1. Introduction

Data communication requires at least two devices working together, one to send data and the other to receive that data along with that a great deal of coordination is required between sender and receiver for a successful exchange to occur. The most important responsibilities of the data link layer are flow control and error control. Collectively, these functions are known as data link control.

2. Flow Control

As any receiving device has a limited speed at which it can process incoming data and a limited amount of memory (buffer) in which to store incoming data. Thus the flow of data from sender must not be allowed to overwhelm the receiver.

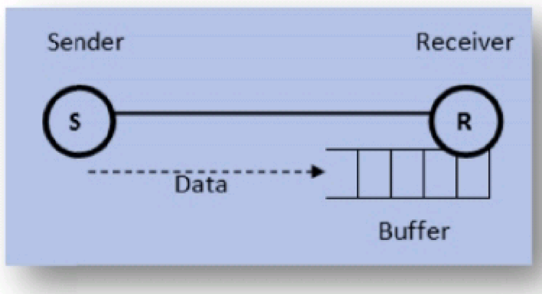


Figure 1.1 Data transfer between sender and receiver

The receiving device must be able to inform the sending device before those limits (processing and memory) are reached and to request that the transmitting device send fewer frames or stop temporarily (Figure 1.1).

Flow control is a set of procedures that tells the sender how much data it can transmit before it must wait for an acknowledgment from the receiver.

3. Error Control

In the data link layer, the term *error control* refers primarily to methods of error detection and retransmission. Error control in the data link layer is based on automatic repeat request, which is the retransmission of data. Any time an error is detected in an exchange, specified frames are retransmitted. This process is called automatic repeat request (ARQ).

4. Protocols for Error Control and Flow Control

Protocols at Data link Layer for flow and error control can be categorized into two types:

- Protocol for noiseless (error-free) channels
- Protocol for noisy (error-prone) channels.

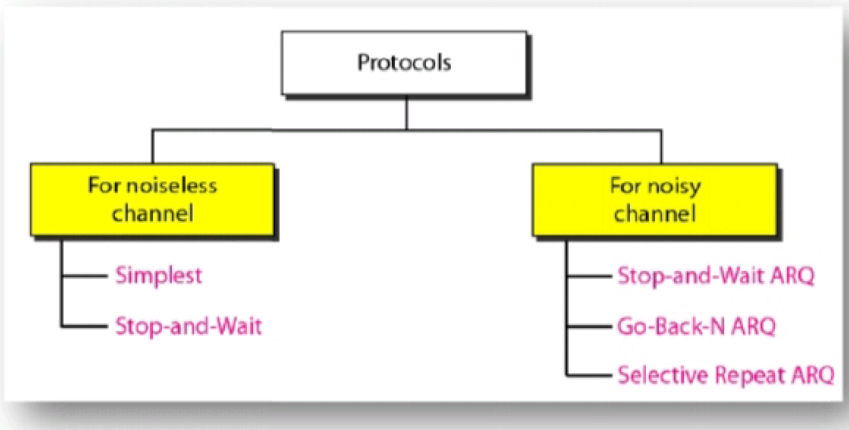


Figure 1.2 Taxonomy of protocols discussed in this chapter

The protocols in the first category cannot be used in real life, but they serve as a basis for understanding the protocols of noisy channels. Figure 1.2 shows the classifications.

4.1. Protocols for Noiseless Channels

Basic assumption: we have an ideal channel in which no frames are lost, duplicated, or corrupted. There are two protocols for this type of channel: the first is a protocol that does not use flow control; the second is the one that does. Neither has error control because we have assumed that the channel is a perfect noiseless channel.

4.1.1. Simplest Protocol

- It is a unidirectional protocol in which data frames are traveling in only one direction-from the sender to receiver.
- Receiver has infinite buffer space and infinite processing speed.
- In other words, the receiver can never be overwhelmed with incoming frames.
- Thus No flow or error control needed.

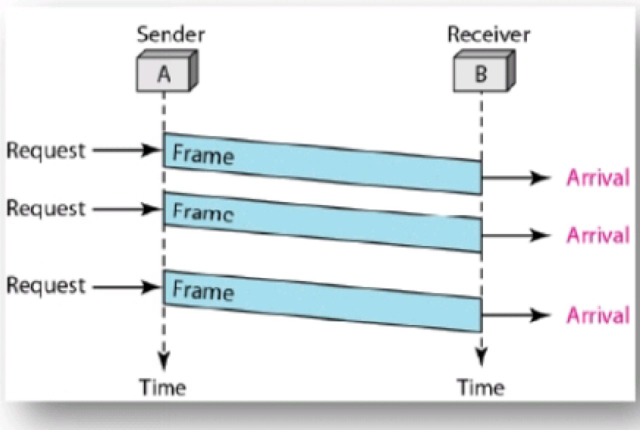


Figure 1.3 Flow diagram for Example 1.1

Example 1.1
Figure 1.3 shows an example of communication using this protocol. It is very simple. The sender sends a sequence of frames without even thinking about the receiver.

4.1.2. Stop-and-Wait Protocol

- Receiver has finite buffer space and infinite processing speed.
- In other words, the receiver can be overwhelmed with incoming frames.
- Thus no error control but flow control is needed.

If data frames arrive at the receiver site faster than they can be processed, the frames must be stored until their use. Normally, the receiver does not have enough storage space, especially if it is receiving data from many sources. This may result in either the discarding of frames or denial of service. To prevent the receiver from becoming overwhelmed with frames, we somehow need to tell the sender to slow down. There must be feedback from the receiver to the sender.

The protocol is called the Stop-and-Wait Protocol because the sender sends one frame, stops until it receives confirmation from the receiver (okay to go ahead), and then sends the next frame.

We still have unidirectional communication for data frames, but auxiliary ACK frames (simple tokens of acknowledgment) travel from the other direction.

Example 1.2
Figure 1.4 shows an example of communication using this protocol. It is still very simple. The sender sends one frame and waits for feedback from the receiver. When the ACK arrives, the sender sends the next frame.

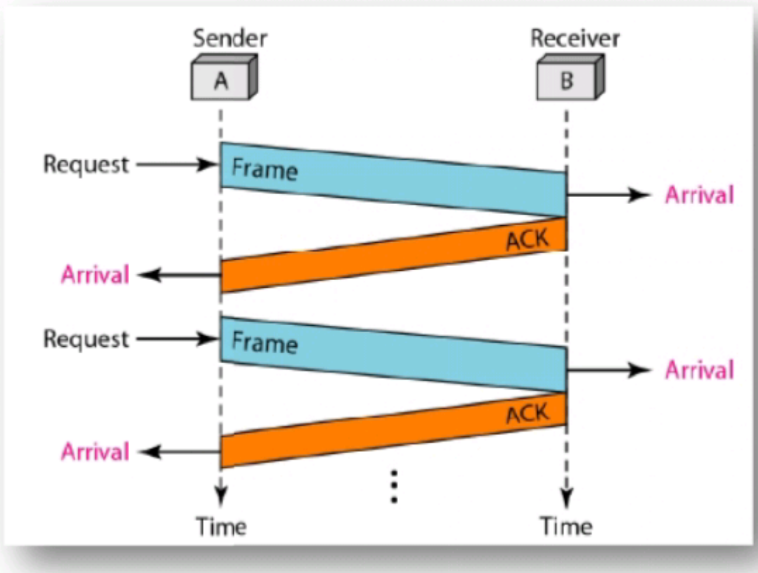


Figure 1.4 Flow diagram for Example 1.2

4.2. Protocol for Noisy Channels

Basic assumption: we have a real channel (noisy) in which frames are lost, duplicated, or corrupted thus error control is required. Receiver has finite buffer space and finite processing speed. In other words, the receiver can be overwhelmed with incoming frames. Thus flow control needed. We discuss three protocols in this section that use error control.

4.2.1. Stop-and-Wait Automatic Repeat Request

- Stop-and-Wait Automatic Repeat Request (Stop-and Wait ARQ), adds a simple error control mechanism to the Stop-and-Wait Protocol.
- To detect and correct corrupted frames, it is needed to add redundancy bits to data frame (using some error correcting code).
- The received frame could be the correct one, or a duplicate, or a frame out of order.
- When the frame arrives at the receiver site, it is checked and if it is corrupted, it is silently discarded. The detection of errors in this protocol is manifested by the silence of the receiver.
- The solution is to number the frames. When the receiver receives a data frame that is out of order, this means that frames were either lost or duplicated.
- The corrupted or lost frames need to be resent in this protocol.
- If the receiver does not respond when there is an error, how can the sender know which frame to resend?
- To remedy this problem, the sender keeps a copy of the sent frame. At the same time, it starts a timer. If the timer expires and there is no ACK for the sent frame, the frame is resent, the copy is held, and the timer is restarted.
- Since an ACK frame can also be corrupted and lost, it too needs redundancy bits and a sequence number.
- The ACK frame for this protocol has a sequence number field.
- In this protocol, the sender simply discards a corrupted ACK frame or ignores an out-of-order one.

Sequence Numbers

- The protocol specifies that frames need to be numbered by using sequence numbers. A field is added to the data frame to hold the sequence number of that frame.
- The sequence numbers are of wrap around in nature. For example, if we decide that the field is *m* bits long, the sequence numbers start from 0, go to 2^m - 1, and then are repeated.

Acknowledgment Numbers

- The acknowledgment numbers always announce the sequence number of the next frame expected by the receiver.
- For *example*, if frame 0 has arrived safe and sound, the receiver sends an ACK frame with acknowledgment 1 (meaning frame 1 is expected next). If frame 1 has arrived safe and sound, the receiver sends an ACK frame with acknowledgment 0 (meaning frame 0 is expected).

Design:

Figure 1.5 shows the design of the Stop-and-Wait ARQ Protocol.

- The sending device keeps a copy of the last frame transmitted until it receives an acknowledgment for that frame. A data frames uses a seqNo (sequence number); an ACK frame uses an ackNo (acknowledgment number).
- The sender has a control variable, which we call *Sn* (sender, next frame to send), that holds the sequence number for the next frame to be sent (0 or 1).
- The receiver has a control variable, which we call *Rn* (receiver, next frame expected), that holds the number of the next frame expected.
- When a frame is sent, the value of *Sn* is incremented (modulo-2), which means if it is 0, it becomes 1 and vice versa.
- When a frame is received, the value of *Rn* is incremented (modulo-2), which mea ns if it is 0, it becomes 1 and vice versa.

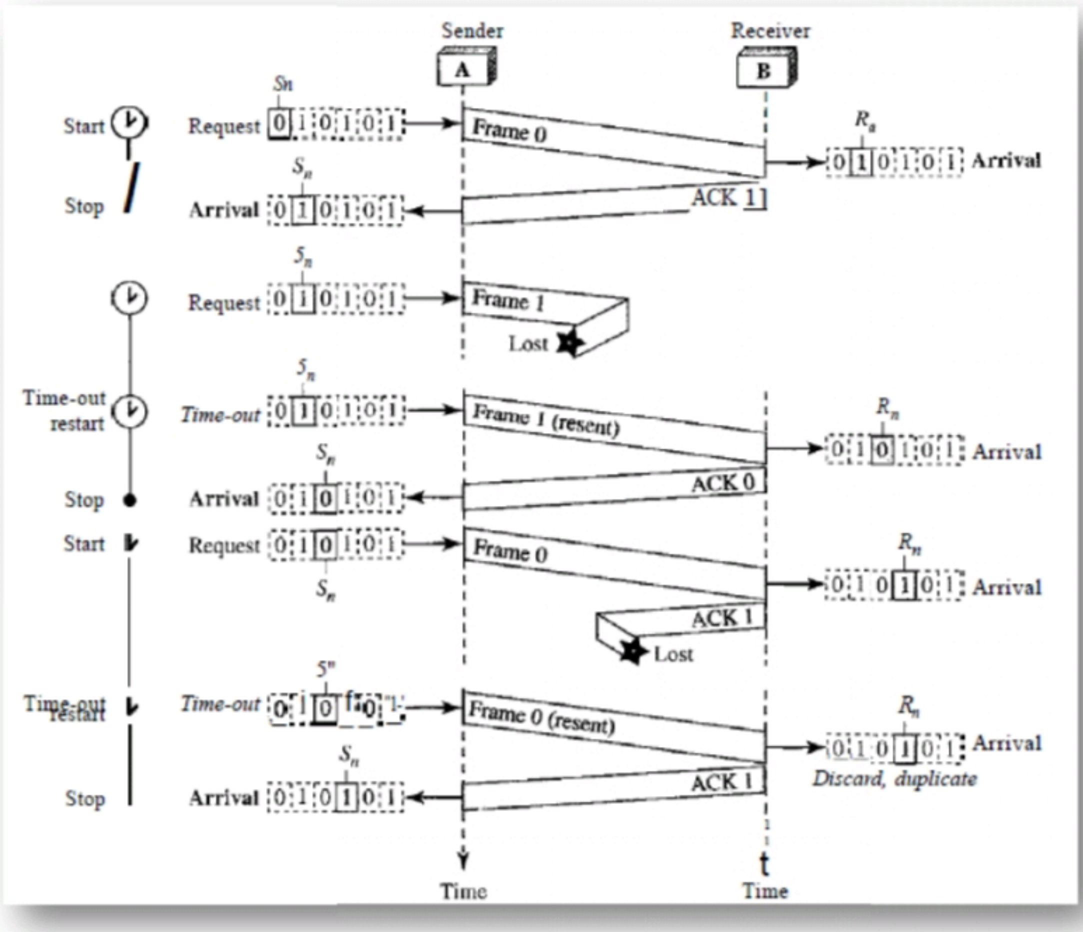


Figure 1.5 Flow diagram for Example 1.3

Example 1.3
Figure 1.5 shows an example of Stop-and-Wait ARQ. Frame 0 is sent and acknowledged. Frame 1 is lost and resent after the time-out. The resent frame 1 is acknowledged and the timer stops. Frame 0 is sent and acknowledged, but the acknowledgment is lost. The sender has no idea if the frame or the acknowledgment is lost, so after the time-out, it resends frame 0, which is acknowledged.

Efficiency
The Stop-and-Wait ARQ discussed in the previous section is very inefficient if the channel is *thick* and *long*. By *thick,* we mean that our channel has a large bandwidth; by *long,* we mean the round-trip delay is long. The product of these two is called the *bandwidth delay product*. We can think of the channel as a pipe. *The bandwidth-delay product then is the volume of the pipe in bits*. The pipe is always there. If we do not use it, we are inefficient. *The bandwidth-delay product is a measure of the number of bits we can send out of our system while waiting for news from the receiver.*

Example 1.4
Assume that, in a Stop-and-Wait ARQ system, the bandwidth of the line is 1 Mbps, and 1 bit takes 20 ms to make a round trip. What is the bandwidth-delay product? If the system data frames are 1000 bits in length, what is the utilization percentage of the link?

Solution
The bandwidth-delay product= $(1x10^6)x(20x10^{-3})$=20,000 bits
The system can send 20,000 bits during the time it takes for the data to go from the sender to the receiver and then back again. However, the system sends only 1000 bits. We can say that the link utilization is only 1000/20,000, or 5 percent. For this reason, for a link with a high bandwidth or long delay, the use of Stop-and-Wait ARQ wastes the capacity of the link.

Example 1.5
What is the utilization percentage of the link in Example 1.4 if we have a protocol that can send up to 15 frames before stopping and worrying about the acknowledgments?
Solution
The bandwidth-delay product is still 20,000 bits. The system can send up to 15 frames or 15,000 bits during a round trip. This means the utilization is 15,000/20,000, or 75 percent. Of course, if there are damaged frames, the utilization percentage is much less because frames have to be resent.

Pipelining
In networking and in other areas, a task is often begun before the previous task has ended. This is known as pipelining. Pipelining does apply to our next two protocols because several frames can be sent before we receive news about the previous frames. Pipelining improves the efficiency of the transmission if the number of bits in transition is large with respect to the bandwidth-delay product.

4.2.2.Go-Back-N Automatic Repeat Request

To improve the efficiency of transmission (filling the pipe), multiple frames must be in transition while waiting for acknowledgment. In other words, we need to let more than one frame be outstanding to keep the channel busy while the sender is waiting for acknowledgment. In Go-Back-N Automatic Repeat Request protocol we can send several frames before receiving acknowledgments; we keep a copy of these frames until the acknowledgments arrive.

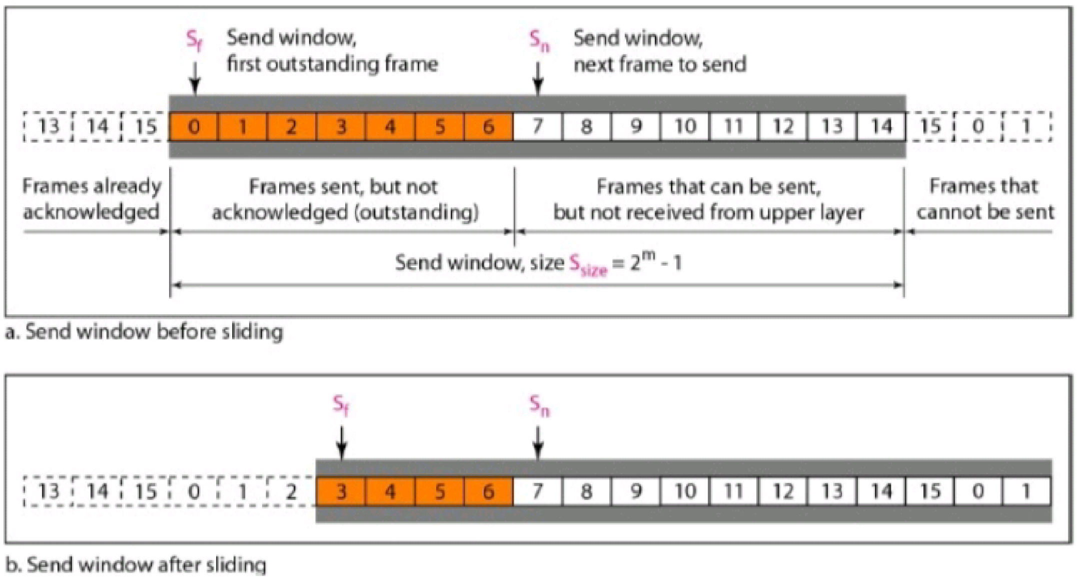
Sequence Numbers

Frames from a sending station are numbered sequentially. If the header of the frame allows m bits for the sequence number, the sequence numbers range from 0 to $2^m - 1$. For example, if m is 4, the only sequence numbers are 0 through 15 inclusive. However, we can repeat the sequence. So the sequence numbers are
0, 1,2,3,4,5,6, 7,8,9, 10, 11, 12, 13, 14, 15,0, 1,2,3,4,5,6,7,8,9,10, 11, ...

Sliding Window

In this protocol (and the next), the sliding window is an abstract concept that defines the range of sequence numbers that is the concern of the sender and receiver. In other words, the sender and receiver need to deal with only part of the possible sequence numbers. The range which is the concern of the sender is called the send sliding window; the range that is the concern of the receiver is called the receive sliding window.

- **The send window** is an imaginary box covering the sequence numbers of the data frames which can be in transit. In each window position, some of these sequence numbers define the frames that have been sent; others define those that can be sent. The maximum size of the window is $2^m - 1$ for reasons that we discuss later. Figure 1.6 shows a sliding window of size 15 *(m* =4).
- The window at any time divides the possible sequence numbers into four regions:
    - The first region, from the far left to the left wall of the window, defines the sequence numbers belonging to frames that are already acknowledged. The sender does not worry about these frames and keeps no copies of them.
    - The second region, colored in Figure 1.5a, defines the range of sequence numbers belonging to the frames that are sent and have an unknown status. The sender needs to wait to find out if these frames have been received or were lost. We call these outstanding frames.
    - The third range, white in the figure, defines the range of sequence numbers for frames that can be sent; however, the corresponding data packets have not yet been received from the network layer.
    - Finally, the fourth region defines sequence numbers that cannot be used until the window slides, as we see next.
- The window itself is an abstraction; three variables define its size and location at any time:
    - S_f(*send* window, the first outstanding frame),
    - S_n (send window, the next frame to be sent), and
    - $S_s$ize (send window, size).



S_f Send window, first outstanding frame
S_n Send window, next frame to send
13 14 15 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 0 1
Frames already acknowledged
Frames sent, but not acknowledged (outstanding)
Frames that can be sent, but not received from upper layer
Frames that cannot be sent
Send window, size $S_{size} = 2^m - 1$
a. Send window before sliding
S_f
S_n
13 14 15 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 0 1
b. Send window after sliding

Figure 1.6 Send window for Go-Back-NARQ

Figure 1.6b shows how a send window can slide one or more slots to the right when an acknowledgment arrives from the other end. Acknowledgments in this protocol are cumulative, meaning that more than one frame can be acknowledged by an ACK frame. In Figure 1.6b, frames 0, 1, and 2 are acknowledged, so the window has slide to the right three slots. Note that the value of S_f is 3 because frame 3 is now the first outstanding frame.

The receive window makes sure that the correct data frames are received and that the correct acknowledgments are sent. The size of the receive window is always 1. The receiver is always looking for the arrival of a specific frame. Any frame arriving out of order is discarded and needs to be resent. Figure 1.7 shows the receive window.

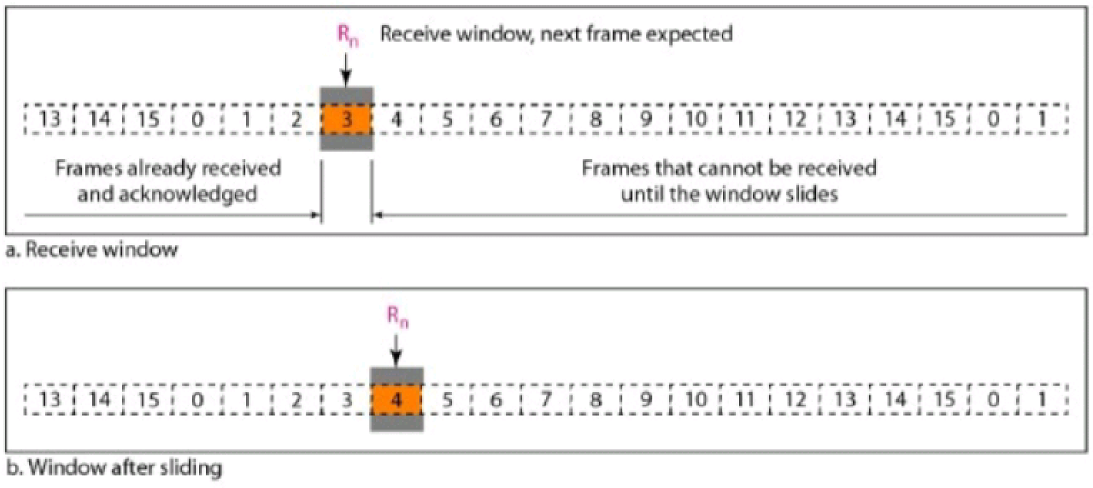


Figure 1.7 Receive window for Go-Back-NARQ

The sequence numbers to the left of the receive window belong to the frames already received and acknowledged; the sequence numbers to the right of this window define the frames that cannot be received. Any received frame with a sequence number in these two regions is discarded. Only a frame with a sequence number matching the value of *Rn* is accepted and acknowledged. The receive window also slides, but only one slot at a time. When a correct frame is received (and a frame is received only one at a time), the window slides.

Timers
Although there can be a timer for each frame that is sent, in this protocol only one timer is used as the timer for the first outstanding frame always expires first; we send all outstanding frames when this timer expires.

Acknowledgment
The receiver sends a positive acknowledgment if a frame has arrived safe and sound and in order. If a frame is damaged or is received out of order, the receiver is silent and will discard all subsequent frames until it receives the one it is expecting. The silence of the receiver causes the timer of the unacknowledged frame at the sender site to expire. This, in turn, causes the sender to go back and resend all frames, beginning with the one with the expired timer. The receiver does not have to acknowledge each frame received. It can send one cumulative acknowledgment for several frames.

Resending a Frame
When the timer expires, the sender resends all outstanding frames. For *example*, suppose the sender has already sent frame 6, but the timer for frame 3 expires. This means that frame 3 has not been acknowledged; the sender goes back and sends frames 3, 4, 5, and 6 again. That is why the protocol is called *Go-Back-N* ARQ.

Design
The idea is similar to Stop-and-Wait ARQ; the difference is that the send window allows us to have as many frames in transition as there are slots in the send window.

Send Window Size
We can now show why the size of the send window must be less than 2^m. As an example, we choose m=2, which means the size of the window can be 2^m- 1, or 3. Figure 1.8 compares a window size of 3 against a window size of 4. If the size of the window is 3 (less than 2^2) and all three acknowledgments are lost, the frame 0 timer expires and all three frames are resent. The receiver is now expecting frame 3, not frame 0, so the duplicate frame is correctly discarded. On the other hand, if the size of the window is 4 (equal to 2^2) and all acknowledgments are lost, the sender will send a duplicate of frame 0. However, this time the window of the receiver expects to receive frame 0, so it accepts frame 0, not as a duplicate, but as the first frame in the next cycle. This is an error.

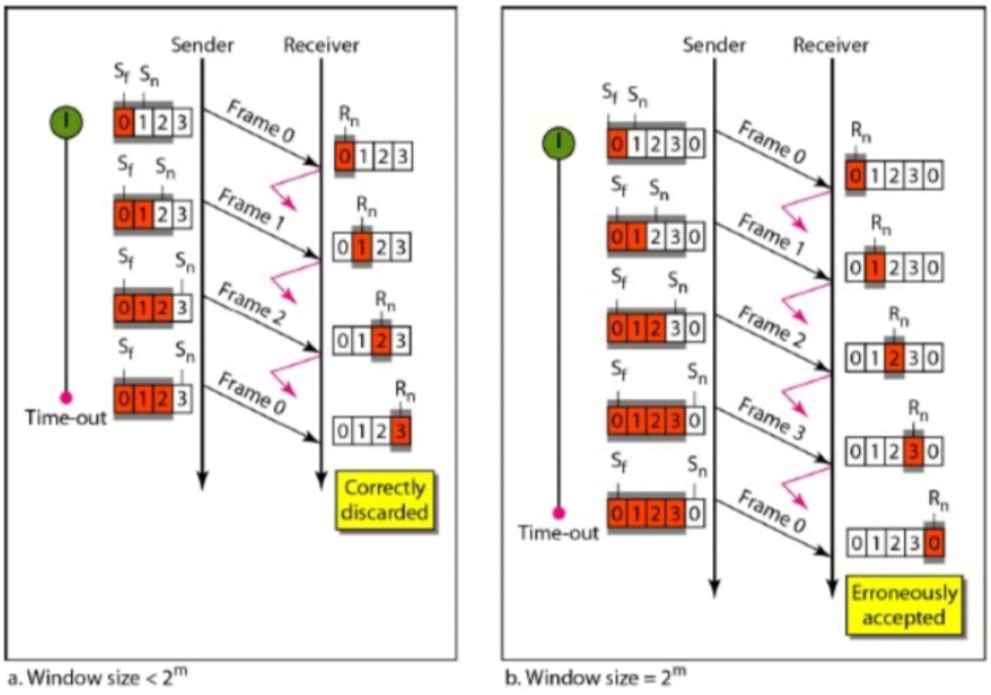


Figure 1.8 Window size for Go-Back-NARQ

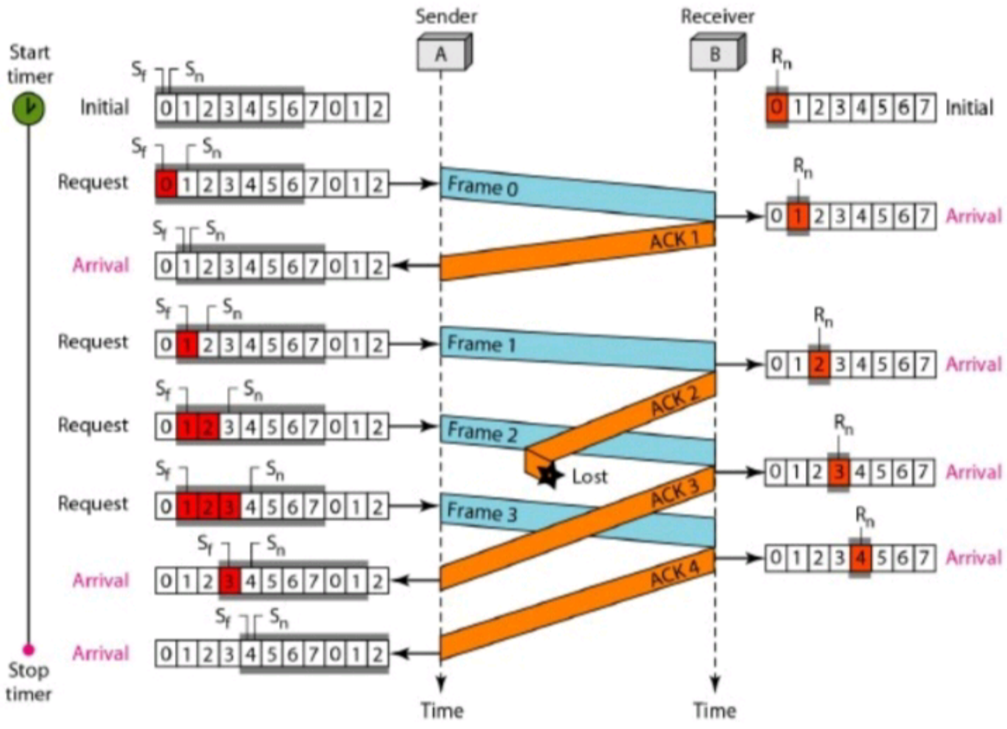


Figure 1.9 Flow diagram for Example 1.6

Example 1.6
Figure 1.9 shows an example of Go-Back-*N*. This is an example of a case where the forward channel is reliable, but the reverse is not. No data frames are lost, but some ACKs are delayed and one is lost. The example also shows how cumulative acknowledgments can help if acknowledgments are delayed or lost. After initialization, there are seven sender events. Request events are triggered by data from the network layer; arrival events are triggered by acknowledgments from the physical layer. There is no time-out event here because all outstanding frames are acknowledged before the timer expires. Note that although ACK 2 is lost, ACK 3 serves as both ACK 2 and ACK3.

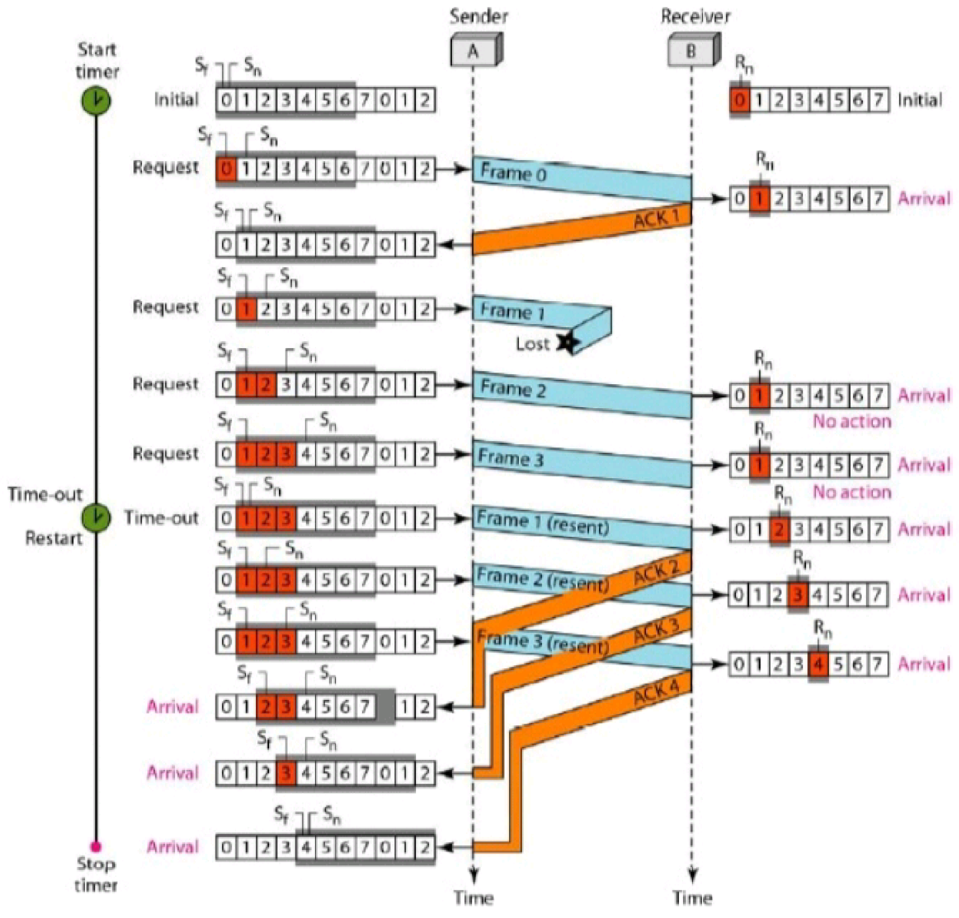


Figure 1.10 Flow diagram for Example 1.7

Example 1.7
Figure 1.10 shows what happens when a frame is lost. Frames 0, 1, 2, and 3 are sent. However, frame 1 is lost. The receiver receives frames 2 and 3, but they are discarded because they are received out of order (frame 1 is expected). The sender receives no acknowledgment about frames 1, 2, or 3. Its timer finally expires. The sender sends all outstanding frames (1, 2, and 3) because it does not know what is wrong. Note that the resending of frames 1, 2, and 3 is the response to one single event. When the sender is responding to this event, it cannot accept the triggering of other events. This means that when ACK 2 arrives, the sender is still busy with sending frame 3. The physical layer must wait until this event is completed and the data link layer goes back to its sleeping state. We have shown a vertical line to indicate the delay. It is the same story with ACK 3; but when ACK 3 arrives, the sender is busy responding to ACK 2. It happens again when ACK 4 arrives.

Note that before the second timer expires, all outstanding frames have been sent and the timer is stopped.

Go-Back-N ARQ versus Stop-and- Wait ARQ

Stop-and-Wait ARQ Protocol is actually a *Go-Back-N* ARQ in which there are only two sequence numbers and the send window size is 1. In other words, $m = 1$, $2^m - 1 = 1$.

Limitation of Go-Back-N ARQ

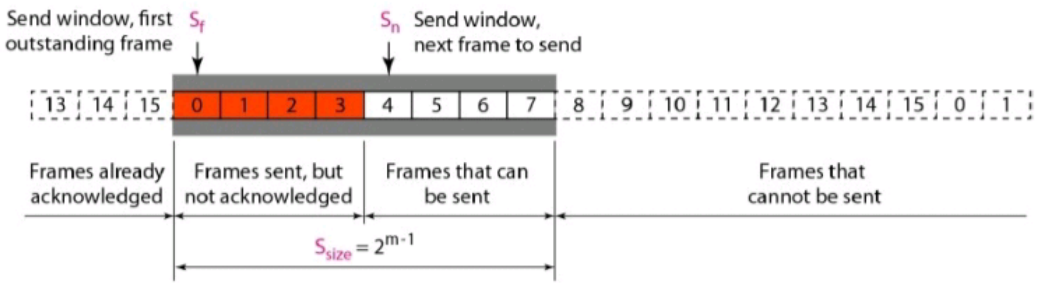
Go-Back-N ARQ simplifies the process at the receiver site. The receiver keeps track of only one variable, and there is no need to buffer out-of-order frames; they are simply discarded. However, this protocol is very inefficient for a noisy link. In a noisy link a frame has a higher probability of damage, which means the resending of multiple frames. This resending uses up the bandwidth and slows down the transmission.

4.2.3. Selective Repeat Automatic Repeat Request

For noisy links, there is another mechanism that does not resend *N* frames when just one frame is damaged; only the damaged frame is resent. This mechanism is called Selective Repeat ARQ. It is more efficient for noisy links, but the processing at the receiver is more complex.

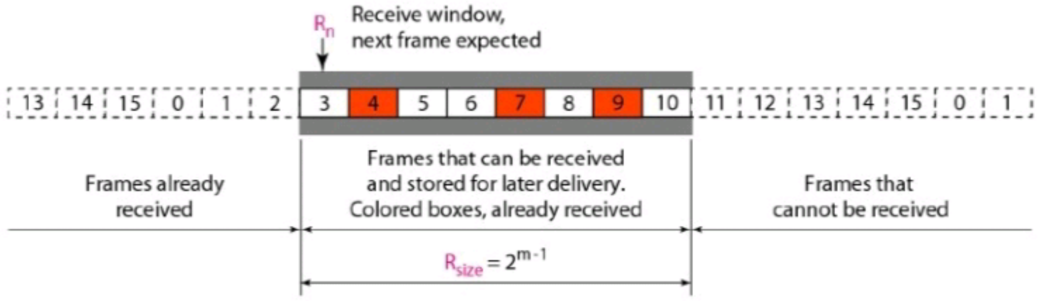
Windows

The Selective Repeat Protocol also uses two windows: a send window and a receive window. However, there are differences between the windows in this protocol and the ones in Go-Back-N. First, the size of the send window is at most one half of 2^m *i.e.* $2^m/2=2^{m-1}$. The reason for this will be discussed later. Second, the receive window is the same size as the send window. For example, if $m = 4$, the sequence numbers go from 0 to 15, but the size of the window is just 8 (it is 15 in the *Go-Back-N* Protocol). The smaller window size means less efficiency in filling the pipe, but the fact that there are fewer duplicate frames can compensate for this. The protocol uses the same variables as we discussed for Go-Back-N. We show the Selective Repeat send window in Figure 1.11 to emphasize the size.



Send window, first outstanding frame
S_f
S_n
Send window, next frame to send
13 14 15 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 0 1
Frames already acknowledged
Frames sent, but not acknowledged
Frames that can be sent
Frames that cannot be sent
$S_{size} = 2^{m-1}$

Figure 1.11 Send window for Selective Repeat ARQ



Rn
Receive window, next frame expected
13 14 15 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 0 1
Frames already received
Frames that can be received and stored for later delivery. Colored boxes, already received
Frames that cannot be received
Rsize = 2m-1

Figure 1.12 Receive window for Selective Repeat ARQ

The receive window in Selective Repeat is totally different from the one in GoBack-N. First, the size of the receive window is the same as the size of the send window *(2^{m-1})*. The Selective Repeat Protocol allows as many frames as the size of the receive window to arrive out of order and be kept until there is a set of in-order frames to be delivered to the network layer. Because the sizes of the send window and receive window are the same, all the frames in the send frame can arrive out of order and be stored until they can be delivered. We need, however, to mention that the receiver never delivers packets out of order to the network layer. Figure 1.12 shows the receive window in this protocol. Those slots inside the window that are colored define frames that have arrived out of order and are waiting for their neighbors to arrive before delivery to the network layer.

Window Sizes

We can now show why the size of the sender and receiver windows must be at most one half of 2^m. For an example, we choose m = 2, which means the size of the window is $2^m/2$, or 2. Figure 1.13 compares a window size of 2 with a window size of 3. If the size of the window is 2 and all acknowledgments are lost, the timer for frame 0 expires and frame 0 is resent. However, the window of the receiver is now expecting frame 2, not frame 0, so this duplicate frame is correctly discarded. When the size of the window is 3 and all acknowledgments are lost, the sender sends a duplicate of frame O. However, this time, the window of the receiver expects to receive frame 0 (0 is part of the window), so it accepts frame 0, not as a duplicate, but as the first frame in the next cycle. This is clearly an error.

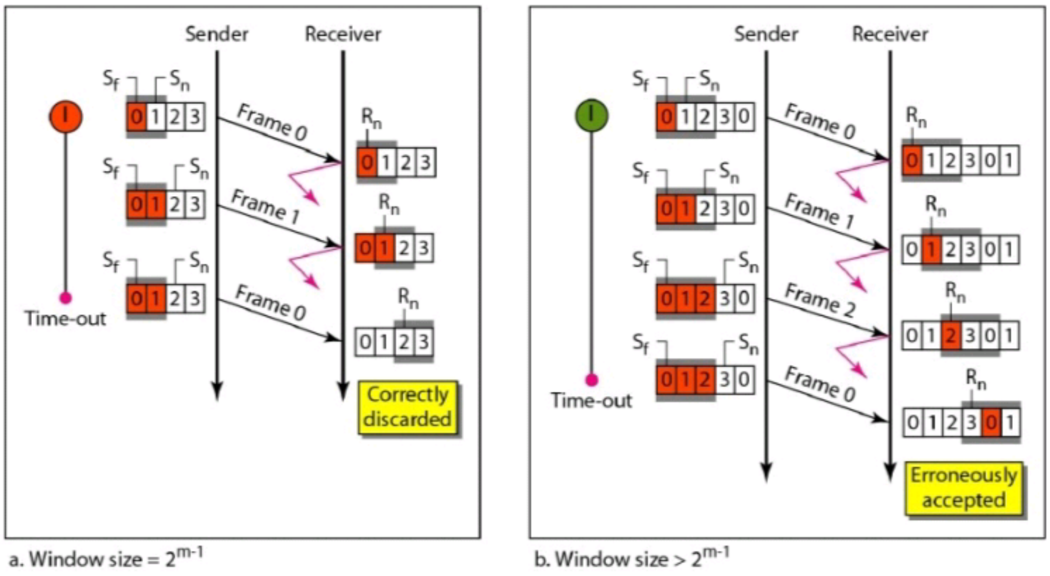


Figure 1.13 Selective Repeat ARQ, window size

***Example* 1.8**
This example is similar to Example 1.3 in which frame 1 is lost. We show how Selective Repeat behaves in this case. Figure 1.14 shows the situation. One main difference is the number of timers. Here, each frame sent or resent needs a timer, which means that the timers need to be numbered (0, 1,2, and 3). The timer for frame 0 starts at the first request, but stops when the ACK for this frame arrives. The timer for frame 1 starts at the second request, restarts when a NAK arrives, and finally stops when the last ACK arrives. The other two timers start when the corresponding frames are sent and stop at the last arrival event.

At the receiver site we need to distinguish between the acceptance of a frame and its delivery to the network layer. At the second arrival, frame 2 arrives and is stored and marked (colored slot), but it cannot be delivered because frame 1 is missing. At the next arrival, frame 3 arrives and is marked and stored, but still none of the frames can be delivered. Only at the last arrival, when finally a copy of frame 1 arrives, can frames 1, 2, and 3 be delivered to the network layer. There are two conditions for the delivery of frames to the network layer: First, a set of consecutive frames must have arrived. Second, the set starts from the beginning of the window. After the first arrival, there was only one frame and it started from the beginning of the window. After the last arrival, there are three frames and the first one starts from the beginning of the window.

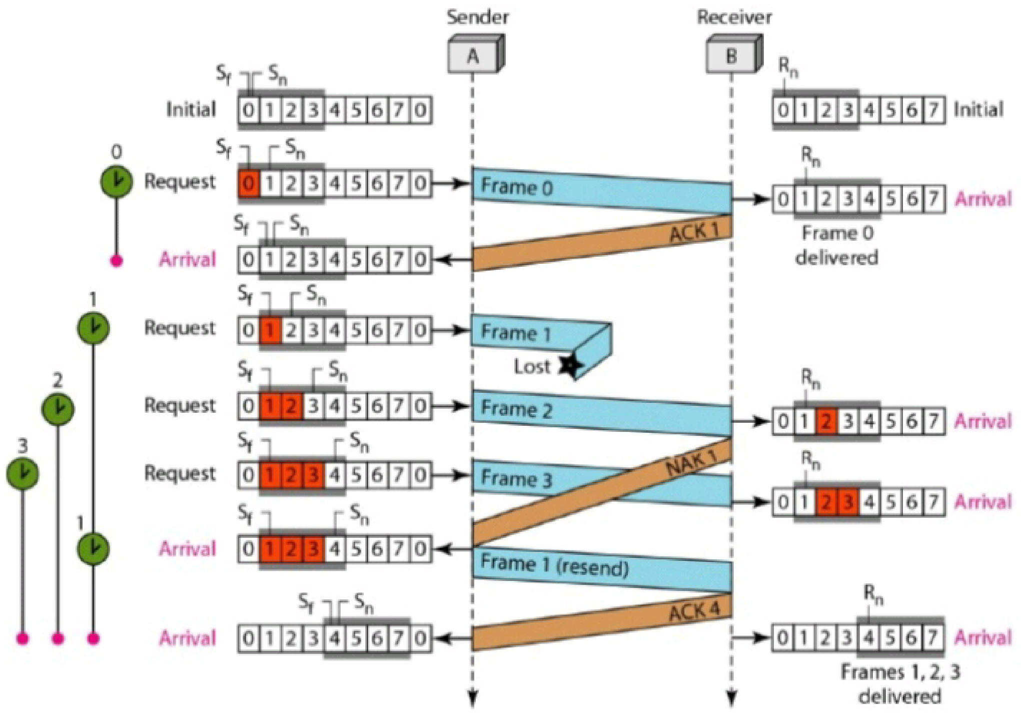


Figure 1.14 Flow diagram for Example 1.8

Another important point is that a NAK is sent after the second arrival, but not after the third, although both situations look the same. The reason is that the protocol does not want to crowd the network with unnecessary NAKs and unnecessary resent frames. The second NAK would still be NAK 1 to inform the sender to resend frame 1 again; this has already been done. The first NAK sent is remembered and is not sent again until the frame slides. A NAK is sent once for each window position and defines the first slot in the window.

The next point is about the ACKs. Notice that only two ACKs are sent here. The first one acknowledges only the first frame; the second one acknowledges three frames. In Selective Repeat, ACKs are sent when data are delivered to the network layer. If the data belonging to *n* frames are delivered in one shot, only one ACK is sent for all of them.

Piggybacking

The three protocols we discussed in this section are all unidirectional: data frames flow in only one direction although control information such as ACK and NAK frames can travel in the other direction. In real life, data frames are normally flowing in both directions: from node A to node B and from node B to node A. This means that the control information also needs to flow in both directions. A technique called **piggybacking** is used to improve the efficiency of the bidirectional protocols. When a frame is carrying data from A to B, it can also carry control information about arrived (or lost) frames from B; when a frame is carrying data from B to A, it can also carry control information about the arrived (or lost) frames from A.