Contents

Unit 6
The Transmission Control Protocol

Structure of the Unit

6.1 Unit Outcomes

After the successful completion of this unit, the student will be able to:

- Identify various connection-oriented transfer protocols.
- Describe the TCP segment header.

6.2 Introduction

In networking, connection-oriented communication is a method or protocol where a semi-permanent connection or communication session is established between the sending and receiving devices before the data is transferred between them. This protocol also ensures that all the packets of a message are delivered to the destination in the order in which they were transmitted from the source.

There are two ways of implementing a connection-oriented protocol: Circuit Switched Connection and Packet Mode Virtual Circuit Connection.

The connection-oriented transport layer protocol used for Packet mode virtual circuit connection is the Transmission Control Protocol (TCP). In virtual circuit networks, a virtual connection is set up and a path is defined for the packets before they can be sent. All the packets in virtual circuit networks contain three important components: source and destination addresses to identify the source and destination hosts and a virtual circuit identifier or a label to differentiate between the different virtual

channels/circuits. When a packet arrives at the router, based on this label, a packet is forwarded to an appropriate output link.

In this unit, we look into Reliable Data Transfer as one of the important services provided by the transport layer. The sub-topics discussed in the following sections are,

- Principles on which reliable data transfer protocols are built
- Different reliable data transfer protocols
- TCP services and its header format

6.3 Principles of Reliable Data Transfer

Reliable Data Transfer is one of the important design issues in networking and the problem of implementing it occurs at three different layers: the Link layer, the Transport layer, and the Application layer. The connection-oriented services are most of the time reliable or sometimes unreliable. In the case of reliable service, an acknowledgment will be sent to the sender by the receiver as positive feedback after successful delivery of the packets or else there is a request for retransmission. There are many versions of reliable data transfer protocols developed for the transport layer. In the following sections the Stop-and-wait, Go-back N, and Selective Repeat versions are studied. But before the specific versions are discussed, the general principles of Reliable Data Transfer are studied.

Fig. 6.1a (left side portion of the figure) shows the service model and its implementation model for reliable data transfer, where the transport layer is shown to act as a reliable channel to the upper layer or application layer entities. With a reliable channel, all the data bits are delivered to the application layer in order, and without being corrupted or lost. This is the service model offered to the Internet by the Transport layer and implemented through the TCP.

Fig. 6.1b (right side portion of the figure) shows the implementation of the reliable service between two endpoints. This implementation is difficult as the layers below it may or may not be reliable. Also, the layer below the two reliably communicating computers may consist of a single physical link (if link layer protocol is considered) or a global internet (if the transport layer protocol is considered). For developing the reliable transport layer protocol, the layer below the transport layer is assumed to be one unreliable point-to-point channel and it is assumed that one or more packets may be lost, they may be out of order and the layers below the transport layer will not reorder them. The transport layer reorders them and passes them to the application process in the same order as they are sent by the sender.

The four interfaces for the transfer protocol are shown in Fig. 6.1b. Here the sending side of the reliable transfer protocol is invoked by calling rdt_send() along with the data to be sent. The transfer protocol packetizes the data and passes it to the unreliable channel by calling udt_send(). The receiver side of the protocol is invoked by calling rdt_rcv() along with the data. At the receiver, the deliver_data() delivers the data to the application layer. The figure shows unidirectional data transfer for simplicity in understanding. But data transfer can be bidirectional also. Here rdt means reliable data transfer and udt

means unreliable data transfer.

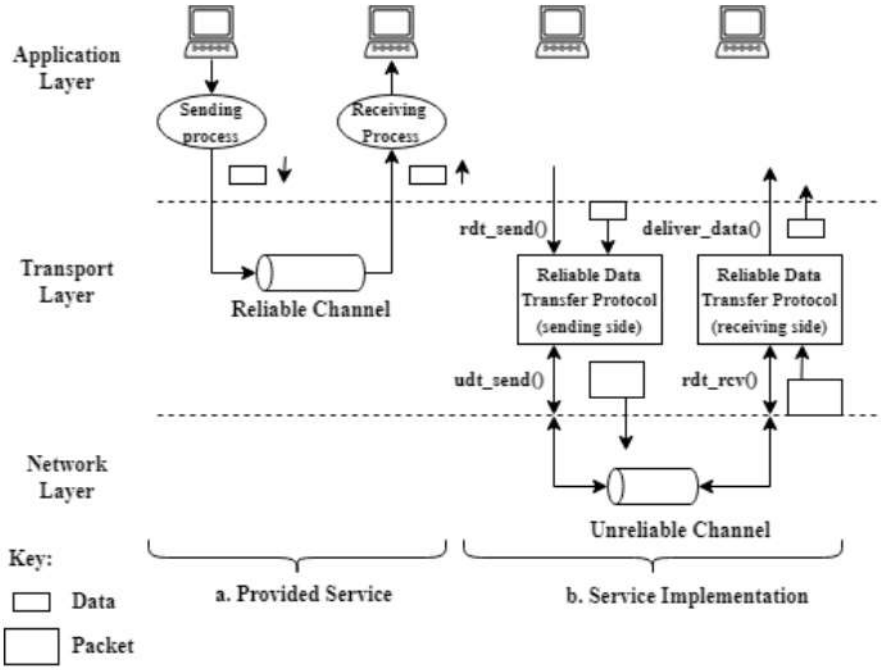


Fig. 6.1: Reliable Data Transfer: Service model and service implementation

6.3.1 Finite State Machine (FSM) for building the reliable transfer protocols

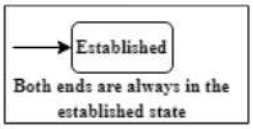
The behavior of the connectionless and connection-oriented protocols can be better represented by an FSM. Figures 6.2a and 6.2b show a representation of the connectionless and connection-oriented transport layer using FSMs.

Here the transport layers, both at the sender and the receiver are thought of as machines having a finite number of states. Until an event occurs, the machine remains in one of the states. In the FSM, the state of the machine is shown in rounded-corner rectangular blocks and the events along the action taken by the machine are shown with a horizontal line separating them besides the state. The absence of an event or an action is represented by a ^ symbol. The machine should start with an initial state at the beginning when the machine is turned on.

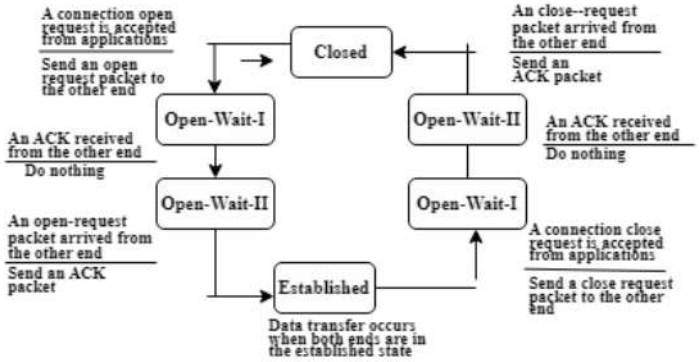
Fig. 6.2a shows the FSM of a connectionless transport protocol with only one state that is the established state. It shows the sender and the receiver in the established state when they are always ready to send and receive the transport layer packets.

Fig. 6.2b shows the FSM of a connection-oriented transport protocol where the machine reaches the established state after going through three states and three more states to close the connection. The closed state of the machine is when there is no connection and there is a change from this state when there is a request for opening the connection by an application process. All the events and the actions along with the states are shown in the figure.

There are different versions of the connection-oriented FSM which will be discussed in the following sections.



a. FSM for Connectionless transport layer



b. FSM for Connection-oriented transport layer

Fig. 6.2: Connectionless and connection-oriented service represented on FSMs

6.4 Building a Reliable Data Transfer Protocol

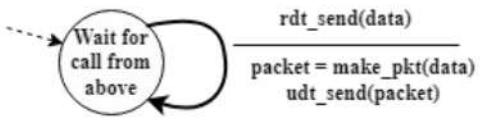
A reliable transfer protocol is built in stages starting from a simple version for a perfectly reliable channel with no bit errors to a complex version for a lossy channel with bit errors.

6.4.1 Reliable Data Transfer Protocol over a Perfectly Reliable Channel: rdt1.0

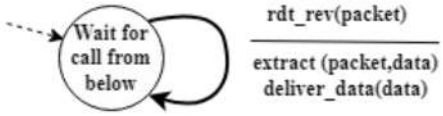
In the first version (rdt1.0) of the protocol, the underlying channel or the network layer is assumed to be completely reliable and free of errors. The FSM for rdt1.0 is shown in Fig. 6.3. It has one FSM for the sender (Fig. 6.3a) and another for the receiver (Fig. 6.3b). The FSMs have only one state and to indicate transitions in the states, curved bold arrows have been used. Also, the initial state of the protocol is indicated by a dashed arrow.

On the sending side, an event is triggered when an application invokes a procedure call through rdt_send() and passes the data along with it. On the sending side, a packet is created by the protocol with the data (make_pkt(data)) and sent on the channel. The FSM is shown in Fig. 6.3a.

On the receiving side, an event is triggered when the network layer invokes a procedure call through rdt_rcv(packet) and passes the packet. On the receiving side, the protocol extracts the data from the packet with extract (packet, data) and passes it to the application through the deliver_data(data). The FSM is shown in Fig. 6.3b.



a: rdt1.0 sending side



b: rdt1.0 receiving side

Fig. 6.3: rdt1.0 - A protocol for a completely reliable channel

The unit of data here is the packet and as this protocol assumes the underlying channel to be completely reliable, no feedback is sent to the sender from the receiver. Also, the receiver receives the data as soon as the sender sends it so, no flow control also is needed.

6.4.2 Reliable Data Transfer Protocol over a Channel with bit errors: rdt2.0

The second version of the protocol is more realistic as it considers the actual scenario of data transmission where there can be bit errors in a packet while being transmitted. Bits in a packet may get corrupted while propagating on a channel, or when they are buffered. So, this version of the protocol is used for error and flow control. To implement a reliable transmission protocol in such situations, the sender sends a packet and waits for feedback to be received within a predefined time from the receiver in the form of an acknowledgment packet.

The implementation can contain two types of acknowledgments: a positive acknowledgment for the correct reception and a negative acknowledgment for erroneous reception. This helps the sender to send the next packet only after a positive acknowledgment is received or repeat the transmission of the previous packet in case of no acknowledgment or a negative acknowledgment in case of erroneous transmission. As this protocol implementation uses retransmissions for reliability, this version of the protocol is known as the Automatic Repeat Request (ARQ) protocol. For retransmission to be made possible, the sender stores a copy of the packets until error-free packets are received. Also, all the received packets are checked for errors by using the checksum bits which are added to every packet by the sender.
Because of the behavior of the protocol, it is called as Stop and Wait protocol.

Fig. 6.4 shows the FSM for rdt2.0 reliable data transfer protocol, which uses packet checksum for error detection, a positive acknowledgment in case of no errors, and a negative acknowledgment in case of erroneous reception. Fig. 6.4a shows the sending side and Fig. 6.4b the receiving side of rdt2.0.

There are two states on the sending. In the first state shown on the left, at the rdt_send(data) event, a packet is created with the data, and a checksum is added using sndpkt = make_pkt(data, checksum). This packet is then sent using the udt_send(sndpkt) function. In the second state, the sender transmits a packet and waits for an ACK or NAK. On the reception of a positive acknowledgment (ACK packet)

from the receiver, the sender moves back to the first state and waits there for an event to occur. If a NAK packet is received, the sender retransmits the previous packet and waits for the ACK packet to be received.

The receiving side has only one state. The receiver sends a reply to the sender after a packet arrives. Based on the error-free or erroneous reception, the reply will be an ACK or an NAK packet.

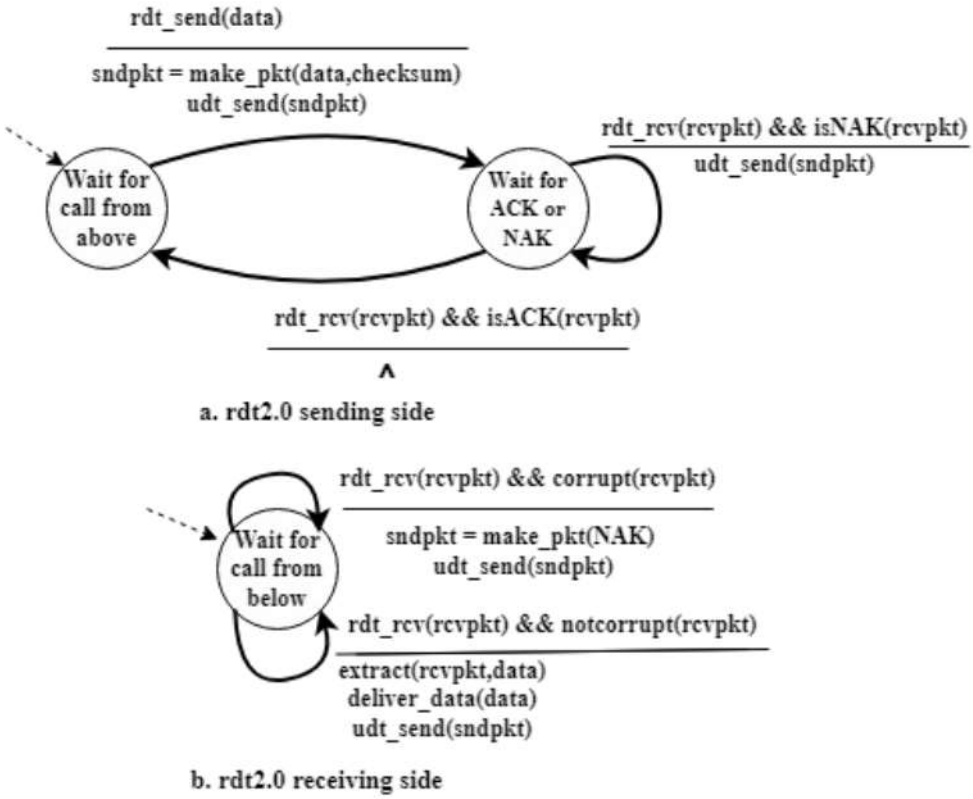


Fig. 6.4: rdt2.0-A protocol for a channel with bit errors

The limitation of this protocol is that if the ACK or NAK packet gets corrupted, the sender cannot understand if the receiver has received the correct data packet. To solve this issue, checksum bits can be added to the ACK/NAK packets, so that the sender can detect errors in these packets or simply resend the previous packet again. Resending packets can result in unnecessary duplication of packets. Duplication also confuses the receiver as it cannot know if the ACK or NAK that it sent previously was received correctly or not and also if the packet it is receiving now is new data or a retransmission. To overcome this problem, a new field is added to the packets by the sender which is a sequence number. The receiver checks this number to identify if any packet is a retransmission. Sequence numbers and their uses are discussed in the next section

6.4.2.1 Sequence Numbers and Acknowledgment Numbers

To identify duplicate packets and avoid them, a sequence number and an acknowledgment number are added to every packet. A sequence number is added to the header of every packet and an acknowledgment number is added by the receiver. This number is the sequence number of the next packet which the receiver expects from the sender. It is in modulo-2 arithmetic.

Eg: If a packet with sequence number x is sent by a sender, then x+1 will be the acknowledgment number sent by the receiver for that packet.
There are three situations occurring when a packet is sent:

1. A packet with sequence number x can be received without any error and an acknowledgment with number x + 1 can be sent back by the receiver as it expects a packet with that sequence number next.
2. The packet can get lost or corrupted, and no acknowledgment is received. In this situation, the sender resends the same packet again after waiting for a predefined time interval or timer time-out.
3. The packet may be received correctly, but the acknowledgment packet may get lost or corrupted, and the sender will have to resend the previous packet with the same sequence number x after the timer time-out interval. This number tells the receiver that it is a duplicate packet and that the acknowledgment packet may have been corrupted or lost. So it sends the acknowledgment number x+1 again but discards the packet.

NOTE: For a lossless channel, the ACK and NAK packets need not send the sequence number of the packets they are acknowledging, as the sender assumes the acknowledgments are always for the recently transmitted.

6.4.2.2 rdt2.1

This is the next version of the rdt2.0 protocol. Here the sender and receiver have twice the states of that of rdt2.0.
rdt2.1 uses both positive and negative acknowledgments. A positive acknowledgment is sent when out-of-order packets are received, and a negative acknowledgment is received when a corrupted packet is received.

6.4.2.3 rdt2.2

In this protocol version, the method of sending an acknowledgment is changed. Instead of using two different acknowledgments, only one type of acknowledgment is used, but the sequence number in the ACK packet indicates whether the packet was received without error or not. Here instead of sending an NAK, an ACK for the previous packet is sent using the sequence number of the previous packet. Thus, the sender receives duplicate ACKs for the same packet indicating the corruption of the packet sent recently. The receiver must now check the sequence number of the acknowledging packet before transmitting a packet.

6.4.3 Reliable Data Transfer Protocol over a Lossy Channel with Bit Errors: rdt3.0

The channel used for transmission can corrupt the bits and also result in packet losses. Two important design issues of the reliable protocol are: detecting packet losses and their recovery. The previous versions of the protocols have only taken care of the bit corruptions in the packet and ways to recover from them by using checksum bits, using sequence numbers for the data packets, and the acknowledgment packets and retransmission. Most of the burden of detecting errors and recovery is on the receiver side. So, the rdt3.0 version of the protocol uses an approach from the sender side to deal with the packet losses.

When a data packet gets lost or its ACK gets lost, the sender waits for a predefined time and then retransmits the packet again. For this purpose, the sender uses a count-down timer which is started when a packet is transmitted or retransmitted and interrupts the sender after the time-out. It is reset again during a new transmission. The design of the time-out of the timer is a major issue. The time-out value should be at least equal to the sum of the round-trip delay between the transmitter and the receiver and the processing time of the packet at the receiver. The worst-case maximum delay is difficult to estimate and for the protocol to recover from the losses, the time-out should be chosen judiciously. A packet is retransmitted if an ACK for it is not received within the time-out. Packet retransmission can occur even in the cases of a packet suffering a transmission delay in the network other than the acknowledgment getting lost.

6.5 Stop-and-Wait Protocol

In this protocol, at any time instance, there is one data packet and one acknowledgment packet.

Fig. 6.5 shows the working of the stop-and-wait protocol.

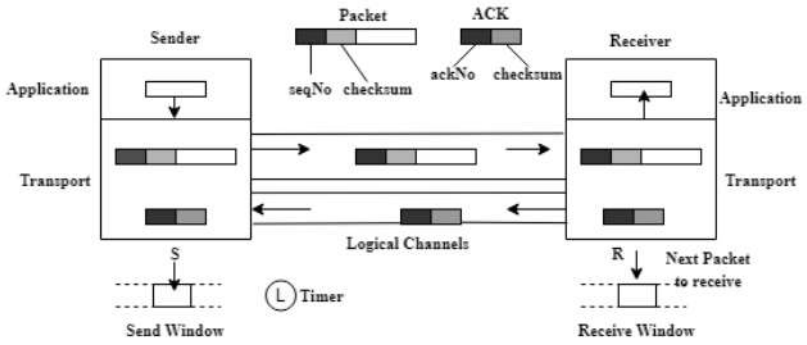


Fig. 6.5: Stop-and-wait protocol

Also, the sliding window size at the sender and the receiver is 1. The sliding window is a method for controlling the flow of data packets between two devices where a reliable and gradual flow of the packets is needed. In the Figure, the sender and the receiver have control variables S and R respectively, which point to the send and the receive windows.

6.5.1 FSM of the Stop-and-wait Protocol

The **sender** has two states: Ready and Blocking

Before any data can be transferred between the sender and the receiver there should be a connection established between them.

The initial state of the Sender is ready state. It keeps transitioning between the ready and blocking states and the initial value of the variable S is set to zero.

Ready State:

In this state, the sender waits for an event to occur. When there is a request coming from an application, a packet with seqNo = S is created, a copy of it is saved, sent to the receiver, the timer is started and then the sender goes into the blocking state.

Blocking State:

Here the sender can respond to the events in the following three ways:

1. The sender can move to the ready state if an ACK packet with an acknowledgment number the next expected packet arrives. The timer is stopped, and the window slides to (S+1) modulo 2 value.
2. The sender can stay in the same state and the packet will be discarded if a packet with a corrupted ACK or an error-free ACK with ackNo ≠ (S+1) modulo 2 arrives.
3. If no ACK is received and the timer times out, the sender retransmits the previous packet and the timer is restarted.

The **receiver** has one state: Ready state

Here the receiver can respond to the events in the following three ways but stays in the same state:

1. The message is extracted from an error-free whose seqNo = R and is given to the specific application process. Also, after sending an ACK packet with ackNo = R the window slides to (R+1) modulo 2 value.
2. A packet arriving with a seqNo ≠ R is discarded as it is a duplicate packet but an ACK with ackNo = R is sent.
3. A corrupted packet also gets discarded.

All the events and actions are illustrated in the FSM as in Fig. 6.6a and Fig. 6.6b.

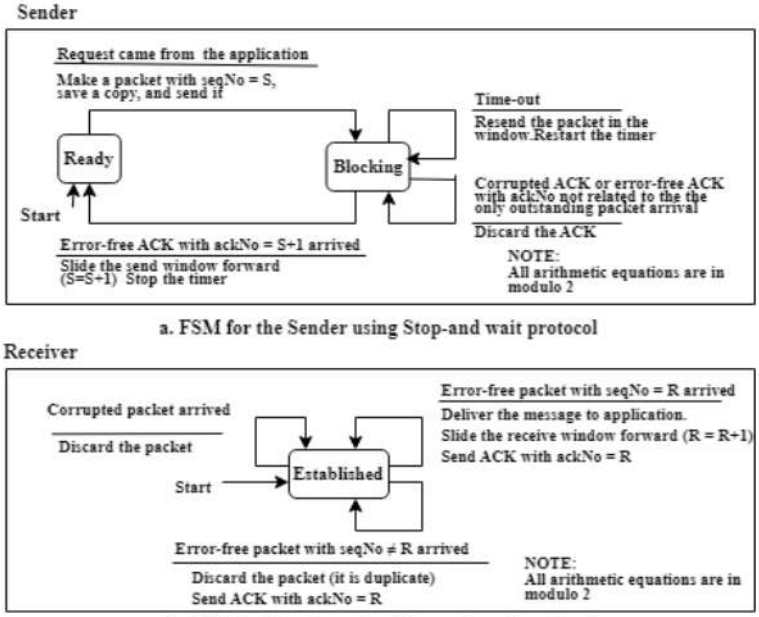


Fig. 6.6: FSMs for the Stop-and-wait protocol

The efficiency of the Stop-and-wait protocol is very low, as the sender has to wait for an acknowledgment to be received before it can send the next packet. So if the bandwidth of the channel is high, it remains idle without data packets in transmission.

6.6 Go-Back-N Protocol

This protocol was developed to improve the efficiency of the Stop-and-wait protocol. Here multiple packets will be transmitted on the channel by the sender while waiting for an acknowledgement. So the channel is busy. Here though there are many packets sent by the sender, only one packet can be buffered by the receiver. The copies of all the packets sent by the sender are saved until the acknowledgments are received. In Fig. 6.7, it can be seen that many data packets as well as acknowledgment packets can be present on the channel at the same time.

6.6.1 Sequence Numbers and Acknowledgement Numbers

If the number of bits in the sequence number field is m, then the packets are given a sequence number which is modulo 2^m, where m is the bits in the sequence number field and the acknowledgment number of a packet is one number more than its sequence number.

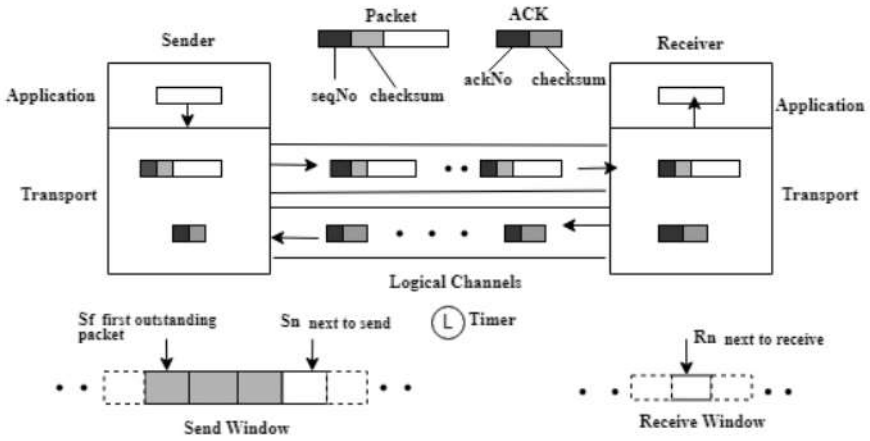


Fig. 6.7: Go-Back-N Protocol

6.6.2 Send Window

This is an imaginary window showing the sequence numbers of the packets that are on the channel and those that are to be sent. The maximum window size is $2^m - 1$, and in this discussion, it is assumed that the window size is set to the maximum value which is fixed here but in other protocols, there may be variable window sizes.

Fig. 6.8 shows a sliding window of size 7 (m =3). The window is an abstraction, divided into four regions: sequence numbers of packets that are already acknowledged, packets sent but not acknowledged, packets ready to be sent, and packets that cannot be sent until the window slides. The three variables S_f, S_n, and S_{size} define the first outstanding packet, the packet to be sent next, and the Send window size respectively.

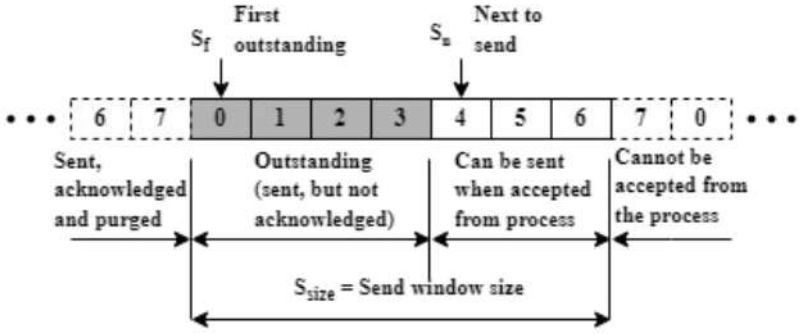


Fig. 6.8: Send-window for Go-Back-N

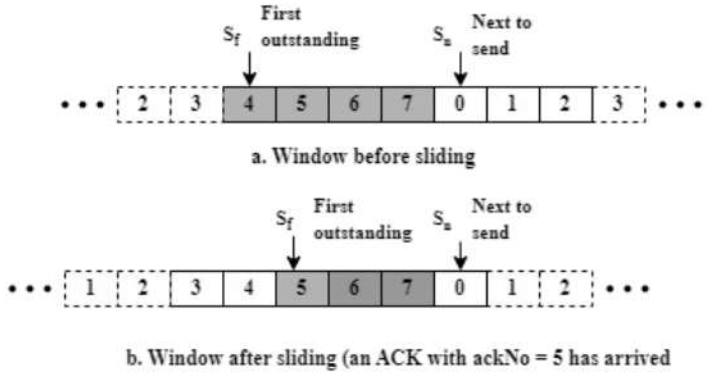


Fig. 6.9: Sliding the send-window for the Go-Back N protocol

The send window can slide when the acknowledgments arrive. Fig. 6.9 shows the sliding of the send window, where the ackNo = 5 has arrived and the receiver is waiting for the packet with sequence number 5.

NOTE: When an error-free ACK packet arrives with an ackNo greater than or equal to S_f and less than S_n (in modulo arithmetic), the sliding window can move one or more slots.

6.6.3 Receive Window

To ensure that the packets are received and acknowledgments are sent correctly, the receive window size is set to 1 in this protocol. Any packet that does not arrive in order is discarded and the sender has to resend it. Fig. 6.10 shows the receive window where only one variable R_n is used to represent the next packet expected. The packets whose sequence numbers are to the left of R_n are received correctly and acknowledged and to its right cannot be received as the window size is 1. So a packet with a matching sequence number with R_n is received and when the receive window slides, it slides one slot at a time.

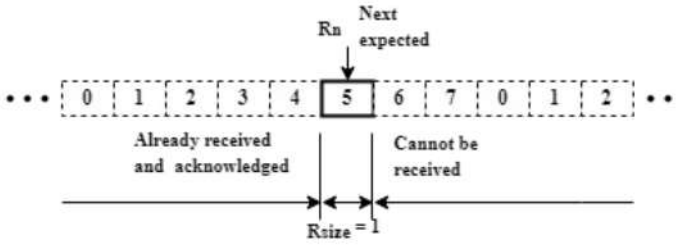


Fig. 6.10: Receive window for Go-Back-N

In this protocol, there is only one timer though there are many packets. When the timer for the first outstanding packet expires, all the outstanding packets are resent.

For Eg: If packet 6 (S_n =7) has already been sent by a sender but the timer expires before packets 3,4,5, and 6 have not been acknowledged (the value of S_f is 3), then the sender goes back and resends these packets again. Because of this, the protocol is called Go-back-N, as the machine goes back to N locations and resends the packets during time-out.

6.6.4 FSM for the Go-back-N protocol

Fig. 6.11 shows the FSM of the Go-back-N protocol.

The sender remains in one of the two states: Ready and Blocking. It is in the Ready state initially. Also, the initial values of the variables S_f and S_n are set to 0.

Ready State:

Here the sender can respond to the events in the following four ways:

1. The sender creates a packet with seqNo = S_n when an application requests it. A copy of this packet is saved first and then sent to the receiver. The timer is started. The value of S_n is incremented by 1 ($S_n = S_n + 1$) modulo 2^m. The sender goes into the blocking state when the window is full or when S_n will be ($S_f + S_{size}$) modulo 2^m.
2. The window slides (S_f = ackNo), whenever an ACK packet arrives with an acknowledgment number related to one of the outstanding packets. The timer is stopped when all the outstanding packets are acknowledged and restarted when all the packets are not acknowledged.
3. An incoming packet is discarded whenever an error-free ACK with ackNo not related to the outstanding packet arrives or a corrupted ACK is received.
4. The sender retransmits all the outstanding packets if a timeout occurs and the timer is restarted.

Blocking State:

Here the sender can respond to the events in the following three ways:

1. The window slides (S_f = ackNo), whenever an ACK packet arrives with an acknowledgment number related to one of the outstanding packets. The timer is stopped when all the outstanding packets are acknowledged and restarted when all the packets are not acknowledged. The sender then moves to the ready state.
2. An incoming packet is discarded whenever an error-free ACK with ackNo not related to the outstanding packet arrives or a corrupted ACK is received.
3. The sender retransmits all the outstanding packets if a timeout occurs and the timer is restarted.

The **receiver** has one state: Ready state

The value of the variable R_n is initialized to zero.

In this state, the receiver responds to the events in one of the following three ways.

1. The required message is extracted from an error-free packet with seqNo = R_n and given to the application process. Also, an ACK packet with ackNo = R_n is sent, and the window slides to (R_n+1) modulo 2^m value.
2. If a packet with seqNo outside the window arrives, the packet is discarded but an ACK with ackNo = R_n is sent.
3. If a corrupted packet arrives, it is discarded.

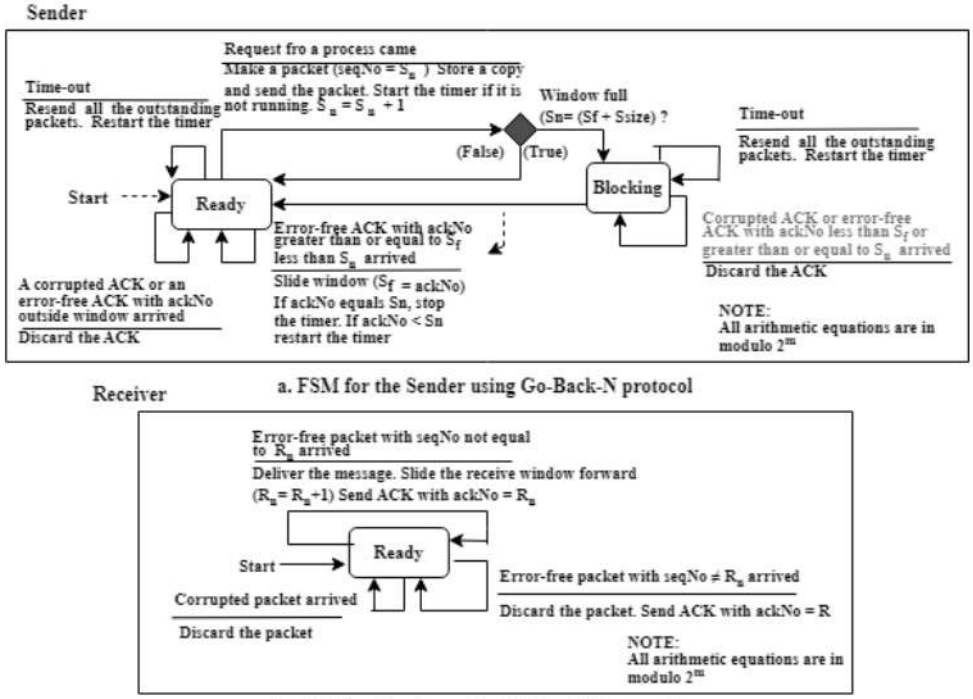


Fig. 6.11: FSMs for the Go-Back-N protocol

6.6.5 Limitations of the Go-Back-N Protocol

The efficiency of this protocol reduces to a very low value when the network layer protocol loses a lot of packets due to congestion. Whenever a packet is lost or corrupted, all the outstanding packets are resent by the sender in spite of some packets being received error-free but out of order. As more packets are resent, there will be more packets on the channel which will worsen the channel congestion. This will in turn lead to more packet drops and increased retransmission and congestion and this can lead to a total collapse in the network.

6.7 Selective-Repeat Protocol

This protocol was developed to improve the efficiency of the Go-Back-N protocol by removing the limitation as discussed in section 6.6.5. This protocol resends only the packets that are lost or resends only selected packets.

6.7.1 Windows

This protocol also uses send and receive windows as in Go-Back-N, but the maximum size of the send window is smaller and is equal to 2^{m-1}. So if the value of m is 4, then the sequence numbers of the packet range from 0 to 15, but the window size is 8. Also, the receive window size is the same as the send window.

Fig. 6.12 shows the outline of the Selective-Repeat protocol and Figures 6.13 and 6.14 show the send and the receive windows in the Selective-Repeat protocol respectively.

All the packets in the send window can arrive at the receive window out of order. The packets will be

stored there until a set of consecutive packets arrive to be passed to the application layer. Figure 6.12 shows the slots inside the receive window, where packets have arrived out of order. These packets will be waiting for the earlier transmitted packets to arrive and then delivered to the application layer.

In this protocol, there is one timer used for each outstanding packet, and when a timer expires, only its associated packet is resent. But most of the protocols used today use only one timer. Also, in this protocol, the ackNo defines the sequence number of a single packet that has been received correctly.

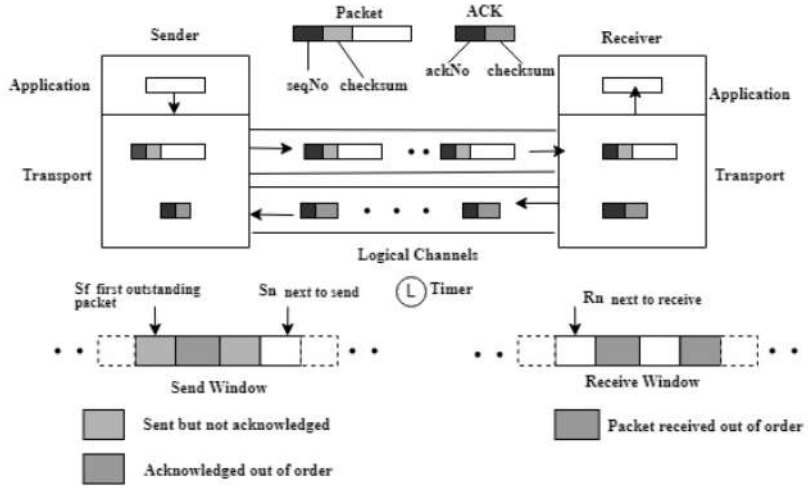


Fig. 6.12: Selective-Repeat Protocol

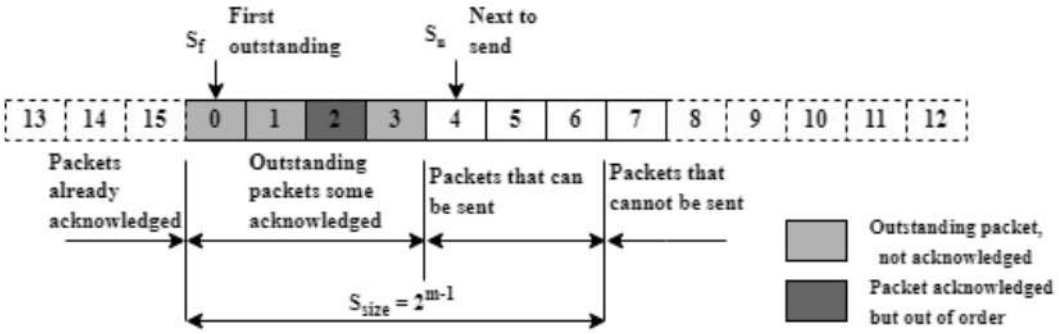


Fig. 6.13: Send-window for Selective-Repeat protocol

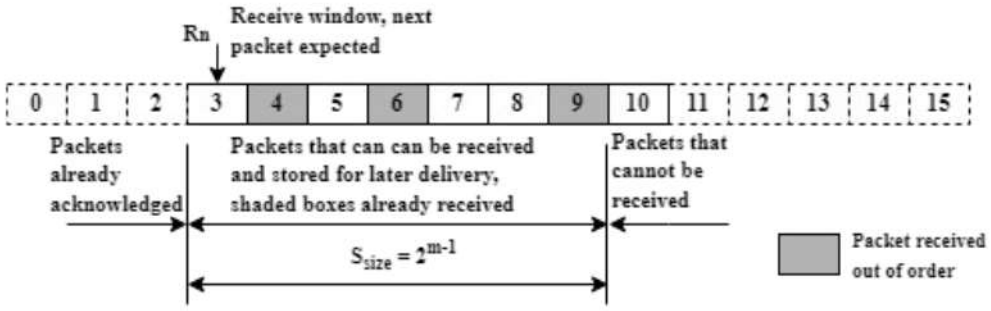


Fig. 6.14: Receive window for Selective-Repeat protocol

6.7.2 FSM for the Selective-Repeat protocol

Fig. 6.15 shows the FSM for the Selective-Repeat protocol.

The sender remains in one of the two states: Ready and Blocking, but initially in the Ready state.

Ready State:

Here the sender can respond to the events in the following four ways:

1. The sender creates a packet with seqNo = S_n when an application requests it. A copy of this packet is saved first and then sent to the receiver. The timer is started. The value of S_n is incremented by 1 ($S_n = S_n + 1$) modulo 2^m. The sender goes into the blocking state when the window is full or when S_n will be ($S_f + S_{size}$) modulo 2^m.
2. The window slides (S_f = ackNo), whenever an ACK packet arrives with an acknowledgment number related to one of the outstanding packets. The timer is stopped when all the outstanding packets are acknowledged and restarted when all the packets are not acknowledged.
3. An incoming packet is discarded whenever an error-free ACK with ackNo not related to the outstanding packet arrives or a corrupted ACK is received.
4. The sender retransmits all the unacknowledged packets if a timeout occurs and the timer is restarted.

Blocking State:

Here the sender can respond to the events in the following three ways:

1. If an error-free ACK with the acknowledgment number related to one of the outstanding packets arrives, the packet is marked as acknowledged. Also, the window slides to the right, if S_f = ackNo. till S_f points to the first unacknowledged packet. The sender moves to the ready state after the window slides.
2. The incoming ACK is discarded if a corrupted ACK or an error-free ACK with ackNo not related to the outstanding packet arrives.
3. The sender retransmits all the unacknowledged packets in the window if a timeout occurs and the timer is restarted.

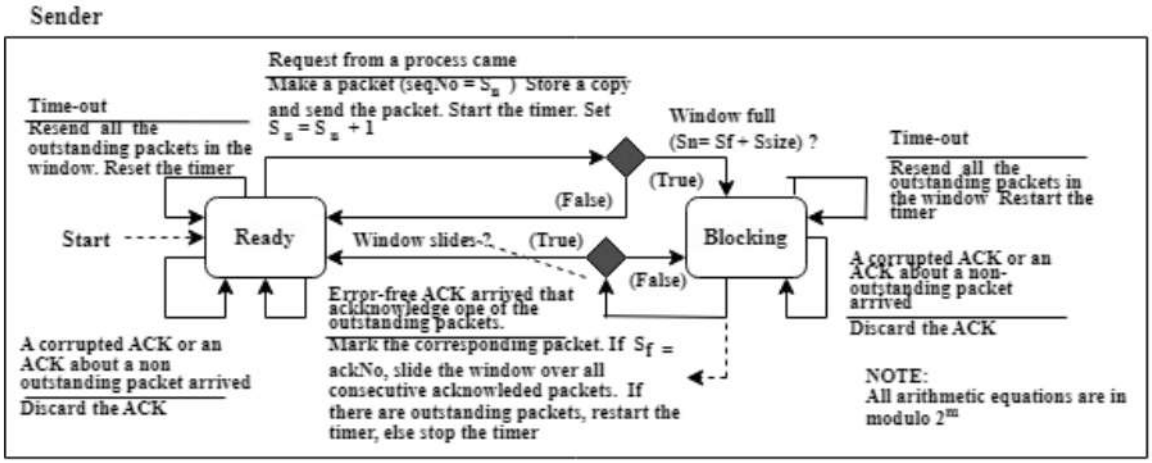
Receiver:

The **receiver** has one state: Ready state

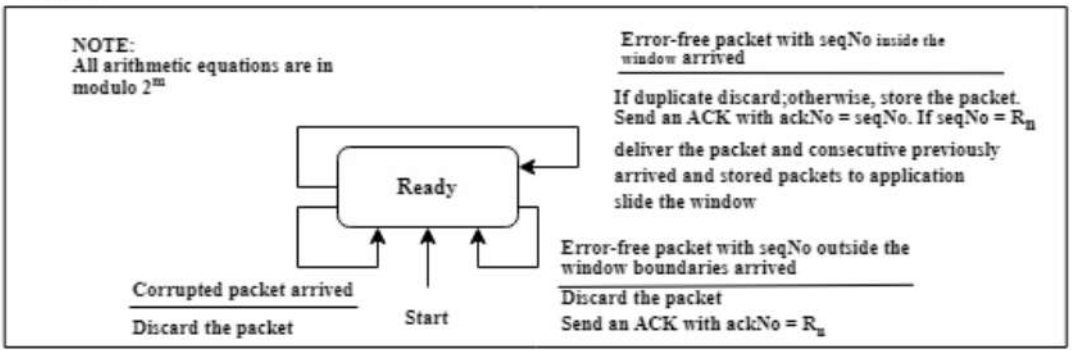
The initial value of the variable R_n is set to zero.

Here the receiver can respond to the events in the following three ways:

1. If an error-free packet with seqNo in the window arrives, the packet is stored and an acknowledgment with ackNo = seqNo is sent. If the seqNo= R_n, then this packet and all the previous packets are passed to the application process. The window slides so that R_n points to the first empty slot.
2. If an error-free packet with seqNo outside the window arrives, the packet is discarded but an ACK with ackNo = R_n is sent.
3. If a corrupted packet arrives, it is discarded.



a. FSM for the Sender using Selective-Repeat protocol



b. FSM for the Receiver using Selective-Repeat protocol

Fig. 6.15: FSMs for the Selective-Repeat protocol

6.8 Transmission Control Protocol (TCP) - Introduction

It is the most popular, connection-oriented, reliable transport layer protocol on the Internet. In this section, first, we will discuss the services provided by TCP and then its features.

6.8.1 TCP services

The services offered by TCP to the application layer processes are as below:

1. Provides process-to-process communication using port numbers.
2. Stream Delivery Service

   In TCP, the sending process passes data as a stream of bytes to the receiver process and not as chunks of data as in UDP. So here, two processes look like being connected by an imaginary tube that carries the data across the Internet.
3. Sending and receiving buffers

   As the sending and receiving processes do not write and read data at the same rate, buffers (are memory locations (thousands of bytes)) used to store data until they are used. There are two buffers, one for sending and another for receiving the TCP segments. These buffers help in implementing flow and error control.

4. Segments

   The network layer provides service to the transport layer and it sends data in the form of packets. So TCP groups the data into some number of bytes called segments and then adds a header to it and passes it to the network layer. The TCP segments are encapsulated into an IP datagram and then transmitted. This whole process is transparent to the receiver, but the receiver is capable of handling the out-of-order reception, corruption, and loss of packets.
5. Full duplex communication

   Using TCP, communication can happen in both directions simultaneously using two TCP endpoints with sending and receiving buffers.
6. TCP performs multiplexing and demultiplexing at the sender and the receiver respectively.
7. As TCP offers connection-oriented service, the transmission of data from the source to the destination occurs in three phases: Logical connection establishment, exchange of data, and connection termination.
8. TCP uses an acknowledgment mechanism to check for the error-free reception of data and this provides reliability.

6.8.2 Features of TCP

For providing the services mentioned in the previous section, TCP includes many features. They are discussed below:

1. Numbering system

   There are two fields in the segment header: sequence number and acknowledgment number, which refer to a byte number in a particular TCP connection and not the segment number. Numbering is different in each direction and independent of each other.

   Each byte received from an application process is numbered by the TCP and these numbers do not start from 0., but can lie in the range of 0 to $2^{32} - 1$.
2. Sequence Number

   TCP assigns a sequence number to each segment that will be transmitted after the bytes are numbered.

   The rules defined for assigning the sequence numbers in each direction are given as follows:

   1. The first segment is assigned any random number.
   2. Next segment is given as the sequence number of the previous segment plus the number of bytes in it.
3. Acknowledgement Number

   This is the number of the next byte that the party expects to receive and it is cumulative. It is the number of the last correctly received byte plus one.

6.8.3 TCP Segment Format

The format of the TCP segment is shown in Fig. 6.16a and its header format is shown in Fig.6.16b

The header is 20 bytes without options and up to 60 bytes with options.

1. Source and Destination port address: These are 16-bit port numbers of the application process in the segment sending and receiving hosts respectively.

2. Sequence number: This is a 32-bit number assigned to the first byte of the data in that segment.
3. Acknowledgement Number: It is a 32-bit number and is discussed in the previous section.
4. Header Length (HLEN): It defines the number of 4-byte words in the header of the TCP segment. As the length of the header can be from 20 bytes to 40 bytes, this value can be between 5 and 15. (5 X 4 = 20 and 15 X 4 = 60)
5. Control: There are 6 different control flags to control the transmission in 6 ways. They can be set or reset to enable or disable the different operations and one or more flag bits can be set at a time. The control operations can be flow control, mode of data transfer, connect establishment, and termination. The description of these bits is given in Fig. 6.17.

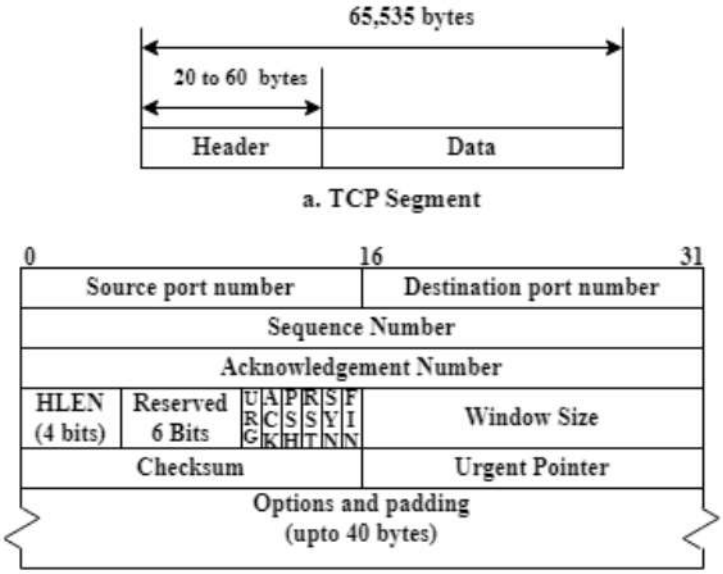


Fig. 6.16: TCP Segment format

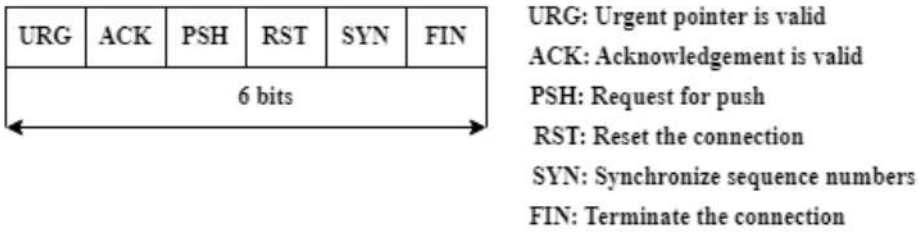


Fig. 6.17: Control field

6. Window size: As the size of this field is 16 bits, the maximum size of the window is 65,535 bytes. This window is determined by the receiver and so is called the receiving window.
7. Checksum: This is a 16-bit field and the procedure used to compute is the same as in UDP. A pseudo-header is added to the segment header similar to UDP. Fig. 6.18 shows a pseudo-

header added to the TCP segment.

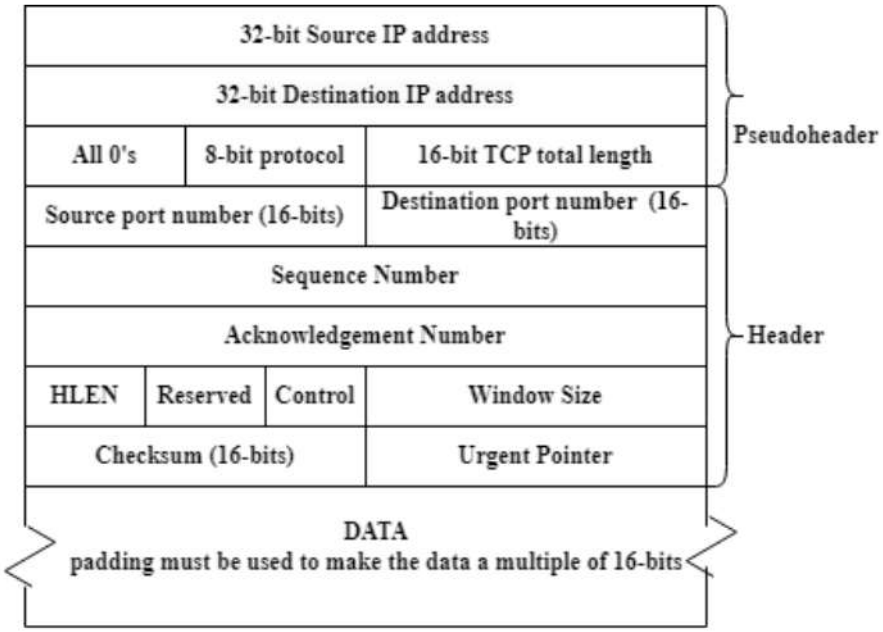


Fig. 6.18: Pseudoheader added to the TCP segment

8. Urgent Pointer: It is a 16-bit value that is considered only if the urgent flag bit is set. Indicates that the segment has to be delivered as fast as possible.
9. Options: Optional information of 40 bytes can be added to the TCP header.

6.8.4 A TCP Connection

As TCP is a connection-oriented transport protocol, a logical connection or path is established between the sender and receiver first and then all the segments of a message are transmitted over this logical path. This single path for transmission helps in receiving acknowledgments, tracking lost frames, and retransmission. Though TCP controls the connection between the sender and the receiver, it uses the network layer services of IP to deliver the segments to the receiver.

There are three phases in a connection-oriented transmission:

1. Connection establishment
2. Data transfer
3. Connection termination

Connection establishment:

TCP transmission uses a full duplex mode and both the sender and the receiver communicate with each other using handshaking signals before transmission.

Three-way handshaking

Let us understand the connection establishment process using a scenario with a client process making a connection with a server process using TCP as shown in Fig. 6.19. Before the connection establishment process can start, the server informs its TCP about its readiness to accept the connection by a request

called passive open. A client wishing to connect to an open server issues a request active open to its TCP. The three-way handshaking will then begin.

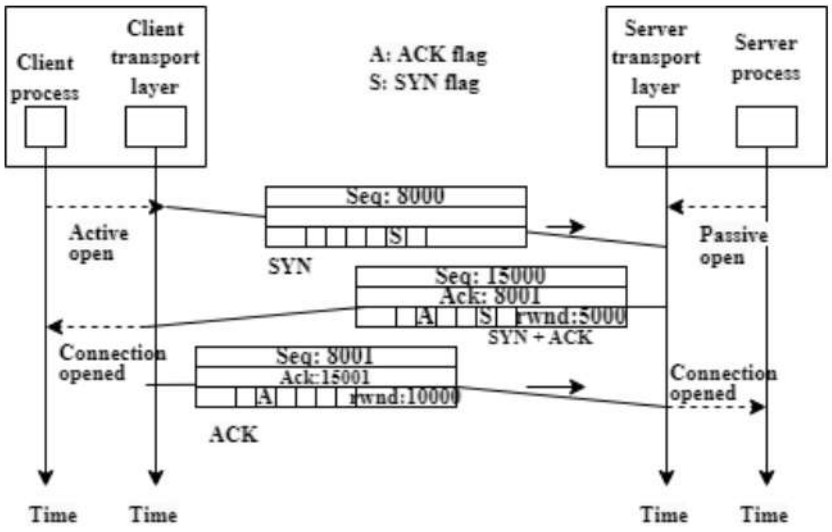


Fig. 6.19: Connection establishment using three-way handshaking

For understanding this process, only a few header field values have been used, although all the fields in the header have their values set.

1. The client sends a SYN segment with a SYN flag bit set for the purpose of synchronizing the sequence numbers. In Fig.6.19, the sequence number randomly chosen by the client is 8000. This segment has no window size defined, no acknowledgment number, and no data in it.
2. The server acknowledges by sending an SYN + ACK segment with the flag bits SYN and ACK set, indicating an SYN for the communication in the opposite direction as well as an acknowledgment for the receipt of an SYN segment from the client. This segment defines the window size for the client. This segment uses one sequence number as it is an SYN segment and requires an acknowledgment. As this is also a control segment, it does not carry data.
3. The client then sends an ACK segment to acknowledge the receipt of the second segment. It does not use any new sequence number. This control segment also does not carry any data.

Data Transfer:

In this phase, data transfer can take place in a duplex mode after connection establishment. To understand the data transfer phase we consider the same scenario as in the connection establishment phase. After the connection establishment, the client and server send 1000 bytes each in one segment The above two segments carry both data and acknowledgment. The client then sends one more segment with only acknowledgment and no data. Fig. 6.20 shows the data transfer phase.

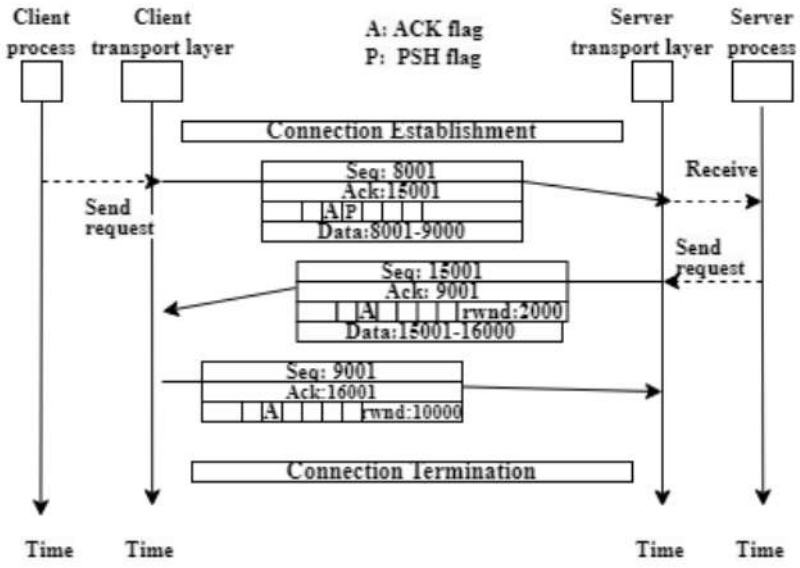


Fig. 6.20: Data Transfer

Connection Termination

The connection in TCP can be closed by either of the communicating nodes. There are two ways of closing a connection: three-way handshaking and four-way handshaking.

Three-way handshaking:

1. First the client process gives a close command to the client TCP which sends a FIN segment either with some remaining data or only control information. This segment has the FIN flag set and if it has only control information, it uses a sequence number as it needs an acknowledgment.
2. After the server TCP receives the FIN segment, it informs its process and sends an acknowledgment segment FIN + ACK in order to confirm that it has received the FIN segment. This segment can carry the remaining data from the server or only the control information and if this is the case it uses one sequence number for acknowledgment purposes.
3. In the final step, an ACK segment with an acknowledgment number added to it is sent by the TCP client to confirm that the FIN segment has been received from the TCP server. The three-way handshaking process for terminating a connection is shown in Fig. 6.21

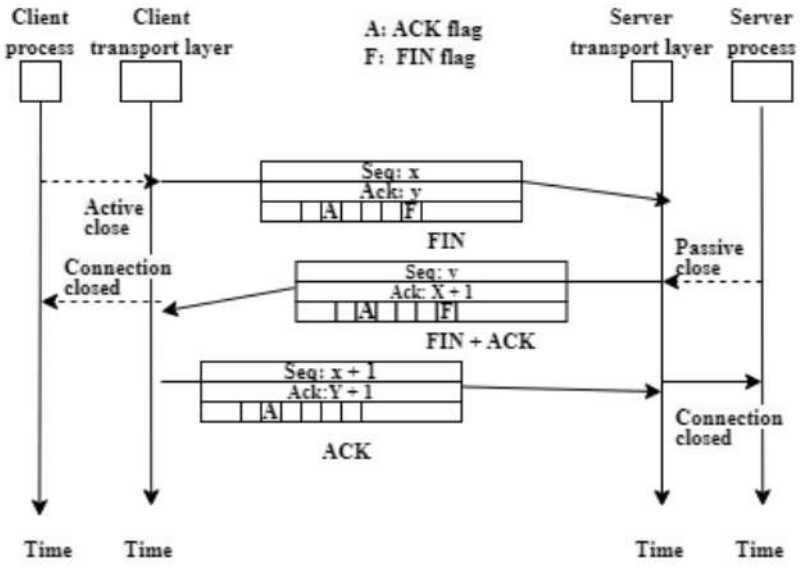


Fig. 6.21: Connection termination using three-way handshaking

6.8.5 Round-trip time

To recover from lost or corrupted segments, the TCP protocol uses the timeout/retransmit protocol mechanism. To implement this mechanism, a very important issue to be considered is the duration of the timeout interval. This time should always be greater than the time required for a data packet to travel from the sender to the receiver plus the time required for the acknowledgment packet to travel from the receiver to the sender. The total time interval is called the round-trip time (RTT). If the timeout is less than RTT, then there will be more retransmissions which may result in congestion on the networks.

6.8.5.1 Estimating the Round-Trip Time

The TCP protocol estimates the round-trip time in the following way. The time needed for a segment to be acknowledged by the receiver after being sent by the sender is represented as SampleRTT. For estimation purposes, the SampleRTT is not measured for all the segments, but it is estimated for only the segment that has been transmitted but not yet acknowledged. So, a new value of SampleRTT is estimated approximately every RTT. This value is measured for only the segments that are transmitted once and not for the ones that have been retransmitted.

The value of SampleRTT keeps changing due to congestion in the routers and changing loads on the end systems. So, the round-trip time is estimated by taking the average of the SampleRTT values. The TCP uses an average of all the SampleRTT values represented as EstimatedRTT. The TCP updates the EstimatedRTT value everytime after obtaining a new value of SampleRTT using the formulae given below:

EstimatedRTT = (1 – α) . EstimatedRTT + α . SampleRTT

The estimated value is the weighted combination of the previous value of EstimatedRTT and the new value of the SampleRTT. The recommended value for α is 0.125.

6.8.6 TCP Applications

This protocol along with IP at the network layer has many applications on the Internet.

1. Web access along with Hyper Text Transfer Protocol (HTTP)
2. Transferring files along with File Transfer Protocol (FTP)
3. Mail transfer along with Simple mail transfer protocol (SMTP)
4. Domain name service (DNS)
5. Remote login through Telnet

6.9 Self-Assessment Questions

Q1. What is a reliable data transfer service? Briefly explain the service model and its implementation model. (6 marks, L2)

Q2. How is reliable data transfer service between two end-points implemented at the transport layer (8 marks, L3)

Q3. Explain the Finite State Machine used for building a reliable data transfer protocol. (5 marks, L4)

Q4. Briefly explain the different versions of the reliable data transfer protocols developed for the transport layer (10 marks, L4)

Q5. Describe the important services and the features provided by the Transmission Control Protocol (12 marks, L3)

Q6. With a neat diagram, explain the TCP Segment format (6 marks, L3)

Q7. Briefly explain the three phases used in a connection-oriented transmission. (6 marks, L4)

Q8. What is round-trip-time and how is it estimated? (6 marks, L4)

6.10 Self-Assessment Activities

A1. Find out some more applications of TCP protocols that you have used for communication on the Internet.

A2. Compare the UDP and TCP protocols with the help of a table indicating the suitability of each of them with applications.

A3. Find out the commands used to find the addresses and port numbers by creating an application process.

6.11 Multiple-Choice Questions

Q1. Which of these is a primary service provided by the TCP protocol? [1 mark, L1]

A. Routing

B. Flow Control
C. Addressing
D. Reliable Data Transfer

Q2. Reliability of data transfer is provided by the TCP with the help of, [1 mark, L1]
A. Multipath routing
B. Error detection and retransmission of lost packets
C. Encryption of data packets
D. None of the above

Q3. Which of the below statements is NOT true about TCP [1 mark, L1]
A. It is a connection-oriented protocol
B. It provides flow control
C. It uses an acknowledgement mechanism
D. It uses best-effort delivery service

Q4. The maximum number of ports available for use in the TCP protocol is __ [1 mark, L1]
A. 64
B. 128
C. 256
D. 65535

Q5. The ____ flag is used by TCP protocol for connection establishment. [1 mark, L1]
A. ACK
B. SYN
C. FIN
D. RST

Q6. The ____ flag is used to indicate connection termination by TCP protocol. [1 mark, L1]
A. ACK
B. FIN
C. PSH
D. None of the above

Q7. Which of the below is TRUE about TCP window size [1 mark, L1]
A. The window size is not adjustable and fixed
B. More the window size, the faster the data transfer
C. The window size is used for the initial connection setup
D. Smaller window size is efficient during high-speed data transfer

Q8. The sequence number is used in TCP for, [1 mark, L1]
A. Keeping track of the number of packets transmitted and received
B. Identifying the source and destination IP addresses
C. Identifying the source and destination port numbers
D. Ordering and reassembling the different segments of a message

6.12 Keys to Multiple-Choice Questions

Q1. Reliable Data Transfer (D)

Q2. Error-detection and retransmission of lost packets (B)
Q3. It uses best-effort delivery service (D)
Q4. 65535 (D)
Q5. SYN (B)
Q6. FIN (B)
Q7. More the window size, the faster the data transfer (B)
Q8. Ordering and reassembling the different segments of a message

6.13 Summary of the Unit

This unit covers three important topics related to the transport layer: The first topic of this unit discusses the Principles of Reliable Data Transfer explained with a Finite State Machine Model for better understanding. This topic also discusses the various stages or versions of building a Reliable data transfer protocol. After an understanding of the principles and the stages, three transport layer protocols and their limitations are discussed in the second topic. These include Stop-and Wait, Go-Back-N, and Selective Repeat protocols. The last topic includes the discussion of the Transmission Control Protocol. The services provided by the protocol and the important features are discussed. The TCP header format, the TCP connection establishment, and the estimation of the round-trip time for the packets are also briefly discussed.

6.14 Recommended Learning Resources

[1] James F Kurose and Keith W Ross, Computer Networking, A Top-Down Approach, Sixth Edition, Pearson,2017.
[2] Behrouz A Forouzan, Data and Communications and Networking, Fifth Edition, McGraw Hill, Indian Edition
[3] https://en.wikipedia.org/wiki/Connection-oriented_communication
[4] https://www.javatpoint.com/sliding-window-protocol