UNIT - 5

Process to Process Delivery in Transport Layer

Process to Process Delivery:

The data link layer is responsible for delivery of frames between two neighboring nodes over a link. This is called node-to-node delivery. The network layer is responsible for delivery of datagrams between two hosts. This is called host-to-host delivery. Real communication takes place between two processes (application programs). We need 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. Figure 4.1 shows these three types of deliveries and their domains

The transport layer is responsible for process-to-process delivery.

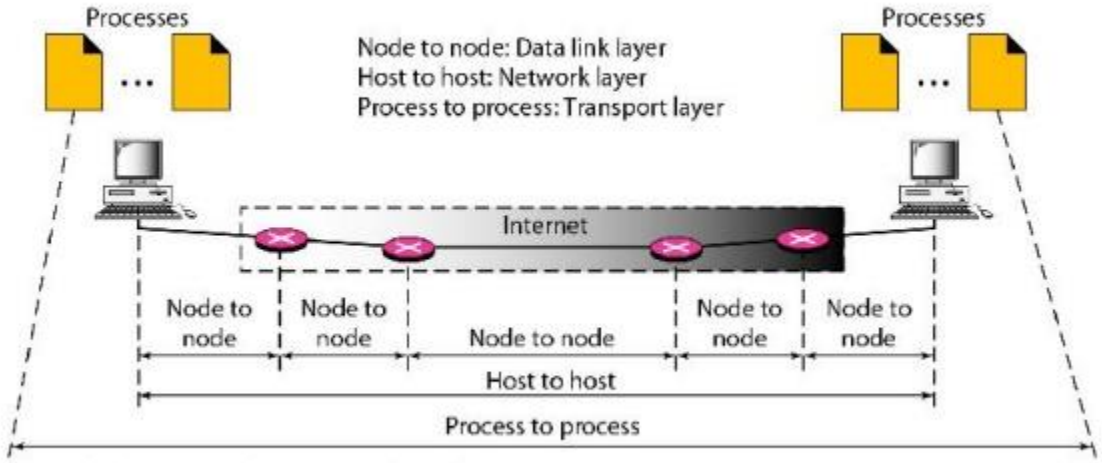

Figure 4.1 Types of data deliveries

1. Client/Server Paradigm

Although 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 data link layer, we need a MAC address to choose one node among several nodes if the connection is not point-to-point. A frame in the data link layer needs a Destination MAC address for delivery and a source address for the next node's reply.

Figure 4.2 shows this concept.

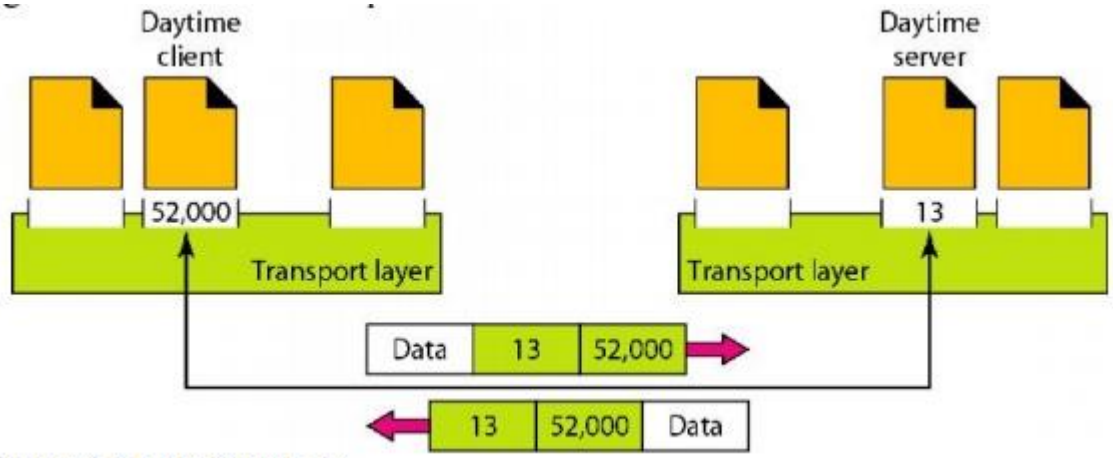

Figure 4.2 Port Numbers

The IP addresses and port numbers play different roles in selecting the final destination of data. The destination IP address defines the host among the different hosts in the world. After the host has been selected, the port number defines one of the processes on this particular host (see Figure 4.3).

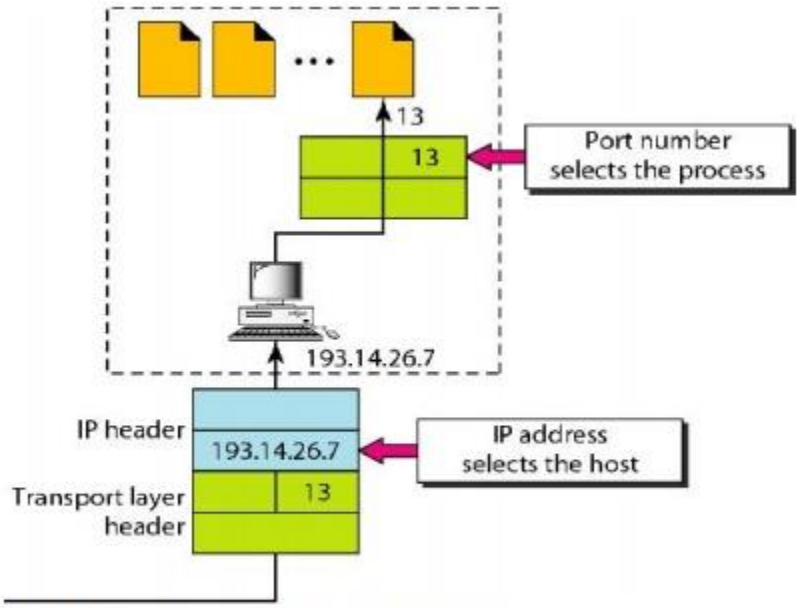

Figure 4.3 IP addresses versus port numbers

3. lANA Ranges

The lANA (Internet Assigned Number Authority) has divided the port numbers into three ranges: well known, registered, and dynamic (or private), as shown in Figure 4.4.

- **Well-known ports**. The ports ranging from 0 to 1023 are assigned and controlledby lANA. These are the well-known ports.

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

- **Dynamic ports**. The ports ranging from 49,152 to 65,535 are neither controllednor registered. They can be used by any process. These are the ephemeral ports.

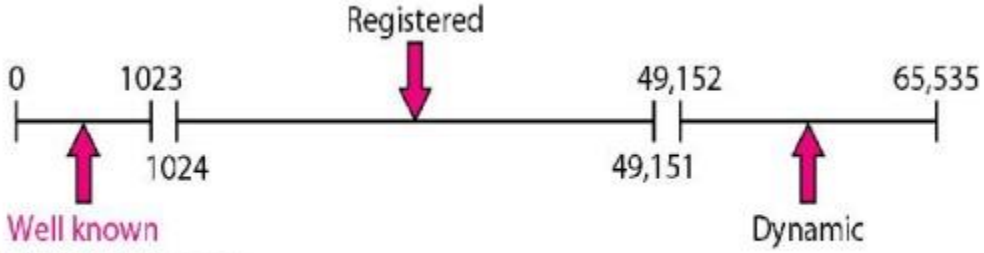

Figure 4.4 lANA Ranges

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 4.5).

UDP or TCP header contains the port numbers.

Figure 4.5 Socket Address

5. Multiplexing and Demultiplexing

The addressing mechanism allows multiplexing and demultiplexing by the transport layer, as shown in Figure 4.6.

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.

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.

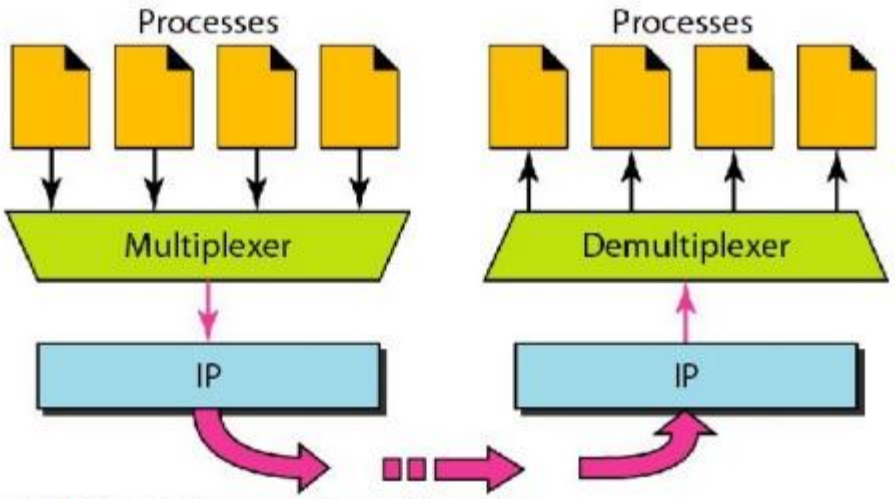

Figure 4.6 Multiplexing and Demultiplexing

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. The packets are not numbered; they may be delayed or lost or may arrive out of sequence. There is no acknowledgment either.

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.

7. 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 means a slower and more complex service.

In the Internet, there are three common different transport layer protocols. 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.

The network layer in the Internet is unreliable (best-effort delivery), we need to implement reliability at the transport layer. To understand that error control at the data link layer does not guarantee error control at the transport layer, let us look at Figure 4.7.

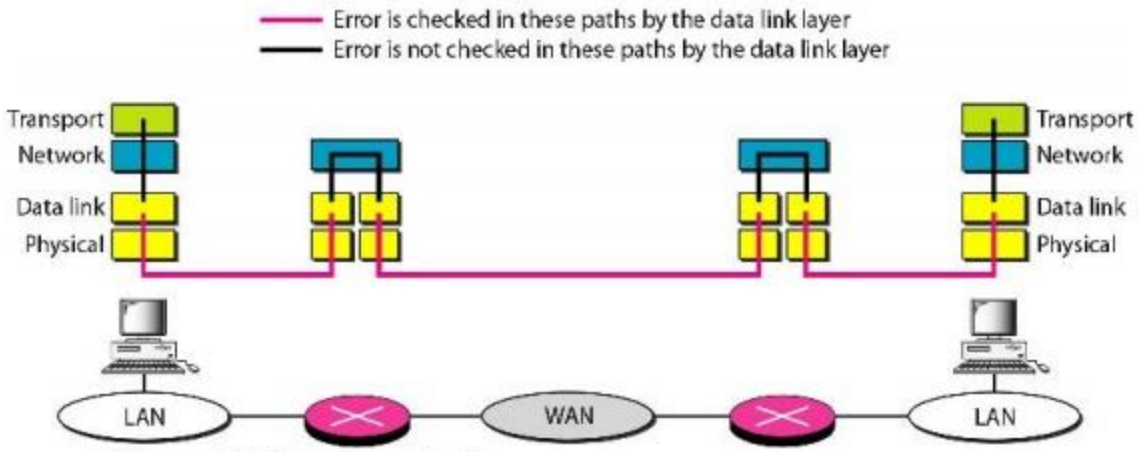

Figure 4.7 Error control

8. Three Protocols

The original TCP/IP protocol suite specifies two protocols for the transport layer: UDP and TCP. We first focus on UDP, the simpler of the two, before discussing TCP. A new transport layer protocol, SCTP, has been designed. Figure 4.8 shows the position of these protocols in the TCP/IP protocol suite.

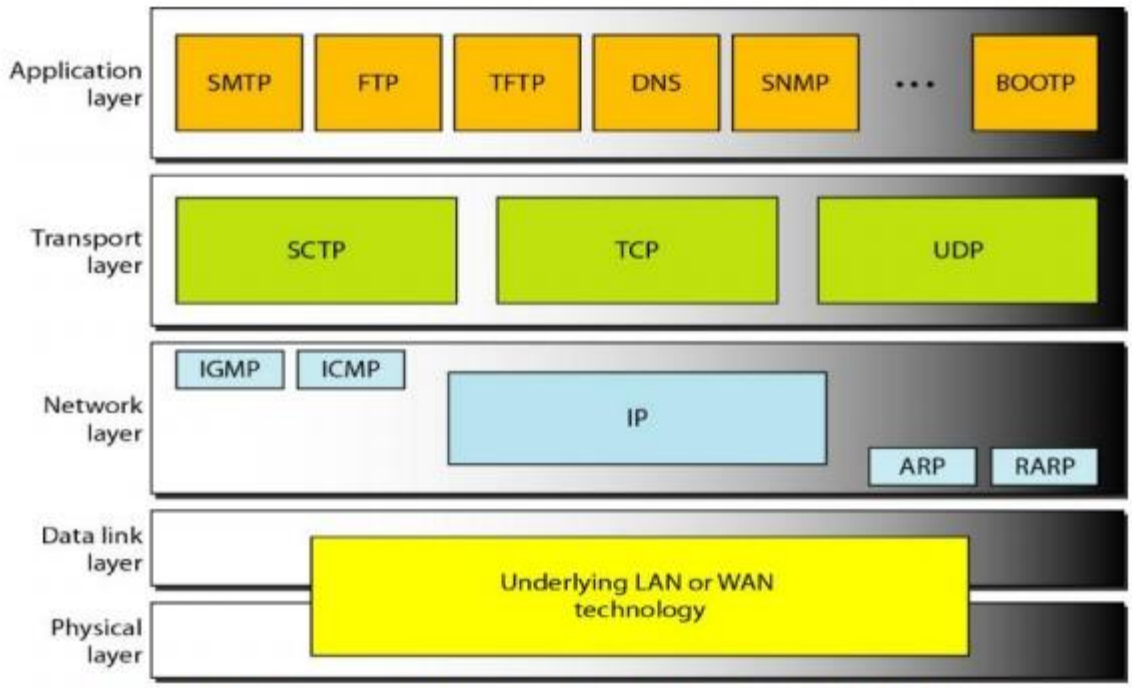

Figure 4.8 Position of UDP, TCP, and SCTP in TCP/IP suite

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. Also, it performs very limited error checking.

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.

1. Well-Known Ports for UDP:

Table 4.1 shows some well-known port numbers used by UDP. Some port numbers can be used by both UDP and TCP1

Table 4.1 Well-known port numbers used by UDP

Port	Protocol	Description
1	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
53	Name server	Domain Name Service
67	BOOTPs	Server port to download bootstrap information
68	BOOTPc	Client port to download bootstrap information
69	TFTP	Trivial File Transfer Protocol
111	RPC	Remote Procedure Call
123	NTP	Network Time Protocol
161	SNMP	Simple Network Management Protocol
162	SNMP	Simple Network Management Protocol (trap)

2. User Datagram

UDP packets, called user datagrams, have a fixed-size header of 8 bytes. Figure 4.9 shows the format of a user datagram.

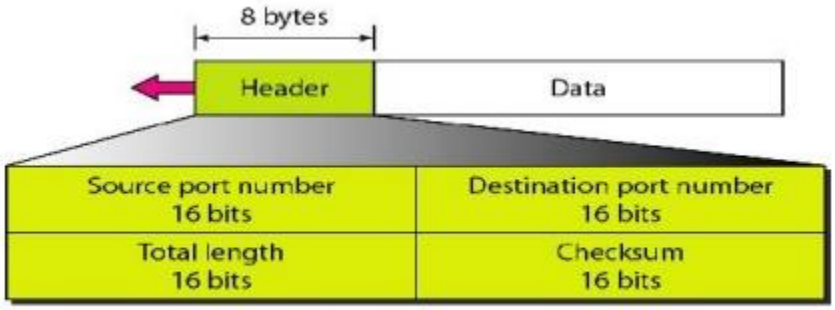

Figure 4.9 User datagram format

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.
- **Destination port number**. This is the port number used by the process running on the destination host. It is also 16 bits long.
- **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.

UDP length = IP length - IP header's length

- **Checksum**. This field is used to detect errors over the entire user datagram (header plus data).

3. Checksum

The UDP checksum calculation is different from the one for IP and ICMP. Here the checksum includes three sections: a pseudo header, the UDP header, and the data coming from the application layer

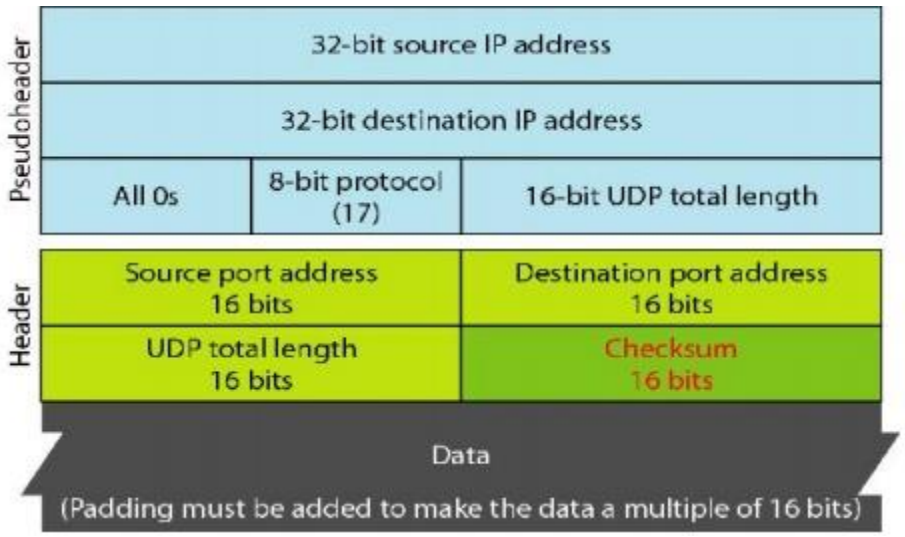

Figure 4.10 Pseudo header for checksum calculation

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 Os (see Figure 4.10).

Example 4.1

Figure 4.11 shows the checksum calculation for a very small user datagram with only 7 bytes of data. Because the number of bytes of data is odd, padding is added for checksum calculation. The pseudo header as well as the padding will be dropped when the user datagram is delivered to IP.

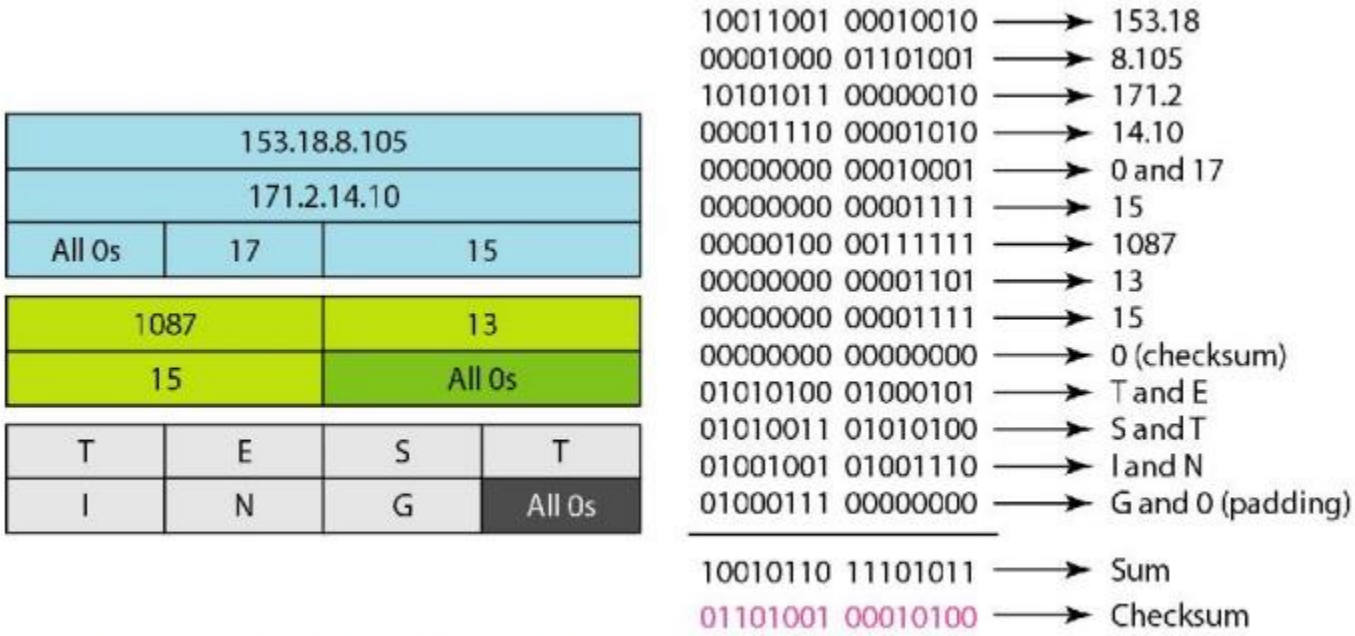

Figure 4.11 Checksum calculation of a simple UDP user datagram

Optional use of the checksum:

The calculation of the checksum and its inclusion in a user datagram are optional. If the checksum is not calculated, the field is filled with 1s. Note that a calculated checksum can never be all 1s because this implies that the sum is all 0s, which is impossible because it requires that the value of fields to be 0s.

4. UDP Operation:

UDP uses concepts common to the transport layer.

Connectionless Services

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. Also, there is no connection establishment and no connection termination. This means that each user datagram can travel on a different path.

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.

Encapsulation and Decapsulation

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

Queuing

In UDP, queues are associated with ports. (Figure 4.12).

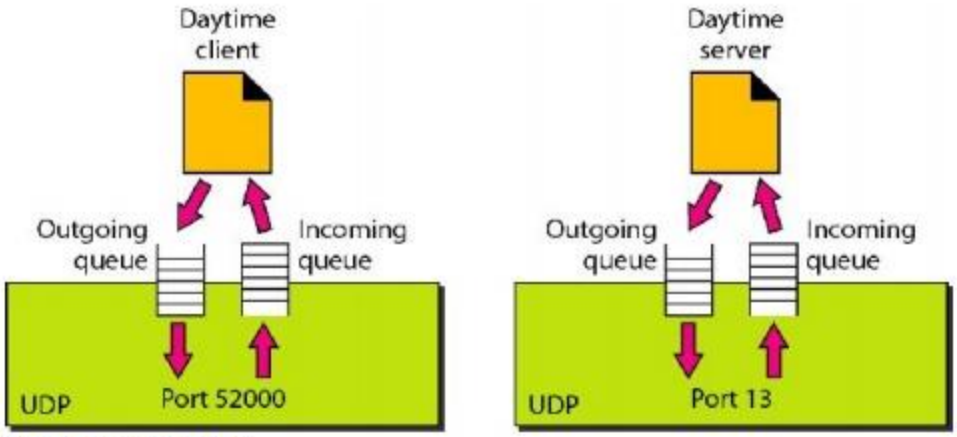

Figure 4.12 Queues in UDP

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.

Note that 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.

5. Use of UDP

The following lists some uses of the UDP protocol:

- 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).

Transmission Control Protocol (TCP)

TCP is a process-to-process (program-to-program) protocol. TCP is called a connection-oriented, reliable transport protocol. TCP uses flow and error control mechanisms at the transport level.

1. TCP Services

The services offered by TCP to the processes at the application layer.

Process-to-Process Communication

TCP provides process-to-process communication using port numbers. Table 4.2 lists some well-known port numbers used by TCP.

Table 4.2 Well-Known Port used by TCP

Port	Protocol	Description
1	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	Bootsrap protocol
79	Finger	Finger
80	HTTP	Hypertext transfer protocol
111	RPC	Remote procedure Call

2. Stream Delivery Service

TCP 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. This imaginary environment is depicted in Figure 4.13. The sending process produces (writes to) the stream of bytes, and the receiving process consumes (reads from) them.

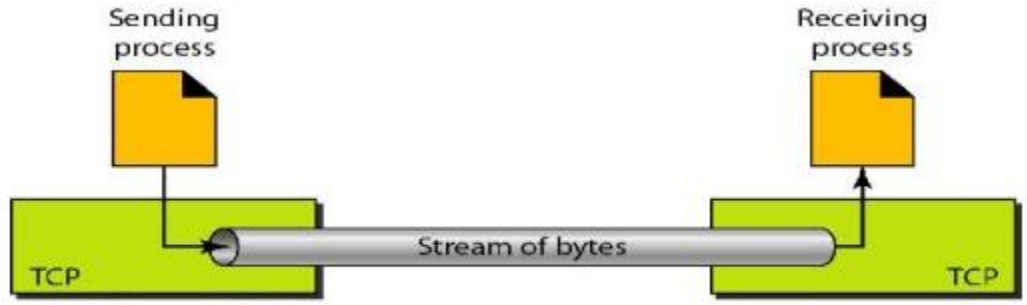

Figure 4.13 ***Stream delivery***

Sending and Receiving Buffers:

Because 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 I-byte locations as shown in Figure 4.14. For simplicity, we have shown two buffers of 20 bytes each; normally the buffers are hundreds or thousands of bytes, depending on the implementation. We also show the buffers as the same size, which is not always the case.

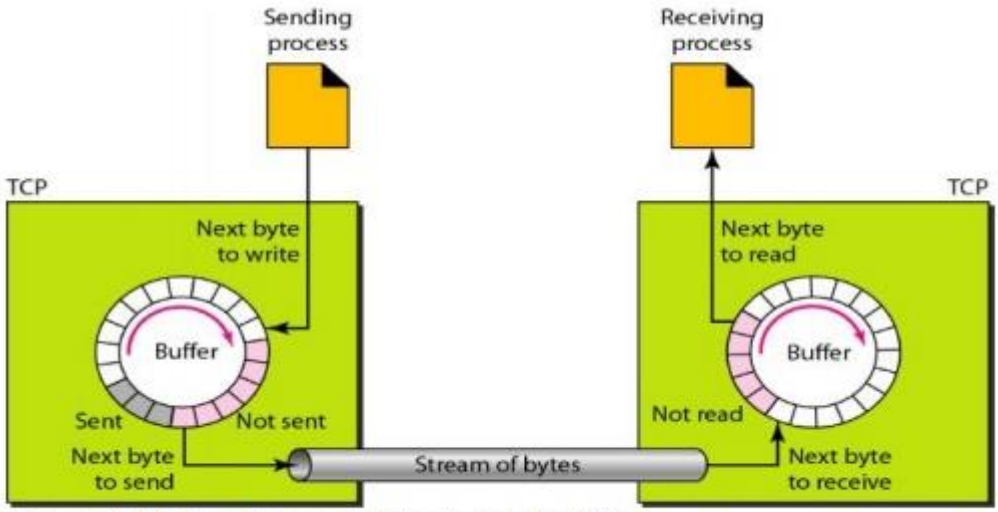

Figure 4.14 Sending and Receiving Buffers

Figure 4.14 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.

Segments

Segments although buffering handles the disparity between the speed of the producing and consuming processes, we need one more step before we can send data. The IP layer, as a service provider for TCP, needs to send data in packets, not as a stream of bytes. At the transport layer, TCP groups a number of bytes together into a packet called a segment. Figure 4.15 show segments are created from the bytes in the buffers.

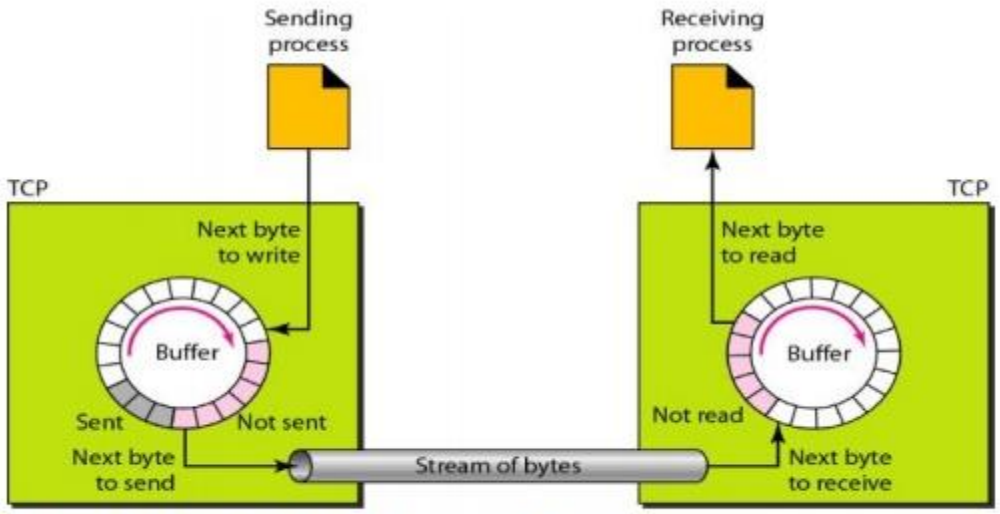

Figure 4.14 Sending and Receiving Buffers

3. 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.

4. Connection-Oriented Service

TCP 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.

Reliable Service

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

5. TCP Features

TCP has several features.

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 the byte number and not the segment number.

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

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.

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

Acknowledgment Number

Communication in TCP is full duplex; when a connection is established, both parties can send and receive data at the same time. Each party numbers the bytes, usually with a different starting byte 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. However, the acknowledgment number defines the number of the next byte that the party expects to receive. In addition, 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 I to it, and announces this sum as the acknowledgment number.

6. Flow Control

TCP provides flow control. 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, TCP implements an error control mechanism. Although error control considers a segment as the unit of data for error detection (loss or corrupted segments), error control is byte-oriented.

Congestion Control

TCP, unlike UDP, takes into account congestion in the network. 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.

7. Segment

A packet in TCP is called a segment.

8. Format

The format of a segment is shown in Figure 4.16

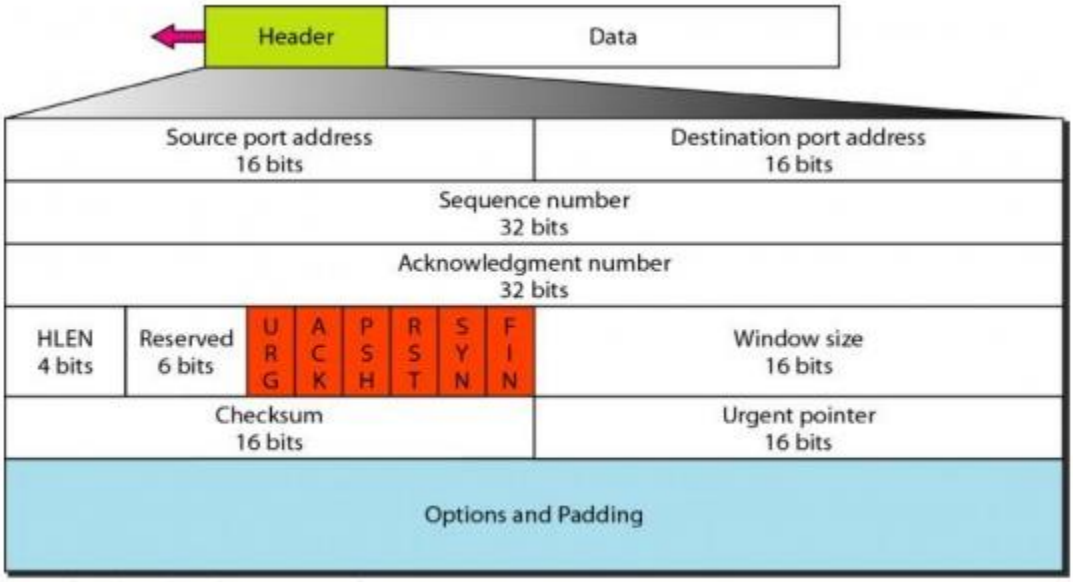

Figure 4.16 TCP Segment format

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.

- **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.

- **Destination port address**. This is a 16-bit field that 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. As we said before, 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.

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

- **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 4.17.One or more of these bits can be set at a time.

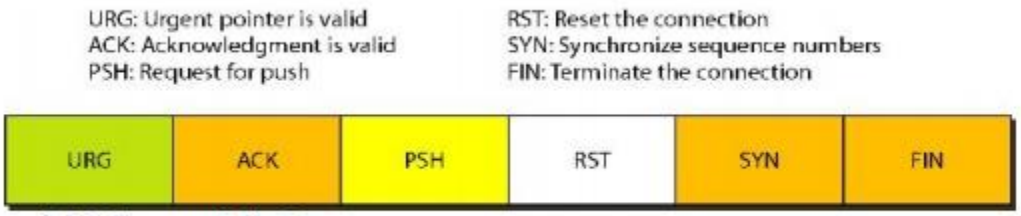

Figure 4.17 Control Field

These bits enable flow control, connection establishment and termination, connection abortion, and the mode of data transfer in TCP. A brief description of each bit is shown in Table 4.3

Table 4.3 Description of flags in the control field

Flag	Description
URG	The value of the urgent pointer field is valid.
ACK	The value of the acknowledge 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. Note that 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. The sender must obey the dictation of the receiver in this case.

- **Checksum**. This 16-bit field contains the checksum. The inclusion of the checksum for TCP is mandatory. For the TCP pseudo header, the value for the protocol field is 6.

- **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.

- **Options.** There can be up to 40 bytes of optional information in the TCP header.

9. A TCP Connection

TCP is connection-oriented. 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.

a. 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 4.18.

To show the process, we use two time lines: one at each site. Each segment has values for all its header fields and perhaps for some of its option fields, too. However, we show only the few fields necessary to understand each phase. 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.

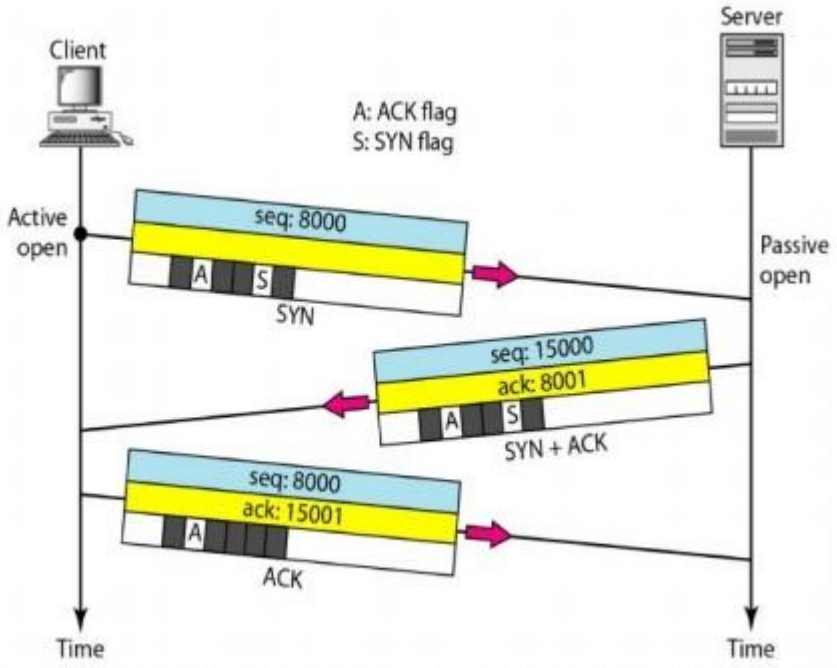

Figure 4.18 Connection establishment using three-way handshaking

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 start, 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.

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

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 segment for communication in the other direction and serves as the acknowledgment for the SYN segment. It consumes one sequence number.

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

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. Note that 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.

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

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 corning from a different client by faking the source IP addresses in the datagrams.

b. Data Transfer

After connection is established, bidirectional data transfer can take place. The client and server can both send data and acknowledgments. The acknowledgment is piggybacked with the data. Figure 4.19 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. 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. Note the values of the sequence and acknowledgment numbers. 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. Most TCP implementations have the option to set or not set this flag.

Pushing Data

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.

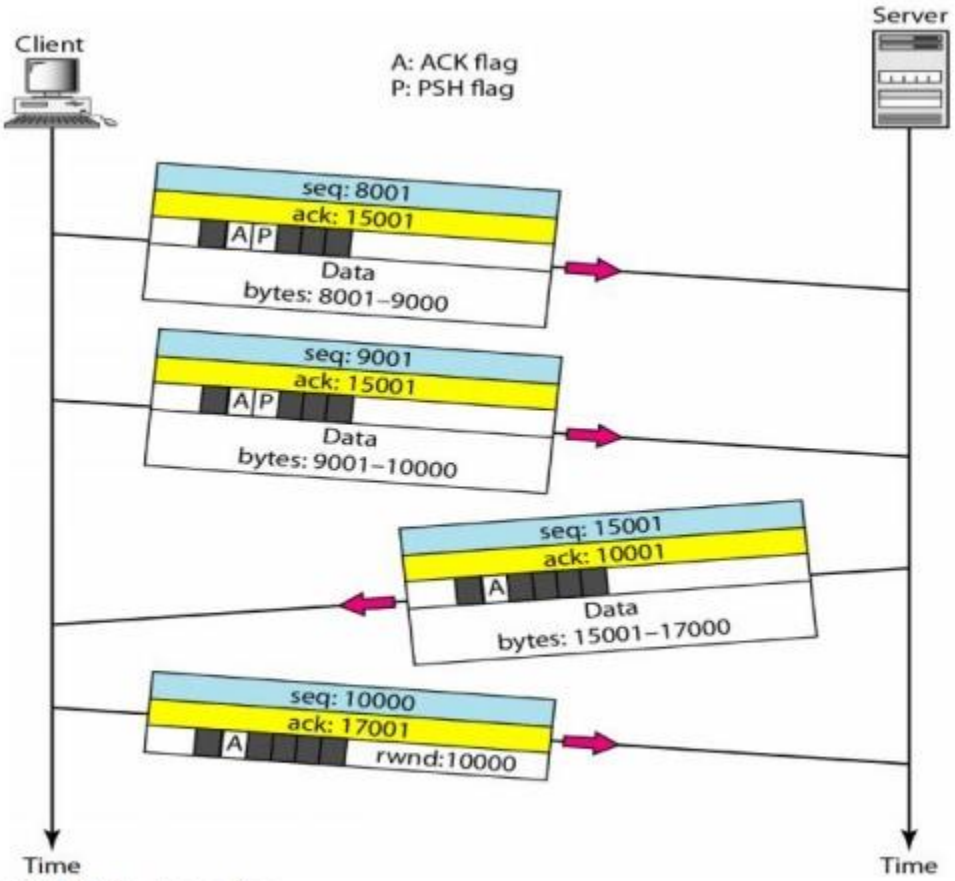

Figure 4.19 Data transfer

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.

Although the push operation can be requested by the application program, most current implementations ignore such requests. TCP can choose whether or not to use this feature.

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, 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.

c. 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.

Three-Way Handshaking

Most implementations today allow three-way handshaking for connection termination as shown in Figure 4.20.

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 4.20. If it is only a control segment, it consumes only one sequence number.

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

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.

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

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.

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 4.21 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. Note the sequence numbers we have used. The second segment (ACK) consumes no sequence number. Although the client has received sequence number y - 1 and is expecting y, the server sequence number is still y - 1. When the connection finally closes, the sequence number of the last ACK segment is still x, because no sequence numbers are consumed during data transfer in that direction.

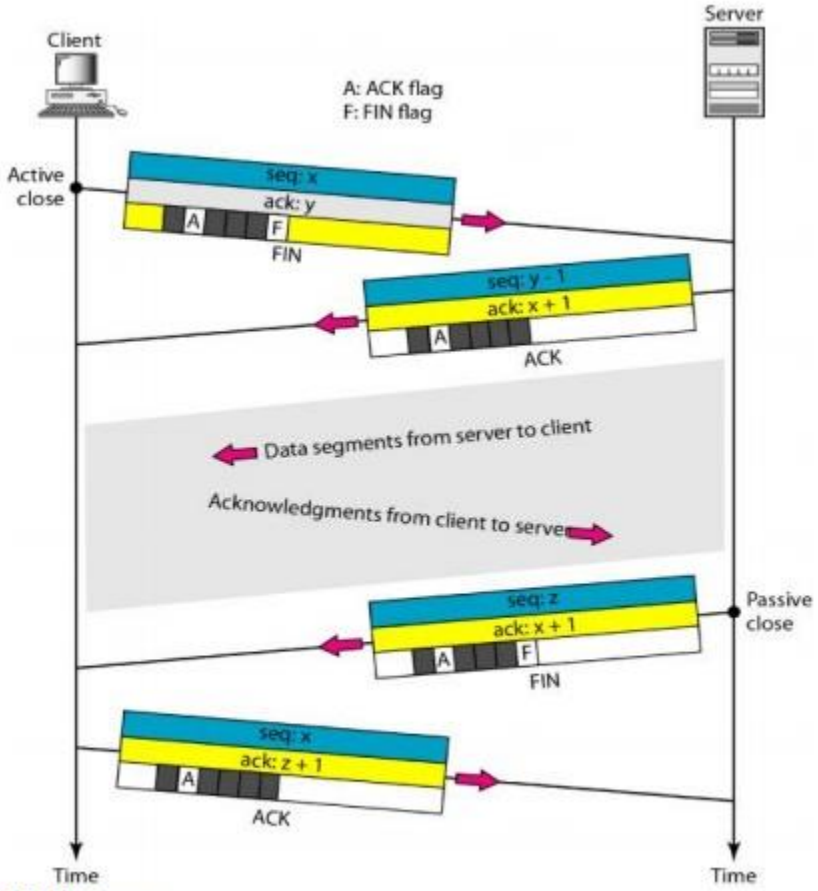

Figure 4.21 Half close

10. 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.

The window is opened, closed, or shrunk. These three activities, as we will see, 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 in this matter.

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.

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.

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 4.22 Sliding Window

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.

11. 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.

TCP provides reliability using error control. Error control includes mechanisms for detecting corrupted segments, lost segments, out-of-order segments, and duplicated segments. Error control also includes a mechanism for correcting errors after they are detected. Error detection and correction in TCP is achieved through the use of three simple tools: checksum, acknowledgment, and time-out.

a. 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. TCP uses a 16-bit checksum that is mandatory in every segment. The 16-bit checksum is considered

inadequate for the new transport layer, SCTP. However, it cannot be changed for TCP because this would involve reconfiguration of the entire header format.

b. 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.

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

c. Retransmission

The heart of the error control mechanism is the retransmission of segments. When a segment is corrupted, lost, or delayed, it is retransmitted. In modern implementations, a segment is retransmitted on two occasions: when a retransmission timer expires or when the sender receives three duplicate ACKs.

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

Note that no retransmission occurs for segments that do not consume sequence numbers. In particular, there is no transmission for an ACK segment.

No retransmission timer is set for an ACK segment.

i. Retransmission After RTO

A recent implementation of TCP maintains one retransmission time-out (RTO) timer for all outstanding (sent, but not acknowledged) segments. When the timer matures, the earliest outstanding segment is retransmitted even though lack of a received ACK can be due to a delayed segment, a delayed ACK, or a lost acknowledgment. Note that no time-out timer is set for a segment that carries only an acknowledgment, which means that no such segment is resent.

ii. Retransmission After Three Duplicate ACK Segments

The previous rule about retransmission of a segment is sufficient if the value of RTO is not very large. Sometimes, however, one segment is lost and the receiver receives so many out-of-order segments that they cannot be saved (limited buffer size).

iii. 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.

Some Scenarios

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.

a. Normal Operation

The first scenario shows bidirectional data transfer between two systems, as in Figure .

The client TCP sends one segment; the server TCP sends three. The figure shows which rule applies to each acknowledgment. There are data to be sent, so the segment displays the next byte expected. When the client receives the first segment from the server, it does not have any more data to send; it sends only an ACK segment

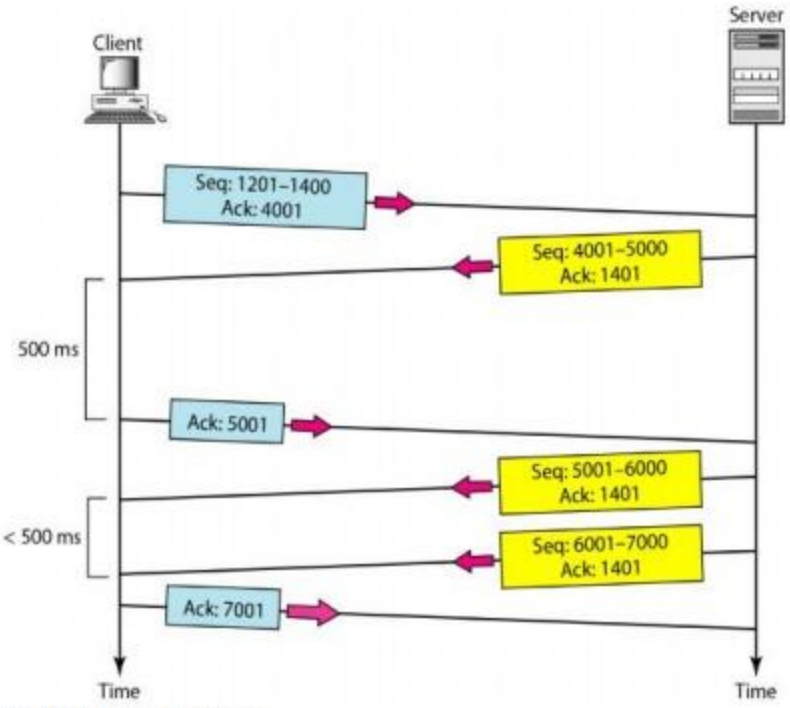

Figure 4.23 Normal operation

b. Lost Segment

In this scenario, we show what happens when a segment is lost or corrupted. A lost

segment and a corrupted segment are treated the same way by the receiver. A lost segment is discarded somewhere in the network; a corrupted segment is discarded by the receiver itself. Both are considered lost. Figure 4.24 shows a situation in which a segment is lost and discarded by some router in the network, perhaps due to congestion.

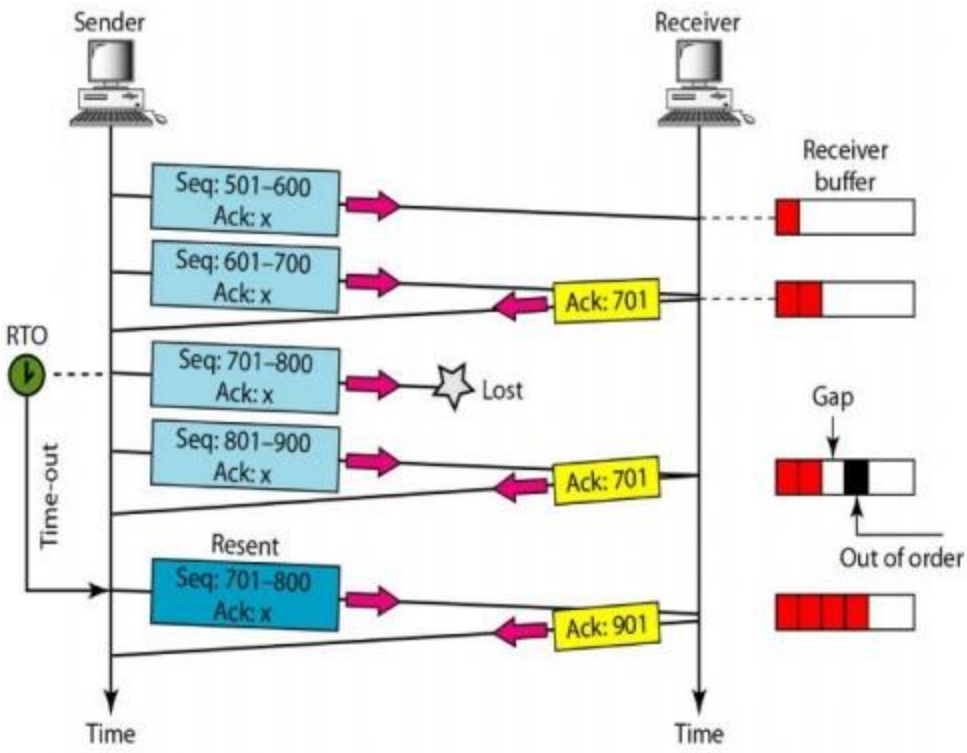

Figure 4.24 Lost segment.

We are assuming that data transfer is unidirectional: one site is sending and the other is receiving. In our scenario, the sender sends segments 1 and 2, which are acknowledged immediately by an ACK. Segment 3, however, are lost. The receiver receives segment 4, which

is out of order. The receiver stores the data in the segment in its buffer but leaves a gap to indicate that there is no continuity in the data. The receiver immediately sends an acknowledgment to the sender, displaying the next byte it expects. Note that the receiver stores bytes 801 to 900, but never delivers these bytes to the application until the gap is filled.

The receiver TCP delivers only ordered data to the process.

c. Fast Retransmission

In this scenario, we want to show the idea of fast retransmission. Our scenario is the same as the second except that the RTO has a higher value (see Figure 4.25).

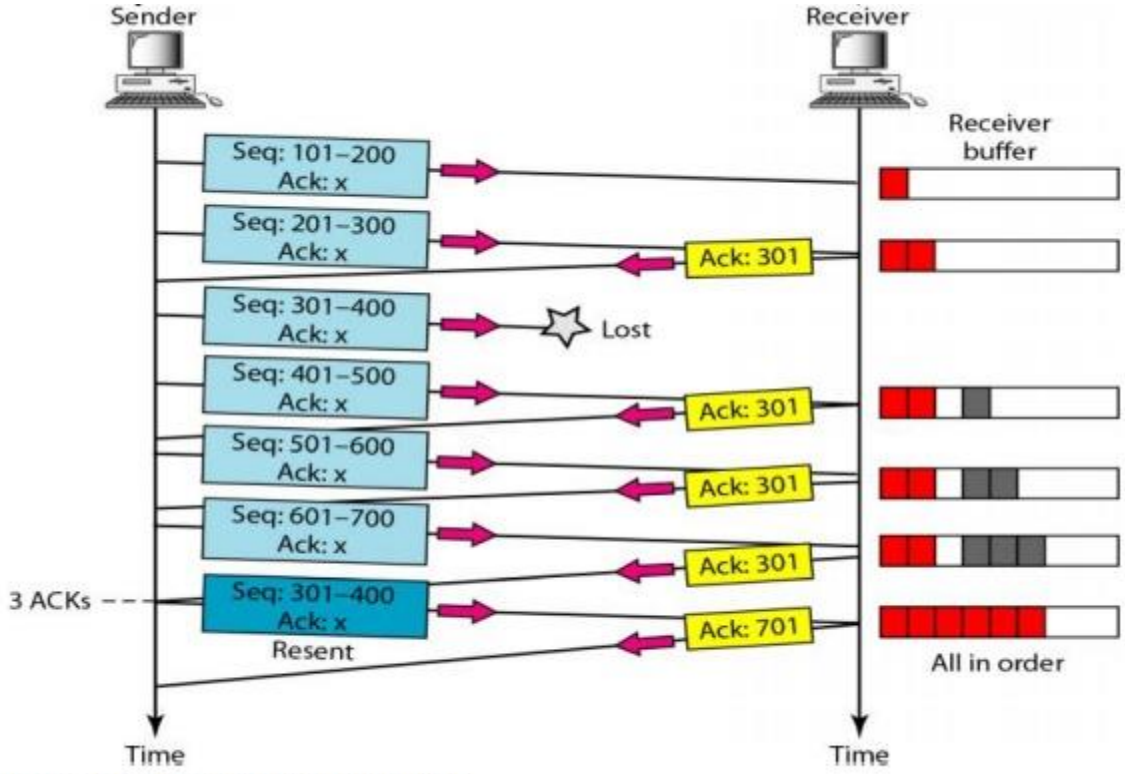

Figure 4.25 Fast Retransmission

When the receiver receives the fourth, fifth, and sixth segments, it triggers an acknowledgment. The sender receives four acknowledgments with the same value (three duplicates). Although the timer for segment 3 has not matured yet, the fast transmission requires that segment 3, the segment that is expected by all these acknowledgments, be resent immediately.

Congestion

Congestion control and quality of service are two issues so closely bound together that improving one means improving the other and ignoring one usually means ignoring the other.

1. Prerequisite discussion:

Most techniques to prevent or eliminate congestion also improve the quality of service in a network.

An important issue in a packet-switched network 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 to control the congestion and keep the load below the capacity. Congestion happens in any system that involves waiting. For example, congestion happens on a freeway because any abnormality in the flow, such as an accident during rush hour, creates blockage.

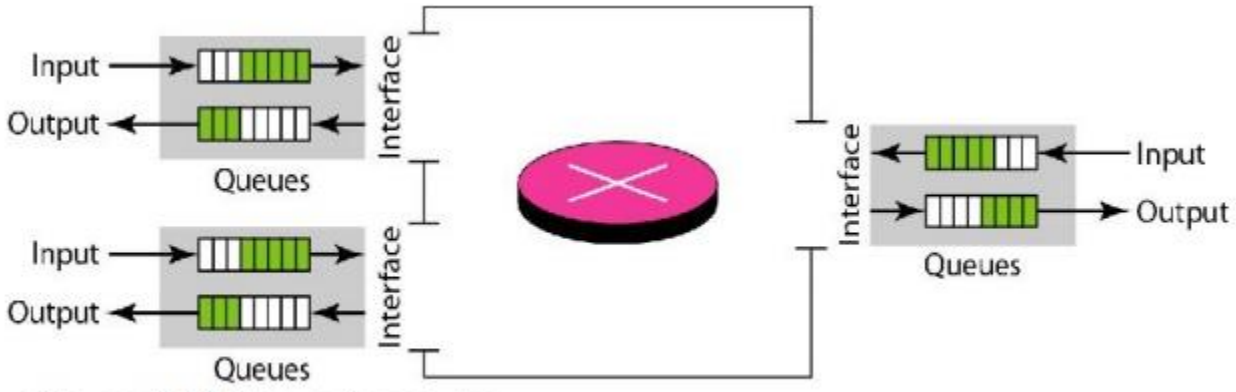

Figure 4. 26 Queues in a router

Congestion in a network or internetwork occurs because routers and switches have queues-buffers that hold the packets before and after processing. A router, for example, has an input queue and an output queue for each interface. When a packet arrives at the incoming interface, it undergoes three steps before departing, as shown in Figure 4.26.

- The packet is put at the end of the input queue while waiting to be checked.

- The processing module of the router removes the packet from the input queue once it reaches the front of the queue and uses its routing table and the destination address to find the route.

- The packet is put in the appropriate output queue and waits its turn to be sent.

We need to be aware of two issues. First, if the rate of packet arrival is higher than the packet processing rate, the input queues become longer and longer. Second, if the packet departure rate is less than the packet processing rate, the output queues become longer and longer.

Congestion Control: Open Loop and Closed Loop

Congestion control refers to techniques and mechanisms that can either prevent congestion, before it happens, or remove congestion, after it has happened.

Congestion Control

Congestion control refers to techniques and mechanisms that can either prevent congestion, before it happens, or remove congestion, after it has happened.

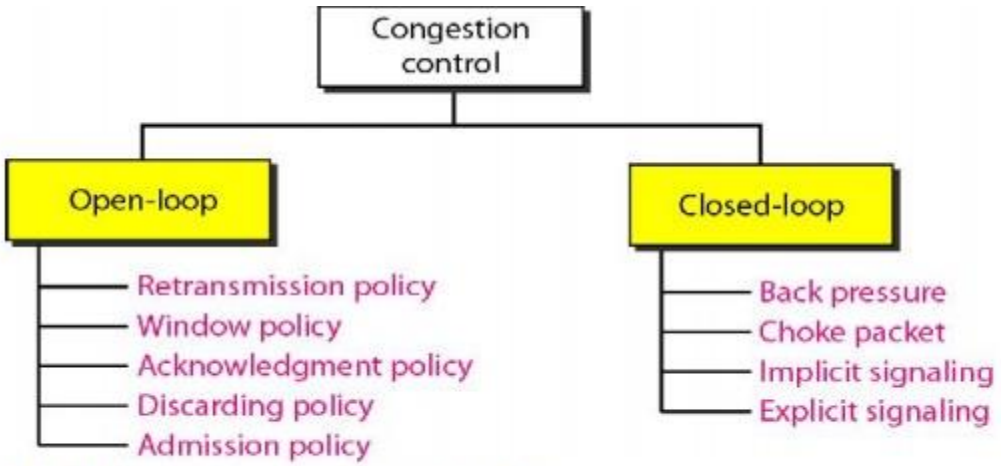

Figure 4. 27 Congestion control categories

In general, we can divide congestion control mechanisms into two broad categories: open-loop congestion control (prevention) and closed-loop congestion control (removal) as shown in Figure 4.27.

1. Open-Loop Congestion Control

In open-loop congestion control, policies are applied to prevent congestion before it happens. In these mechanisms, congestion control is handled by either the source or the destination.

a. Retransmission Policy

Retransmission is sometimes unavoidable. If the sender feels that a sent packet is lost or corrupted, the packet needs to be retransmitted. Retransmission in general may increase congestion in the network. However, a good retransmission policy can prevent congestion. The retransmission policy and the retransmission timers must be designed to optimize efficiency and at the same time prevent congestion. For example, the retransmission policy used by TCP (explained later) is designed to prevent or alleviate congestion.

b. Window Policy

The type of window at the sender may also affect congestion. The Selective Repeat window is better than the Go-Back-N window for congestion control. In the Go-Back-N window, when the timer for a packet times out, several packets may be resent, although some may have arrived safe and sound at the receiver. This duplication may make the congestion worse. The Selective Repeat window, on the other hand, tries to send the specific packets that have been lost or corrupted.

c. Acknowledgment Policy

The acknowledgment policy imposed by the receiver may also affect congestion. If the receiver does not acknowledge every packet it receives, it may slow down the sender and help prevent congestion. Several approaches are used in this case. A receiver may send an acknowledgment only if it has a packet to be sent or a special timer expires. A receiver may decide to acknowledge only N packets at a time. We need to know that the acknowledgments are also part of the load in a network. Sending fewer acknowledgments means imposing fewer loads on the network.

d. Discarding Policy

A good discarding policy by the routers may prevent congestion and at the same time may not harm the integrity of the transmission. For example, in audio transmission, if the policy is to discard less sensitive packets when congestion is likely to happen, the quality of sound is still preserved and congestion is prevented or alleviated.

e. Admission Policy

An admission policy, which is a quality-of-service mechanism, can also prevent congestion in virtual-circuit networks. Switches in a flow, first check the resource requirement of a flow before admitting it to the network. A router can deny

establishing a virtual circuit connection if there is congestion in the network or if there is a possibility of future congestion.

2. Closed-Loop Congestion Control

Closed-loop congestion control mechanisms try to alleviate congestion after it happens. Several mechanisms have been used by different protocols.

a. Backpressure

The technique of backpressure refers to a congestion control mechanism in which a congested node stops receiving data from the immediate upstream node or nodes. This may cause the upstream node or nodes to become congested, and they, in turn, reject data from their upstream nodes or nodes. And so on. Backpressure is a node-to-node congestion control that starts with a node and propagates, in the opposite direction of data flow, to the source. The backpressure technique can be applied only to virtual circuit networks, in which each node knows the upstream node from which a flow of data is corning. Figure 4.28 shows the idea of backpressure.

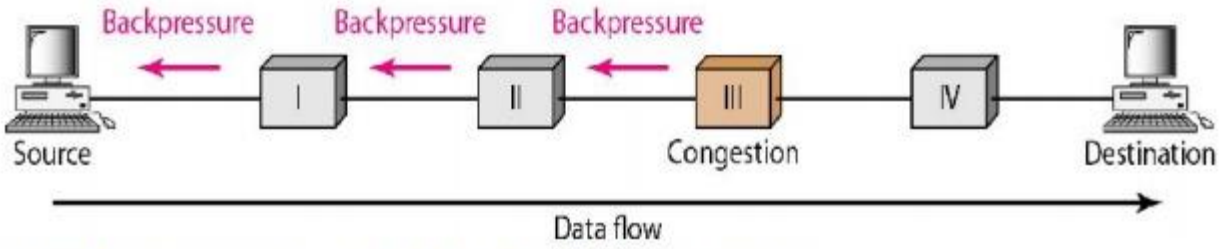

Figure 4.28 Backpressure method for alleviating congestion

Node III in the figure has more input data than it can handle. It drops some packets in its input buffer and informs node II to slow down. Node II, in turn, may be congested because it is slowing down the output flow of data. If node II is congested, it informs node I to slow down, which in turn may create congestion. If so, node I inform the source of data to slow down. This, in time, alleviates the congestion. Note that the pressure on node III is moved backward to the source to remove the congestion.

b. Choke Packet

A choke packet is a packet sent by a node to the source to inform it of congestion. Note the difference between the backpressure and choke packet methods. In backpressure, the warning is from one node to its upstream node, although the warning may eventually reach the source station. In the choke packet method, the warning is from the router, which has encountered congestion, to the source station directly. The intermediate nodes through which the packet has traveled are not warned. We have seen an example of this type of control in ICMP. When a router in the Internet is overwhelmed with IP datagrams, it may discard some of them; but it informs the source host, using a source quench ICMP message. The warning message goes directly to the source station; the intermediate routers, and does not take any action. Figure 4.29 shows the idea of a choke packet.

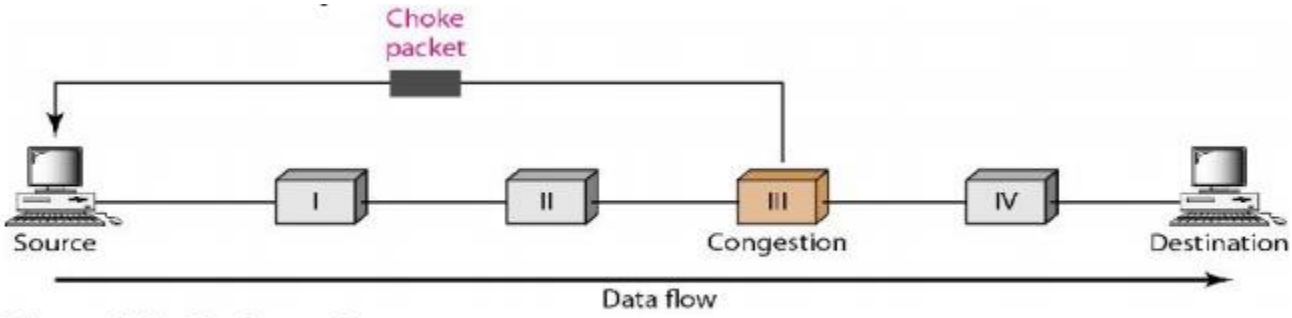

Figure 4.29 Choke packet

c. Implicit Signaling

In implicit signaling, there is no communication between the congested node or nodes and the source. The source guesses that there is congestion somewhere in the network from other symptoms. For example, when a source sends several packets and there is no acknowledgment for a while, one assumption is that the network is congested. The delay in receiving an acknowledgment is interpreted as congestion in the network; the source should slow down.

d. Explicit Signaling

The node that experiences congestion can explicitly send a signal to the source or destination. The explicit signaling method, however, is different from the choke packet method. In the choke packet method, a separate packet is used for this purpose; in the explicit signaling method, the signal is included in the packets that carry data. Explicit signaling, as we will see in Frame Relay congestion control, can occur in either the forward or the backward direction.

i. Backward Signaling

A bit can be set in a packet moving in the direction opposite to the congestion. This bit can warn the source that there is congestion and that it needs to slow down to avoid the discarding of packets.

ii. Forward Signaling

A bit can be set in a packet moving in the direction of the congestion. This bit can warn the destination that there is congestion. The receiver in this case can use policies, such as slowing down the acknowledgments, to alleviate the congestion.