BACHELOR OF COMPUTER APPLICATIONS

SEMESTER 3

DCA2201

COMPUTER NETWORKING

Unit 5

Data Link Layer – Sliding Window Protocols

Table of Contents

1. INTRODUCTION

In the previous unit, we discussed the issues of datalink layer design and elementary datalink protocols. Data link layer is responsible for implementing flow control mechanisms. Data link layer protocols are procedures for achieving reliable, efficient communication of frames between two machines connected by a communication channel. In this unit, we will discuss various sliding window protocols. The essence of sliding window protocol is that both sender and receiver agree on the number of data frames it sends and receives before sending an acknowledgement. This protocols define a window that slides from left to right over time, hence called sliding window protocols. The datalink protocol used in ARPANET provides an alternative to the sliding window protocol. The underlying concept of ARPANET is concurrent logical channels.

We will start this unit with the description of sliding window protocols. This unit progresses with the discussion of different sliding window protocols. In the last session, we will discuss concurrent logical channels.

1.1 Objectives:

After studying this unit, you should be able to:

- *Describe sliding window protocols*
- *Differentiate go-back-n and selective repeat protocols*
- *Explain about concurrent logical channels*

2. SLIDING WINDOW PROTOCOLS

In Elementary data link protocols, data frames were transmitted in one direction only. But there is a need to transmit data in both directions. One way to achieve both way transmission of data is by using the same link for data in both directions. In elementary datalink stop and wait protocol, each link is composed of a forward channel (for data) and a reverse channel (for acknowledgements). In this, frames are transmitted in both ways and the reverse channel has the same capacity as the forward channel and the capacity of the reverse channel is almost entirely wasted.

In sliding window protocol, data frames from A to B are intermixed with the acknowledgement frames from A to B. By looking at the kind field in the header of an incoming frame, the receiver can tell whether the frame is data or an acknowledgement.

Another method is, when a data frame arrives, instead of immediately sending a separate control frame, the receiver restrains itself and waits until the network layer passes it the next packet. The acknowledgement is attached to the outgoing data frame (using the ack field in the frame header). In effect, the acknowledgement gets a free ride on the next outgoing data frame. The technique of temporarily delaying outgoing acknowledgements so that they can be hooked onto the next outgoing data frame is known as **piggybacking**. This results the better use of available channel bandwidth. The *ack* field in the frame header costs only a few bits, whereas a separate frame would need a header, the acknowledgement, and a checksum. Also, generally fewer frames sent means a lighter processing load at the receiver. The problem in the case of piggy backing is that, waiting time for the next packet is unknown. That is, for how long the data link layer has to wait for a packet to piggyback the acknowledgement is unspecified. If the data link layer waits longer than the sender's timeout period, the frame will be retransmitted. In order to resolve this issue, the datalink layer must resort to some specific scheme such as waiting for a fixed number of milliseconds. If a new packet arrives immediately, the acknowledgement is piggybacked onto it. Else, if no new packet has arrived by the end of the specific time period, the data link layer just sends a separate acknowledgement frame.

Now, we will discuss three different sliding window protocols in the coming sections. In Sliding window protocol, each outbound frame contains a sequence number, ranging from 0 up to some maximum. The maximum is usually *2n – 1* so the sequence number fits exactly in an *n-bit* field. The stop-and-wait sliding window protocol uses *n = 1*, restricting the sequence numbers to 0 and 1. In sliding window protocol, at any instant of time, the sender maintains a set of sequence numbers corresponding to frames it is permitted to send. These frames fall within the **sending window**. Similarly, the receiver also maintains a **receiving window** corresponding to the set of frames it is permitted to accept. The sender's window and the receiver's window need not have the same lower and upper limits or even have the same size. In some sliding window protocols, window size is fixed. But in some others window size can grow or shrink over the course of time as frames are sent and received.

The sequence numbers within the sender's window represent frames that have been sent or can be sent but are as yet not acknowledged. Whenever a new packet arrives from the network layer, it is given the next highest sequence number, and the upper edge of the window is advanced by one. When an acknowledgement comes in, the lower edge is advanced by one. So, the window continuously maintains a list of unacknowledged frames. Figure 5.1 shows an example of a sliding window.

Since frames currently within the sender's window may ultimately be lost or damaged in transit, the sender must keep all of these frames in its memory for possible retransmission. Thus, if the maximum window size is *n*, the sender needs *n* buffers to hold the unacknowledged frames. If the window ever grows to its maximum size, the sending data link layer must forcibly shut off the network layer until another buffer becomes free.

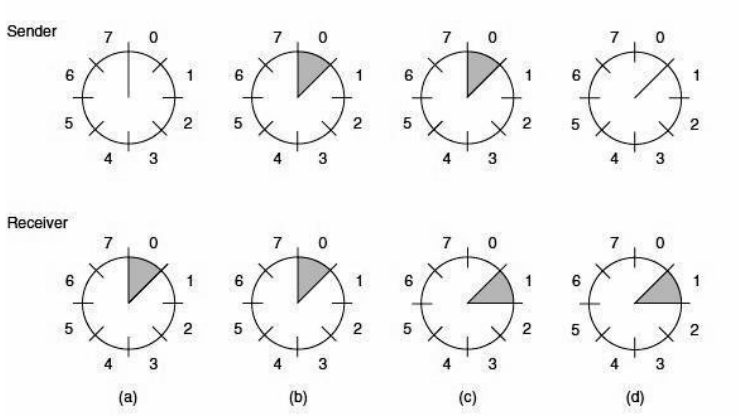

Figure 5.1: A sliding window of size 1, with a 3-bit sequence number.

(a) Initially. (b) After the first frame has been sent. (c) After the first frame has been received. (d) After the first acknowledgment has been received.

The receiving data link layer's window holds the frames which it accepts. Frames falling within the window are kept in the receiver's buffer. When a frame whose sequence number is equal to the lower edge of the window is received, it is passed to the network layer and the window is rotated by one. Any frame falling outside the window is discarded. In all of these cases, a subsequent acknowledgement is generated so that the sender may work out how to proceed.

2.1 One-Bit Sliding Window Protocol

A sliding window protocol with a window size of 1, uses the stop-and-wait method since the sender transmits a frame and waits for its acknowledgement before sending the next one. Figure 5.2 shows the procedure for one-bit sliding window protocol. Here, *Next_frame_to_send* tells which frame the sender is trying to send. Similarly, *frame_expected* tells which frame the receiver is expecting. In both cases, 0 and 1 are the only possibilities. The procedure is that the sender machine fetches the first packet from its network layer, builds a frame from it and sends it. At the receiver side, the receiving datalink layer checks to see if it is a duplicate. If the frame is the one expected, it is passed to the network layer and the receiver's window is moved up. Acknowledgment contains the number of the last frame received without error. If this number agrees with the sequence number of the frame the

sender is trying to send, the sender knows it is done with the frame stored in buffer and can fetch the next packet from its network layer. If the sequence number disagrees, it must continue trying to send the same frame.

```
/* sliding window protocol*/ #define MAX_SEQ 1

typedef enum {frame_arrival, cksum_err,timeout} event_type; #include “protocol.h”

void protocol4 (void)

{

seq_nr next_frame_to_send;  /*0 or 1 only */

seq_nr frame_expected;        /* 0 or 1 only*/ frame r,s;

packet buffer; /*current packet being sent*/ event_type event;


next_frame_to_send=0; frame_expected =0;


from_network_layer (&buffer);       /*fetch          packet from    network layer*/

s.info =buffer;

s.seq = next_frame_to_send; /*  insert  seq  no  to  the  frame*/  s.ack=1-frame_expected;
        /*piggy back acknowledgement*/ to_physical_layer (&s);        /*transmit        the
frame*/

start_timer (s.seq); while (true) {

Wait_for_event (&event);

If (event == frame_arrival)   /* frame has arrived undamaged*/

{
```

```
From_physical_layer (&r);

If (r.seq==frame_expected){

to_network_layer (&r.info);  /*pass packet to network layer*/

inc (frame_expected);          /* invert seq number expected next*/

}


If (r.ack == next_frame_to_send) {

Stop_timer (r.ack);   /*turn the timer off*/ From_network_layer (&buffer);   /*fetch new packet from network layer*/ Inc (next_frame_to_send);  /*invert sender's sequence number*/

}

}

s.info=buffer; s.seq=next_frame_to_send;

s.ack=1-frame_expected;     /*sequence number of last received frame*/ to_physical_layer (&s);   /*transmit the frame

start_timer (s.deq);   /*start the timer running*/

}

}
```

Figure 5.2: A 1-bit sliding window protocol

A trouble in this protocol is that if both sender, say A and receiver, say B initiate communication simultaneously then their first frame cross each other and this will result in the sending of duplicate packets and hence wasting bandwidth.

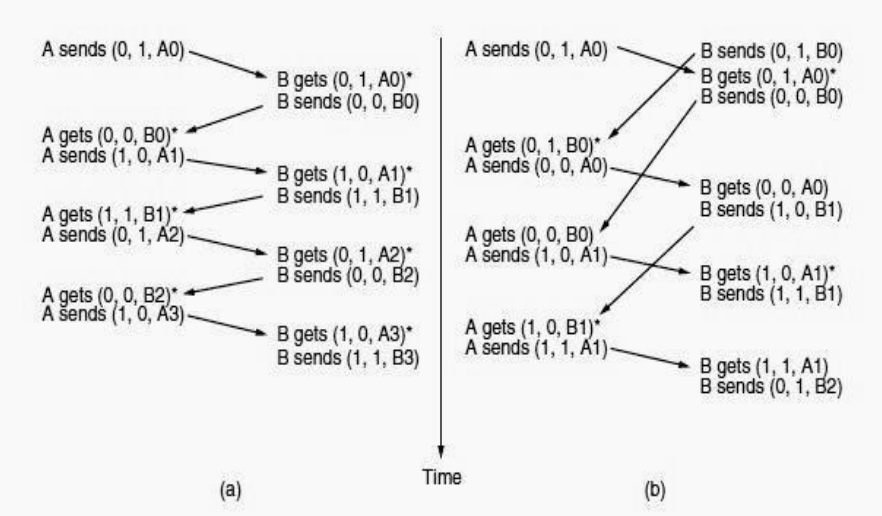

Figure 5.3: (a) Normal Case. (b) Abnormal case. The notation is (seq, ack, packet number). An asterisk indicates where a network layer accepts a packet.

Figure 5.3 (a) shows the normal operation, where each frame arrives properly and there are no duplicates. In Figure 5.3 (b) half of the frames contain duplicates even though there are no transmission errors. Even if one side clearly starts first, similar situations can occur as a result of premature timeouts.

2.2 Protocol Using Go-Back-N

The problem we have faced in the previous algorithm is that the sender has to wait for an acknowledgment before sending another frame. If we avoid that constraint, much better efficiency can be achieved. The solution lies in allowing the sender to transmit up to w frames before blocking, instead of 1. With a large enough choice of w, the sender will be able to continuously transmit frames since the acknowledgements will arrive for previous frames before the window becomes full, preventing the sender from blocking.

The value of w is determined by the number of frames that can fit inside the channel as they propagate from sender to receiver. This capacity is determined by the bandwidth in bits/sec multiplied by the one-way transit time, or the bandwidth-delay product of the link. The technique of keeping multiple frames in trajectory is an example of pipelining. Pipelining frames over an unreliable communication channel raises some serious issues. If a frame in the middle of a long stream is damaged or lost, succeeding frames will arrive at the receiver

before the sender even finds out that anything is wrong. Two basic approaches are available for dealing with errors in the presence of pipelining, which are shown in below figure 5.4.

One approach is called ***go-back-n***. In this, the receiver simply discards all subsequent frames, sending no acknowledgements for the discarded frames. We can see that this approach corresponds to a receive window of size 1. That is, here the data link layer refuses to accept any frame except the next one it must give to the network layer. If the sender's window fills up before the timer runs out, the pipeline will begin to empty. Eventually, the sender will time out and retransmit all unacknowledged frames in order, starting with the damaged or lost one. This approach can waste a lot of bandwidth if the error rate is high.

In figure 5.4 (b), we see ***go-back-n*** for the case in which the receiver's window is large. Frames 0 and 1 are correctly received and acknowledged. Frame 2, however, is damaged or lost. The sender, unaware of this problem, continues to send frames until the timer for frame 2 expires. Then it backs up to frame 2 and starts over with it, sending 2, 3, 4, etc. all over again.

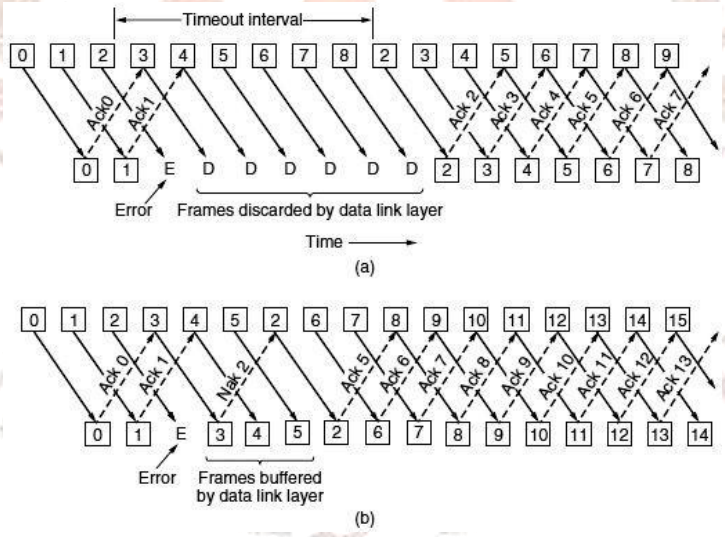

Figure 5.4: Pipelining and error recovery. Effect of an error when

(a) receiver's window size is 1 and (b) receiver's window size is large.

The second approach for handling errors when frames are pipelined is called selective repeat. When this is used, a bad frame that is received is discarded, but any good frames

received after it are accepted and buffered. When the sender times out, only the oldest unacknowledged frame is retransmitted. If that frame arrives correctly, the receiver can deliver to the network layer, in sequence, all the frames it has buffered. Selective repeat corresponds to a receiver window larger than 1. This approach can require large amounts of data link layer memory if the window is large.

Mostly, selective repeat is combined with having the receiver send a negative acknowledgement (NAK) when it detects an error. NAKs stimulate retransmission before the corresponding timer expires and thus improve performance. In figure 5.4 (b), frames 0 and 1 are correctly received and acknowledged and frame 2 is lost. When frame 3 arrives at the receiver, the data link layer notices that it has missed a frame, so it sends back a NAK for 2 but buffers 3. When frames 4 and 5 arrive, they, too, are buffered by the data link layer instead of being passed to the network layer. Eventually, the NAK 2 gets back to the sender, which immediately resends frame 2. When that arrives, the data link layer now has 2, 3, 4, and 5 and can pass all of them to the network layer in the correct order. It can also acknowledge all frames up to and including 5.

In a ***go-back-n*** protocol, the receiving data link layer only accepts frames in order. Frames following an error are discarded. Although ***go-back-n*** protocol does not buffer the frames arriving after an error, it does not escape the problem of buffering altogether. Since a sender may have to retransmit all the unacknowledged frames at a future time, it must keep all transmitted frames until it knows that they have been accepted by the receiver. When an acknowledgement comes in for frame n, frames n − 1, n − 2, and so on are also automatically acknowledged. This type of acknowledgement is called a ***cumulative acknowledgement***. This property is very important because some of the previous acknowledgement-bearing frames were lost or garbled. Whenever any acknowledgement comes in, the data link layer checks to see if any buffers can now be released. If buffers can be released, a previously blocked network layer can be available to cause more network layer ready events.

2.3 Protocol Using Selective Repeat

In the case of *go-back-n* protocol, if there are more errors then it wastes a lot of bandwidth on retransmitted frames. This protocol is best suited if errors are rare. An *alternative*

strategy is *selective repeat protocol*. Here, the receiver can accept and buffer the frames following a damaged or lost one. In this protocol, both sender and receiver maintain a window of outstanding and acceptable sequence numbers respectively.

Sender's window size starts at 0 and grows to some predefined maximum. But receiver's window is always fixed in size and equal to the predefined maximum. The receiver has a buffer reserved for each sequence number within its fixed window. Associated with each buffer is a bit telling whether the buffer is full or empty. Whenever a frame arrives, its sequence number is checked to see if it falls within the window. If so and if it has not already been received, it is accepted and stored. This action is taken without regard to whether or not the frame contains the next packet expected by the network layer. Of course, it must be kept within the data link layer and not passed to the network layer until all the lower-numbered frames have already been delivered to the network layer in the correct order.

A problem associated with this nonsequential receive is illustrated here with an example. Suppose that we have a 3-bit sequence number, so that the sender is permitted to transmit up to seven frames before being required to wait for an acknowledgement. Initially, the sender's and receiver's windows are as shown in figure 5.5 (a). The sender now transmits frames 0 through 6. The receiver's window allows it to accept any frame with a sequence number between 0 and 6 inclusive. All seven frames arrive correctly, so the receiver acknowledges them and advances its window to allow receipt of 7, 0, 1, 2, 3, 4, or 5, as shown in figure 5.5 (b). All seven buffers are marked empty. Suppose at this point all the acknowledgements lost, the protocol should operate correctly regardless of this trouble. The sender eventually times out and retransmits frame 0. When this frame arrives at the receiver, a check is made to see if it falls within the receiver's window. Unfortunately, in Figure 5.5 (b) frame 0 is within the new window, so it is accepted as a new frame. The receiver also sends a (piggybacked) acknowledgement for frame 6, since 0 through 6 have been received.

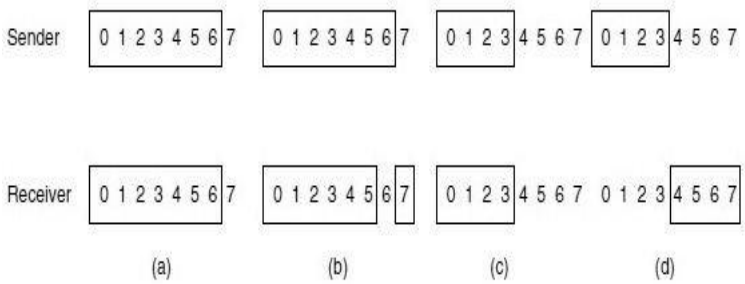

Figure 5.5: (a) Initial situation with a window of size 7 (b) After 7 frames have been sent and received but not acknowledged. (c) Initial situation with a window size of 4. (d) After 4 frames have been sent and received but not acknowledged.

The sender is happy to learn that all its transmitted frames did actually arrive correctly, so it advances its window and immediately sends frames 7, 0, 1, 2, 3, 4, and 5. Frame 7 will be accepted by the receiver and its packet will be passed directly to the network layer. Immediately thereafter, the receiving data link layer checks to see if it has a valid frame 0 already, discovers that it does, and passes the old buffered packet to the network layer as if it were a new packet. Consequently, the network layer gets an incorrect packet, and the protocol fails.

We can say, the essence of this problem is that after the receiver advanced its window, the new range of valid sequence numbers overlapped the old one. Therefore, the following batch of frames might be either duplicates or new ones. The receiver has no way to distinguish these two cases.

This problem can be resolved by making sure that after the receiver has advanced its window there is no overlap with the original window. So, in order to ensure this, the maximum window size should be at most half the range of the sequence numbers. This situation is shown in figure 5.5 (c) and Fig 5.5 (d). With 3 bits, the sequence numbers range from 0 to 7. Only four unacknowledged frames should be outstanding at any instant. That way, if the receiver has just accepted frames 0 through 3 and advanced its window to permit acceptance of frames 4 through 7, it can unambiguously tell if subsequent frames are retransmissions (0 through 3) or new ones (4 through 7).

Self-Assessment Questions - 1

1. The underlying concept of ARPANET is _____ .
2. Which protocols define a window that slides from left to right over time?

   (a) Sliding window protocol (b) stop and wait (c) simplex

3. The technique of temporarily delaying outgoing acknowledgements so that they can be hooked onto the next outgoing data frame is known as _____
4. A sliding window protocol with a window size of 1, uses _____since the sender transmits a frame and waits for its acknowledgement before sending the next one.
5. The technique of keeping multiple frames in trajectory is an example of_____.
6. In ____protocol, the receiver simply discards all subsequent frames, sending no acknowledgements for the discarded frames.
7. In _____protocol, a bad frame that is received is discarded, but any good frames received after it are accepted and buffered.
8. When an acknowledgement comes in for frame n, frames *n – 1, n – 2,* and so on are also automatically acknowledged. This type of acknowledgement is called _____.

3. CONCURRENT LOGICAL CHANNELS

The datalink protocol used in ARPANET provides an alternative to the sliding window protocol, in that it is able to keep the link full while using the simple stop and wait algorithm. One important consequence of this approach is that the frames sent over a given link are not kept in any particular order. The protocol also implies nothing about flow control.

The idea underlying the ARPANET protocol, which we refer to as *concurrent logical channels*, is to multiplex several logical channels onto a single point-to-point link and to run the ***stop-and-wait*** algorithm on each of these logical channels. There is no relationship maintained among the frames sent on any of the logical channels, yet because a different frame can be outstanding on each of the several logical channels the sender can keep the link full.

In this protocol, the sender keeps 3 bits of state for each channel. They are: a Boolean, saying whether the channel is currently busy; the 1-bit sequence number to use the next time a frame is sent on this logical channel; and the next sequence number to expect on a frame that arrives on this channel. When the node has a frame to send, it uses the lowest idle channel, and otherwise it behaves just like stop-and-wait.

Self-Assessment Questions - 2

9. The technique to multiplex several logical channels onto a single point-to-point link and to run the stop-and-wait algorithm on each of these logical channels is called _____.
10. In concurrent logical channels protocol, the sender keeps _____ bits of state for each channel.

4. SUMMARY

Let us recapitulate the important concepts discussed in this unit:

- Sliding window protocols define a window that slides from left to right over time, hence called sliding window protocols.
- The datalink protocol used in ARPANET provides an alternative to the sliding window protocol. The underlying concept of ARPANET is concurrent logical channels.
- The technique of temporarily delaying outgoing acknowledgements so that they can be hooked onto the next outgoing data frame is known as piggybacking.
- In Elementary data link protocols, data frames were transmitted in one direction only.
- In Sliding window protocol, each outbound frame contains a sequence number, ranging from 0 up to some maximum.
- The maximum is usually *2n – 1* so the sequence number fits exactly in an *n-bit* field.
- A sliding window protocol with a window size of 1, uses stop-and-wait since the sender transmits a frame and waits for its acknowledgement before sending the next one.
- In go-back-n, if an error occurred, the receiver simply discards all subsequent frames, sending no acknowledgements for the discarded frames.
- In selective repeat protocol, the receiver can accept and buffer the frames following a damaged or lost one. In this protocol, both sender and receiver maintain a window of outstanding and acceptable sequence numbers respectively.
- The idea underlying the ARPANET protocol, which we refer to as *concurrent logical channels,* is to multiplex several logical channels onto a single point-to-point link and to run the stop-and-wait algorithm on each of these logical channels.
- In *concurrent logical channels* protocol, the sender keeps 3 bits of state for each channel. They are: a Boolean, saying whether the channel is currently busy; the 1-bit sequence number to use the next time a frame is sent on this logical channel; and the next sequence number to expect on a frame that arrives on this channel.

5. TERMINAL QUESTIONS

1. Explain one-bit sliding window protocol.
2. Differentiate between go-back-n and selective repeat protocols.
3. Describe concurrent logical channels.
4. Explain go-back-n protocol.
5. Describe selective repeat protocol

6. ANSWERS

Self-Assessment Questions

1. Concurrent logical channels
2. Sliding window protocols
3. Piggy backing
4. Stop and wait
5. Pipelining
6. Go-back-n
7. Selective repeat
8. Cumulative acknowledgement
9. Concurrent logical channels
10. Three

Terminal Questions

1. A sliding window protocol with a window size of 1, uses stop-and-wait since the sender transmits a frame and waits for its acknowledgement before sending the next one. (Refer section 5.2.1 for more details).
2. One approach is called go-back-n. In this, the receiver simply discards all subsequent frames, sending no acknowledgements for the discarded frames. An alternative strategy is selective repeat protocol. Here, the receiver can accept and buffer the frames following a damaged or lost one. In this protocol, both sender and receiver maintain a window of outstanding and acceptable sequence numbers respectively. (Refer section 5.2.2 and 5.2.3 for more details)

3. The datalink protocol used in ARPANET provides an alternative to the sliding window protocol, in that it is able to keep the link full while using the simple stop and wait algorithm. One important consequence of this approach is that the frames sent over a given link are not kept in any particular order. The protocol also implies nothing about flow control. (Refer section 5.3 for more details).
4. In this, the receiver simply discards all subsequent frames, sending no acknowledgements for the discarded frames. We can see that this approach corresponds to a receive window of size 1. That is, here the data link layer refuses to accept any frame except the next one it must give to the network layer. (Refer section 5.2.2 for more details).
5. In In case of ***go-back-n*** protocol, if there are more errors then it wastes a lot of bandwidth on retransmitted frames. This protocol is best suited if errors are rare. An alternative strategy is selective repeat protocol. Here, the receiver can accept and buffer the frames following a damaged or lost one. In this protocol, both sender and receiver maintain a window of outstanding and acceptable sequence numbers respectively. (Refer section 5.2.3 for more details).

References:

- Andrew S Tanenbaum, David J. Wetherall, *"Computer Networks,"* Fifth edition.
- Larry L. Peterson, Bruce S. Davie, *"Computer Networks – a Systems Approach,"* Fifth edition.
- James F. Kurose, Keith W. Ross, *"Computer Networking – A top-down approach,"* Sixth edition.
- Behrouz A. Forouzan, Sophia Chung Fegan, *"Data Communication and Networking,"* Fourth edition.
- William Stallings, *"Computer Networking with Internet Protocols and Technology,"* Third edition.