable Service

- TCP is a reliable transport protocol.
- It uses an acknowledgment mechanism to check the safe and sound arrival of data.

TCP Features:

1.Numbering System:

- 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 the byte number and not the segment number.

2.Byte Number:

- TCP numbers all data bytes that are transmitted in a connection. Numbering is independent in each direction.
- When TCP receives bytes of data from a process, it stores them in the sending buffer and numbers them.

- The numbering does not necessarily start from O.
- Instead, TCP generates a random number between 0 and

(2 power 32) - 1 for the number of the first byte.

For example, if the random number happens to be 1057 and the total data to be sent are 6000 bytes, the bytes are numbered from 1057 to 7056

3. Sequence Number

- After the bytes have been numbered, TCP assigns a sequence number to each segment that is being sent.
- The sequence number for each segment is the number of the first byte carried in that segment

- When a segment carries a combination of data and control information (piggybacking), it uses a sequence number.
- If a segment does not carry user data, it does not logically define a sequence number.
- The field is there, but the value is not valid.
- However, some segments, when carrying only control information, need a sequence number to allow an acknowledgment from the receiver.
- These segments are used for connection establishment, termination, or abortion.

- Each of these segments consumes one sequence number as though it carried 1 byte, but there are no actual data.
- If the randomly generated sequence number is x, the first data byte is numbered x + 1.
- The byte x is considered a phony byte that is used for a control segment to open a connection

Acknowledgment Number:

- The sequence number in each direction shows the number of the first byte carried by the segment.
- Each party also uses an acknowledgment number to confirm the bytes it has received.
- the acknowledgment number defines the number of the next byte that the party expects to receive.
- the acknowledgment number is cumulative, which means that the party takes the number of the last byte that it has received, safe and sound, adds 1 to it, and announces this sum as the acknowledgment number.

- The term cumulative here means that if a party uses 5643 as an acknowledgment number, it has received all bytes from the beginning up to 5642.
- Note that this does not mean that the party has received 5642 bytes because the first byte number does not have to start from O

Note

The bytes of data being transferred in each connection are numbered by TCP. The numbering starts with a randomly generated number.

Example 23.3

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)

Note

The value in the sequence number field of a segment defines the number of the first data byte contained in that segment.

Note

The value of the acknowledgment field in a segment defines the number of the next byte a party expects to receive. The acknowledgment number is cumulative.

Flow Control TCP:

- The receiver of the data controls the amount of data that are to be sent by the sender.
- This is done to prevent the receiver from being overwhelmed with data.
- The numbering system allows TCP to use a byte-oriented flow control.

Error Control To provide reliable service,

- error control considers a segment as the unit of data for error detection (loss or corrupted segments), error control is byte-oriented,

Congestion Control

The amount of data sent by a sender is not only controlled by the receiver (flow control), but is also determined by the level of congestion in the network

TCP SEGMENT FORMAT:-

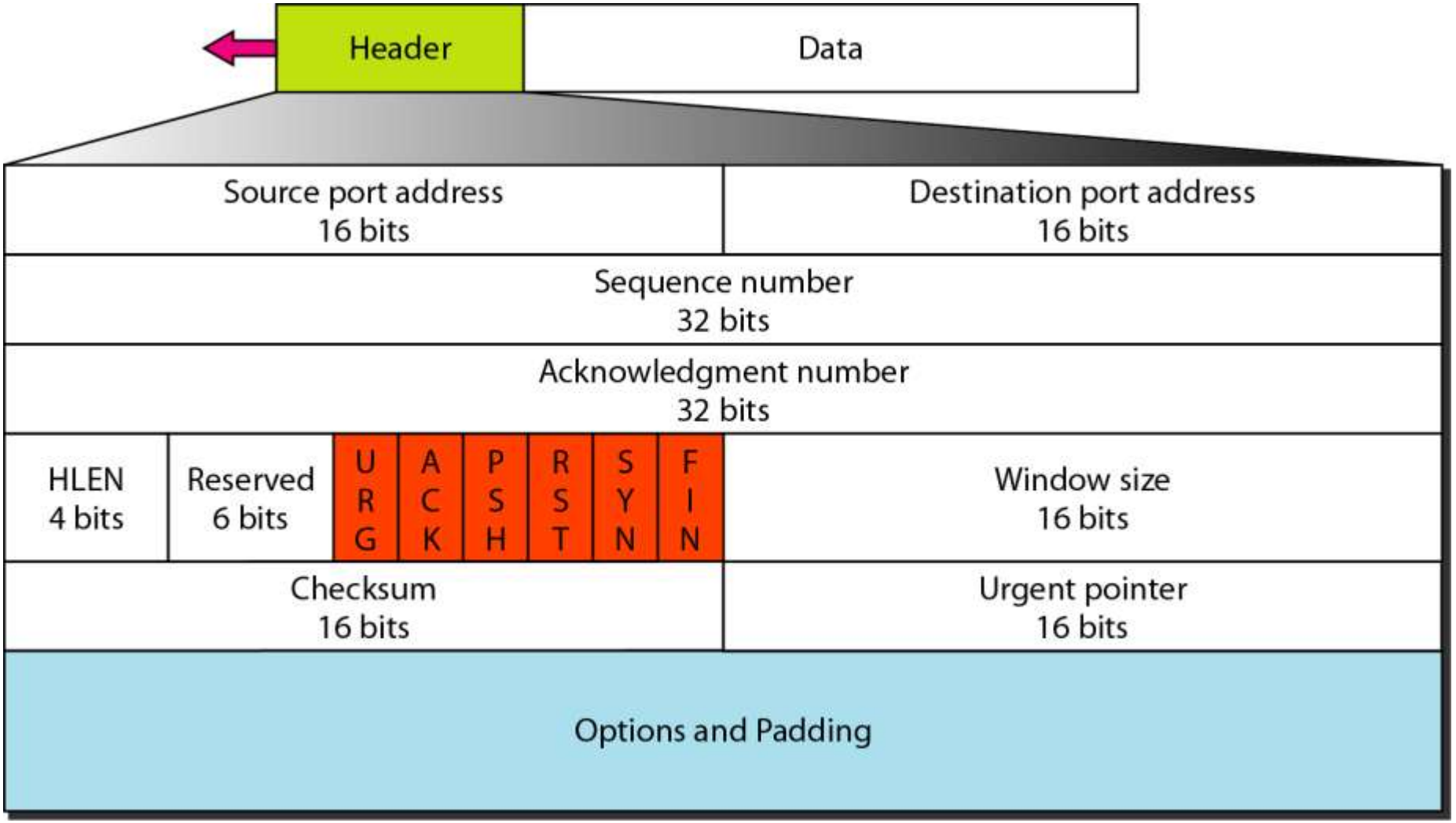
The format of a segment is shown in Figure 23.16.

- ✓ The segment consists of a 20- to 60-byte header, 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.

Header fields:

- **Source port address**. This is a 16-bit field that defines the port number of the application program in the host that is sending the segment. This serves the same purpose as the source port address in the UDP header

Figure 23.16 *TCP segment format*



Destination port address.

- ✓ This is a 16-bit field defines the port number of the application program in the host that is receiving the segment. This serves the same purpose as the destination port address in the UDP header .

Sequence number.

- ✓ This 32-bit field defines the number assigned to the first byte of data contained in this segment.
- ✓ TCP is a stream transport protocol.
- ✓ To ensure connectivity, each byte to be transmitted is numbered.
- ✓ The sequence number tells the destination which byte in this sequence comprises the first byte in the segment.
- ✓ During connection establishment, each party uses a random number generator to create an initial sequence number (ISN), which is usually different in each direction.

Acknowledgment number:

This 32-bit field defines the byte number that the receiver of the segment is expecting to receive from the other party.

- If the receiver of the segment has successfully received byte number x from the other party,
- it defines x + I as the acknowledgment number.
- Acknowledgment and data can be piggybacked together

Header length:

This 4-bit field indicates the number of 4-byte words in the TCP header. The length of the header can be between 20 and 60 bytes. Therefore, the value of this field can be between 5 (5 x 4 =20) and 15 (15 x 4 =60).

Reserved. This is a 6-bit field reserved for future use

Control. This field

defines 6 different control bits or flags as shown in Figure 23.17. One or more of these bits can be set at a time.

Figure 23.17 *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

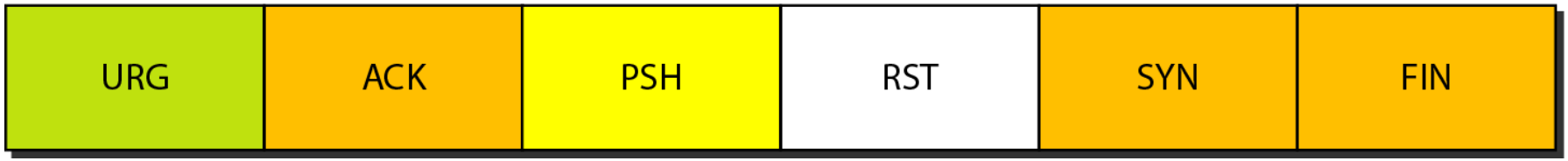


Table 23.3 ***Description of flags in the control field***

Flag	*Description*
URG	The value of the urgent pointer field is valid.
ACK	The value of the acknowledgment field is valid.
PSH	Push the data.
RST	Reset the connection.
SYN	Synchronize sequence numbers during connection.
FIN	Terminate the connection.

Window size.

- This field defines the size of the window, in bytes, that the other party must maintain.
- the length of this field is 16 bits, which means that the maximum size of the window is 65,535 bytes.
- This value is normally referred to as the receiving window (rwnd) and is determined by the receiver.

Checksum.

- ✓ This 16-bit field contains the checksum.
- ✓ The calculation of the checksum for TCP follows the same procedure as the one described for UDP

Urgent pointer. This l6-bit field, which is valid only if the urgent flag is set, is used when the segment contains urgent data. It defines the number that must be added to the sequence number to obtain the number of the last urgent byte in the data section of the segment.

TCP Connection:

- A connection-oriented transport protocol establishes a virtual path between the source and destination.
- All the segments belonging to a message are then sent over this virtual path.
- Using a single virtual pathway for the entire message facilitates the acknowledgment process as well as retransmission of damaged or lost frames
- TCP, uses the services of IP, a connectionless protocol, can be connection-oriented.
- TCP uses the services of IP to deliver individual segments to the receiver, but it controls the connection itself
- If a segment is lost or corrupted, it is retransmitted

In TCP, connection-oriented transmission requires three phases: connection establishment, data transfer, and connection termination.

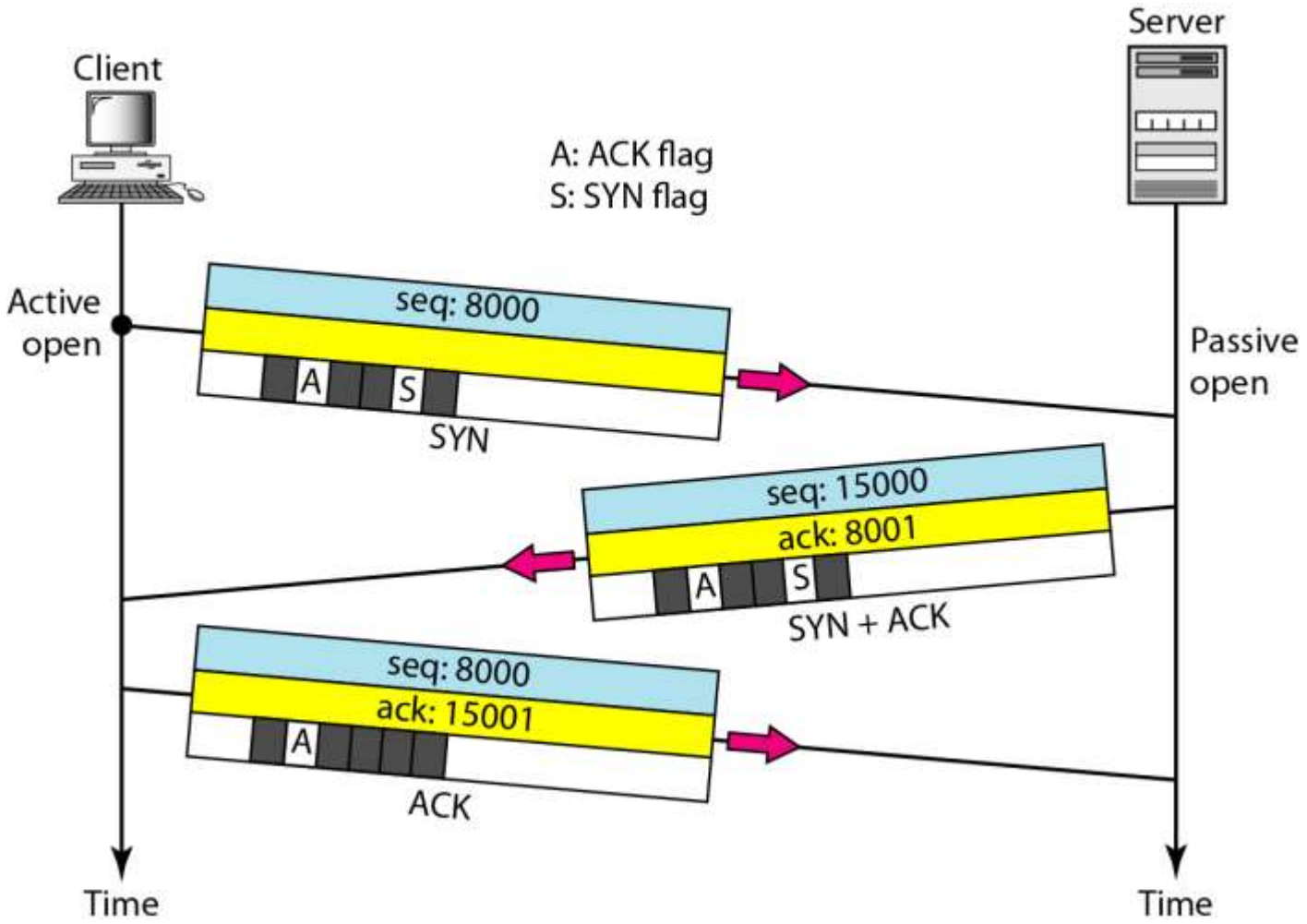
1. Connection Establishment:

- 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.

Three-Way Handshaking: The connection establishment in TCP is called three way handshaking. In our example, an application program, called the client, wants to make a connection with another application program, called the server, using TCP as the transport layer protocol.

- The process starts with the server. The server program tells its TCP that it is ready to accept a connection. This is called a request for a passive open.
- Although the server TCP is ready to accept any connection from any machine in the world, it cannot make the connection itself.
- The client program issues a request for an active open
- A client that wishes to connect to an open server tells its TCP that it needs to be connected to that particular server.
- TCP can now start the three-way handshaking process as shown in Figure 23.18.

Figure 23.18 *Connection establishment using three-way handshaking*



- We show the sequence number, the acknowledgment number, the control flags (only those that are set), and the window size, if not empty

The three steps in this phase are as follows.:

Step 1:-

- The client sends the first segment, a SYN segment, in which only the SYN flag is set.
- This segment is for synchronization of sequence numbers .
- It consumes one sequence number. When the data transfer starts, the sequence number is incremented by 1.
- We can say that the SYN segment carries no real data, but we can think of it as containing 1 imaginary byte.

- Step 2:-
- The server sends the second segment, a SYN + ACK segment, with 2 flag bits set: SYN and ACK. This segment has a dual purpose. It is a SYN for communication in the other direction and serves as the acknowledgment for the SYN segment. It consumes one sequence number.
- Step 3:-
- The client sends the third segment. This is just an ACK segment. It acknowledges the receipt of the second segment with the ACK flag and acknowledgment number field.
- the sequence number in this segment is the same as the one in the SYN segment; the ACK segment does not consume any sequence numbers.

Simultaneous Open

- A rare situation, called a simultaneous open, may occur when both processes issue an active open.
- In this case, both TCPs transmit a SYN + ACK segment to each other, and one single connection is established between them

SYN Flooding Attack:

- The connection establishment procedure in TCP is susceptible to a serious security problem called the SYN flooding attack.
- This happens when a malicious attacker sends a large number of SYN segments to a server, pretending that each of them is coming from a different client by faking the source IP addresses in the datagrams

- The server, assuming that the clients are issuing an active open, allocates the necessary resources, such as creating communication tables and setting timers.
- The TCP server then sends the SYN +ACK segments to the fake clients, which are lost.
- During this time, however, a lot of resources are occupied without being used
- If, during this short time, the number of SYN segments is large, the server eventually runs out of resources and may crash

- This SYN flooding attack belongs to a type of security attack known as a denial-of-service attack, in which an attacker monopolizes a system with so many service requests that the system collapses and denies service to every request
- Some implementations of TCP have strategies to alleviate the effects of a SYN attack.
- Some have imposed a limit on connection requests during a specified period of time.

Note

A SYN segment cannot carry data, but it consumes one sequence number.

Note

A SYN + ACK segment cannot carry data, but does consume one sequence number.

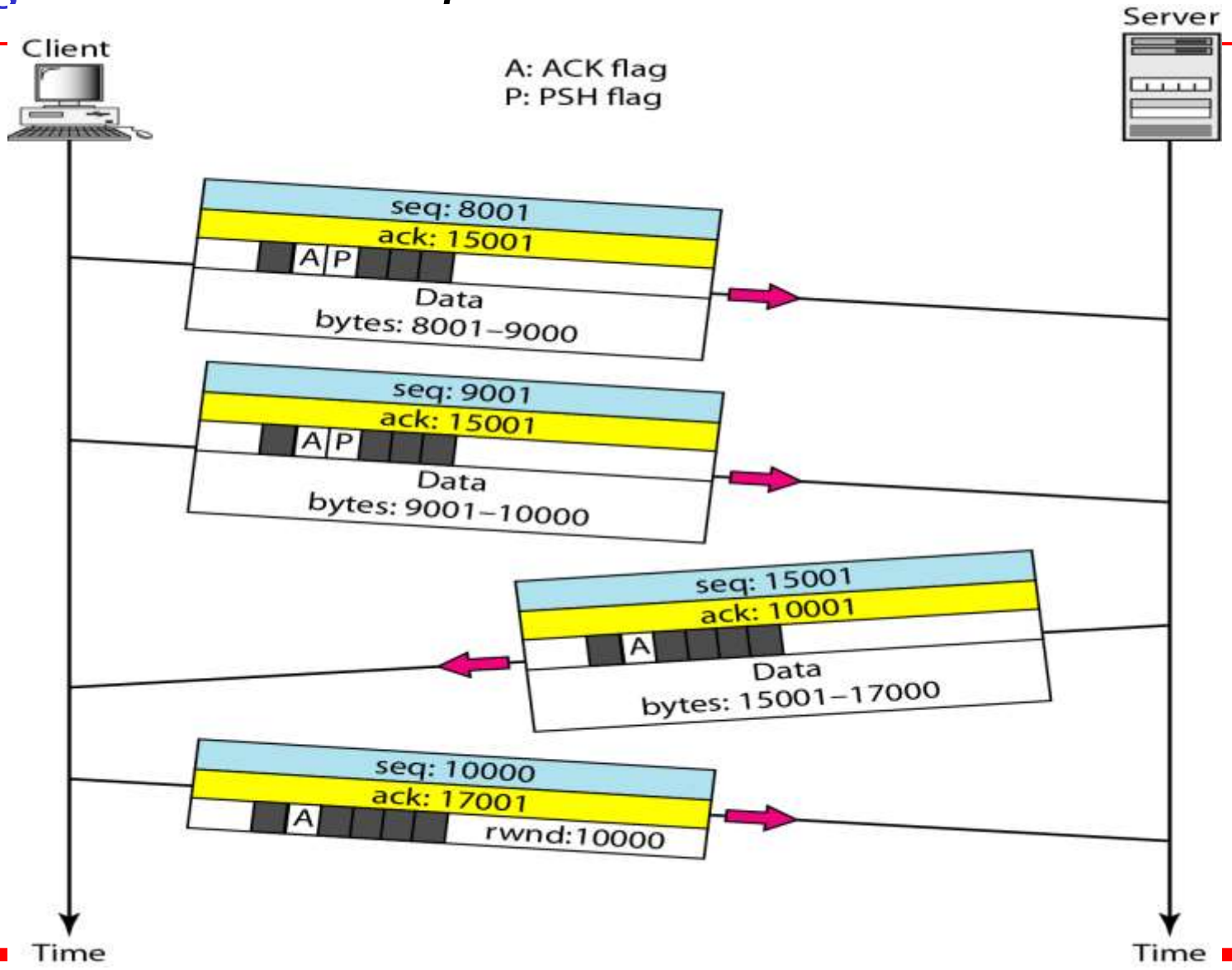
Note

An ACK segment, if carrying no data, consumes no sequence number.

2. Data Transfer:

- After connection is established, bidirectional data transfer can take place.
- The client and server can both send data and acknowledgments.
- 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 23.19 shows an example.
- In this example, after connection is established (not shown in the figure), the client sends 2000 bytes of data in two segments.
- The server then sends 2000 bytes in one segment.
- The client sends one more segment.

Figure 23.19 *Data transfer*



- The first three segments carry both data and acknowledgment, but the last segment carries only an acknowledgment because there are no more data to be sent.
- The data segments sent by the client have the PSH (push) flag set so that the **server TCP knows to deliver data to the server process as soon as they are received**.
- The segment from the server, on the other hand, does not set the push flag.

Pushing Data:

- We saw that the sending TCP uses a buffer to store the stream of data coming from the sending application program.
- The sending TCP can select the segment size.

- The receiving TCP also buffers the data when they arrive and delivers them to the application program
- when the application program is ready or when it is convenient for the receiving TCP.
- This type of flexibility increases the efficiency of TCP.
- However, on occasion the application program has no need for this flexibility_
- For example, consider an application program that communicates interactively with another application program on the other end.
- The application program on one site wants to send a keystroke to the application at the other site and receive an immediate response.
- Delayed transmission and delayed delivery of data may not be acceptable by the application program

- TCP can handle such a situation.
- The application program at the sending site can request a push operation.
- This means that the sending TCP must not wait for the window to be filled.
- It must create a segment and send it immediately.
- The sending TCP must also set the push bit (PSH) **to let the receiving TCP know that the segment includes data that must be delivered to the receiving application program as soon as possible and not to wait for more data to come.**

- Urgent Data
- TCP is a stream-oriented protocol.
- This means that the data are presented from the application program to TCP as a stream of bytes.
- Each byte of data has a position in the stream.
- However, on occasion an application program needs to send urgent bytes.
- This means that the sending application program wants a piece of data to be read out of order by the receiving application program.
- As an example, suppose that the sending application program is sending data to be processed by the receiving application program.
- When the result of processing comes back, the sending application program finds that everything is wrong

- It wants to abort the process, but it has already sent a huge amount of data.
- If it issues an abort command (control +C), these two characters will be stored at the end of the receiving TCP buffer.
- It will be delivered to the receiving application program after all the data have been processed.
- The solution is to send a segment with the URG bit set.
- The sending application program tells the sending TCP that the piece of data is urgent.
- The sending TCP creates a segment and inserts the urgent data at the beginning of the segment.
- The rest of the segment can contain normal data from the buffer.
- **The urgent pointer field in the header defines the end of the urgent data and the start of normal data**

- When the receiving TCP receives a segment with the URG bit set,
- it extracts the urgent data from the segment, using the value of the urgent pointer,
- **and delivers them, out of order, to the receiving application program**

3. Connection Termination:

- Any 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.

1.Three-Way Handshaking:

- Most implementations today allow three-way handshaking for connection termination as shown in Figure 23.20.

Step 1:- In a normal situation, the client TCP, after receiving a close command from the client process, sends the first segment, a FIN segment in which the FIN flag is set.

- Note that a FIN segment can include the last chunk of data sent by the client,
- or it can be just a control segment as shown in Figure 23.20.
- If it is only a control segment, it consumes only one sequence number.

Step 2:-

The server TCP, after receiving the FIN segment, informs its process of the situation and sends the second segment, a FIN + ACK segment, to confirm the receipt of the FIN segment from the client.

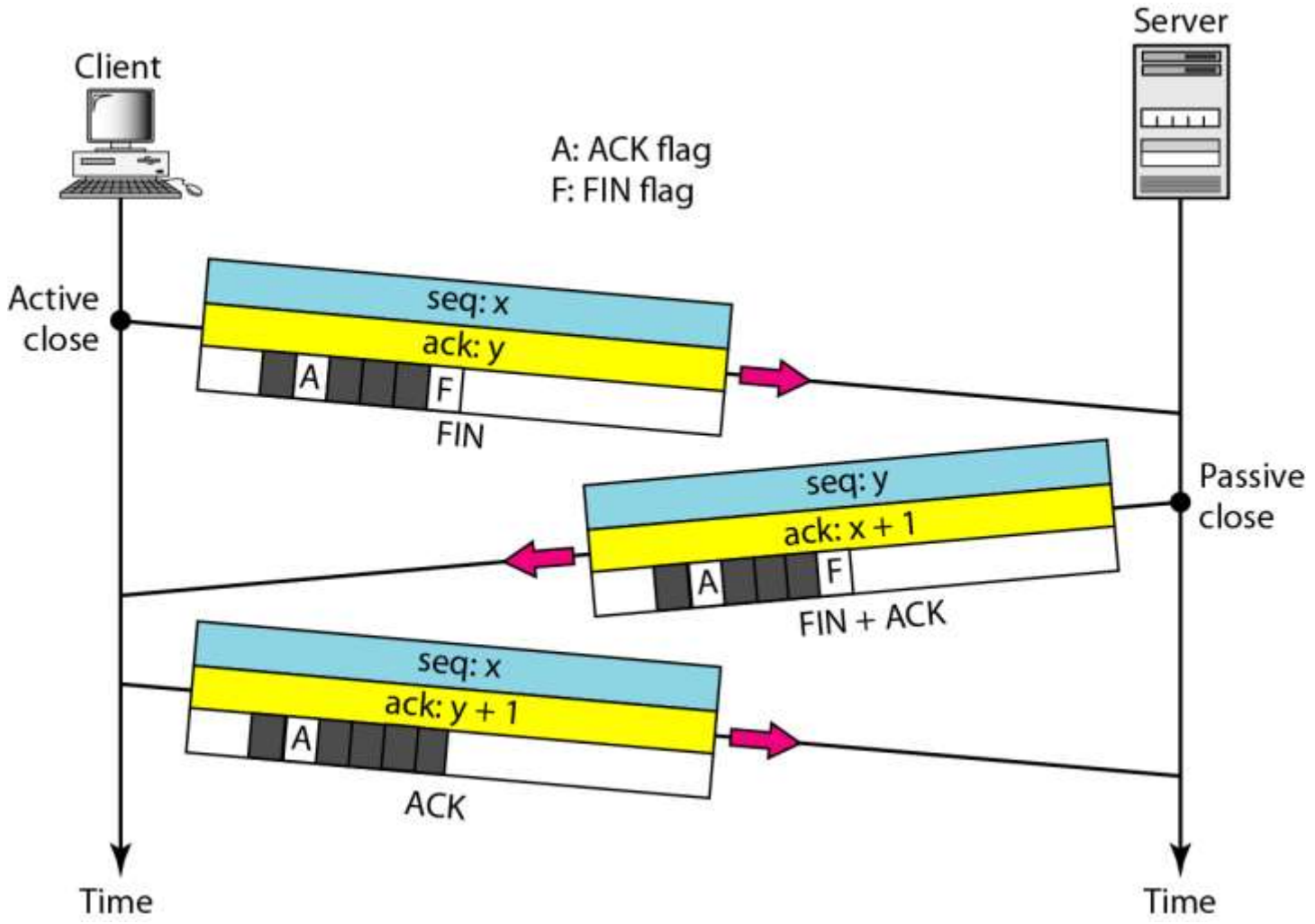
- and at the same time to announce the closing of the connection in the other direction
- This segment can also contain the last chunk of data from the server.
- If it does not carry data, it consumes only one sequence number

Step 3:-

The client TCP sends the last segment, an ACK segment, to confirm the receipt of the FIN segment from the TCP server.

- This segment contains the acknowledgment number, which is 1 plus the sequence number received in the FIN segment from the server.
- This segment cannot carry data and consumes no sequence numbers

Figure 23.20 *Connection termination using three-way handshaking*



Note

The FIN segment consumes one sequence number if it does not carry data.

Note

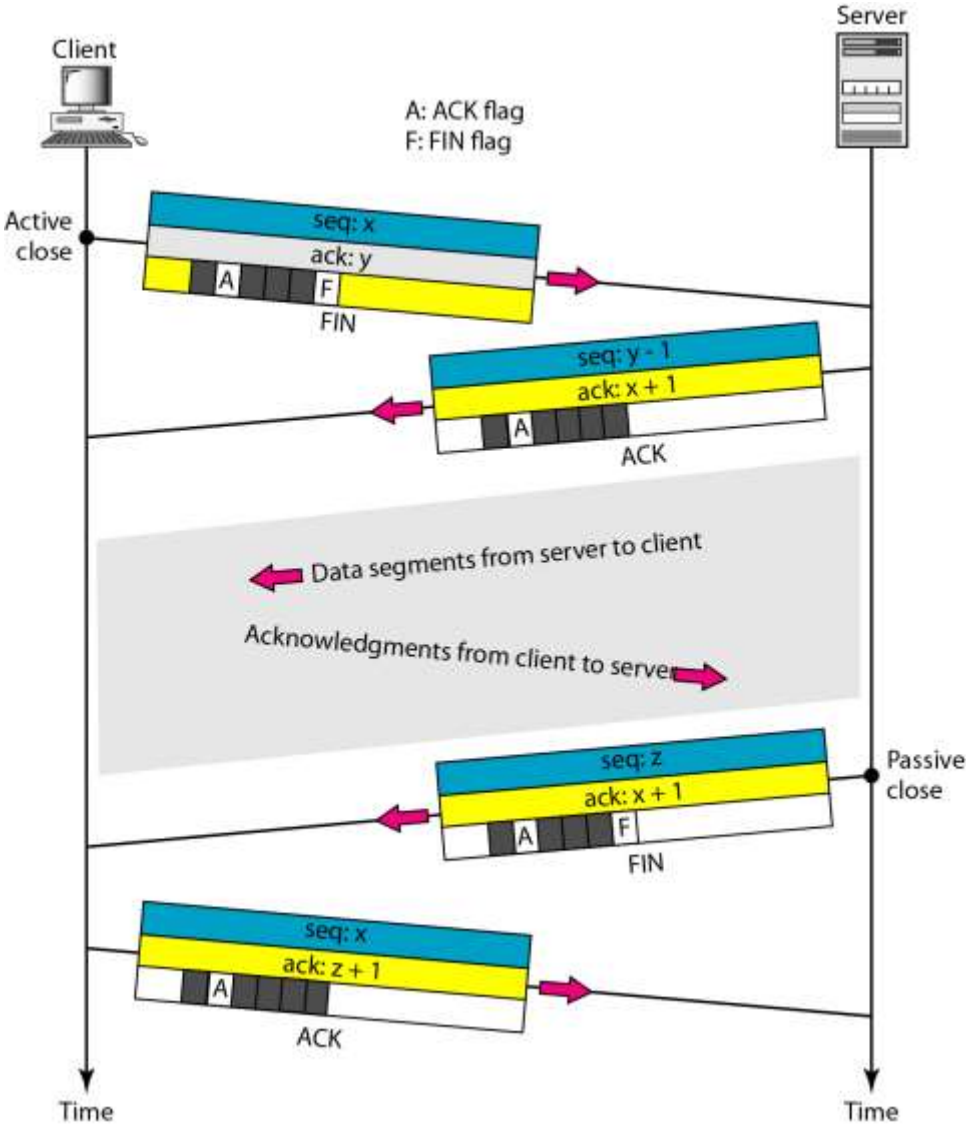
The FIN + ACK segment consumes one sequence number if it does not carry data.

2. Half-Close :

- In TCP, one end can stop sending data while still receiving data. This is called a half-close.
- Although either end can issue a half-close
- it is normally initiated by the client.
- It can occur when the server needs all the data before processing can begin.
- A good example is sorting.
- When the client sends data to the server to be sorted, the server needs to receive all the data before sorting can start.
- This means the client, after sending all the data, can close the connection in the outbound direction.
- However, the inbound direction must remain open to receive the sorted data.

- The server, after receiving the data, still needs time for sorting; its outbound direction must remain open.
- Figure 23.21 shows an example of a half-close.
- The client half-closes the connection by sending a FIN segment.
- The server accepts the half-close by sending the ACK segment.
- The data transfer from the client to the server stops.
- The server, however, can still send data.
- When the server has sent all the processed data, it sends a FIN segment, which is acknowledged by an ACK from the client.
- After half-closing of the connection, data can travel from the server to the client and acknowledgments can travel from the client to the server.
- The client cannot send any more data to the server

Figure 23.21 *Half-close*



Flow Control:

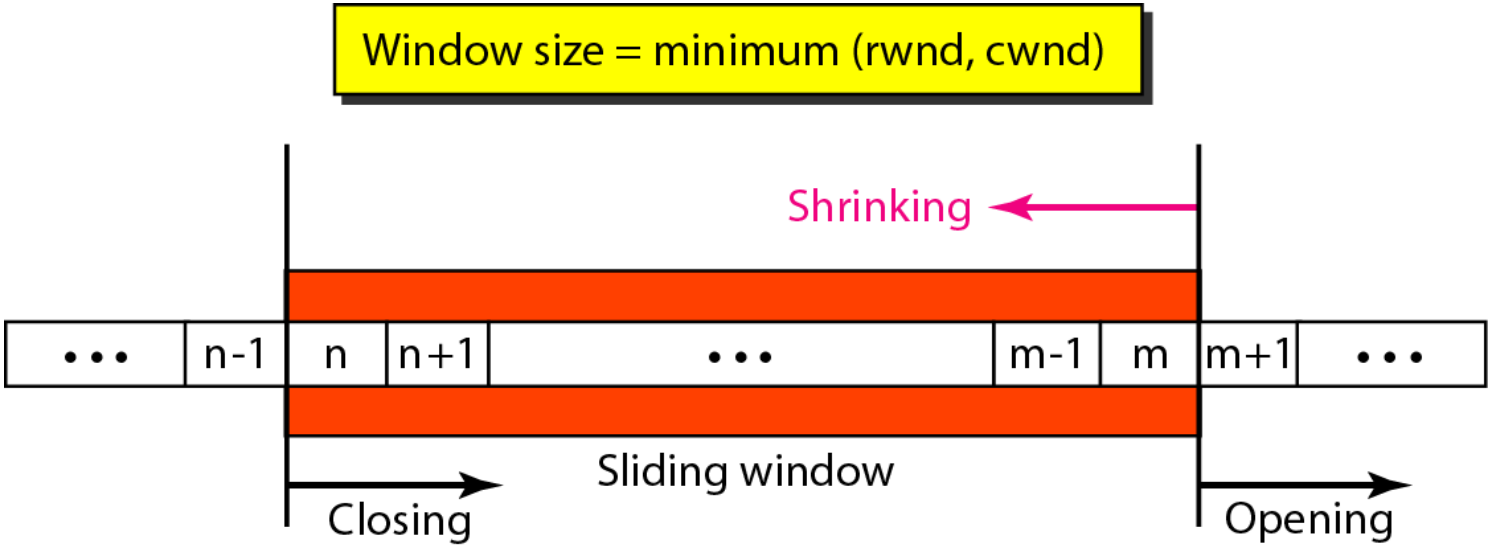
- TCP uses a sliding window, to handle flow control.
- The sliding window protocol used by TCP, however, is something between the Go-Back-N and Selective Repeat sliding window.
- The sliding window protocol in TCP looks like the Go-Back-N protocol because it does not use NAKs;
- it looks like Selective Repeat because the receiver holds the out-of-order segments until the missing ones arrive.
- There are two big differences between this sliding window and the one we used at the data link layer.
- First, the sliding window of TCP is byte-oriented; the one we discussed in the data link layer is frame-oriented.
- Second, the TCP's sliding window is of variable size; the one we discussed in the data link layer was of fixed size

- Figure 23.22 shows the sliding window in TCP.
- The window spans a portion of the buffer containing bytes received from the process.
- The bytes inside the window are the bytes that can be in transit;
- they can be sent without worrying about acknowledgment.
- The imaginary window has two walls: one left and one right.
- The window is opened, closed, or shrunk.
- These three activities,, are in the control of the receiver (and depend on congestion in the network), not the sender.
- The sender must obey the commands of the receiver
- Opening a window means moving the right wall to the right. This allows more new bytes in the buffer that are eligible for sending.
- Closing the window means moving the left wall to the right.

- This means that some bytes have been acknowledged and the sender need not worry about them anymore.
- Shrinking the window means moving the right wall to the left
- This is strongly discouraged and not allowed in some implementations because it means revoking the eligibility of some bytes for sending.
- This is a problem if the sender has already sent these bytes.
- Note that the left wall cannot move to the left because this would revoke some of the previously sent acknowledgments.
- The size of the window at one end is determined by the lesser of two values: receiver window (rwnd) or congestion window (cwnd).

- The receiver window is the value advertised by the opposite end in a segment containing acknowledgment.
- It is the number of bytes the other end can accept before its buffer overflows and data are discarded.
- The congestion window is a value determined by the network to avoid congestion

Figure 23.22 *Sliding window*



Note

A sliding window is used to make transmission more efficient as well as to control the flow of data so that the destination does not become overwhelmed with data.
TCP sliding windows are byte-oriented.

Example 23.4

What is the value of the receiver window (rwnd) for host A if the receiver, host B, has a buffer size of 5000 bytes and 1000 bytes of received and unprocessed data?

Solution

The value of rwnd = 5000 − 1000 = 4000. Host B can receive only 4000 bytes of data before overflowing its buffer. Host B advertises this value in its next segment to A.

Example 23.5

What is the size of the window for host A if the value of rwnd is 3000 bytes and the value of cwnd is 3500 bytes?

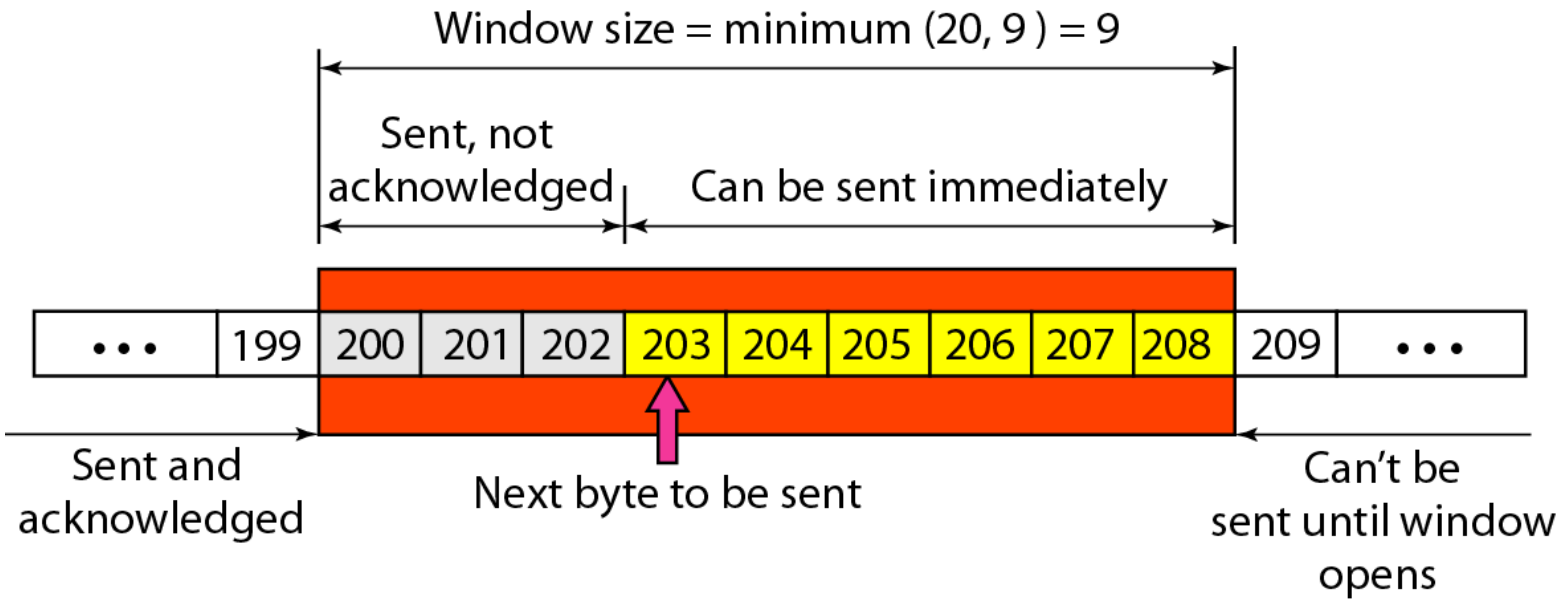
Solution

The size of the window is the smaller of rwnd and cwnd, which is 3000 bytes.

Example 23.6

Figure 23.23 shows an unrealistic example of a sliding window. The sender has sent bytes up to 202. We assume that cwnd is 20 (in reality this value is thousands of bytes). The receiver has sent an acknowledgment number of 200 with an rwnd of 9 bytes (in reality this value is thousands of bytes). The size of the sender window is the minimum of rwnd and cwnd, or 9 bytes. Bytes 200 to 202 are sent, but not acknowledged. Bytes 203 to 208 can be sent without worrying about acknowledgment. Bytes 209 and above cannot be sent.

Figure 23.23 *Example 23.6*



Note

Some points about TCP sliding windows:

- The size of the window is the lesser of rwnd and cwnd.
- The source does not have to send a full window's worth of data.
- The window can be opened or closed by the receiver, but should not be shrunk.
- The destination can send an acknowledgment at any time as long as it does not result in a shrinking window.
- The receiver can temporarily shut down the window; the sender, however, can always send a segment of 1 byte after the window is shut down.

- 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
- TCP provides reliability using error control
- Error detection and correction in TCP is achieved through the use of three simple tools: checksum, acknowledgment, and time-out.

1.Checksum:- Each segment includes a checksum field which is used to check for a corrupted segment.

If the segment is corrupted, it is discarded by the destination TCP and is considered as lost

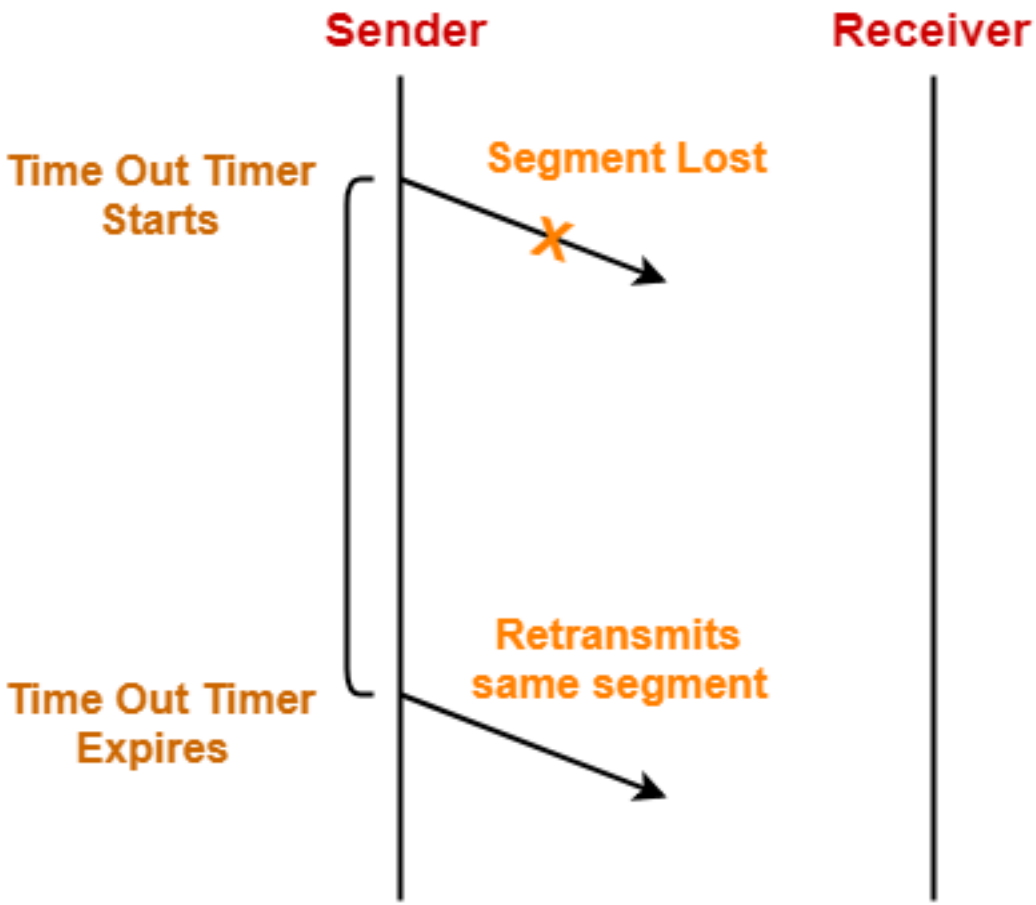
2. Acknowledgment TCP uses acknowledgments to confirm the receipt of data segments.

3. Retransmission The heart of the error control mechanism is the retransmission of segments. When a segment is corrupted, lost, or delayed, it is retransmitted

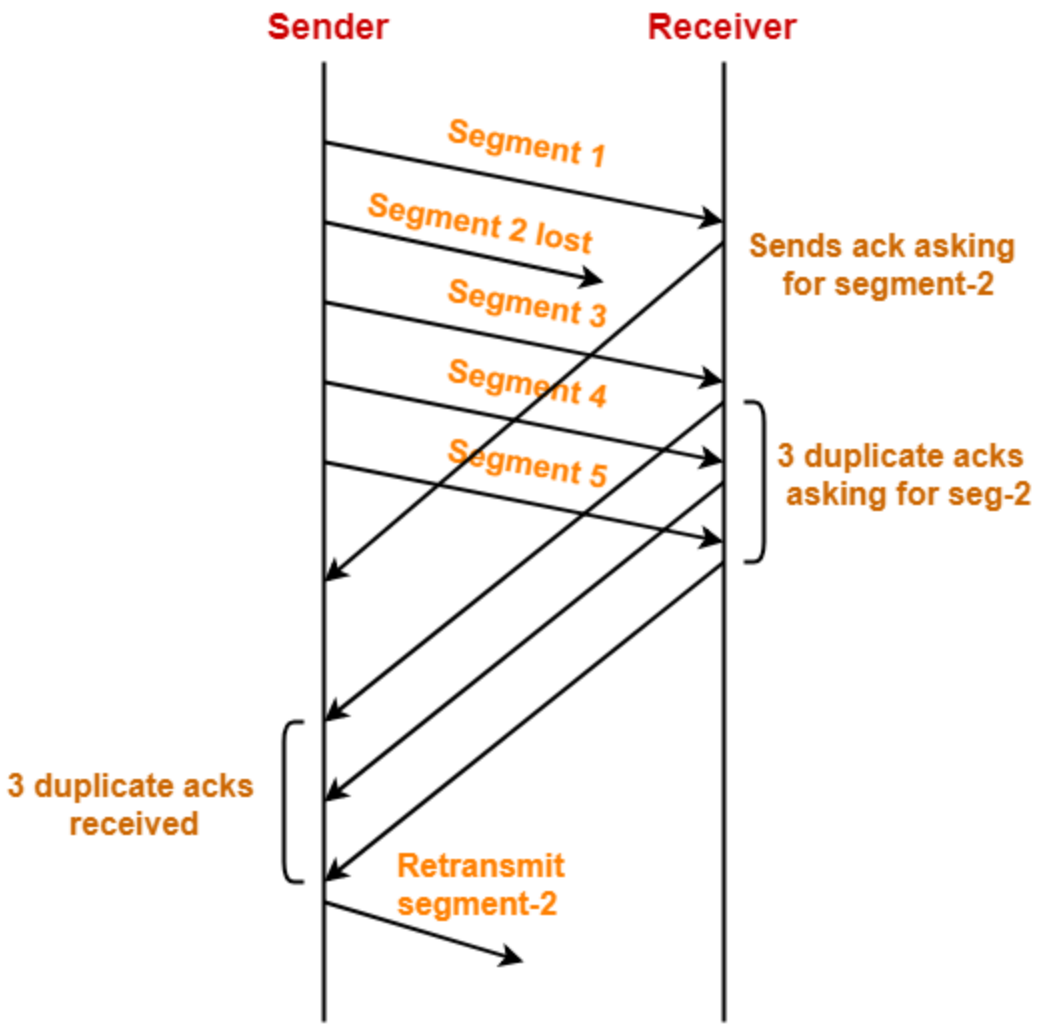
1.Retransmission After RTO.

When the timer matures, the earliest outstanding segment is retransmitted.

2. Retransmission After 3 Duplicate ACK Segments



Retransmission after Time Out Timer Expiry



Retransmission after receiving 3 duplicate acks

- Out-of-Order Segments
- When a segment is delayed, lost, or discarded, the segments following that segment arrive out of order.
- Originally, TCP was designed to discard all out-of-order segments, resulting in the retransmission of the missing segment and the following segments.
- Most implementations today do not discard the out-of-order segments.
- They store them temporarily and flag them as out-of-order segments until the missing segment arrives.

Note

In modern implementations, a retransmission occurs if the retransmission timer expires or three duplicate ACK segments have arrived.

Note

No retransmission timer is set for an ACK segment.

Note

Data may arrive out of order and be temporarily stored by the receiving TCP, but TCP guarantees that no out-of-order segment is delivered to the process.

Figure 23.24 TCP-*Normal operation*

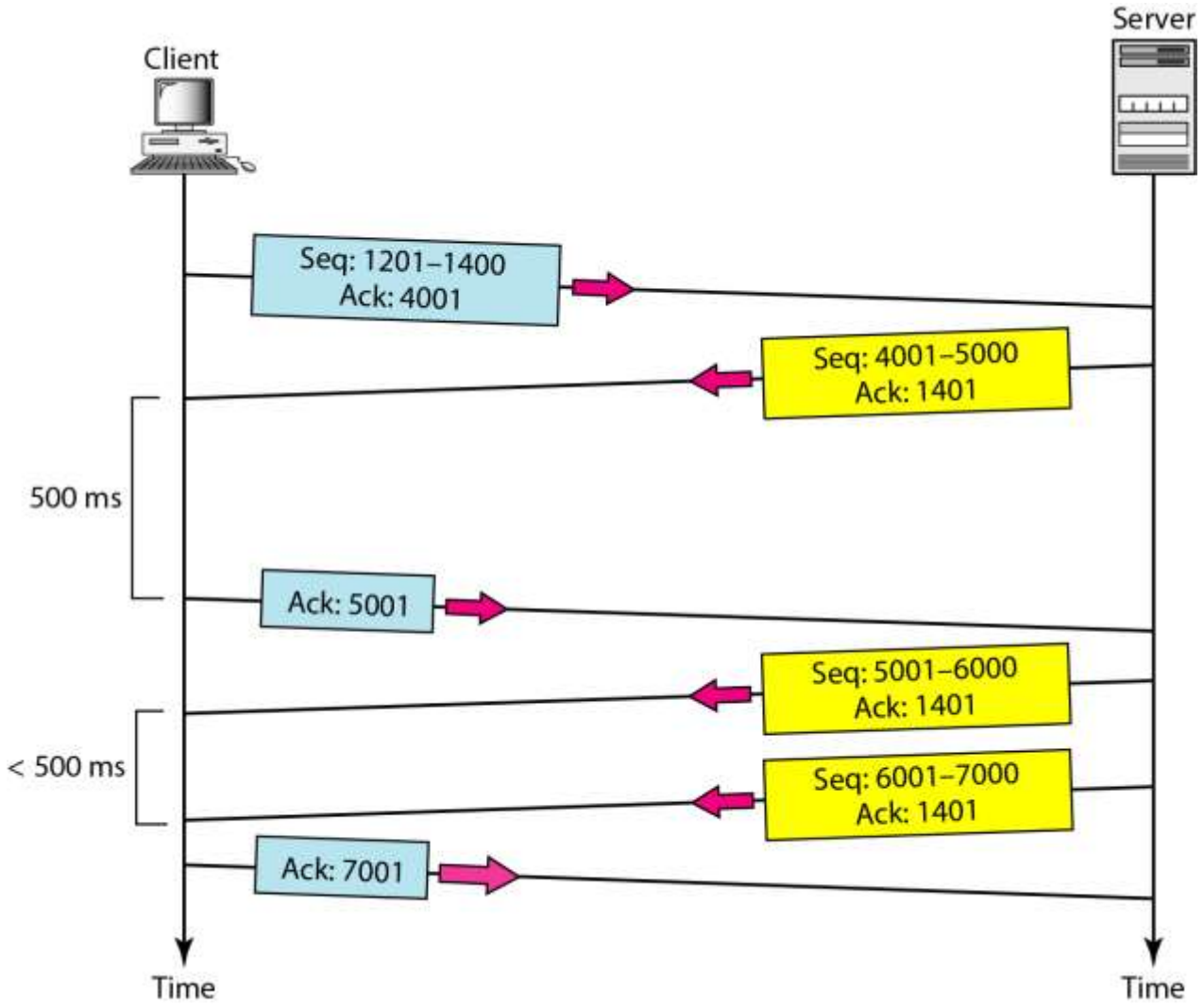
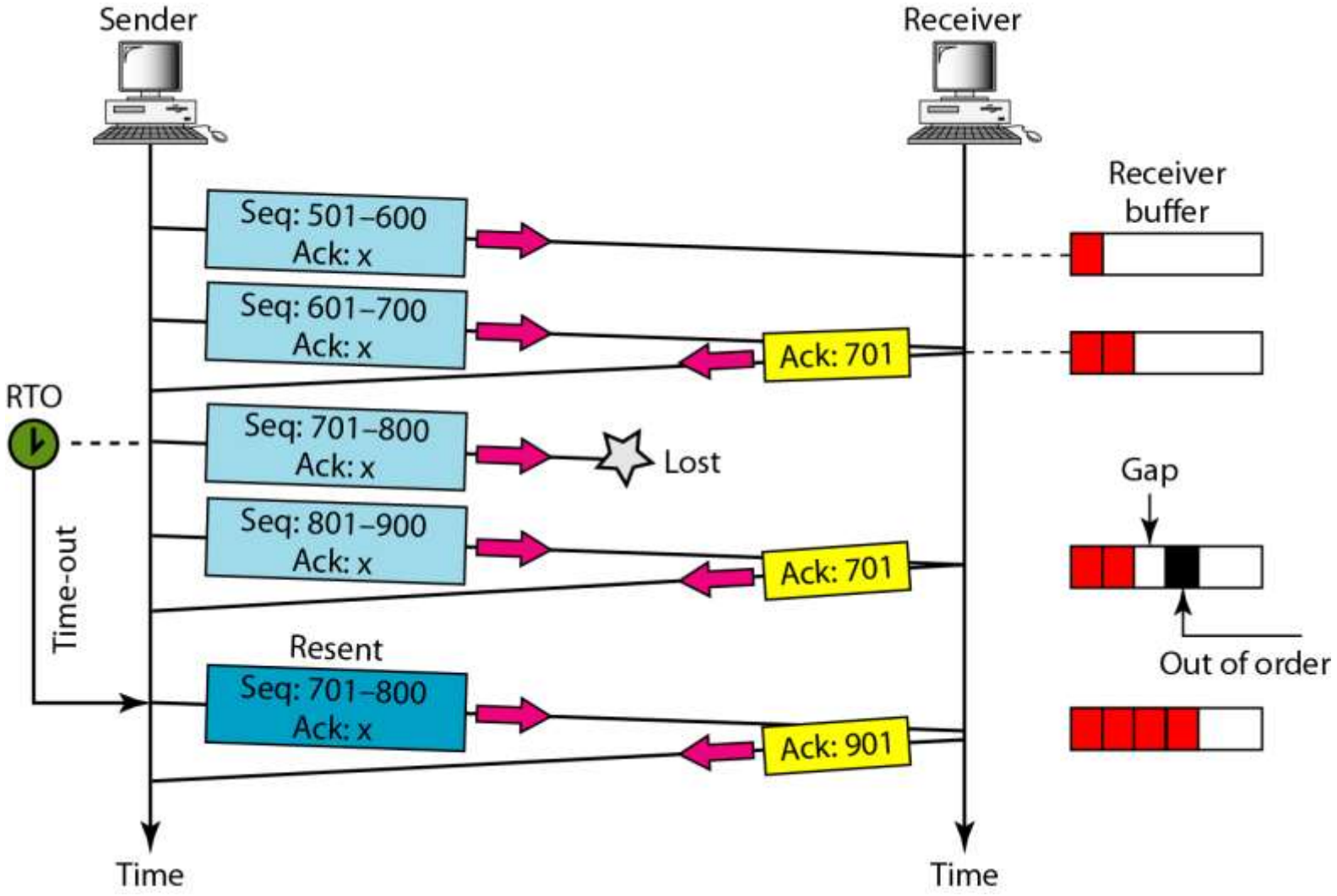


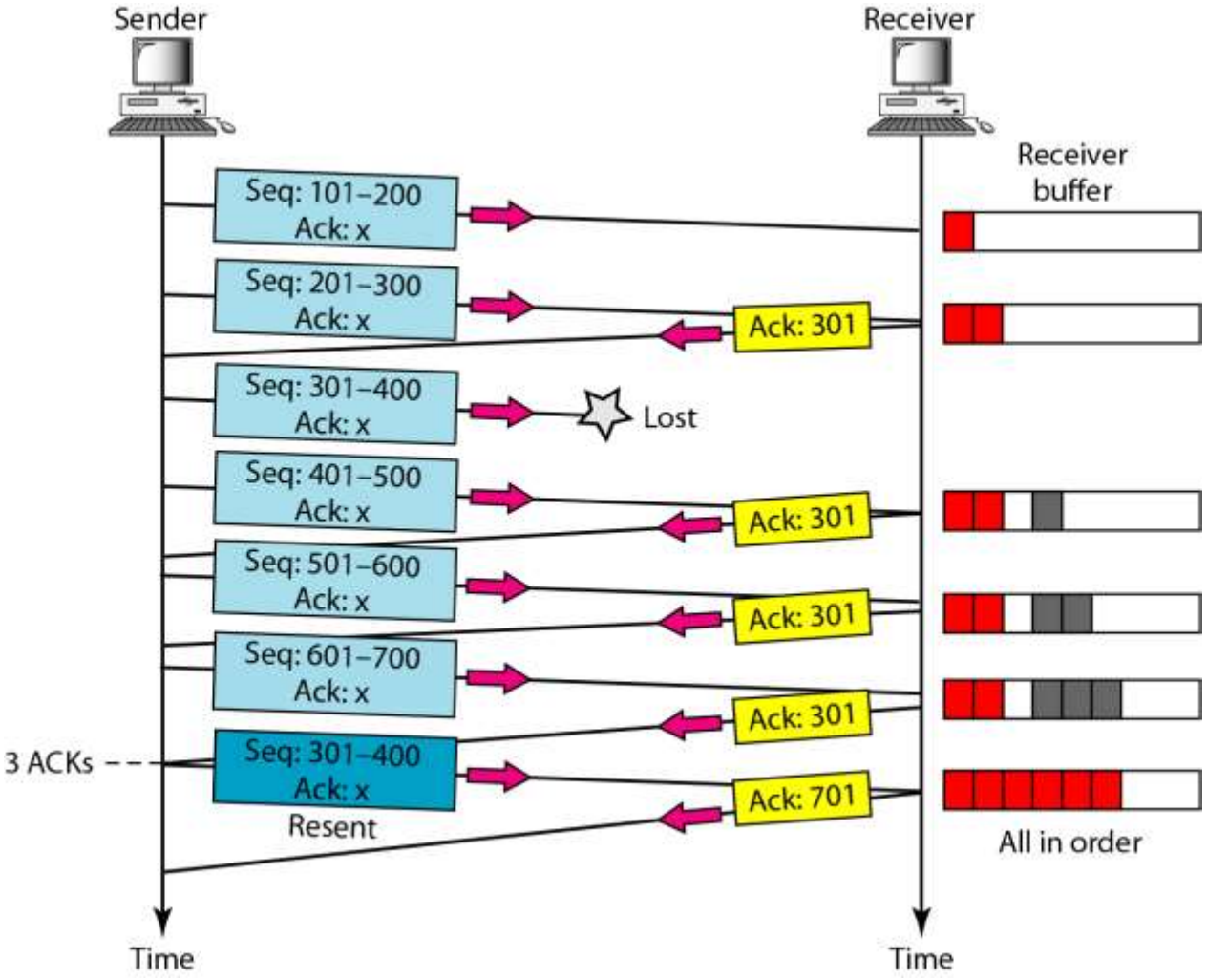
Figure 23.25 TCP- *Lost segment*



Note

The receiver TCP delivers only ordered data to the process.

Figure 23.26 *Fast retransmission*



Compare UDP and TCP:

- UDP is a message-oriented protocol.
- A process delivers a message to UDP, which is encapsulated in a user datagram and sent over the network.
- UDP conserves the message boundaries; each message is independent of any other message.
- UDP is unreliable; the sender cannot know the destiny of messages sent.
- A message can be lost, duplicated, or received out of order. UDP also lacks features, such as congestion control and flow control

- TCP is a byte-oriented protocol.
- It receives a message or messages from a process, stores them as a stream of bytes, and sends them in segments.
- There is no preservation of the message boundaries.
- However, TCP is a reliable protocol.
- The duplicate segments are detected, the lost segments are resent, and the bytes are delivered to the end process in order.
- TCP also has congestion control and flow control mechanisms.