FLOW CONTROL ,ERROR CONROL AND CONGESTION CONTROL AT TRANSPORT LAYER

Flow Control at Transport Layer:

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

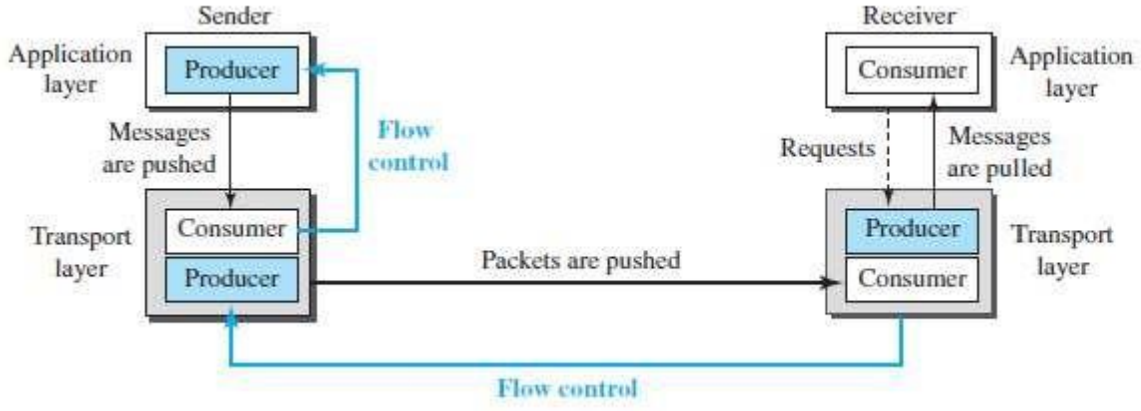


Fig: Flow control at the transport layer.

Buffers:

- Although flow control can be implemented in several ways, one of the solutions is normallyto usetwo buffers: one atthe sending transport layer andthe other atthe receivingtransport layer.
- A buffer is a set of memory locations that can hold packets at thesender and receiver.

Error Control:

- Error control at the transport layer is responsible for

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

Sequence Numbers

- Error control requires that the sending transport layer knows which packet is to be resent and the receiving transport layer knows which packet is a duplicate, or which packet has arrived out of order. This can be done if the packets are numbered.
- We can add a field to the transport-layer packet to hold the **sequence number** of the packet. When a packet is corrupted or lost, the receiving transport layer can somehow inform the sending transport layer to resend that packet using the sequence number.
- The receiving transport layer can also detect duplicate packets if two received packets have the same sequence number.
- The out-of-order packets can be recognized by observing gaps in the sequence numbers.

Acknowledgment:

- The receiver side can send an acknowledgment (ACK) for each of a collection of packets that have arrived safe and sound.

If an ACK does

- not arrive before the timer expires, the sender resends the packet.

Combination of Flow and Error Control:

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

Sliding Window:

- The buffer is represented as a set of slices, called the ***sliding window,*** that occupies part of the circle at any time. At the sender site, when a packet is sent, the corresponding slice is marked.

- When all the slices are marked, it means that the buffer is full and no further messages can be accepted from the application layer. When an acknowledgment arrives.

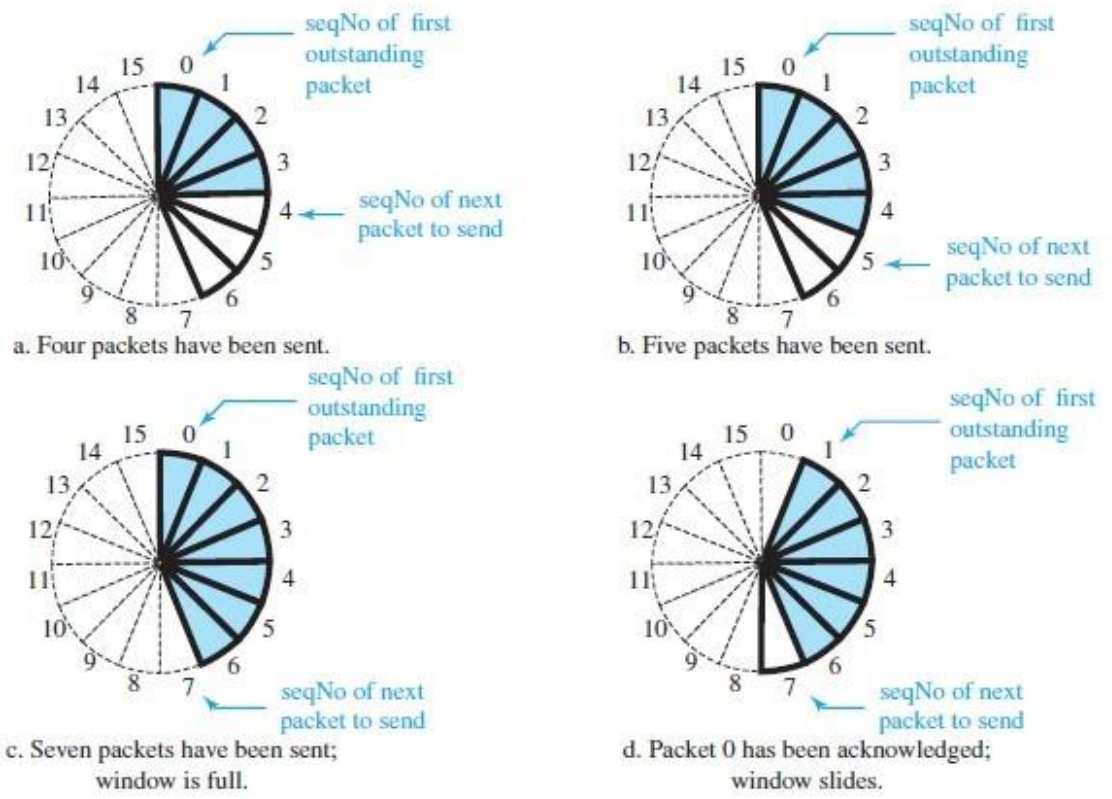


Fig: Sliding window in circular format.

- Most protocols show the sliding window using linear representation. The idea is the same.

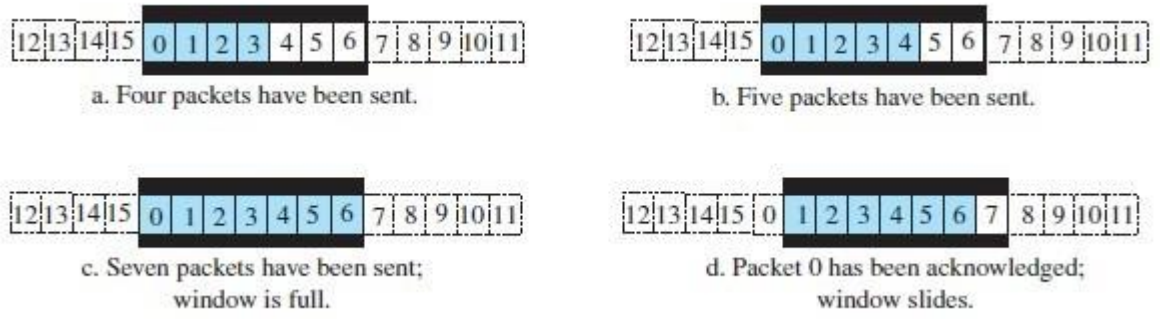


Fig: Sliding window in linear format.

Congestion Control:

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

Connectionless and Connection-Oriented Protocols:

- A transport-layer protocol, like a network-layer protocol, can provide two types of services: connectionless and connection-oriented.

Connectionless Service:

- In a connectionless service, the source process (application program) needs to divide its message into chunks of data of the size acceptable by the transport layer and deliver them to the transport layer one by one.
- The transport layer treats each chunk as a single unit without any relation between the chunks. When a chunk arrives from the application layer, the transport layer encapsulates it in a packet and sends it.
- To show the independency of packets, assume that a client process has three chunks of messages to send to a server process.
- The packets may arrive out of order at the destination and will be delivered out oforder to the server process.

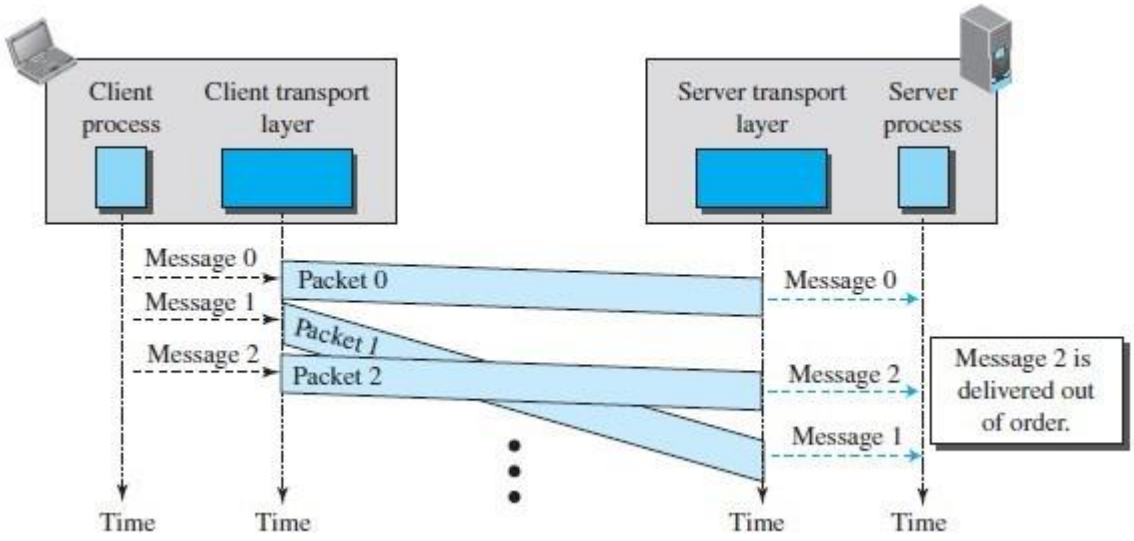


Fig: Connectionless service:

- If these three chunks of data belong to the same message, the server process may have received a strange message. The situation would be worse if one of the packets were lost.
- Since there is no numbering on the packets, the receiving transport layer has no idea that one of the messages has been lost. It just delivers two chunks of data to the server process.
- Two problems arise from the fact that the two transport layers do not coordinate with each other.
- No flow control, error control, or congestion control can be effectively implemented in a connectionless service.

Go-Back-N Protocol (GBN)

- To improve the efficiency of transmission (to fill the pipe), multiple packets must be in transition while the sender is waiting for acknowledgment.
- The key to Go-back-N is that we can send several packets before receiving acknowledgments, but the receiver can only buffer one packet. We keep a copy of the sent packets until the acknowledgments arrive.

Sequence Numbers:

- The sequence numbers are modulo 2m,

Acknowledgment Numbers:

- **In the Go-Back-N protocol, the acknowledgment number is cumulative and defines the sequence number of the next packet expected to arrive.**

Send Window:

- The send window is an imaginary box covering the sequence numbers of the data packets that can be in transit or can be sent.
- The send window at any time divides the possible sequence numbers into four regions. The first region, left of the window, defines the sequence numbers belonging to packets that are already acknowledged. The sender does not worry about these packets and keeps no copies of them.
- The second region, colored, defines the range of sequence numbers belonging to the packets that have been sent, but have an unknown status. The sender needs to wait to find out if these packets have been received or were lost. We call these outstanding packets.
- The third range, white in the figure, defines the range of sequence numbers for packets that can be sent; however, the corresponding data have not yet been received from the application layer.
- Finally, the fourth region, right of the window, defines sequence numbers that cannot be used until the window slides.

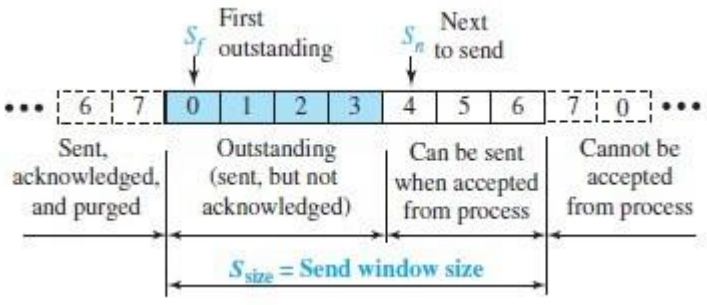


Fig: Send window for Go – Back – N.

Receive Window:

- The receive window makes sure that the correct data packets are received and that the correct acknowledgments are sent. In Go-Back-N, the size of the receive window is always 1.
- The receiver is always looking for the arrival of a specific packet. Any packetarriving out of order is discarded and needs to be resent.

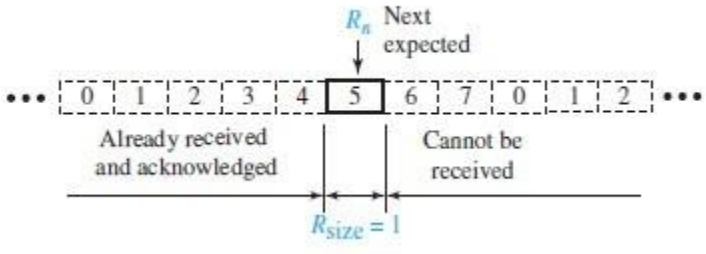


Fig: Receive window for Go- back –N.

- **The receive window is an abstract concept defining an imaginary box of size 1 with a single variable Rn. The window slides when a correct packet has arrived; sliding occurs one slot at a time.**

Timers:

- Although there can be a timer for each packet that is sent, in our protocol we use only one. The reason is that the timer for the first outstanding packet always expires first. Were send all outstanding packets when this timer expires.

Resending packets:

- When the timer expires, the sender resends all outstanding packets.

FSMs

Sender

- The sender starts in the ready state, but thereafter it can be in one of the two states :ready or blocking.

Ready state. Four events may occur when the sender is in ready state.

a. If a request comes from the application layer, the sender creates a packet with the sequence number set to Sn. A copy of the packet is stored, and the packet is sent. If the window is full, Sn
= (Sf +Ssize) modulo 2m, the sender goes to the blocking state.

b. If an error-free ACK arrives with ackNo related to one of the outstanding packets, the sender slides the window (set Sf = ackNo), and if all outstanding packets are acknowledged (ackNo =

Sn), then the timer is stopped. If all outstanding packets are not acknowledged, the timer is restarted.

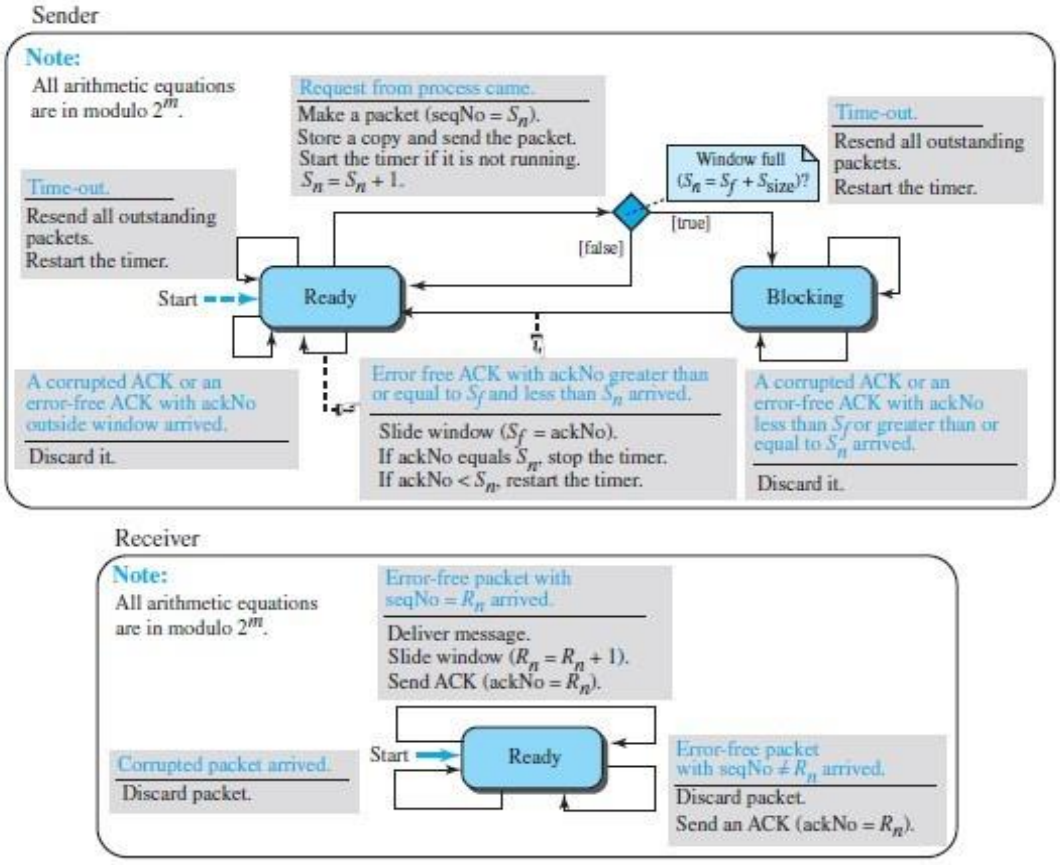


Fig: FSMs for the Go-Back-N protocol

c. If a corrupted ACK or an error-free ACK with ackNo not related to the outstanding packet arrives, it is discarded.

d. If a time-out occurs, the sender resends all outstanding packets and restarts the timer.

Blocking state. Three events may occur in this case:

a. If an error-free ACK arrives with ackNo related to one of the outstanding packets, the sender slides the window (set Sf = ackNo) and if all outstanding packets are acknowledged (ackNo = Sn), then the timer is stopped.

b. If all outstanding packets are not acknowledged, the timer is restarted. The sender then moves to the ready state.

c. If a corrupted ACK or an error-free ACK with the ackNo not related to the outstanding packets arrives, the ACK is discarded.

d. If a time-out occurs, the sender sends all outstanding packets and restarts the timer.

Receiver

- The receiver is always in the ready state. The only variable, Rn, is initialized to 0.

Three events may occur:

a. If an error-free packet with seqNo = Rn arrives, the message in the packet is delivered to the application layer. The window then slides, Rn = (Rn + 1) modulo 2m. Finally an ACK is sent with ackNo = Rn.

b. If an error-free packet with seqNo outside the window arrives, the packet is discarded, but an ACK with ackNo = Rn is sent.

c. If a corrupted packet arrives, it is discarded.

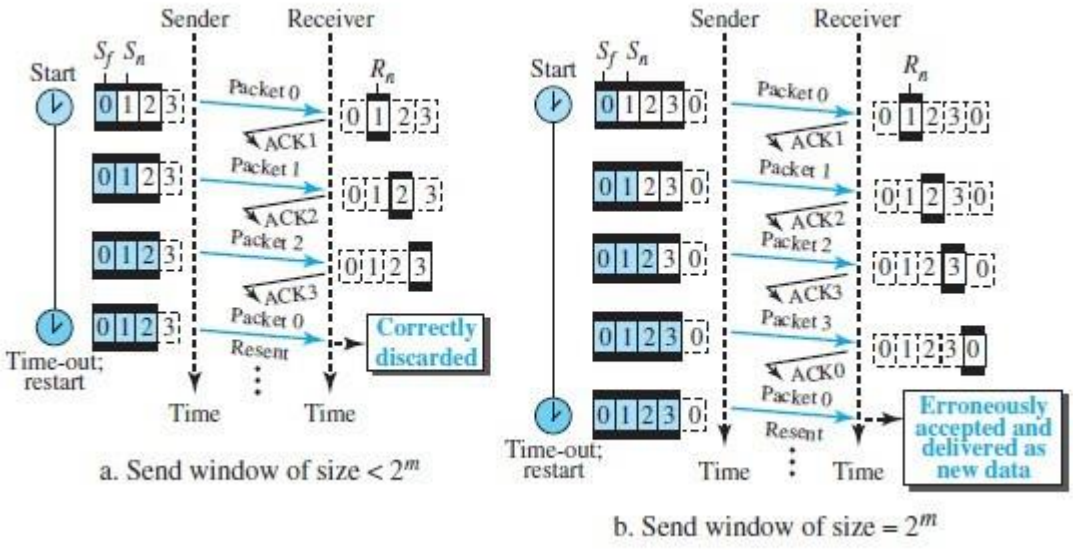
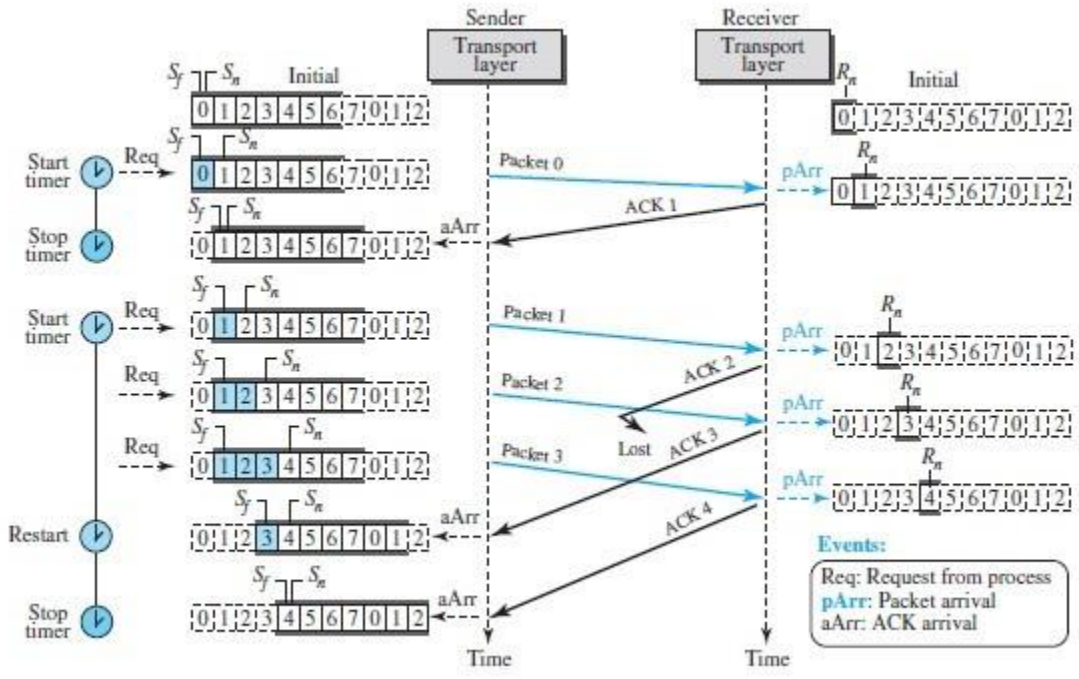


Fig: Send window size for Go – Back –N.



Fig; shows an example of Go- back – N.

Example: 1

Figure 2.3 shows what happens when a packet is lost. Packets 0, 1, 2, and 3 are sent. However, packet 1 is lost. The receiver receives packets 2 and 3, but they are discarded because they are received out of order (packet 1 is expected). When the receiver receives packets 2 and 3, it sends ACK1 to show that it expects to receive packet 1. However, these ACKs are not useful for the sender because the ackNo is equal to Sf , not greater than Sf. So the sender discards them. When the time-out occurs, the sender resends packets 1, 2, and 3, which are acknowledged.

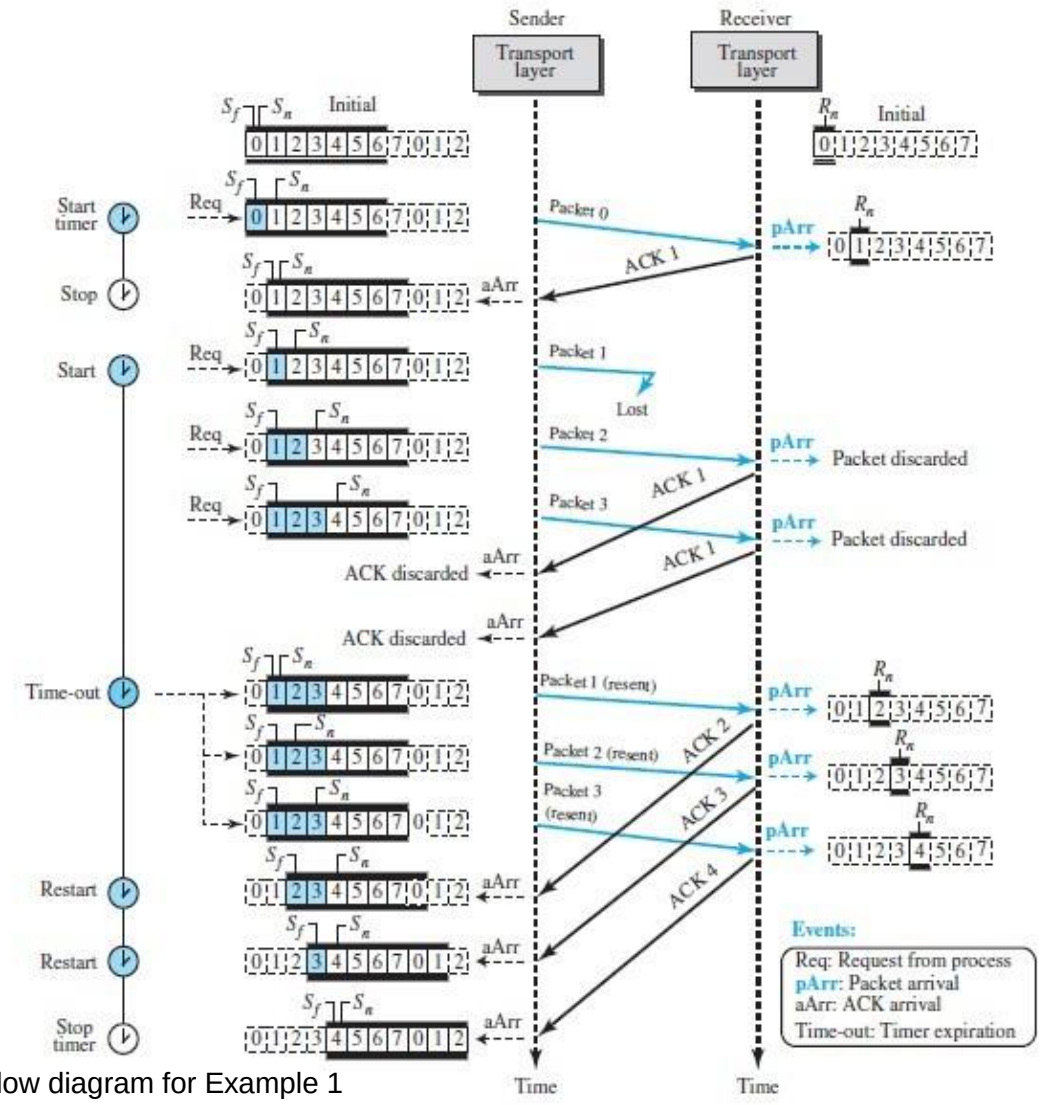


Fig: Flow diagram for Example 1

INTRODUCTION TO TRANSPORT LAYER

INTRODUCTION:

- The transport layer is located between the application layer and the network layer. It provides a process-to-process communication between two application layers.

Transport-Layer Services:

Process-to-Process Communication:

- The first duty of a transport-layer protocol is to provide **process-to-process communication.**

Addressing: Port Numbers:

- Although there are a few ways to achieve process-to-process communication, the most common is through the **client-server paradigm.**
- A process on the lo cal host, called a *client,* needs services from a process usually on the remote host, called a *server.*
- The local host and the remote host are defined using IP addresses (discussed in Chapter 18). To define the processes, we need second identifiers, called **port numbers.** In the TCP/IP protocol suite, the port numbers are integers between 0 and 65,535 (16 bits).
- The client program defines itself with a port number, called the **ephemeral port number.** The word *ephemeral* means“short-lived”.
- TCP/IP has decided to use universal port numbers for servers; these are called **well-known port numbers.**

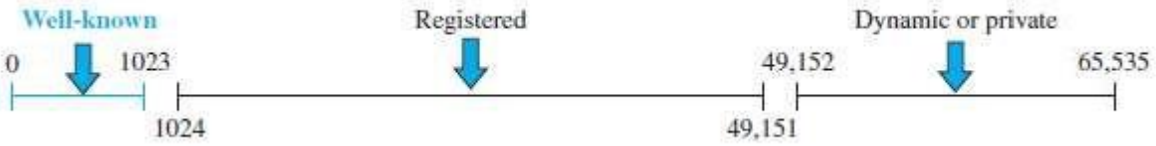


Fig: ICANN ranges.

- **Well-known ports.** The ports ranging from 0 to 1023 are assigned and controlled by ICANN. These are the well-known ports.
- **Registered ports.** The ports ranging from 1024 to 49,151 are not assigned or controlled by ICANN. They can only be registered with ICANN to prevent duplication.
- **Dynamic ports.** The ports ranging from 49,152 to 65,535 are neither controlled nor registered. They can be used as temporary or private port numbers.

Encapsulation and Decapsulation:

- To send a message from one process to another, the transport-layer protocol encapsulates and decapsulates messages.
- When a process has a message to send, it passes the message to the transport layer along with a pair of socket addresses and some other pieces of information, which depend on the transport- layer protocol.
- The transport layer receives the data and adds the transport-layer header. The packets at the transport layer in the Internet are called *user datagrams, segments,* or *packets,* depending on what transport-layer protocol we use.
- Decapsulation happens at the receiver site. When the message arrives at the destination transport layer, the header is dropped and the transport layer delivers the message to the process running at the application layer.
- The sender socket address is passed to the process in case it needs to respond to the message received.

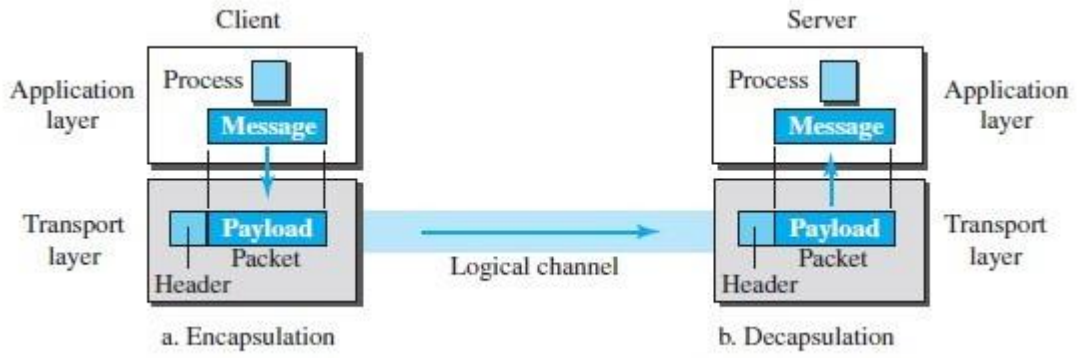


Fig: Encapsulation and decapsulation.

Multiplexing and Demultiplexing:

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

Flow Control:

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

Pushing or Pulling:

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

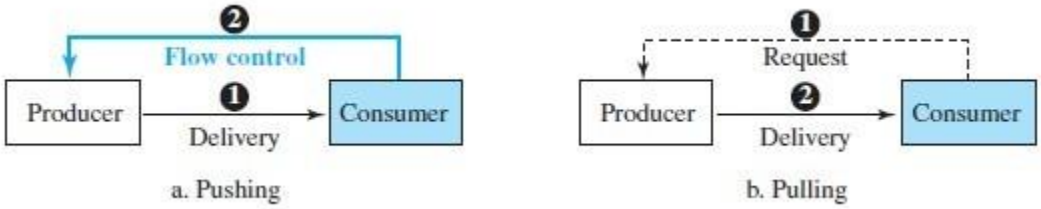


Fig: Pushing or pulling.

When the producer *pushes* the items, the consumer may be overwhelmed and there is a need for flow control.

Selective-Repeat Protocol

- Go-Back-*N* protocol Each time a single packet is lost or corrupted, the sender resends all outstanding packets, to avoid this Another protocol, called the **Selective-Repeat (SR) protocol** is used resends only selective packets, those that are actually lost.

Windows:

- The Selective-Repeat protocol also uses two windows: a send window and a receive window. The send window maximum size can be 2^{m-1}. The receive window in Selective-Repeat is totally different from the one in Go-Back-*N*. The size of the receive window is the same as the size of the send window(maximum $2m$-1).
- The Selective-Repeat protocol allows as many packets as the size of the receive window to arrive out of order and be kept until there is a set of consecutive packets to be delivered to the application layer.

Fig: Send window for Selective-Repeat protocol.

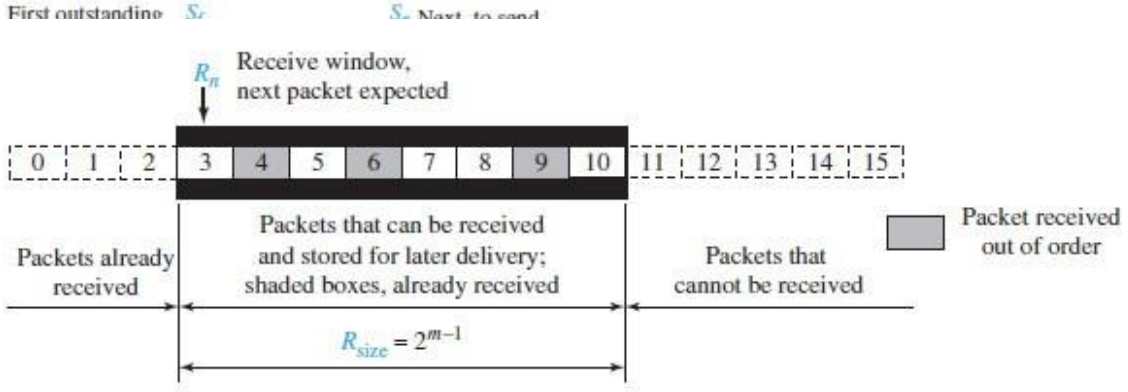


Fig: Receive window for Selective-Repeat protocol

Timer:

- Selective-Repeat uses one timer for each outstanding packet. When a timer expires, only the corresponding packet is resent.

Acknowledgments:

- In the Selective-Repeat protocol, an acknowledgment number defines the sequence number of the error-free packet received.

FSMs:

FSMs for the Selective-Repeat protocol.

Sender:

- The sender starts in the *ready* state, but later it can be in one of the two states: *ready* or *blocking*.

Ready state.

Four events may occur in this case:

a. If a request comes from the application layer, the sender creates a packet with the sequence number set to *Sn*. A copy of the packet is stored, and the packet is sent. If the timer is not running, the sender starts the timer. The value of *Sn* is now incremented, $Sn = (Sn + 1) \text{ modulo } 2^m$. If the window is full, $Sn = (Sf + S\text{size}) \text{ modulo } 2^m$, the sender goes to the blocking state.

b. If an error-free ACK, If there are outstanding packets, the timer is restarted; otherwise, the timer is stopped.

c. If a corrupted ACK or an error-free ACK with ackNo not related to an outstanding packet arrives, it is discarded.

d. If a time-out occurs, the sender resends all unacknowledged packets in the window and restarts the timer.

Blocking state.

Three events may occur in this case:

- If an error-free ACK arrives with ackNo related to one of the outstanding packets, that packet is marked as acknowledged.
- If a corrupted ACK or an error-free ACK with the ackNo not related to outstanding packets arrives, the ACK is discarded.

- If a time-out occurs, the sender resends all unacknowledged packets in the window and restarts the timer.

Receiver:

The receiver is always in the *ready* state. Three events may occur:

a. If an error-free packet with seqNo in the window arrives, the packet is stored and an ACK with ackNo = seqNo is sent. In addition, if the seqNo = *Rn*, then the packet and all previously arrived consecutive packets are delivered to the application ayer and the window slides.

b. If an error-free packet with seqNo outside the window arrives, the packet is discarded, but an ACK with ackNo = *Rn* is returned to the sender.

c. If a corrupted packet arrives, the packet is discarded.

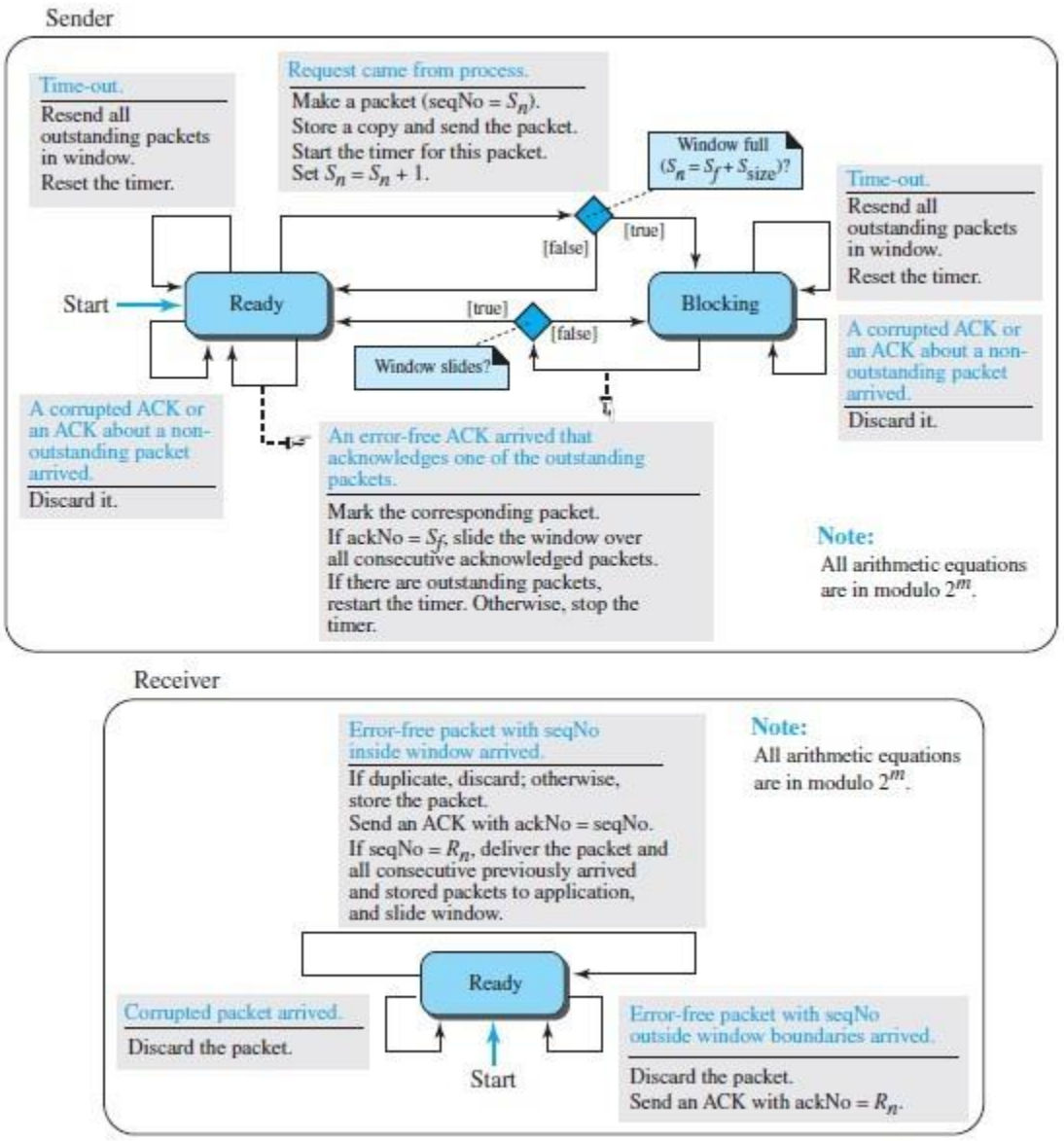


Fig: FSMs for SR protocol.

Example: 2
This example show how Selective-Repeat behaves in this case. Fig shows the situation.

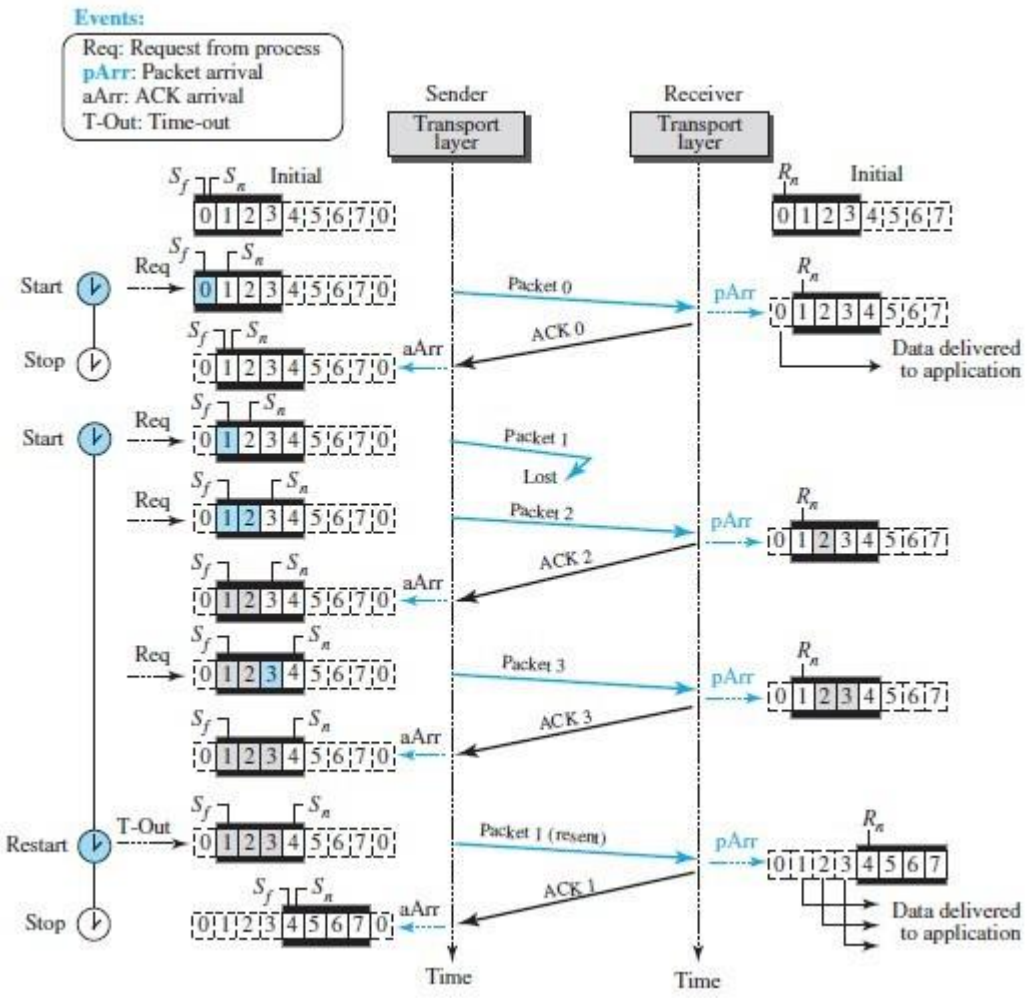
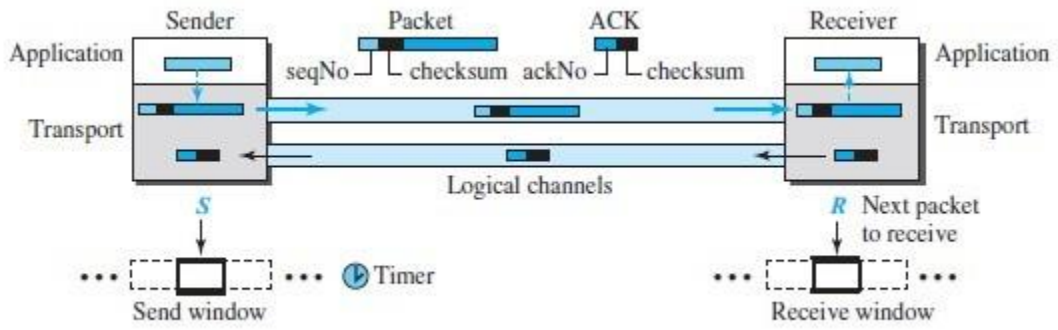


Fig: Flow diagram for example 2

Stop-and-Wait Protocol

- Connection-oriented protocol called the **Stop-and-Wait protocol,** which uses both flow and error control. Both the sender and the receiver use a sliding window of size 1. The sender sends one packet at a time and waits for an acknowledgment before sending the next one.
- To detect corrupted packets, we need to add a checksum to each data packet. When a packet arrives at the receiver site, it is checked. If its checksum is incorrect, the packet is corrupted and silently discarded.
- The silence of the receiver is a signal for the sender that a packet was either corrupted or lost. Every time the sender sends a packet, it starts a timer. If an acknowledgment arrives before the timer expires, the timer is stopped and the sender sends the next packet (if it has one to send).
- If the timer expires, the sender resends the previous packet, assuming that the packet was either lost or corrupted. This means that the sender needs to keep a copy of the packet until its acknowledgment arrives.

Fig: Stop – and – wait protocol.



- The Stop-and-Wait protocol is a connection-oriented protocol that provides flow and error control.

Sequence Numbers:

- To prevent duplicate packets, the protocol uses sequence numbers and acknowledgment numbers. The packet arrives safe and sound at the receiver site; the receiver sends an acknowledgment.
- The acknowledgment arrives at the sender site, causing the sender to send the next packet numbered $x + 1$.

- The packet is corrupted or never arrives at the receiver site; the sender resends the packet (numbered *x*) after the time-out.
- The packet arrives safe and sound at the receiver site; the receiver sends an acknowledgment, but the acknowledgment is corrupted or lost.
- The sender resends the packet (numbered *x*) after the time-out. Note that the packet here is a duplicate. The acknowledgment numbers always announce the sequence number of the *next packet expected* by the receiver.
- **In the Stop-and-Wait protocol, the acknowledgment number always announces, inmodulo-2 arithmetic, the sequence number of the next packet expected.**

FSMs:

- FSMs for the Stop-and-Wait protocol. Since the protocol is aconnection-oriented protocol, both ends should be in the *established* state beforeexchanging data packets. The states are actually nested in the *established* state.

Sender

- The sender is initially in the ready state, but it can move between the ready and blockingstate. The variable *S* is initialized to 0.

Ready state.

- When the sender is in this state, it is only waiting for one event to occur. If a request comes from the application layer, the sender creates a packet with the sequence number set to *S*. A copy of the packet is stored, and the packet is sent. The sender then starts the only timer. The sender then moves to the blocking state.

Blocking state.

- When the sender is in this state, three events can occur:
  a. If an error-free ACK arrives with the ackNo related to the next packet to be sent, then the timer is stopped. Finally, the sender moves to the ready state.
  b. If a corrupted ACK or an error-free ACK with the ackNo ≠ (S + 1) modulo 2arrives, the ACK is discarded.
  c. If a time-out occurs, the sender resends the only outstanding packet and restarts the timer.

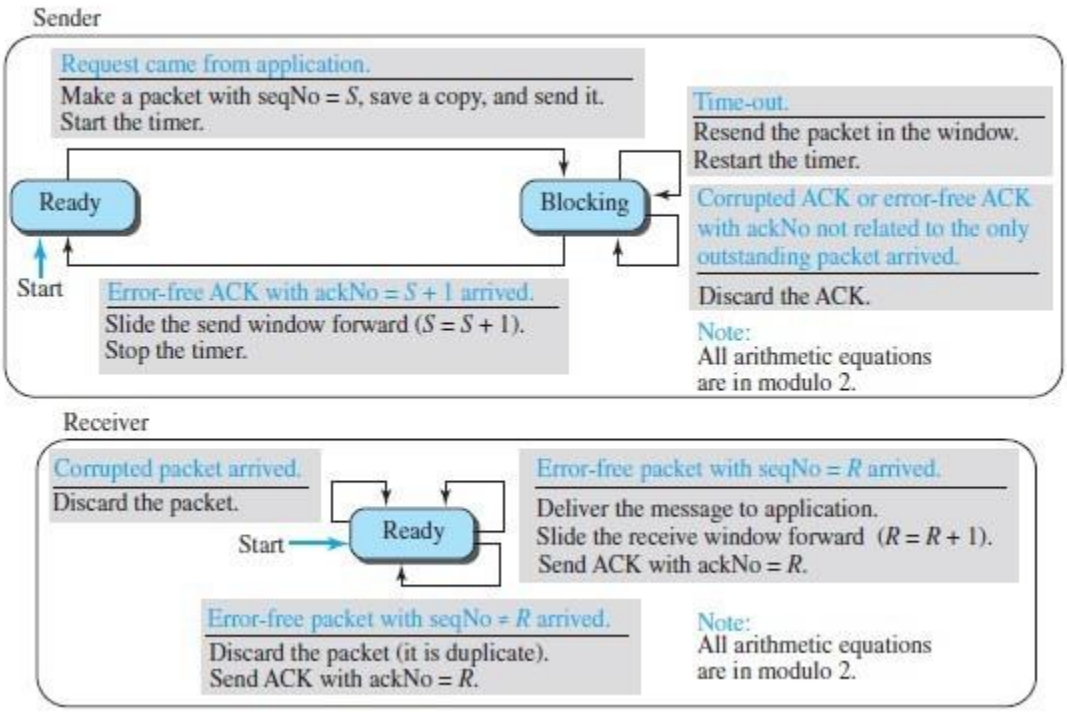


Fig: FSMs for the Stop – and – wait protocol.

Receiver

- The receiver is always in the *ready* state. Three events may occur:

a. If an error-free packet with seqNo = R arrives, the message in the packet is delivered to the application layer. The window then slides, $R = (R + 1)$ modulo 2. Finally an ACK with ackNo = R is sent.

b. If an error-free packet with seqNo $\neq R$ arrives, the packet is discarded, but an ACK with ackNo = R is sent.

c. If a corrupted packet arrives, the packet is discarded.

Efficiency:

- The Stop-and-Wait protocol is very inefficient if our channel is *thick* and *long*. By *thick,* we mean that our channel has a large bandwidth (high data rate); by *long,* we mean the round-trip delay is long.

TCP CONNECTION

- **Connection Establishment:**
- **Three Way Handshaking:**
- The connection establishment in TCP is called three way handshaking. The process starts with the server. The server program tells its TCP that it is ready to accept a c nnection. This is called a request for a passive open.
- The client program issues a request for an active open. A client that wishes to connect to particular server.

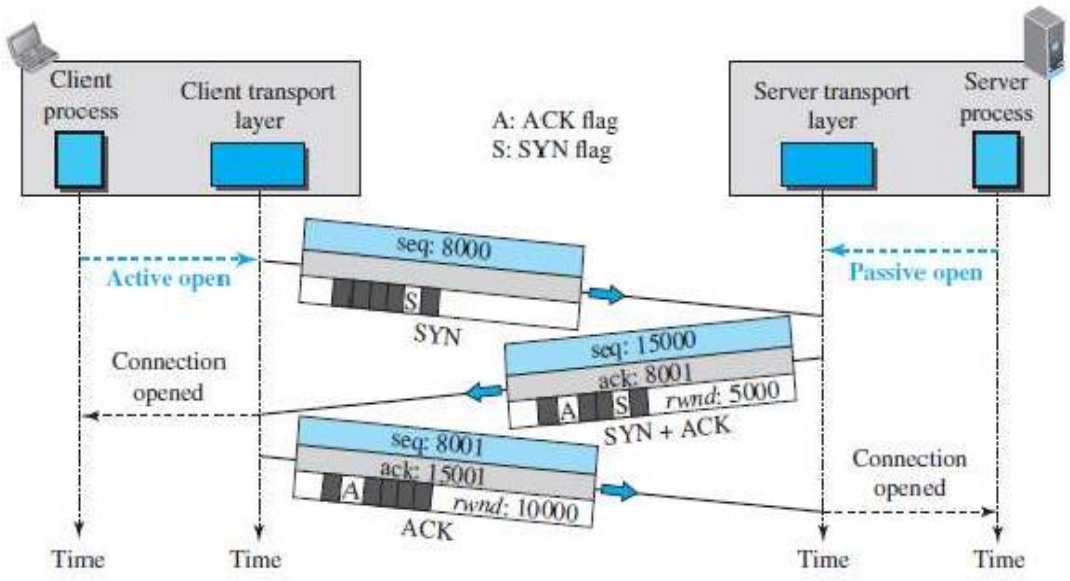


Fig: Connectionestablishment using three-way handshaking

The three steps in this phase are as follows.

- The client sends the first segment, a SYN segment,A SYN segment cannot carry data, but it consumes one sequence number.
- The server sends the second segment, a SYN +ACK segment, with 2 flag bits set: SYN and ACK. A SYN +ACK segment cannot carry data, but does consume one sequence number.
- The client sends the third segment. This is just an ACK segment. An ACK segment, if carrying no data, consumes no sequence number.

Data Transfer:

- After connection is established, bidirectional data transfer can take place. The clientand server can both send data and acknowledgments.

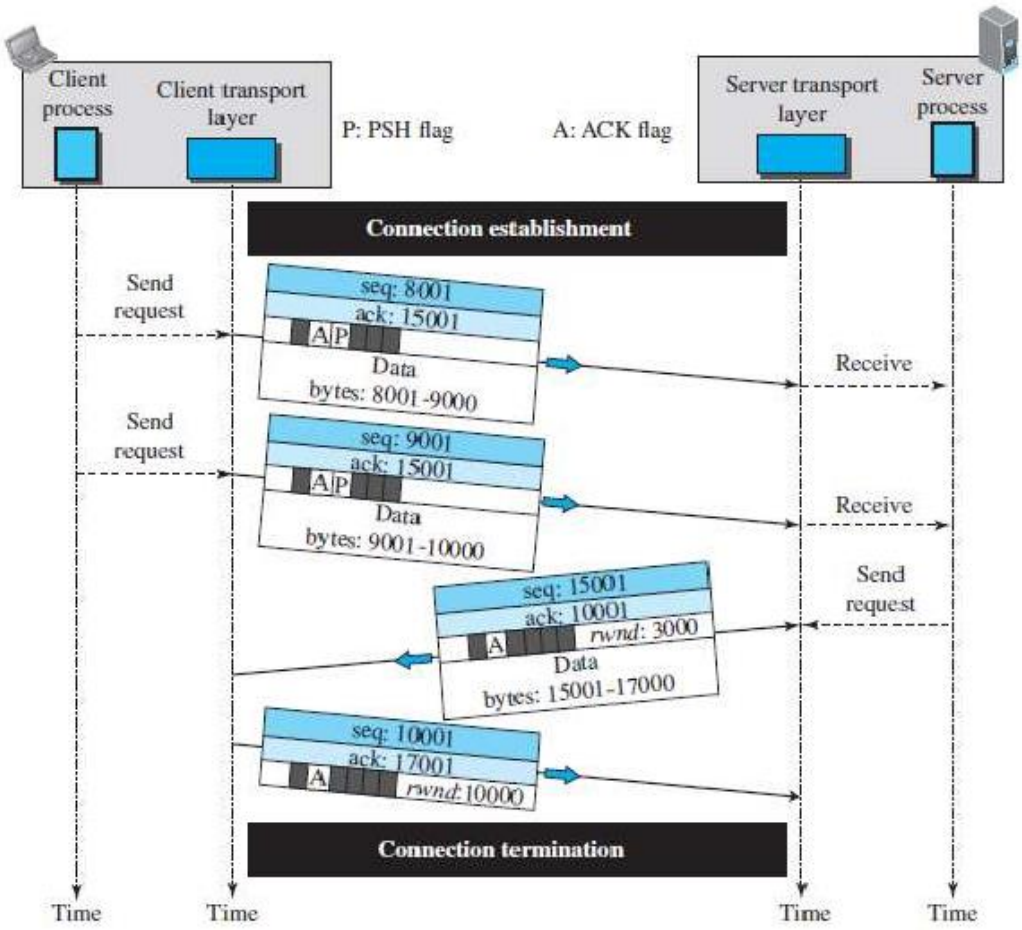


Fig: Data transfer.

Connection Termination:

- Two options for connection termination: three-way handshaking and four-way handshaking with a half-close option.

Three-Way Handshaking:

1. Sends the first segment, a FIN segment.
2. The server TCP, after receiving the FIN segment, informs its process of the situation and sends the second segment, a FIN +ACK segment.
3. The client TCP sends the last segment, an ACK segment, to confirm the receipt ofthe FIN segment from the TCP server.

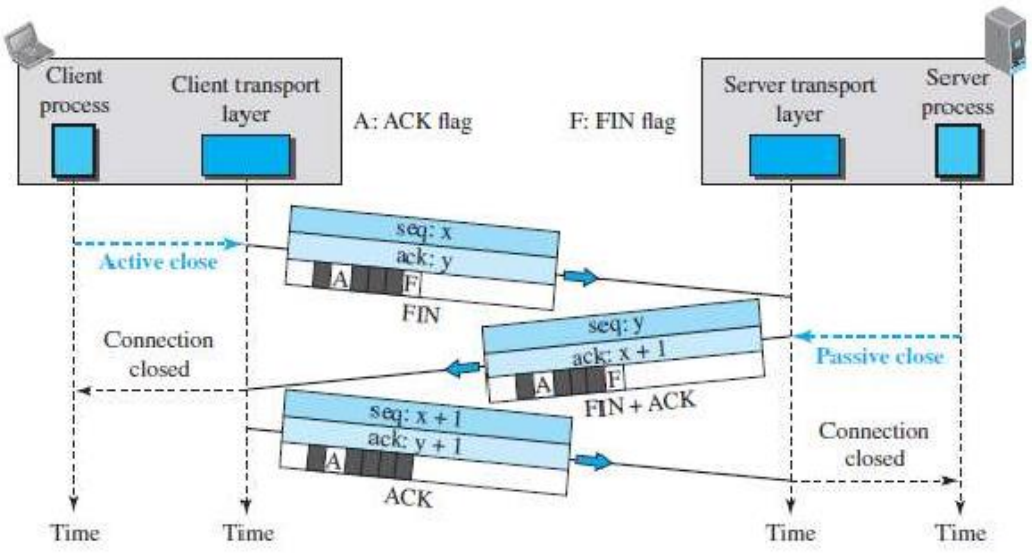


Fig: Connection termination using three-way handshaking.

- **Half-Close:**
  - In TCP, one end can stop sending data while still receiving data. This is called a half-close.

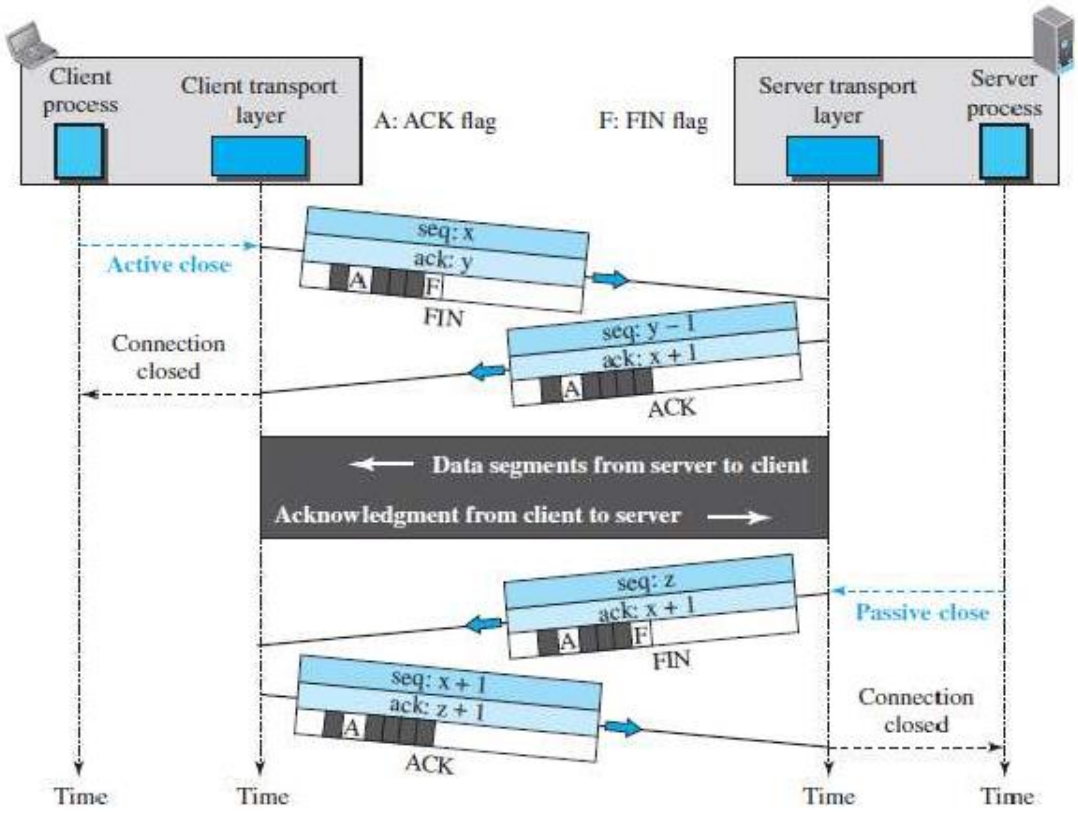


Fig: Half-close.

TRANSMISSION CONTROL PROTOCOL TCP)

The second transport layer protocol is called Transmission Control Protocol (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. TCP is called a connection-oriented, reliable transport protocol.

TCP Services:

Process-to-Process Communication:

- TCP provides process-to-process communication using port numbers.

Table: 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	Quoto	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	Tenninal 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

Stream Delivery Service:

- TCP is a stream-oriented protocol. It allows the sending process to deliver data as a stream ofbytes and allows the receiving process to obtain data as a stream of bytes. The sending

process produces (writes to) the stream of bytes, and thereceiving process consumes (reads from) them.

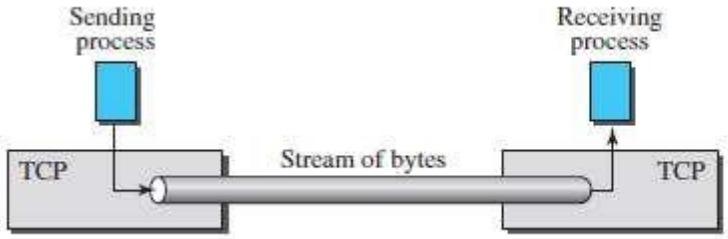


Fig: Stream delivery.

- Sending and Receiving Buffers Because the sending and the receiving processes maynot write or read data at the same speed, TCP needs buffers for storage. buffers are also necessary for flow and error control mechanisms.

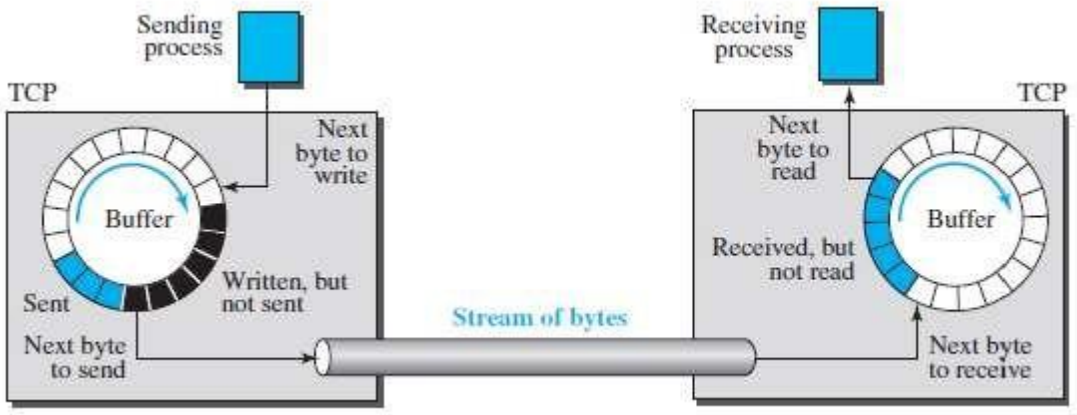


Fig: Sending and receiving buffers.

- 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.
- At the receiver site is simpler. 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.

Segments:

- 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.

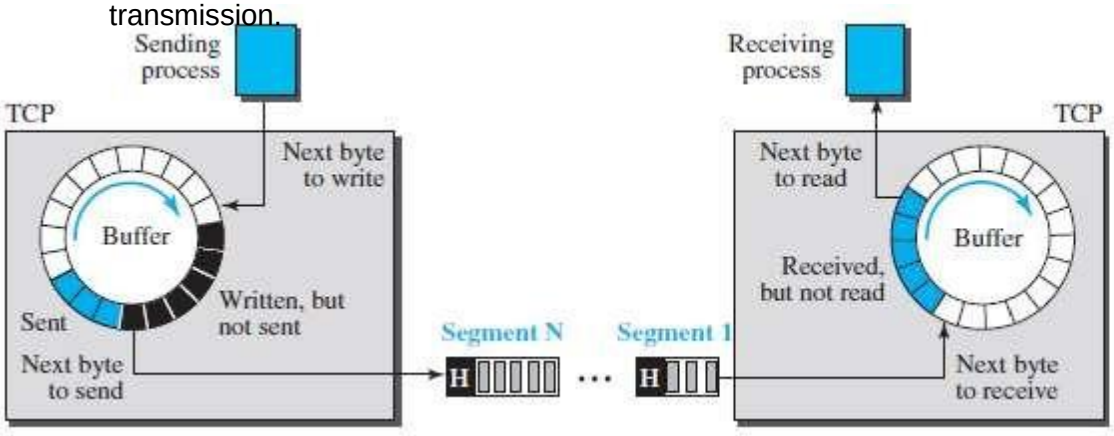


Fig: TCP segments.

Full-Duplex Communication:

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

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.

TCP Features:

Numbering System:

- There are two fields called the sequence number and the acknowledgment number.

Byte Number:

- 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.

Acknowledgment Number:

- 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, unlike UDP, provides flow control. This is done to prevent the receiver from being over whelmed with data.

Error Control

- To provide reliable service, TCP implements an error control mechanism.

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.

Segment:

- A packet in TCP is called a segment.

Format:

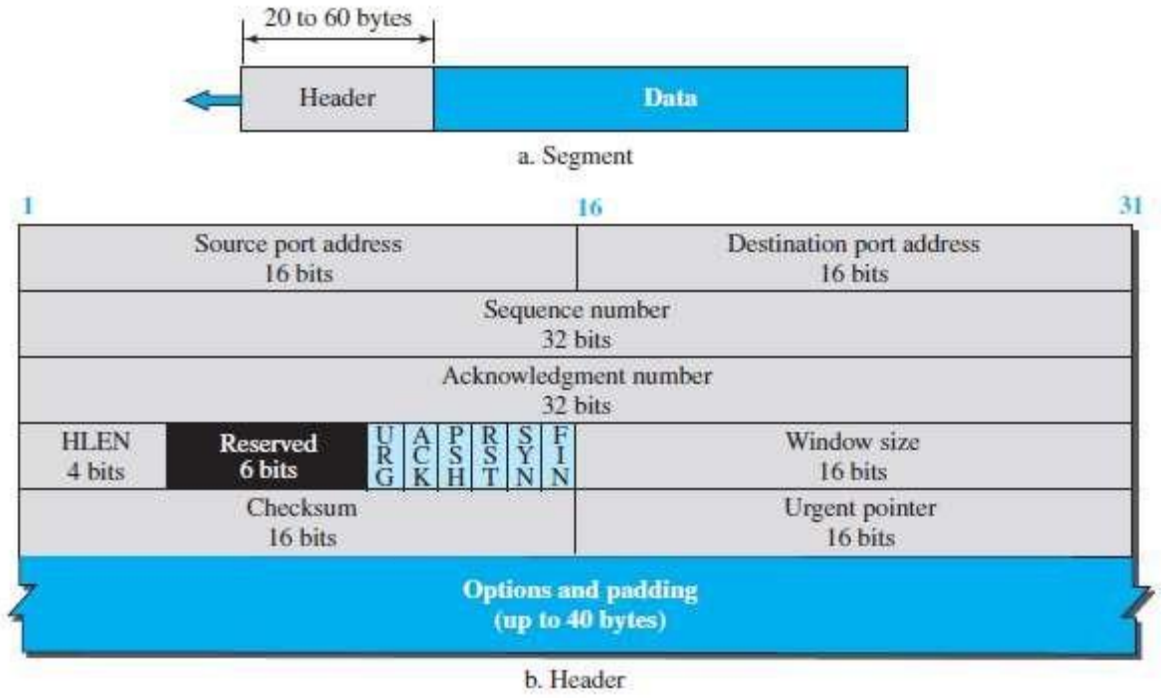


Fig: TCP segment format.

- **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.
- **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.
- **Sequence address:**
  This 32-bit field defines the number assigned to the first byte of data contained in this 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.
- **Reserved:**
  This is a 6-bit field reserved for future use.
- **Control:**

This field defines 6 different control bits or flags.

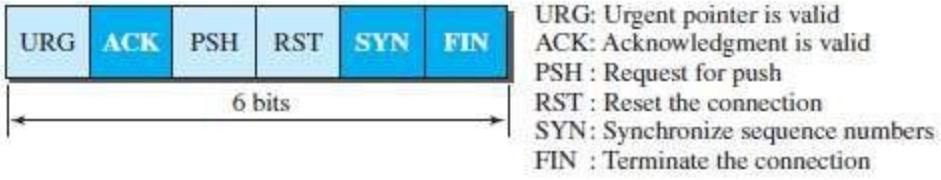


Fig: Control field.

Table: 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.

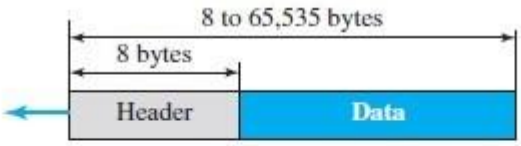
USER DATAGRAM PROTOCOL

- The **User Datagram Protocol (UDP)** is a connectionless, unreliable transport protocol. 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 using UDP takes much less interaction between the sender and receiver than using TCP.

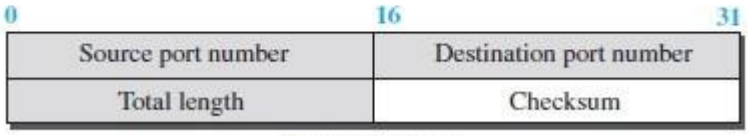
User Datagram:

- UDP packets, called ***user datagrams,*** have a fixed-size header of 8 bytes made of four fields, each of 2 bytes (16 bits). The first two fields define the source and destination port numbers. The third field defines the total length of the user datagram, header plus data.

Fig: User datagram packet format



a. UDP user datagram



b. Header format

Example:

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

CB84000D001C001C

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

Solution

a. The source port number is the first four hexadecimal digits (CB84)16, which means that the source port number is 52100.
b. The destination port number is the second four hexadecimal digits (000D)16, which

means that the destination port number is 13.

c. The third four hexadecimal digits (001C)16 define the length of the whole UDP packet as 28 bytes.

d. The length of the data is the length of the whole packet minus the length of the header, or 28 −8 20 bytes.

e. Since the destination port number is 13 (well-known port), the packet is from the client to the server.

f. The client process is the Daytime (see Table1).

UDP Services:

Process-to-Process Communication:

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

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

Flow Control:

- UDP is a very simple protocol. There is no flow control, and hence no window mechanism. The receiver may overflow with incoming messages.

Error Control:

- There is no error control mechanism in UDP except for the checksum. This means that the sender does not know if a message has been lost or duplicated.
- When the receiver detects an error through the checksum, the user datagram is silently discarded.

Checksum:

- UDP checksum calculation includes three sections: a pseudo header, the UDP header, and the data coming from the application layer. The pseudo header is the part of the header of the IP packet in which the user datagram is to be encapsulated with some fields filled with 0s.
- If the checksum does not include the pseudo header, a user datagram may arrive safe and sound. However, if the IP header is corrupted, it may be delivered to the wrong host.