# Chapter 10: Data Link Control

**Sunghyun Cho**
**School of Computer Science**
**Hanyang University**
**chopro@hanyang.ac.kr**

## Data Link Layer

- Two main functions
  - Data link control
    - manage the design and procedures for communication between two adjacent nodes: node-to-node comm.
    - framing, flow and error control, s/w protocols
  - Media access control
    - how to share the link

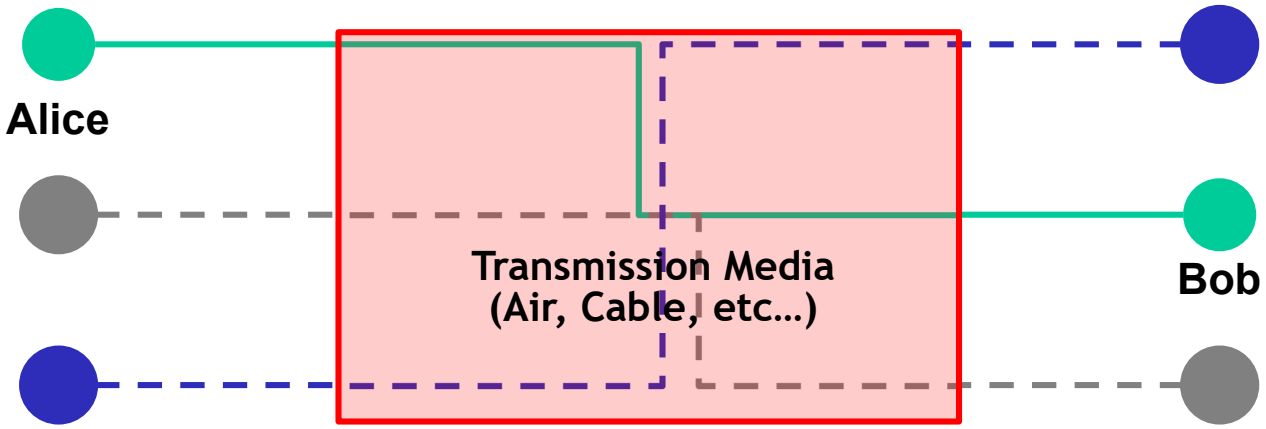
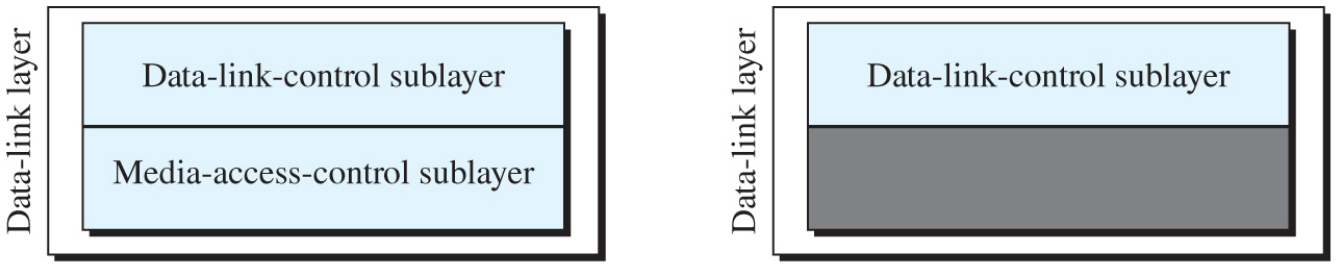


**Figure.** *Dividing the data-link layer into two sublayers*




a. Data-link layer of a broadcast link

b. Data-link layer of a point-to-point link



# 11-1 FRAMING

***The data link layer needs to pack bits into frames, so that each frame is distinguishable from another. Our postal system practices a type of framing. The simple act of inserting a letter into an envelope separates one piece of information from another; the envelope serves as the delimiter.***

***Topics discussed in this section:***

**Fixed-Size Framing**
**Variable-Size Framing**

**Figure.** *Example of frame in data-link layer*

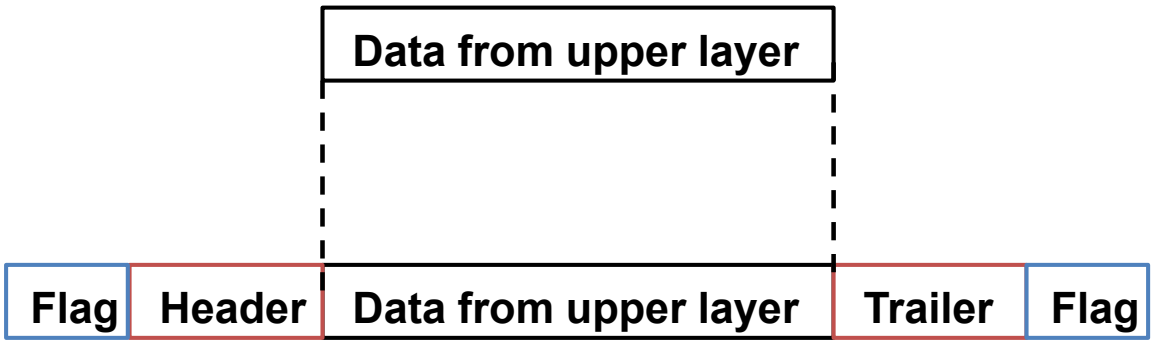
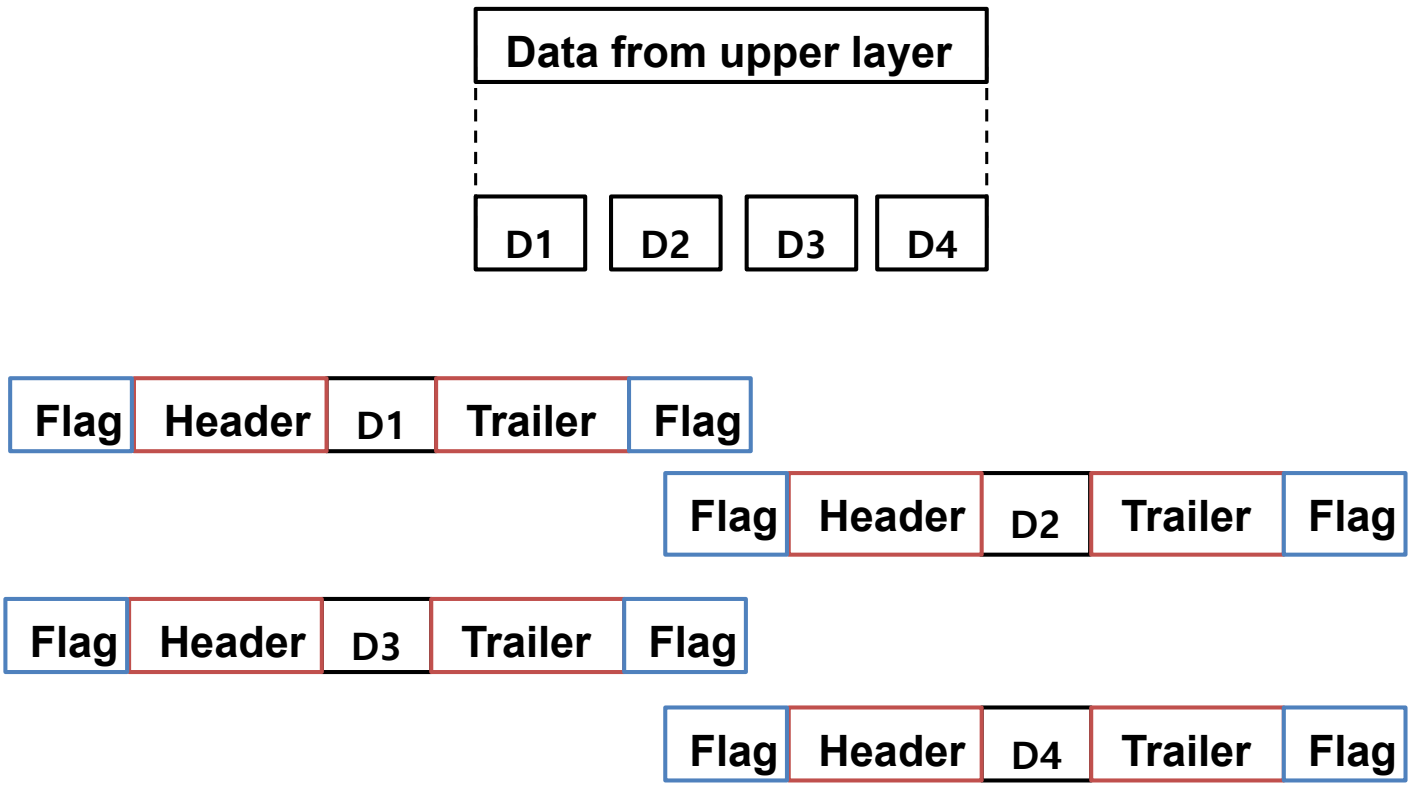





**Figure.** *Example of frame in data-link layer*

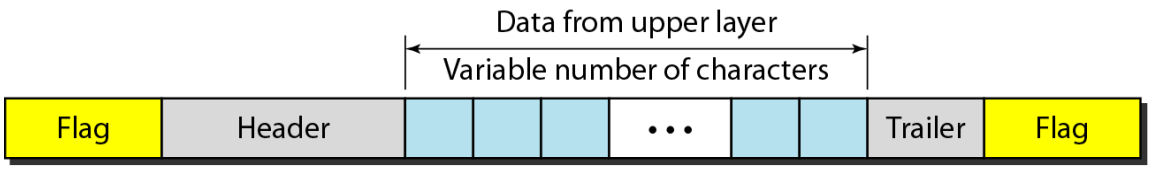


# Fixed vs. Variable Size Framing

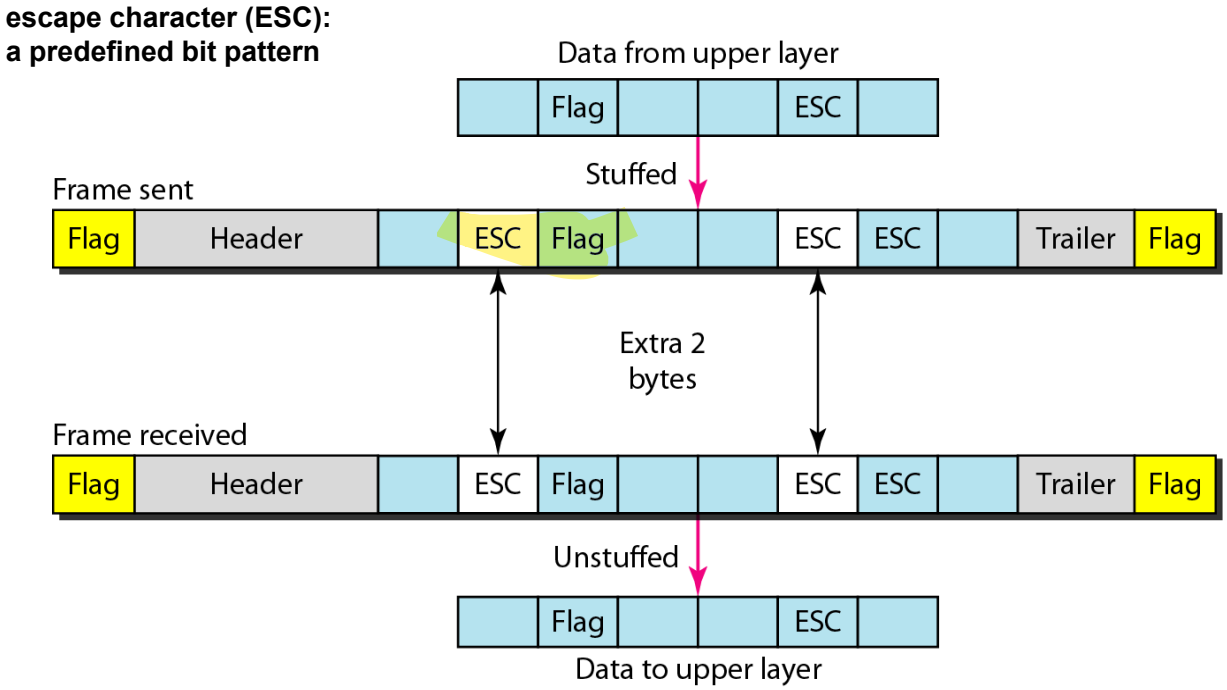
- Fixed-size framing
  - no need for defining the boundaries of the frames
  - ATM (53 octets)
- Variable-size framing
  - need a way to define the end of the frame and the beginning of the next
- Character-oriented protocols
  - byte stuffing
  - Flag: 1 byte pattern added at the beginning and the end of a frame, composed of protocol-dependent special characters
- Bit-oriented protocols
  - bit stuffing



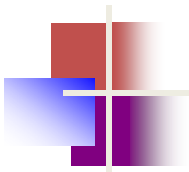
**Figure 11.1** *A frame in a character-oriented protocol*



## Figure 11.2 *Byte stuffing and unstuffing*



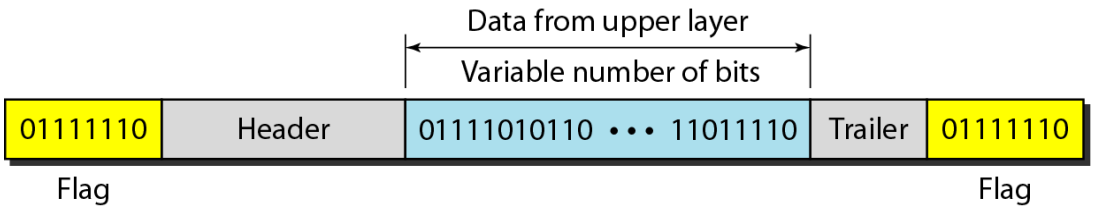


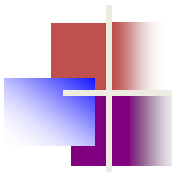
*Note*

**Byte stuffing is the process of adding 1 extra byte whenever there is a flag or escape character in the text.**

## Figure 11.3 *A frame in a bit-oriented protocol*



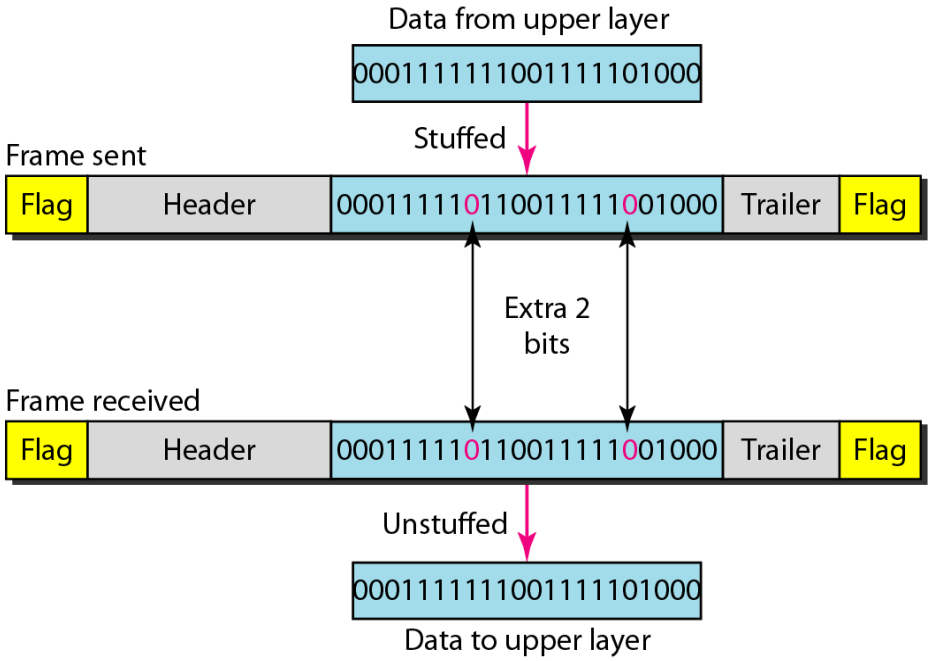


*Note*

**Bit stuffing is the process of adding one extra 0 whenever five consecutive 1s follow a 0 in the data, so that the receiver does not mistake the pattern 0111110 for a flag.**

## Figure 11.4 *Bit stuffing and unstuffing*






# 11-2 FLOW AND ERROR CONTROL

***The most important responsibilities of the data link layer are flow control and error control. Collectively, these functions are known as data link control.***

***Topics discussed in this section:***

**Flow Control**
**Error Control**

Note

**Flow control refers to a set of procedures used to restrict the amount of data that the sender can send before waiting for acknowledgment.**



Note

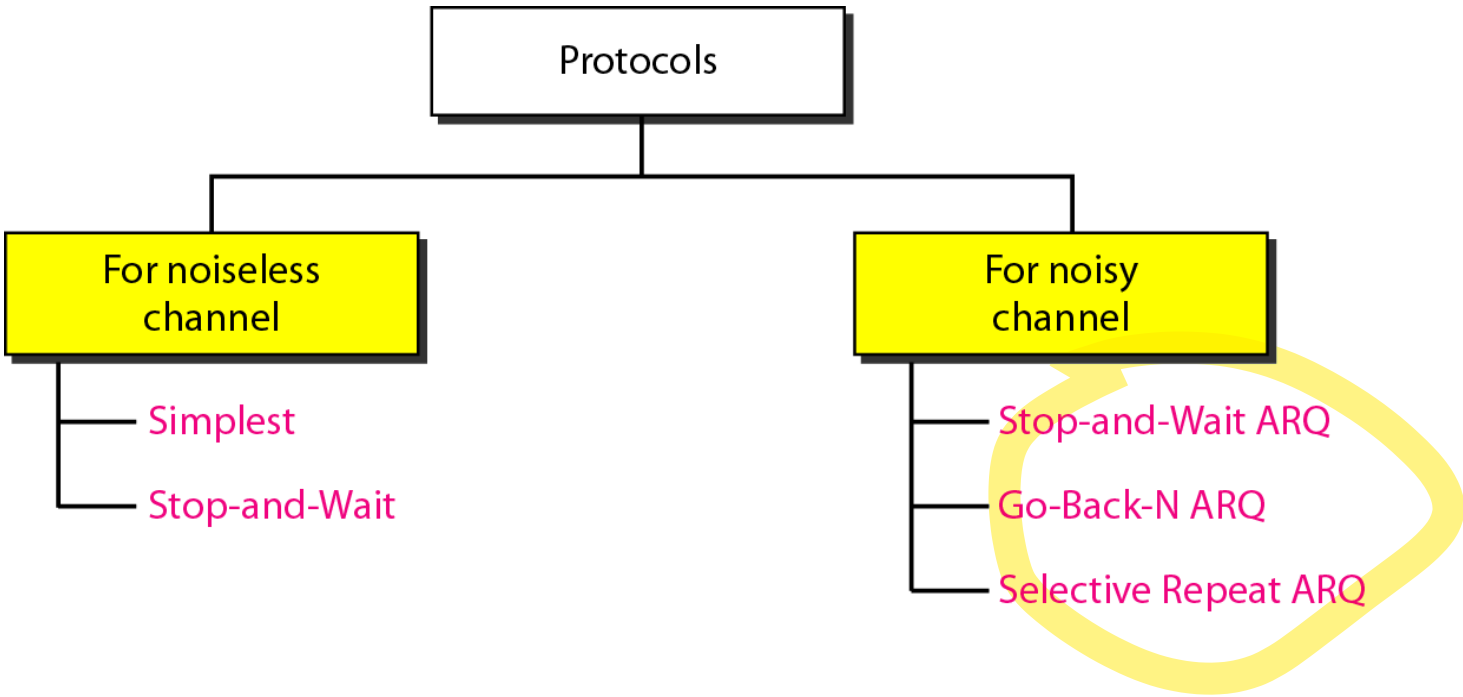
**Error control in the data link layer is based on automatic repeat request, which is the retransmission of data.**

# 11-3 PROTOCOLS

***Now let us see how the data link layer can combine framing, flow control, and error control to achieve the delivery of data from one node to another. The protocols are normally implemented in software by using one of the common programming languages. To make our discussions language-free, we have written in pseudocode a version of each protocol that concentrates mostly on the procedure instead of delving into the details of language rules.***



**Figure 11.5** *Taxonomy of protocols discussed in this chapter*



# 11-4 NOISELESS CHANNELS

***Let us first assume we have an ideal channel in which no frames are lost, duplicated, or corrupted. We introduce two protocols for this type of channel.***

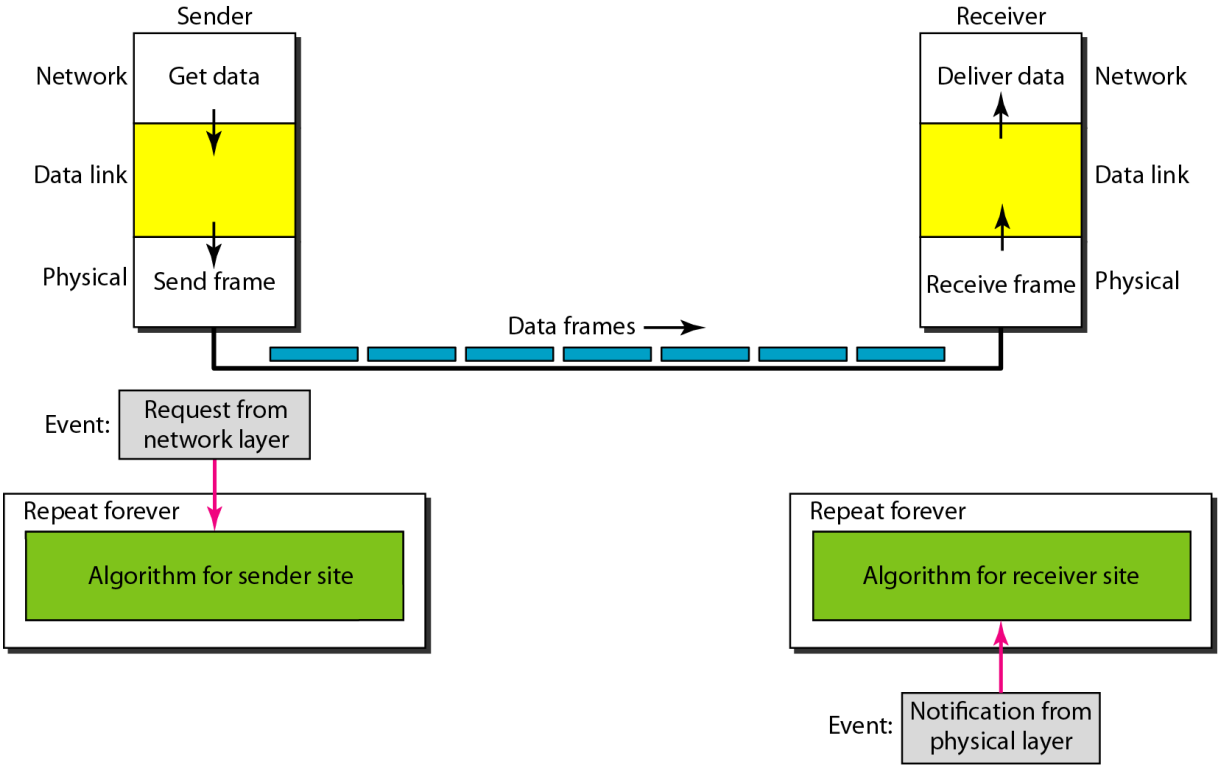
***Topics discussed in this section:***

**Simplest Protocol**
**Stop-and-Wait Protocol**



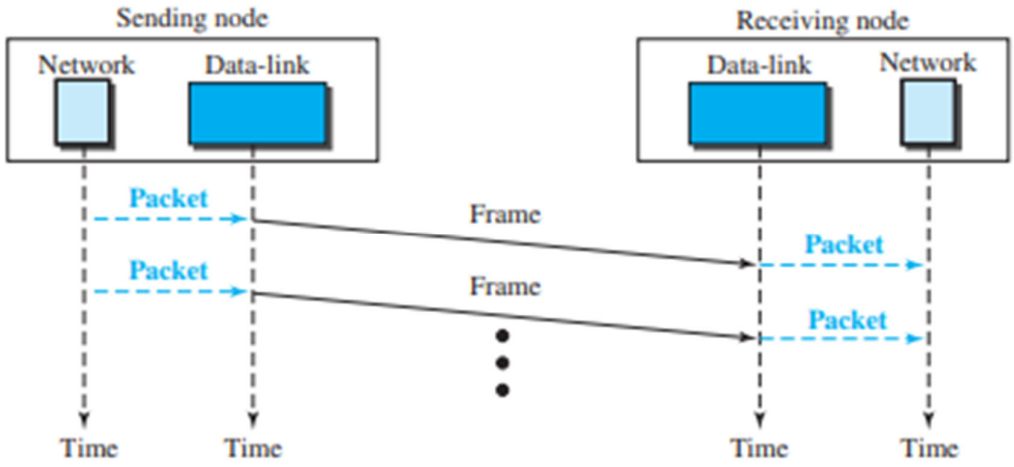
## Simplest Protocol

## Figure 11.6 *The design of the simplest protocol with no flow or error control*





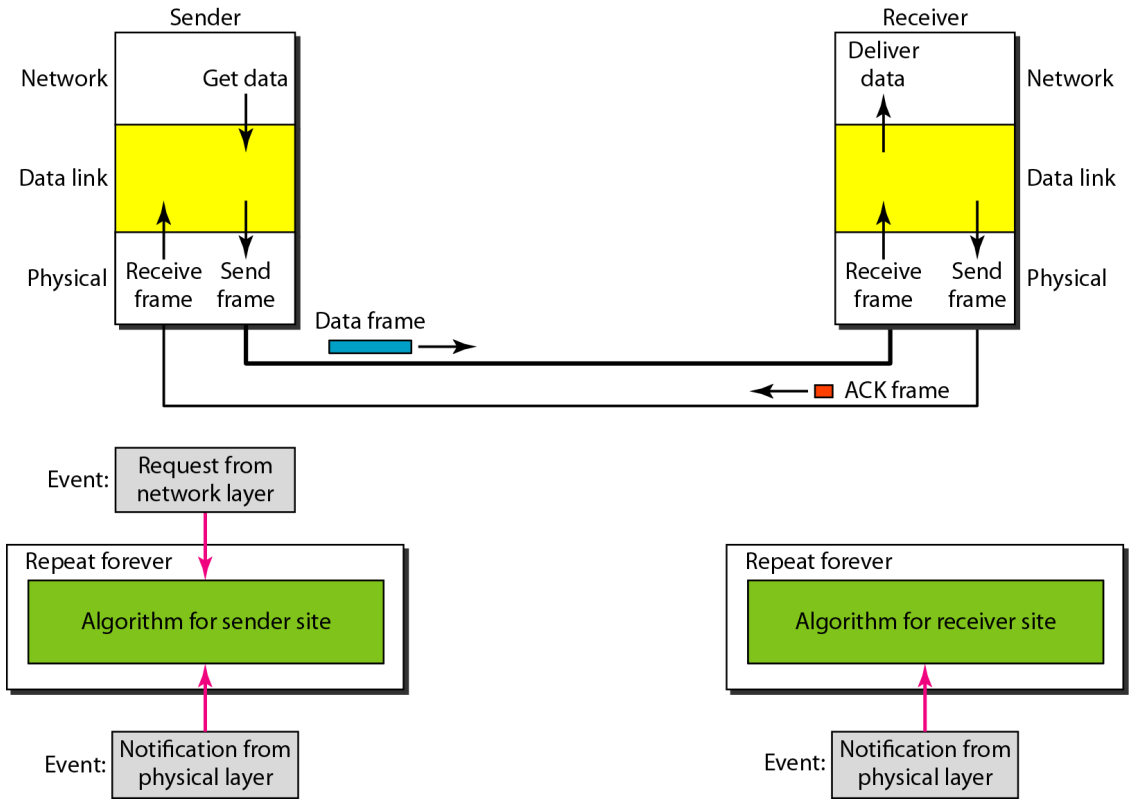

## Figure . *Flow diagram for simplest protocol*



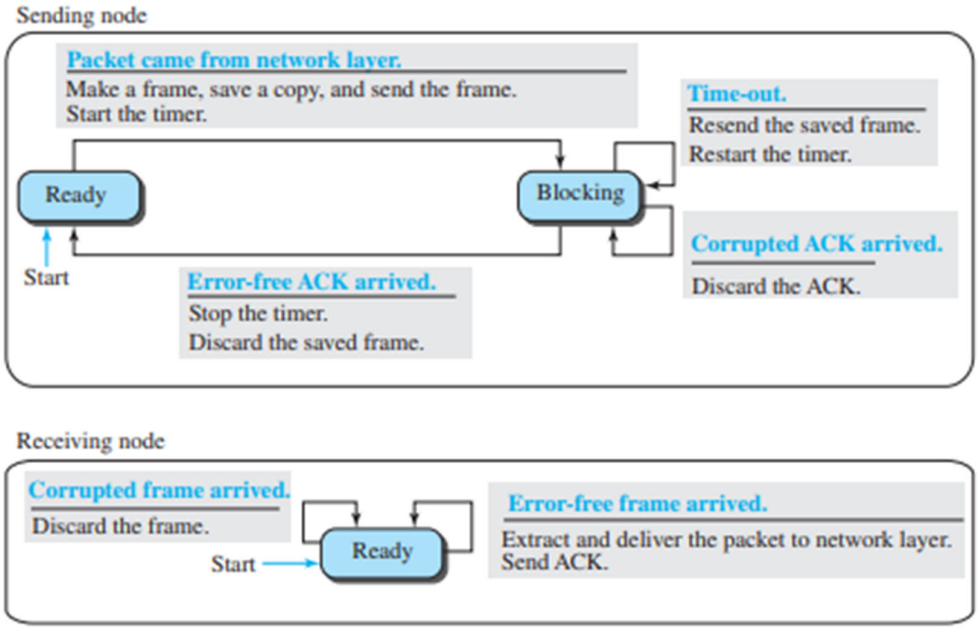
# Stop-and-Wait Protocol



## Figure 11.8 *Design of Stop-and-Wait Protocol*



## Figure 11.8 *State diagram for stop-and-wait protocol*

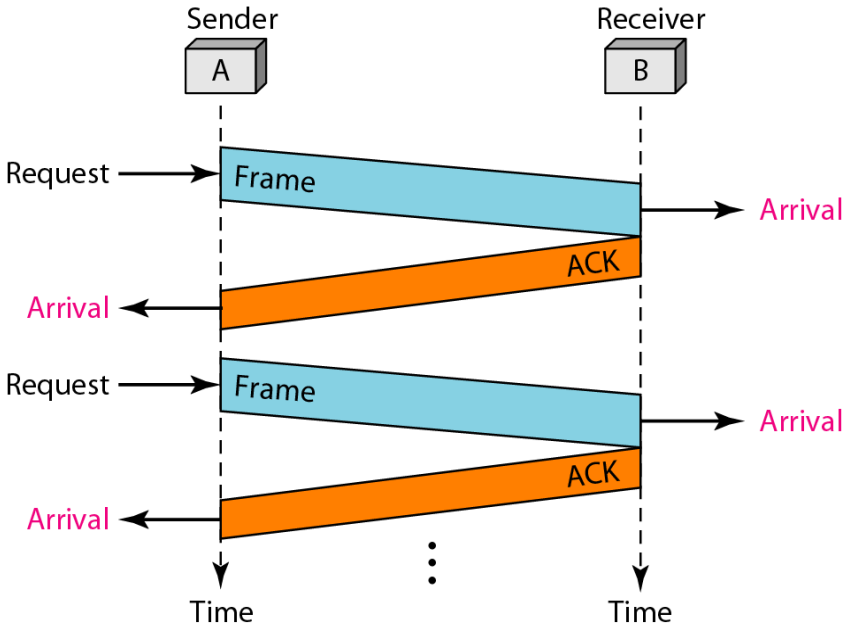





## *Example 11.2*

***Figure 11.9 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. Note that sending two frames in the protocol involves the sender in four events and the receiver in two events.***

**Figure 11.9** *Flow diagram for Example 11.2*






# 11-5 NOISY CHANNELS

***Although the Stop-and-Wait Protocol gives us an idea of how to add flow control to its predecessor, noiseless channels are nonexistent. We discuss three protocols in this section that use error control.***

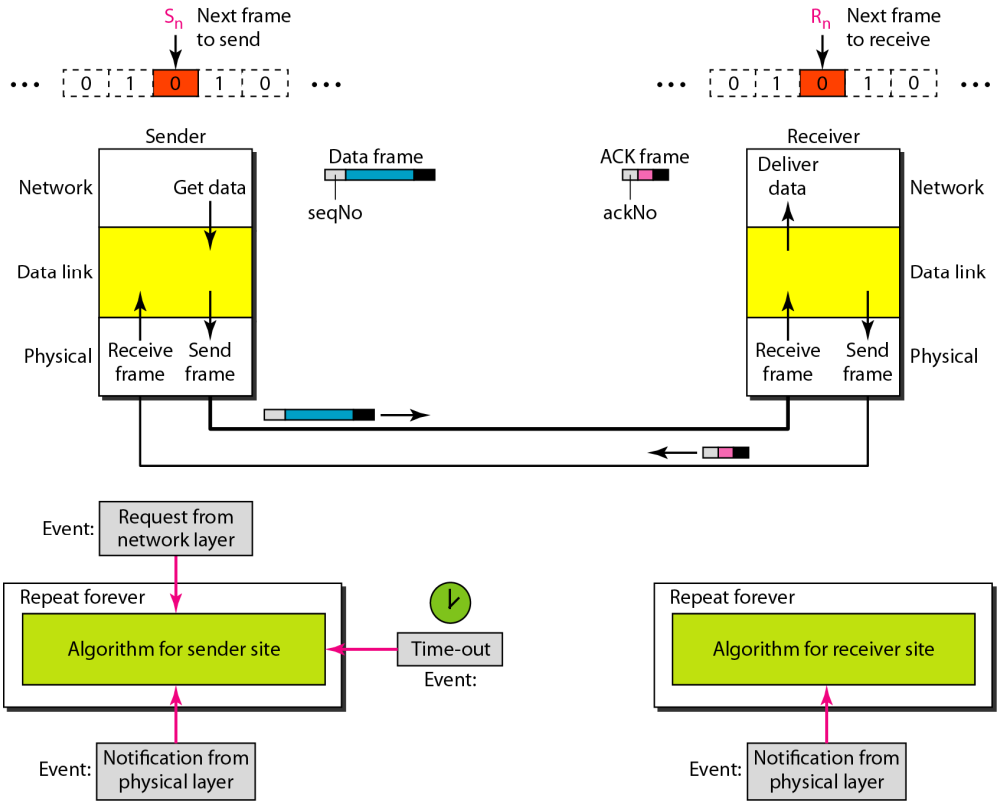
***Topics discussed in this section:***

**Stop-and-Wait Automatic Repeat Request**
**Go-Back-N Automatic Repeat Request**
**Selective Repeat Automatic Repeat Request**

# Stop-and-Wait Protocol



## Figure 11.10 *Design of the Stop-and-Wait ARQ Protocol*

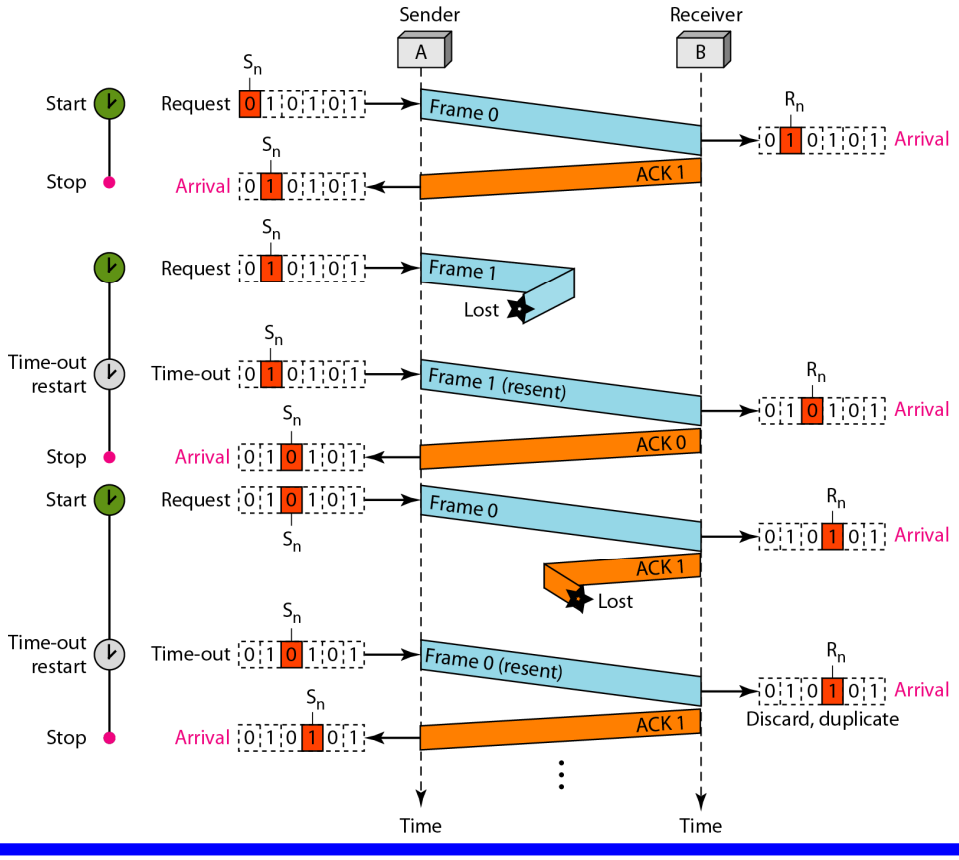


# Example 11.3

*Figure 11.11 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.*



**Figure 11.11** *Flow diagram for Example 11.3*



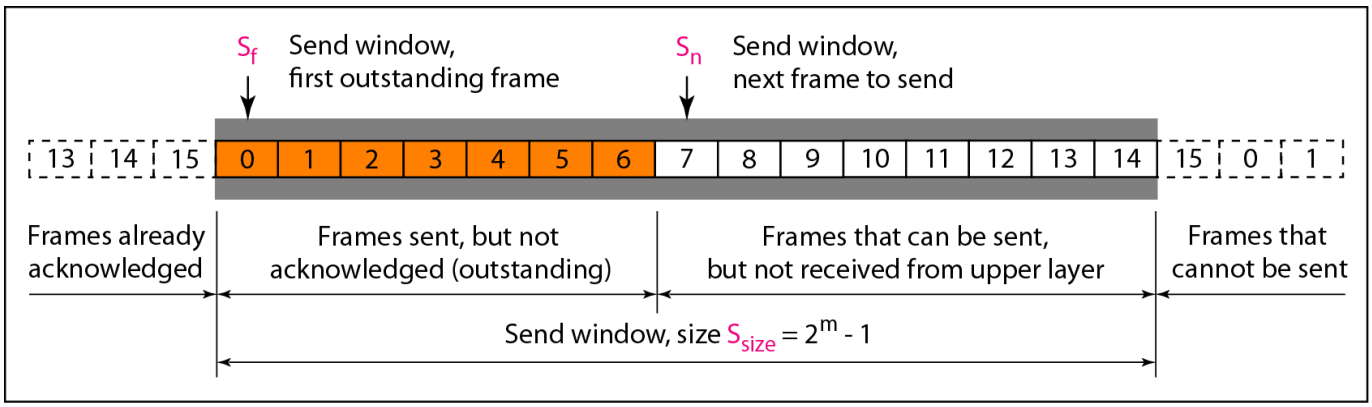
# Go-Back-N ARQ



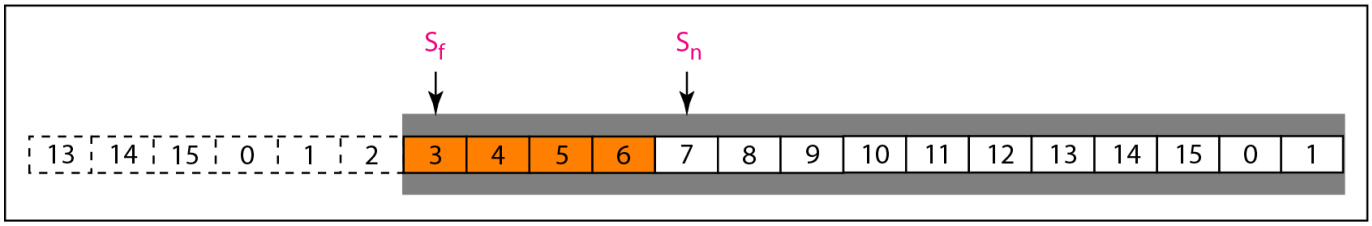
*Note*

**In the Go-Back-N Protocol, the sequence numbers are modulo $2^m$, where m is the size of the sequence number field in bits.**

## Figure 11.12 *Send window for Go-Back-N ARQ*

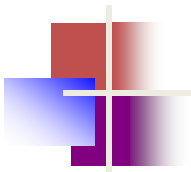



a. Send window before sliding




b. Send window after sliding





**Note**

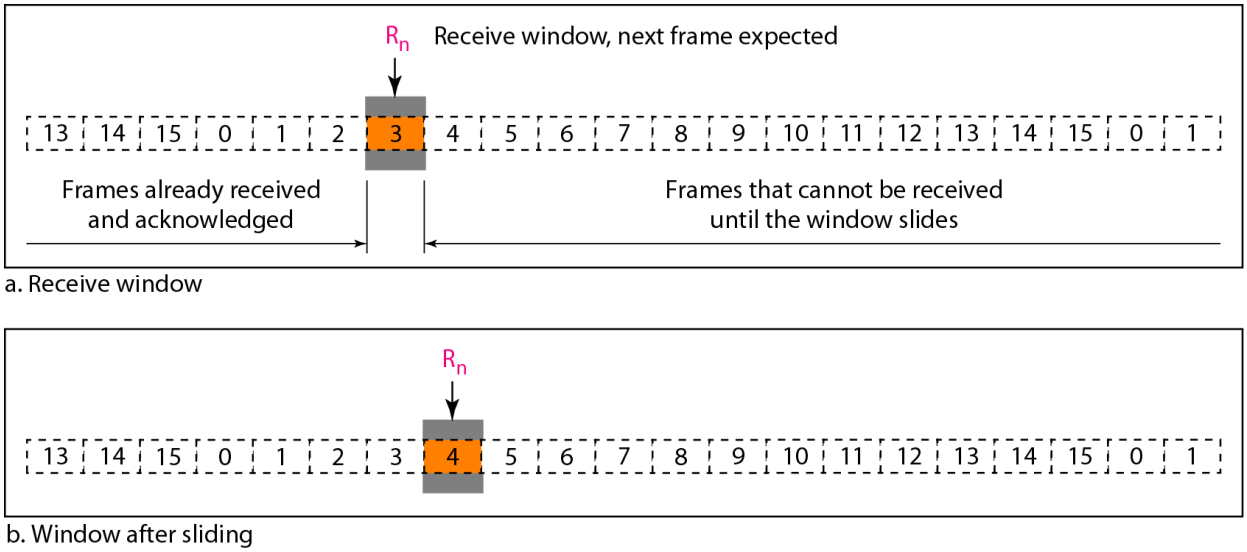
**The send window is an abstract concept defining an imaginary box of size $2^m - 1$ with three variables: $S_f$, $S_n$, and $S_{size}$.**

**Note**

**The send window can slide one or more slots when a valid acknowledgment arrives.**



**Figure 11.13** *Receive window for Go-Back-N ARQ*

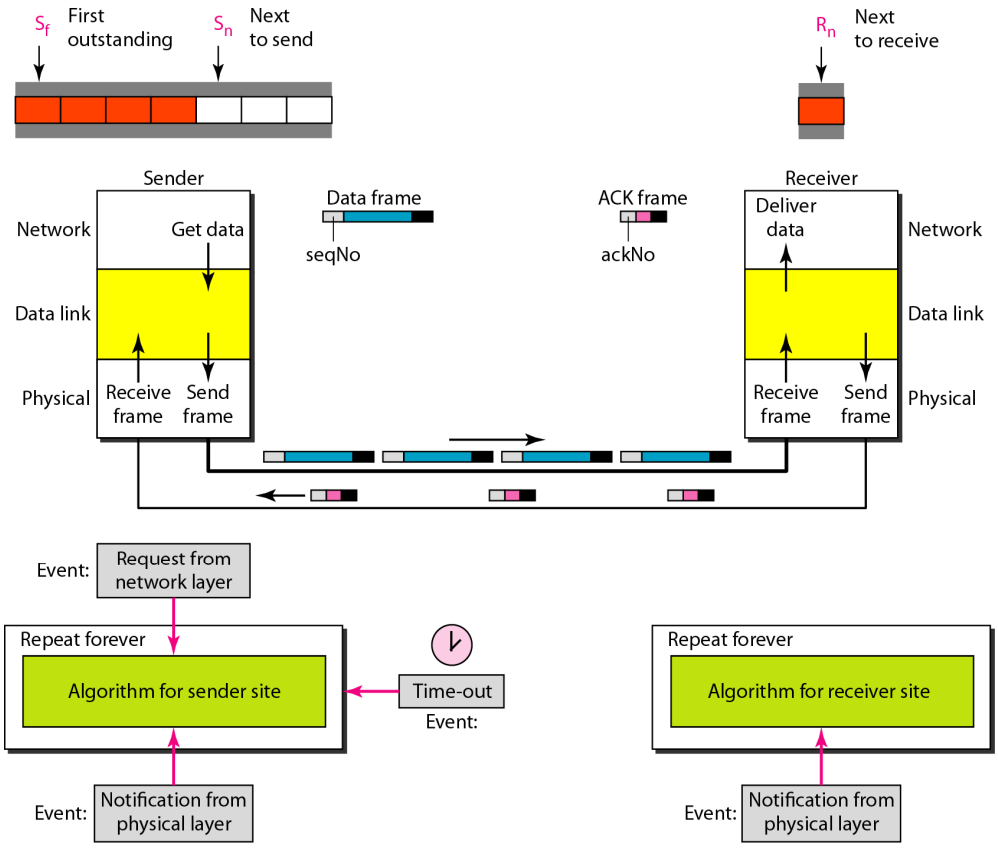


**Note**

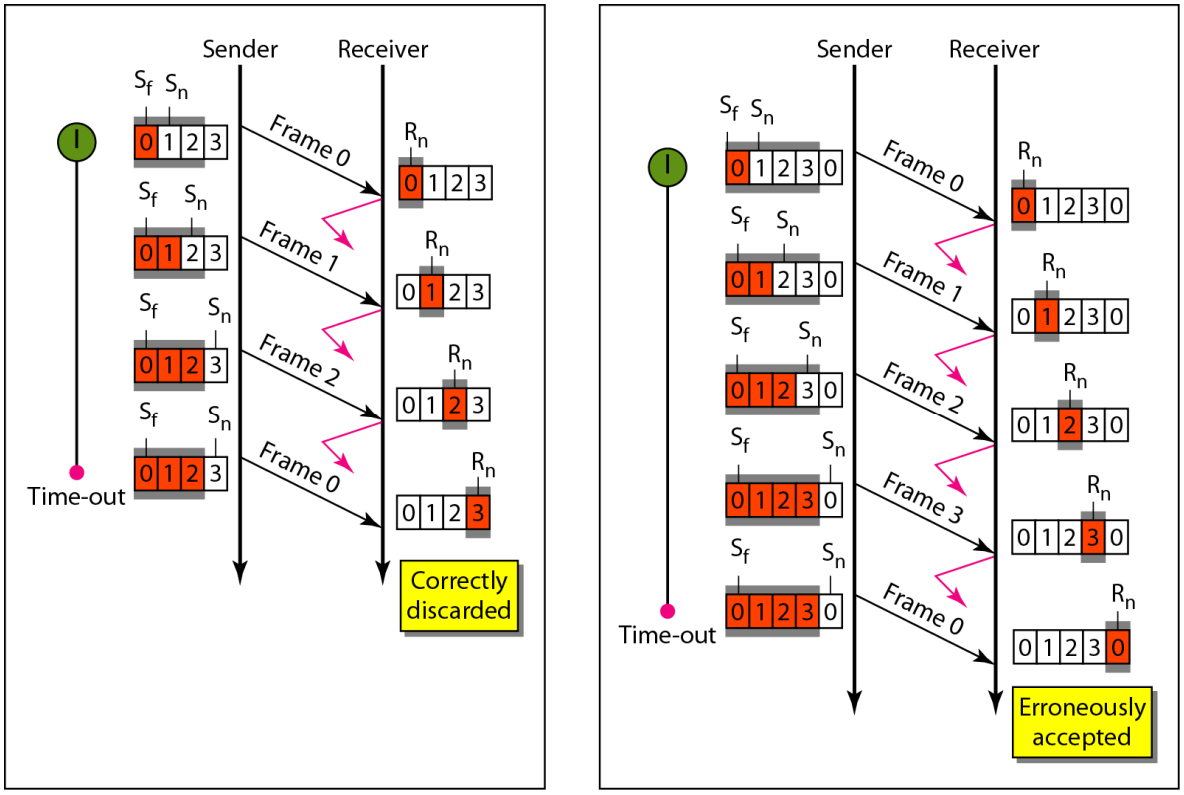
**The receive window is an abstract concept defining an imaginary box of size 1 with one single variable $R_n$. The window slides when a correct frame has arrived; sliding occurs one slot at a time.**



## Figure 11.14 *Design of Go-Back-N ARQ*



## Figure 11.15 *Window size for Go-Back-N ARQ*




a. Window size < $2^m$

b. Window size = $2^m$



*Note*

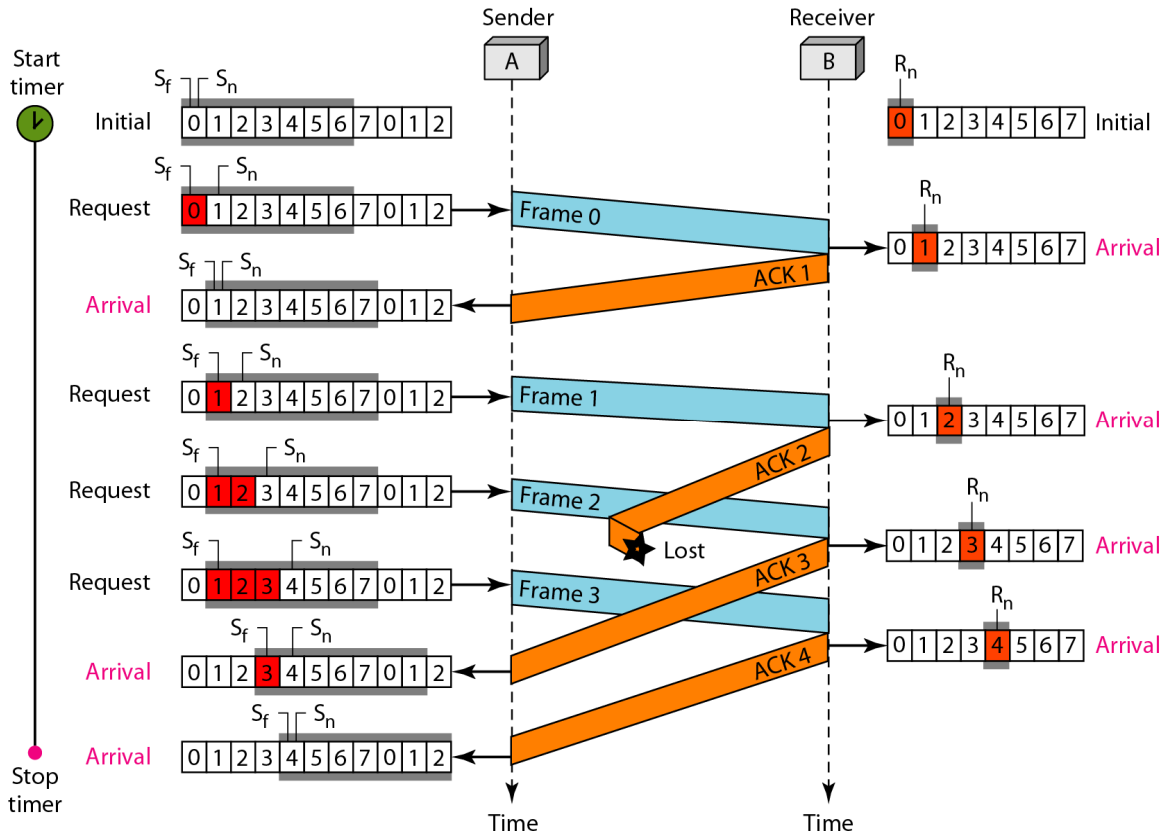
**In Go-Back-N ARQ, the size of the send window must be less than $2^m$; the size of the receiver window is always 1.**

# Example 11.6

*Figure 11.16 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 ACK 3.*



**Figure 11.16** *Flow diagram for Example 11.6*



## Example 11.7

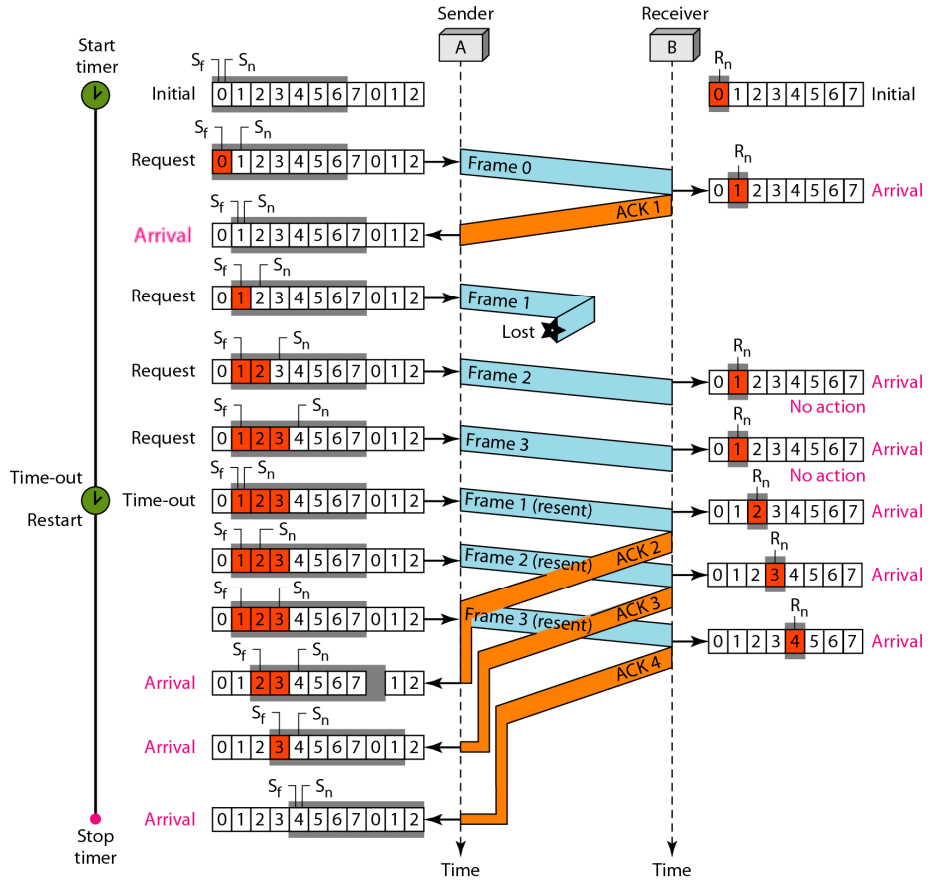
*Figure 11.17 shows what happens when a frame is lost. Frames 0, 1, 2, and 3 are sent. However, frame 1 is lost. The receiver receives frames 2 and 3, but they are discarded because they are received out of order. The sender receives no acknowledgment about frames 1, 2, or 3. Its timer finally expires. 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.*



## Example 11.7 (continued)

*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.*

## Figure 11.17 *Flow diagram for Example 11.7*






*Note*

**Stop-and-Wait ARQ is a special case of Go-Back-N ARQ in which the size of the send window is 1.**

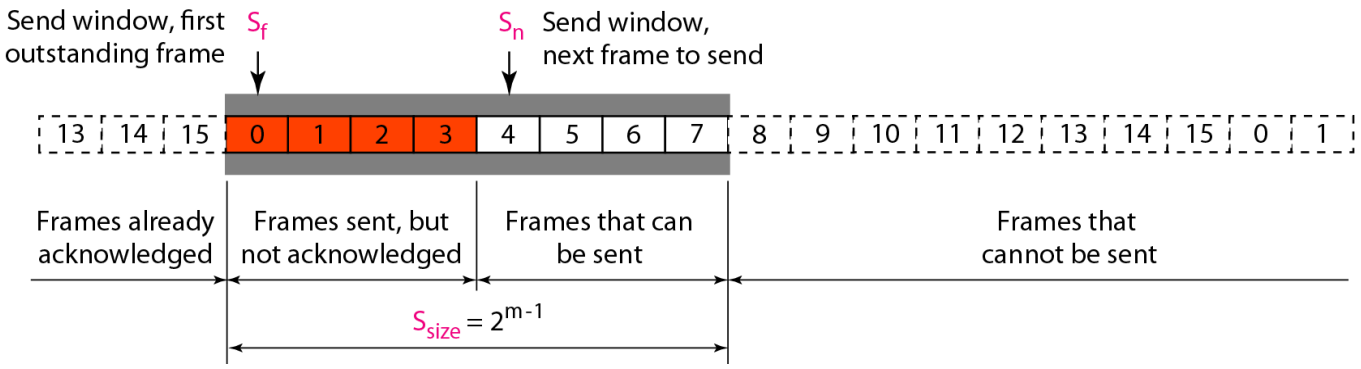
# Selective Repeat ARQ



# Selective Repeat ARQ

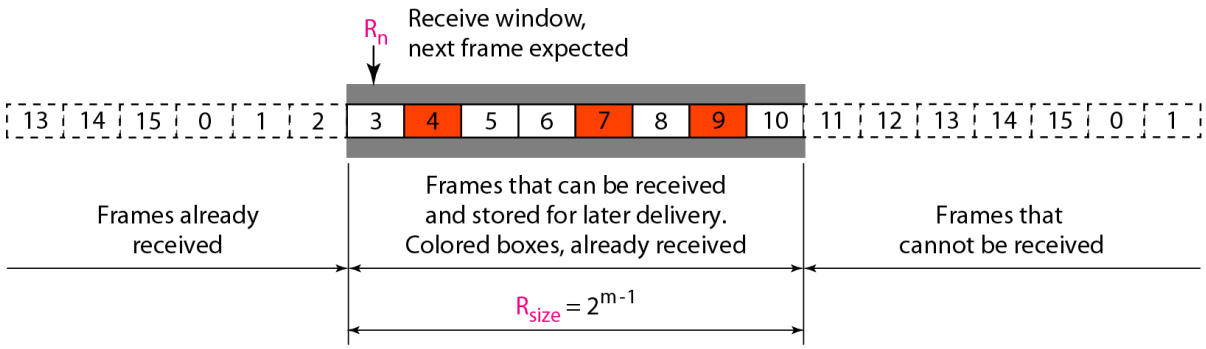
- Go-Back-N ARQ
  - simplifies the process at the receiver site
    - the size of receive window is one
  - the receiver keeps track of only one variable
  - there is no need to buffer out-of-order frames
  - very inefficient for a noisy link
- Selective Repeat ARQ
  - only the damaged frame is resent
  - efficient for a noisy link but complex process at the receiver site
  - large window size ($2^{m-1}$) at the receiver site
    - the receive window is totally different from the one in Go-Back-N
  - the window size at the sender site is also $2^{m-1}$

## Figure 11.18 *Send window for Selective Repeat ARQ*

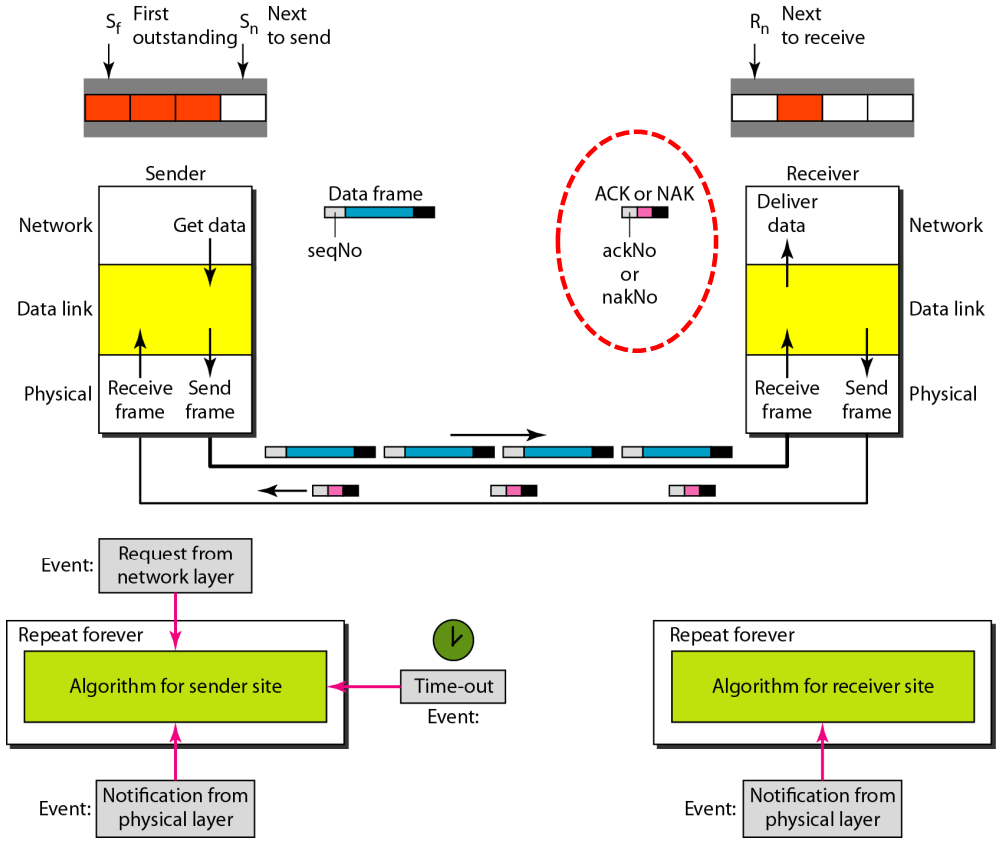





## Figure 11.19 *Receive window for Selective Repeat ARQ*

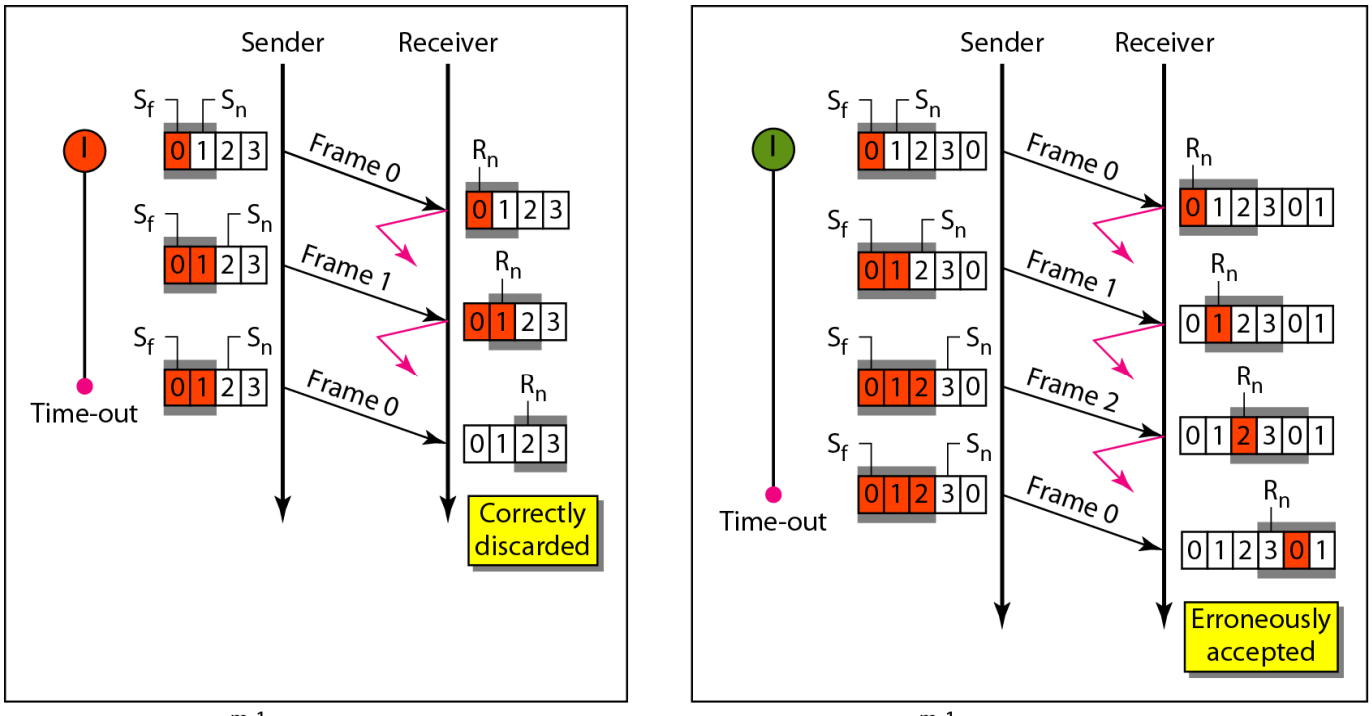


# Figure 11.20 *Design of Selective Repeat ARQ*






# Figure 11.21 *Selective Repeat ARQ, window size*




a. Window size = $2^{m-1}$

b. Window size > $2^{m-1}$

Note

**In Selective Repeat ARQ, the size of the sender and receiver window must be at most one-half of $2^m$.**



## *Example 11.8*

*This example is similar to Example 11.3 in which frame 1 is lost. We show how Selective Repeat behaves in this case. Figure 11.23 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.*

## Example 11.8 (continued)

*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, 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.*



## Example 11.8 (continued)

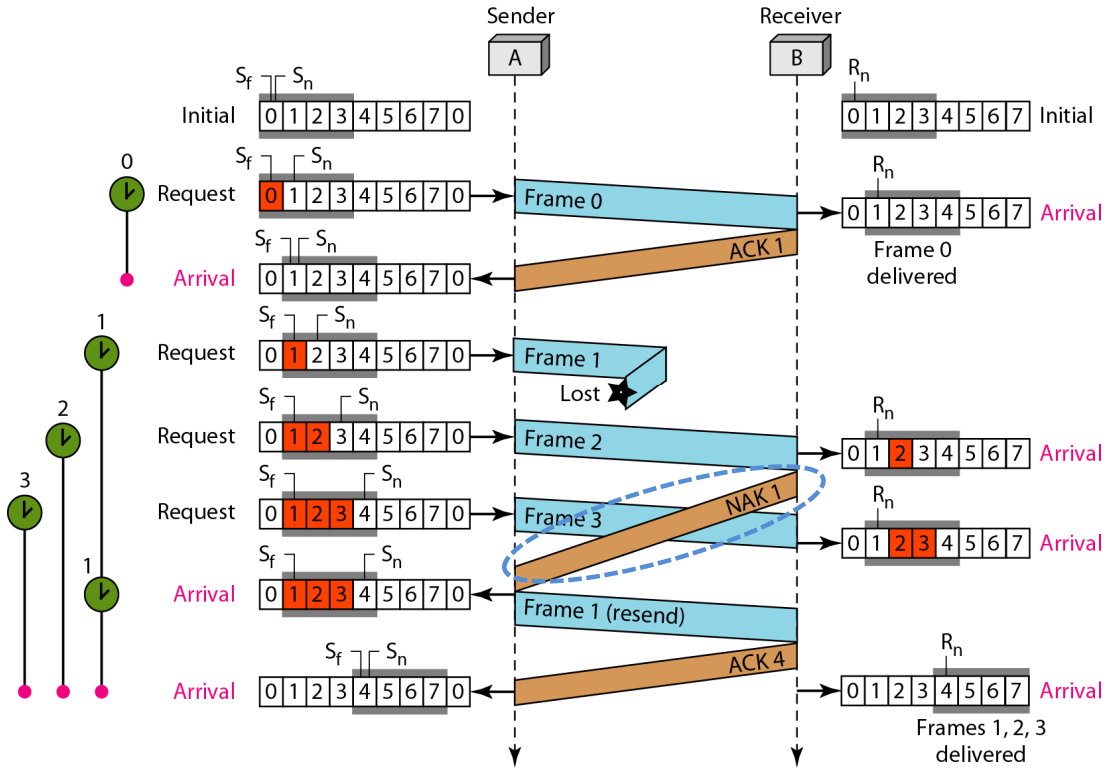
*Another important point is that a NAK is sent after the second arrival, but not after the third, although both situations look the same. The reason is that the protocol does not want to crowd the network with unnecessary NAKs and unnecessary resent frames. The second NAK would still be NAK1 to inform the sender to resend frame 1 again; this has already been done. The first NAK sent is remembered (using the nakSent variable) 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.*

# Example 11.8 (continued)

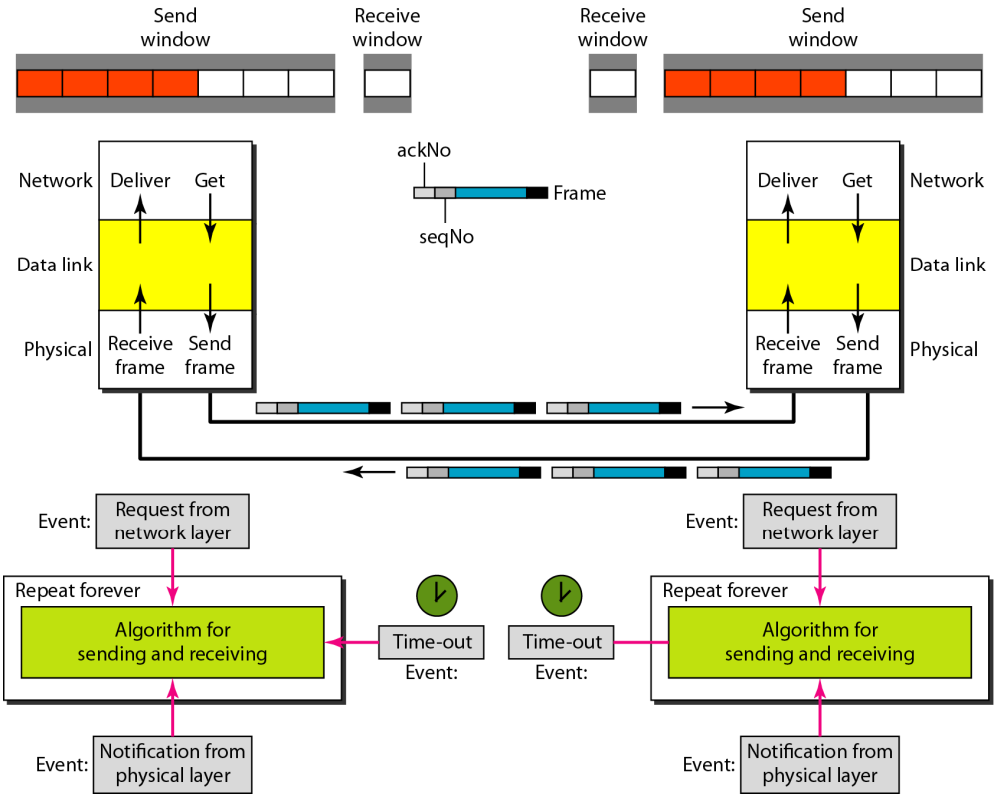
*The next point is about the ACKs. Notice that only two ACKs are sent here. The first one acknowledges only the first frame; the second one acknowledges three frames. In Selective Repeat, ACKs are sent when data are delivered to the network layer. If the data belonging to n frames are delivered in one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



## Figure 11.23 *Flow diagram for Example 11.8*



# Figure 11.24 *Design of piggybacking in Go-Back-N ARQ*






# Figure 11.25 *Design of piggybacking in Go-Back-N ARQ*