Data Link Layer:
Framing
and
Connecting Devices

AGENDA

- **Framing**
- Fixed and Variable Size Framing,
- Flow and Error Control,
- Noiseless Channels: Simplest and Stop-and-Wait,
- Noisy Channels: Stop-and-Wait Automatic Repeat Request, Go-back-N Automatic Repeat Request

Data-Link Layer Functions

- The two main functions of the data link layer are data link control and media access control.
- The first, data link control, deals with the design and procedures for communi cation between two adjacent nodes: node-to-node communication.
- Data link control functions include framing, flow and error control, and software implemented protocols that provide smooth and reliable transmission of frames between nodes.

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.
- Framing in the data-link layer separates a message from one source to a destination by adding a sender address and a destination address.
- The destination address defines where the packet is to go; the sender address helps the recipient acknowledge the receipt.

Framing

- Although the whole message could be packed in one frame, that is not normally done as:
- A frame can be very large, making flow and error control very inefficient. When a message is carried in one very large frame, even a single-bit error would require the retransmission of the whole frame.
- When a message is divided into smaller frames, a single-bit error affects only that small frame.

Framing

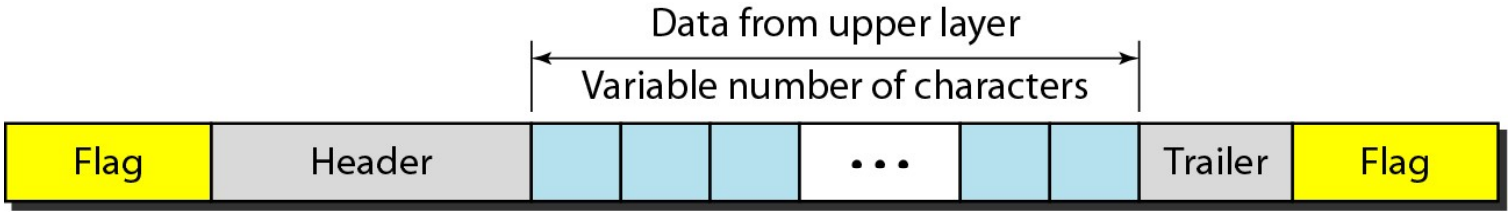
- Frame Size
- Frames can be of fixed or variable size.
- In fixed-size framing, there is no need for defining the boundaries of the frames; the size itself can be used as a delimiter.
- Example: ATM wide-area network, which uses frames of fixed size
- called cells.
- Variable-size framing is prevalent in local-area networks and we need a way to define the end of the frame and the beginning of the next.
- It is divided into: character-oriented approach and bit-oriented approach.

Framing

- Character-Oriented Framing
- In character-oriented (or byte-oriented) framing, data to be carried are 8-bit characters from a coding system such as ASCII.
- The header, which normally carries the source and destination addresses and other control information, and the trailer, which carries error detection redundant bits, are also multiples of 8 bits.
- To separate one frame from the next, an 8-bit (1-byte) flag is added at the beginning and the end of a frame.
- The flag, composed of protocol-dependent special characters, signals the start or end of a frame. Figure 11.1 shows the format of a frame in a character-oriented protocol.

Framing

- Process of wrapping data with certain info before sending out
- Figure 11.1 A frame in a character-oriented protocol



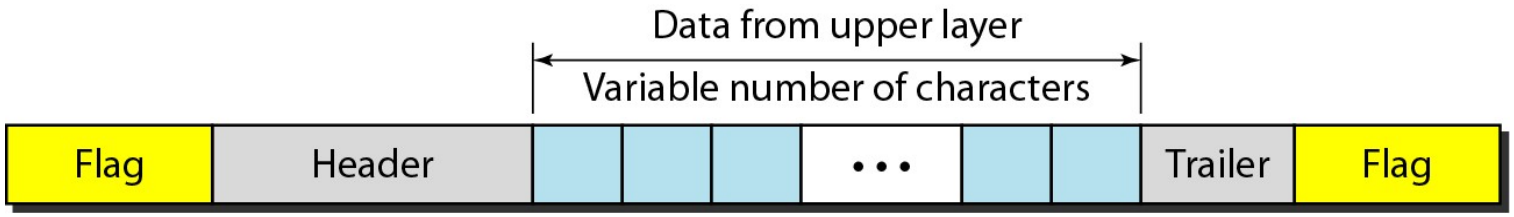
- A frame typically consists of
  - Flag: indication for start and end of a frame
  - Header: source/destination addresses, as well as other control information
  - Data from the upper layer
  - Trailer: error detection/correction code

Framing

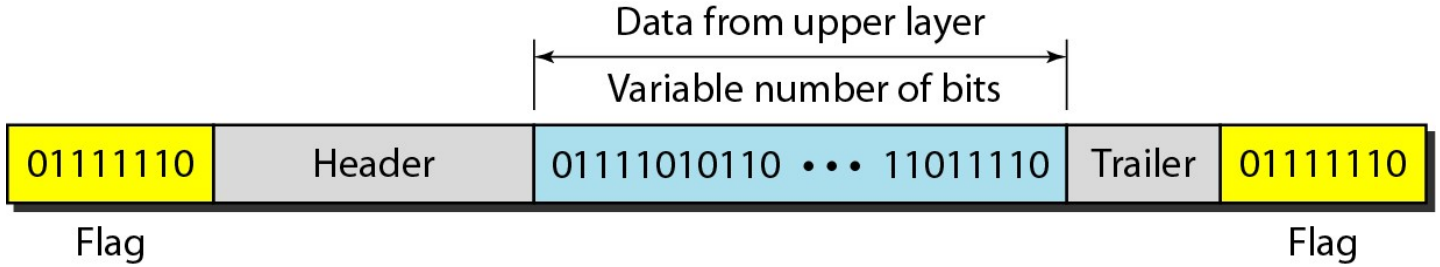
- Character-Oriented Framing
- Character-oriented framing was popular when only text was exchanged by the data-link layers. The flag could be selected to be any character not used for text com munication.
- Now we send other types of information such as graphs, audio, and video; any character used for the flag could also be part of the information.
- If this happens, the receiver thinks it has reached the end of the frame. To fix this problem, a byte-stuffing strategy was added to character-oriented framing.

Byte vs. Bit Oriented

- Framing in byte-oriented protocols

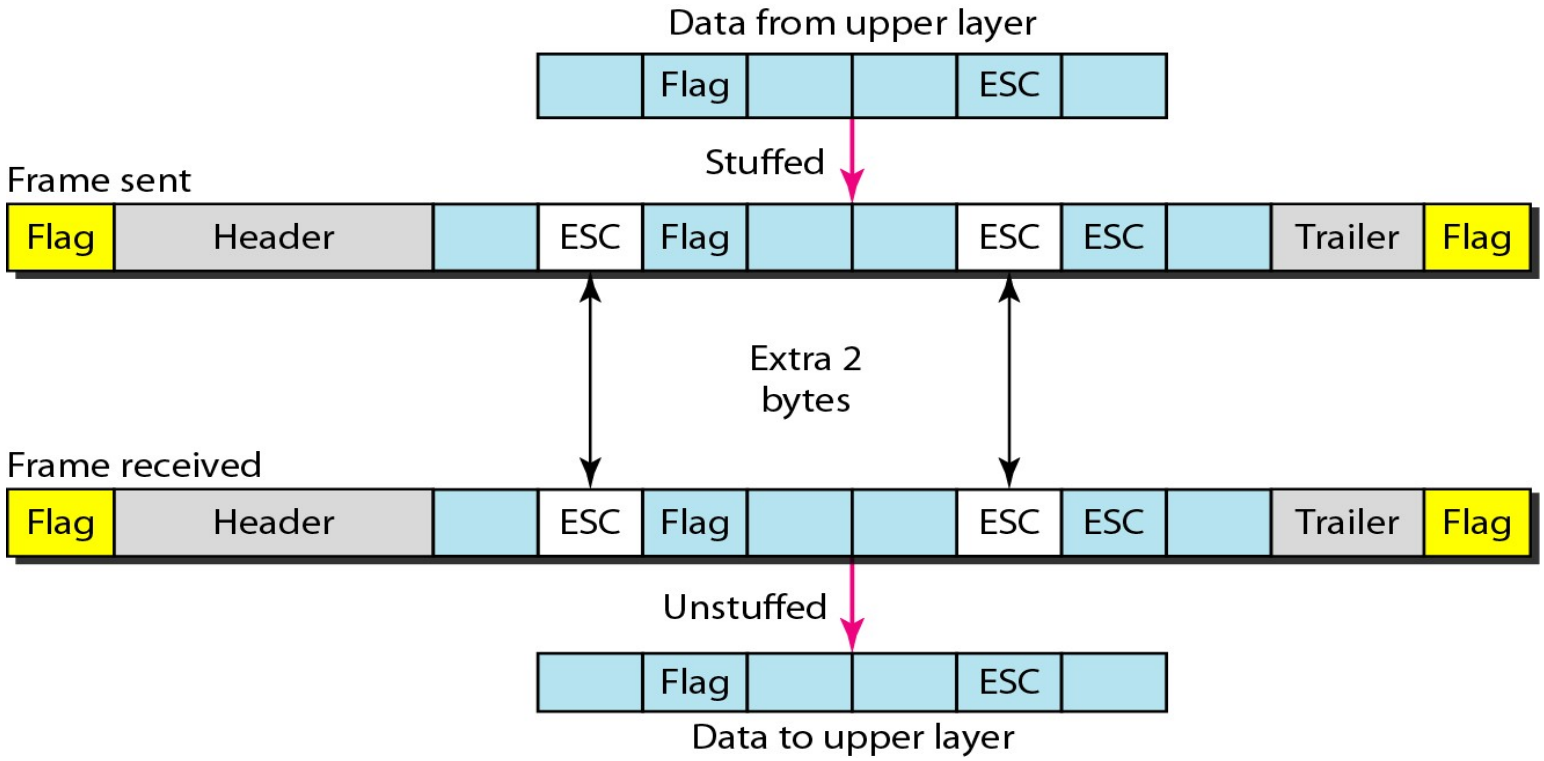


- Framing in bit-oriented protocols



Byte Stuffing

- In byte stuffing (or character stuffing), a special byte is added to the data section of the frame when there is a character with the same pattern as the flag. The data section is stuffed with an extra byte.
- This byte is usually called the escape character (ESC) and has a predefined bit pattern. Whenever the receiver encounters the ESC character, it removes it from the data section and treats the next character as data, not as a delimiting flag

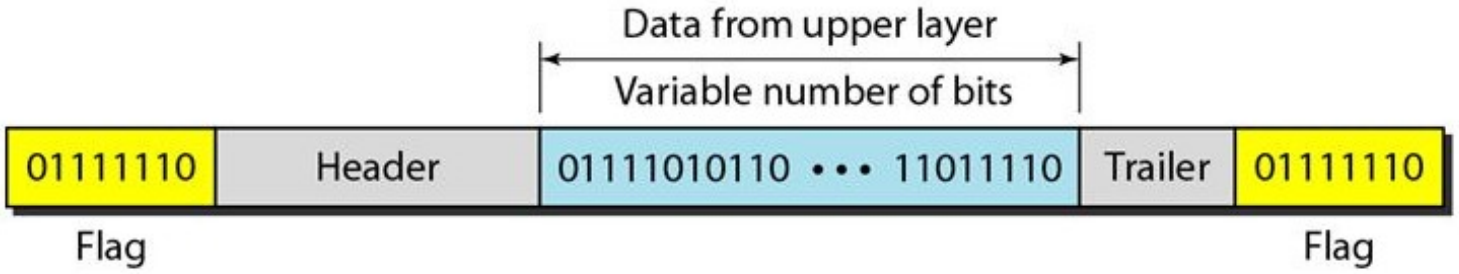


Byte Stuffing

- What happens if the text contains one or more escape characters followed by a byte with the same pattern as the flag?
- The receiver removes the escape character, but keeps the next byte, which is incorrectly interpreted as the end of the frame.
- To solve this. That is if the escape character is part of the text, another escape character is added to show that the second one is part of the text.
- Character-oriented protocols present another problem in data communications.
- The universal coding systems in use today, such as Unicode, have 16-bit and 32-bit characters that conflict with 8-bit characters. Hence we can use bit stuffing approach as well.

Bit Stuffing

- Bit-Oriented Protocols
- In bit-oriented framing, the data section of a frame is a sequence of bits to be interpreted by the upper layer as text, graphic, audio, video, and so on.
- However, in addition to headers (and possible trailers), we still need a delimiter to separate one frame from the other.
- Most protocols use a special 8-bit pattern flag, 01111110, as the delimiter to define the begin ning and the end of the frame, as shown in Figure 11.3 below.

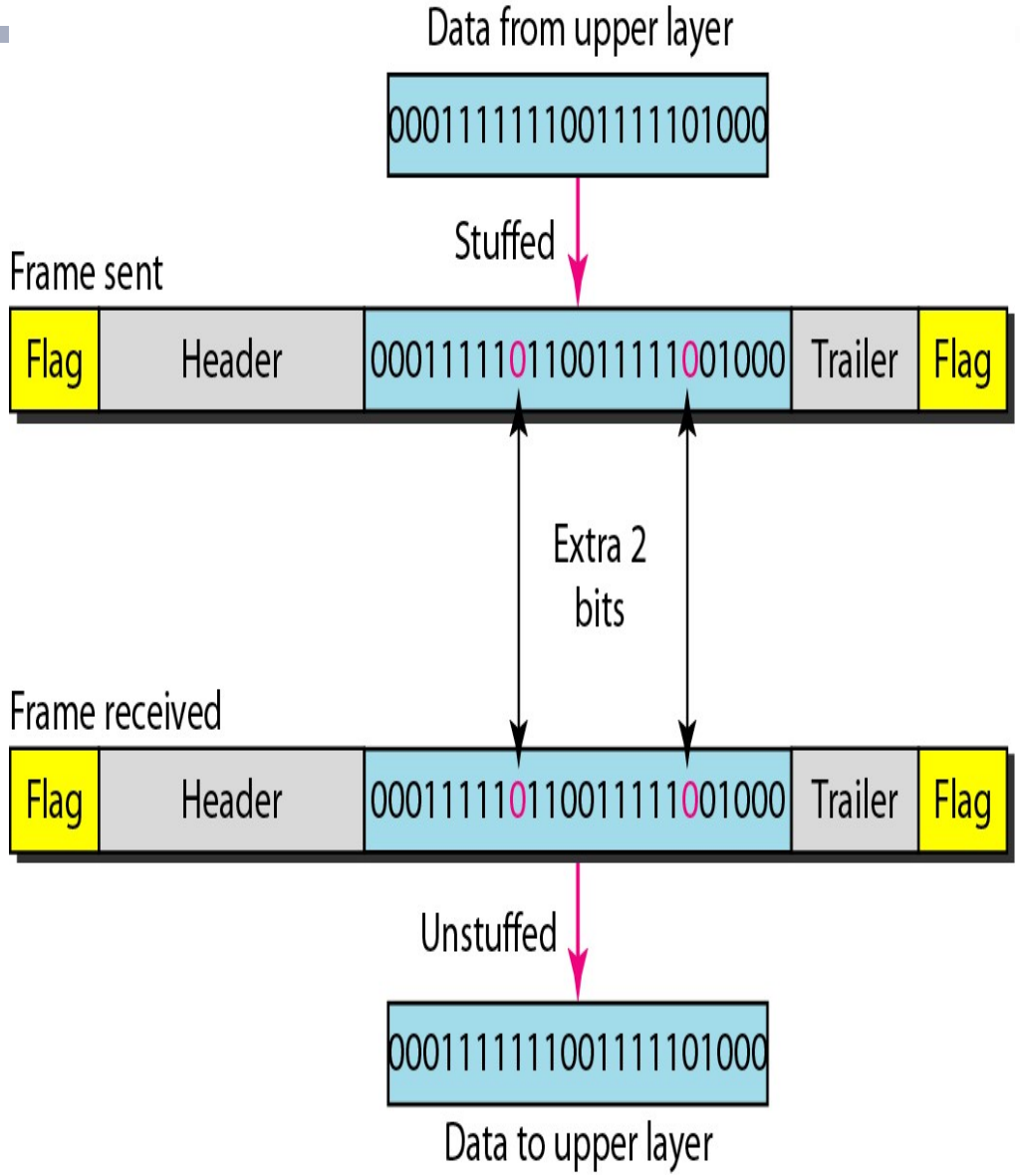


Bit Stuffing

- This flag can create the same type of problem we saw in the character-oriented protocols.
- We do this by stuffing 1 single bit (instead of 1 byte) to prevent the pattern from looking like a flag. The strategy is called bit stuffing.
- In bit stuffing, if a 0 and five consecutive 1 bits are encountered, an extra 0 is added. This extra stuffed bit is eventually removed from the data by the receiver.
- This guarantees that the flag field sequence does not inadvertently appear in the frame.

Bit Stuffing

- Bit-Oriented Framing
- Figure 11.4 shows bit stuffing at the sender and bit removal at the receiver.
- Note that even if we have a 0 after five 1s, we still stuff a 0. The 0 will be removed by the receiver.
- This means that if the flag like pattern 01111110 appears in the data, it will change to 011111010 (stuffed) and is not mistaken for a flag by the receiver.
- The real flag 01111110 is not stuffed by the sender and is recognized by the receiver



Flow Control and Error Control

- Flow control
  - Flow control refers to a set of procedures used to restrict the amount of data that the sender can send before waiting for acknowledgment.
  - Flow control refers to a set of procedures used to restrict the amount of data that the sender can send before waiting for acknowledgment.
  - The receiving device must be able to inform the sending device before those limits are reached and to request that the transmitting device send fewer frames or stop temporarily

Flow Control and Error Control

- Flow control
  - Each receiving device has a block of memory, called a buffer, reserved for storing incom ing data until they are processed. If the buffer begins to fill up, the receiver must be able to tell the sender to halt transmission until it is once again able to receive.

Flow Control and Error Control

- Error control
  - Error control is both error detection and error correction.
  - It allows the receiver to inform the sender of any frames lost or damaged in transmission and coordinates the retransmission of those frames by the sender.
  - Any time an error is detected in an exchange, specified frames are retransmitted. This process is called automatic repeat request (ARQ)

Protocol

- The protocols are normally implemented in software by using one of the common programming languages.
- Pseudocode of each protocol concentrates mostly on the procedure instead of delving into the details of language rules.

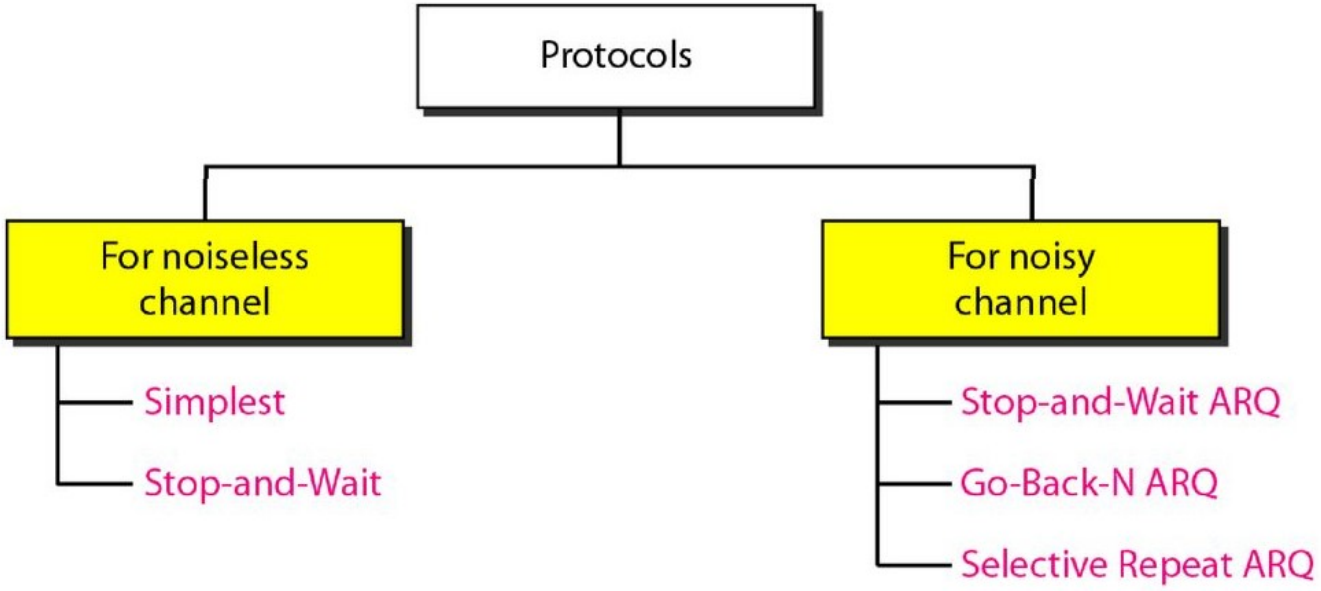


Figure 11.5 *Taxonomy of protocols discussed in this chapter*

Noiseless Channel

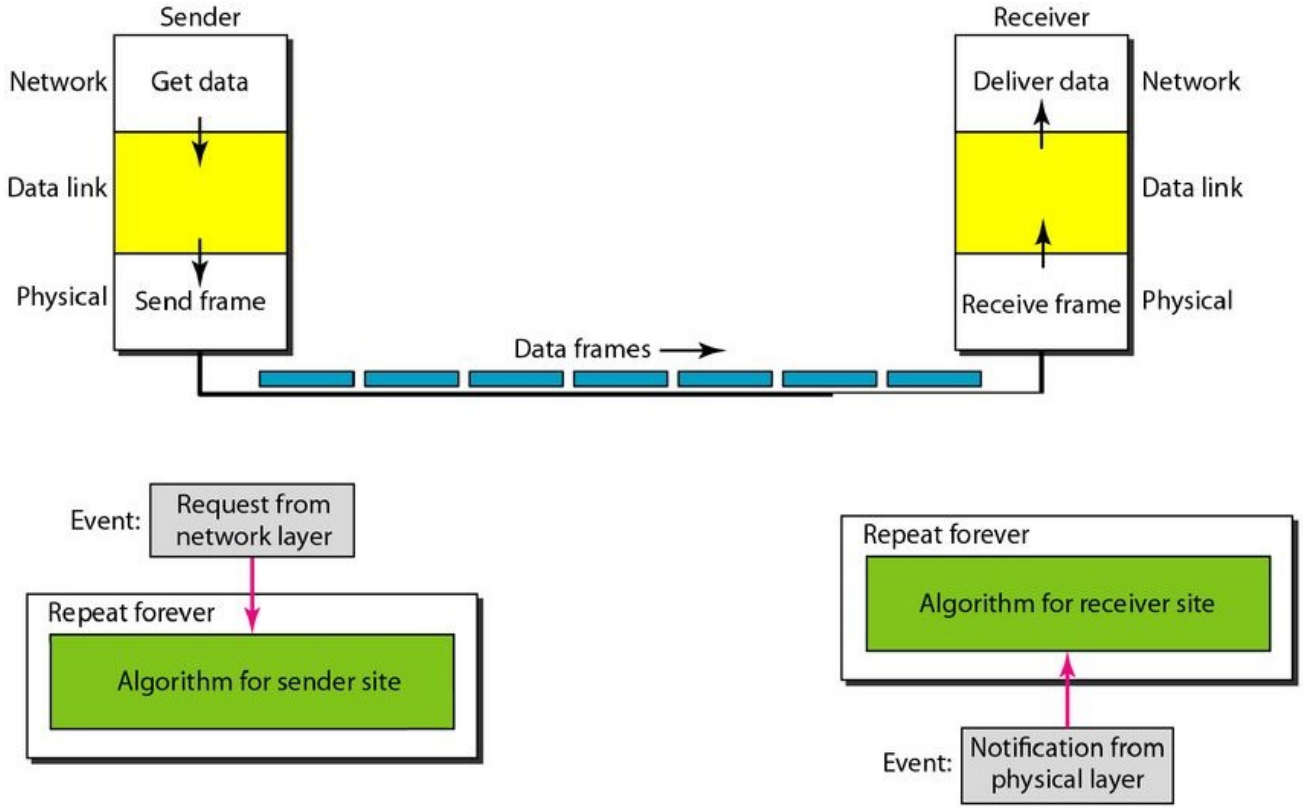
- **Simplest Protocol**
- The protocol has no flow or error control. It is a unidirectional protocol in which data frames are travelling in only one direction-from thesender to receiver.
- Assume that the receiver can immediately handle any received frame with a negligible processing time.
- Receivers data link layer removes the header from the frame and hands the data packet to its network layer.
- The receiver can never be overwhelmed with incoming frames.

Noiseless Channel

- **Simplest Protocol**
- Design
- The data link layer at the sender gets data from network layer, makes a frame out of the data, and sends it.
- The data link layer at the receiver site receives a frame from its physical layer, extracts data from the frame, and delivers the data to its network layer.

Noiseless Channel

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



Noiseless Channel

- **Simplest Protocol**
- Design
- The procedure at the sender site is constantly running; there is no action until there is a request from the network layer.
- The procedure at the receiver site is also constantly running, but there is no action until notification from the physical layer arrives.

Noiseless Channel

- **Simplest Protocol**
- Design
- Pseudocode is for the main process only.
- It does not show any details for the modules GetData, MakeFrame, and SendFrame.
- GetDataO takes a data packet from the network layer, MakeFrameO adds a header and delimiter flags to the data packet to make a frame, and SendFrameO delivers the frame to the physical layer for transmission.

Algorithm 11.1 *Sender-site algorithm for the simplest protocol*

```
1  while(true)                          // Repeat forever
2  {
3    WaitForEvent();                    // Sleep until an event occurs
4    if(Event(RequestToSend))           //There is a packet to send
5    {
6       GetData();
7       MakeFrame();
8       SendFrame();                    //Send the frame
9    }
10 }
```

Algorithm 11.2 *Receiver-site algorithm for the simplest protocol*

```
1  while(true)                           // Repeat forever
2  {
3    WaitForEvent();                     // Sleep until an event occurs
4    if(Event(ArrivalNotification))      //Data frame arrived
5    {
6       ReceiveFrame();
7       ExtractData();
8       DeliverData();                   //Deliver data to network layer
9    }
10 }
```

Noiseless Channel

- **Simplest Protocol**
- Sending side
- Pseudocode is for the main process only.
- It does not show any details for the modules GetData, MakeFrame, and SendFrame.
- GetDataO takes a data packet from the network layer, MakeFrameO adds a header and delimiter flags to the data packet to make a frame, and SendFrameO delivers the frame to the physical layer for transmission.

Noiseless Channel

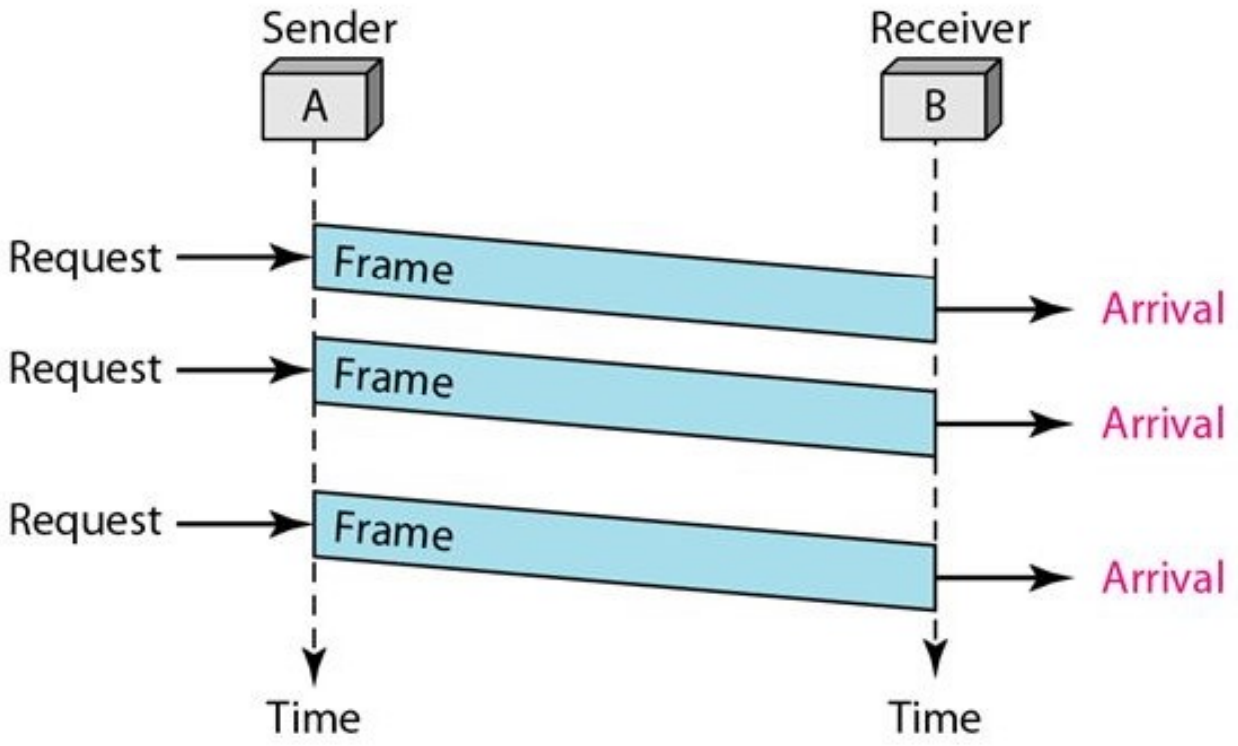
- **Simplest Protocol**
- Receiving side
- The event here is the arrival of a data frame.
- After the event occurs, the data link layer receives the frame from the physical layer using the ReceiveFrameO process, extracts the data from the frame using the ExtractDataO process, and delivers the data to the network layer using the DeliverDataO process.

Noiseless Channel

- **Simplest Protocol**
- Example 11.1
- Figure 11.7 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.
- To send three frames, three events occur at the sender site and three events at the receiver site.
- Note that the data frames are shown by tilted boxes; the height of the box defines the transmission time difference between the first bit and the last bit in the frame.

Noiseless Channel

- **Simplest Protocol**
- Example 11.1



Noiseless Channel

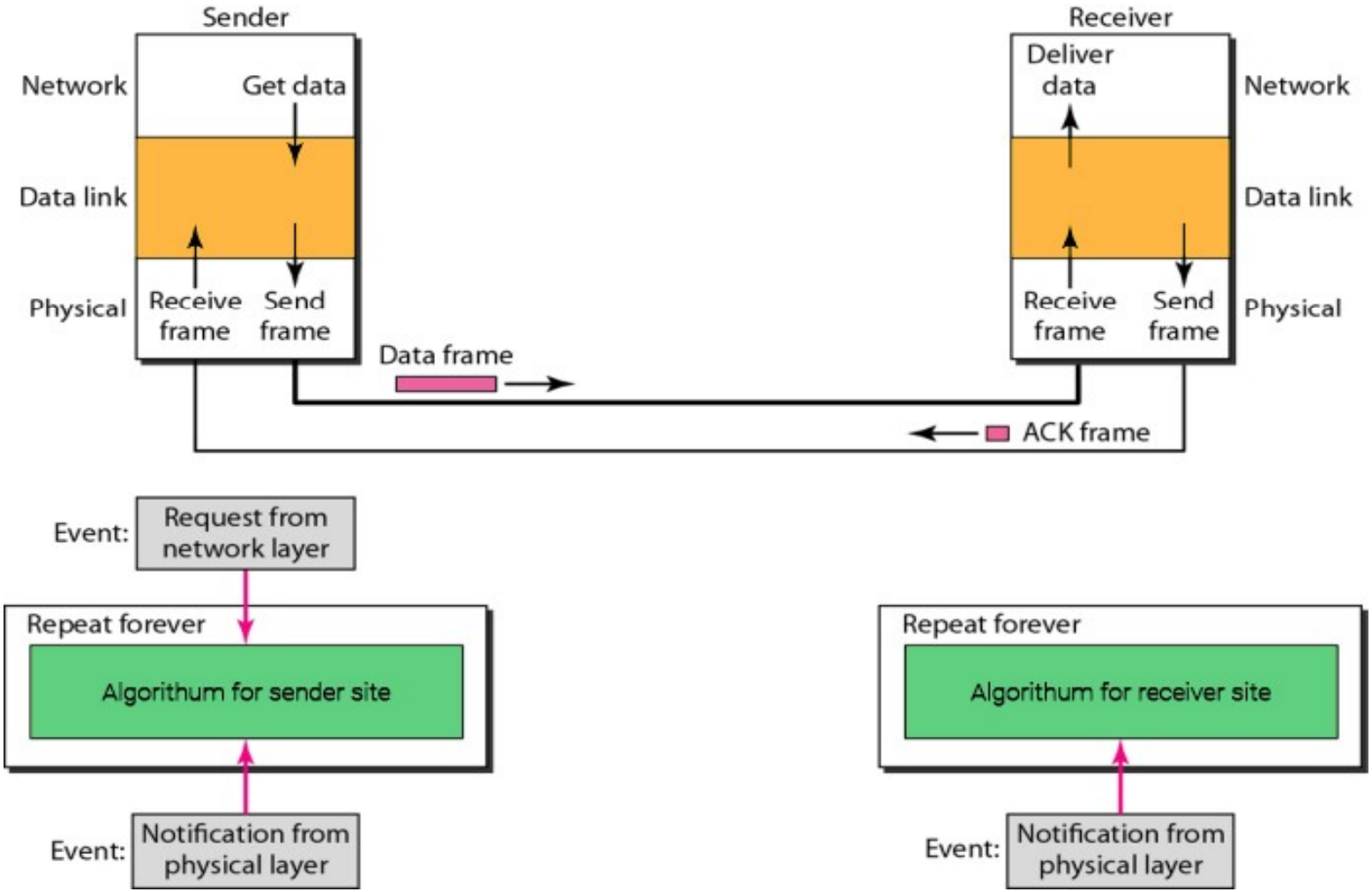
- **Stop-and-Wait Protocol**
- If data frames arrive at the receiver site faster than they can be processed, the frames must be stored until their use.
- If receiver does not have enough storage or is receiving from many sources, this may result in either the discarding of frames or denial of service.
- We somehow need to tell the sender to slow down.

Noiseless Channel

- **Stop-and-Wait Protocol**
- The protocol is called the Stop-and-Wait Protocol because the sender sends one frame, stops until it receives confirmation from the receiver, and then sends the next frame.
- There is unidirectional communication for data frames, but auxiliary ACK frames goes in the other direction.
- At any time, there is either one data frame on the forward channel or one ACK frame on the reverse channel, needing a half-duplex link.

Noiseless Channel

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



Noiseless Channel

Algorithm 11.3 *Sender-site algorithm for Stop-and-Wait Protocol*

```
1  while(true)                          //Repeat forever
2  canSend = true                       //Allow the first frame to go
3  {
4    WaitForEvent();                    // Sleep until an event occurs
5    if(Event(RequestToSend) AND canSend)
6    {
7       GetData();
8       MakeFrame();
9       SendFrame();                    //Send the data frame
10      canSend = false;                //Cannot send until ACK arrives
11   }
12   WaitForEvent();                    // Sleep until an event occurs
13   if(Event(ArrivalNotification)      // An ACK has arrived
14    {
15      ReceiveFrame();                 //Receive the ACK frame
16      canSend = true;
17    }
18 }
```

Noiseless Channel

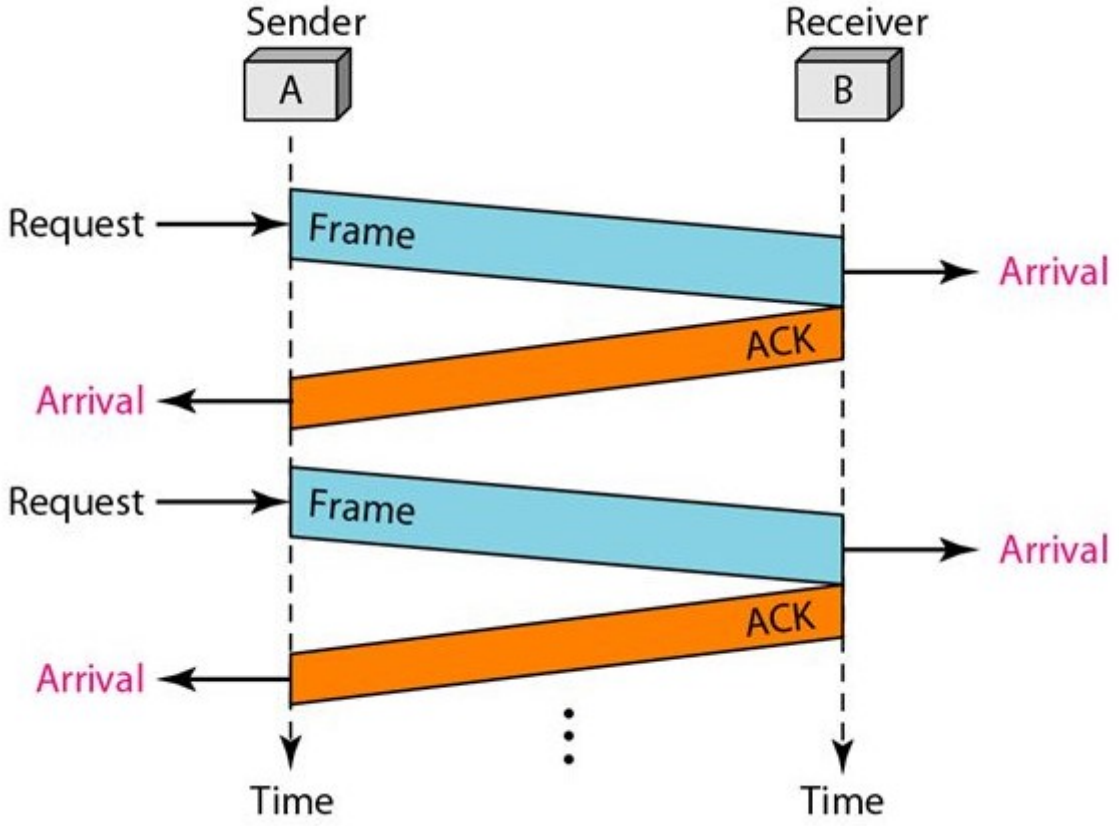
Algorithm *Receiver-site for Stop-and-Wait Protocol*

```
1  while(true)                          //Repeat forever
2  {
3    WaitForEvent();                    // Sleep until an event occurs
4    if(Event(ArrivalNotification))     //Data frame arrives
5    {
6       ReceiveFrame();
7       ExtractData();
8       Deliver(data);                  //Deliver data to network layer
9       SendFrame();                    //Send an ACK frame
10   }
11 }
```

Noiseless Channel

- **Example 11.2**
- As shown in figure 11.9, 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.



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 can ignore the error (as we sometimes do), or we need to add error control to our protocols.

Noisy Channels

- **Stop-and-Wait Automatic Repeat Request**
- This protocol adds a simple error control mechanism to the Stop-and-Wait Protocol.
- To detect and correct corrupted frames, we need to add redundancy bits to our data frame.
- Lost frames are more difficult to handle than corrupted ones.
- The received frame could be the correct one, or a duplicate, or a frame out of order.
- 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.

Noisy Channels

- **Stop-and-Wait Automatic Repeat Request**
- The corrupted and lost frames need to be resent in this protocol. For that 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.
- The sender simply discards a corrupted ACK frame or ignores an out-of-order one.

Noisy Channels

- **Stop-and-Wait Automatic Repeat Request**
- Sequence Numbers
- A field is added to the data frame to hold the sequence number of that frame.
- Since we want to minimize the frame size, we look for the smallest range that provides unambiguous communication.
- 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.
- The sequence numbers are based on modulo-2 arithmetic.

Noisy Channels

- **Stop-and-Wait Automatic Repeat Request**
- Out of Range Sequence Numbers
- Assume we have used x as a sequence number; we only need to use x + 1 after that, no need for x + 2. Three things are possible: Three things can happen.
- **1.** The frame arrives safe and sound at the receiver site; the receiver sends an acknowledgment.
- The acknowledgment arrives at the sender site, causing the sender to send the next frame numbered x + 1.

Noisy Channels

- **Stop-and-Wait Automatic Repeat Request**
- **2.** The frame arrives safe and sound at the receiver site; the receiver sends an acknowledgment, but the acknowledgment is corrupted or lost.
- The sender resends the frame (numbered x) after the time-out. (here is a duplicate frame)
- The receiver can recognize this fact because it expects frame x + I but frame x was received.
- **3.** The frame is corrupted or never arrives at the receiver site; the sender resends the frame (numbered x) after the time-out.

Noisy Channels

- **Stop-and-Wait Automatic Repeat Request**
- We can see that there is a need for sequence numbers x and x + I because the receiver needs to distinguish between different cases.

Noisy Channels

- **Stop-and-Wait Automatic Repeat Request**
- Acknowledgement 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).

Noisy Channels

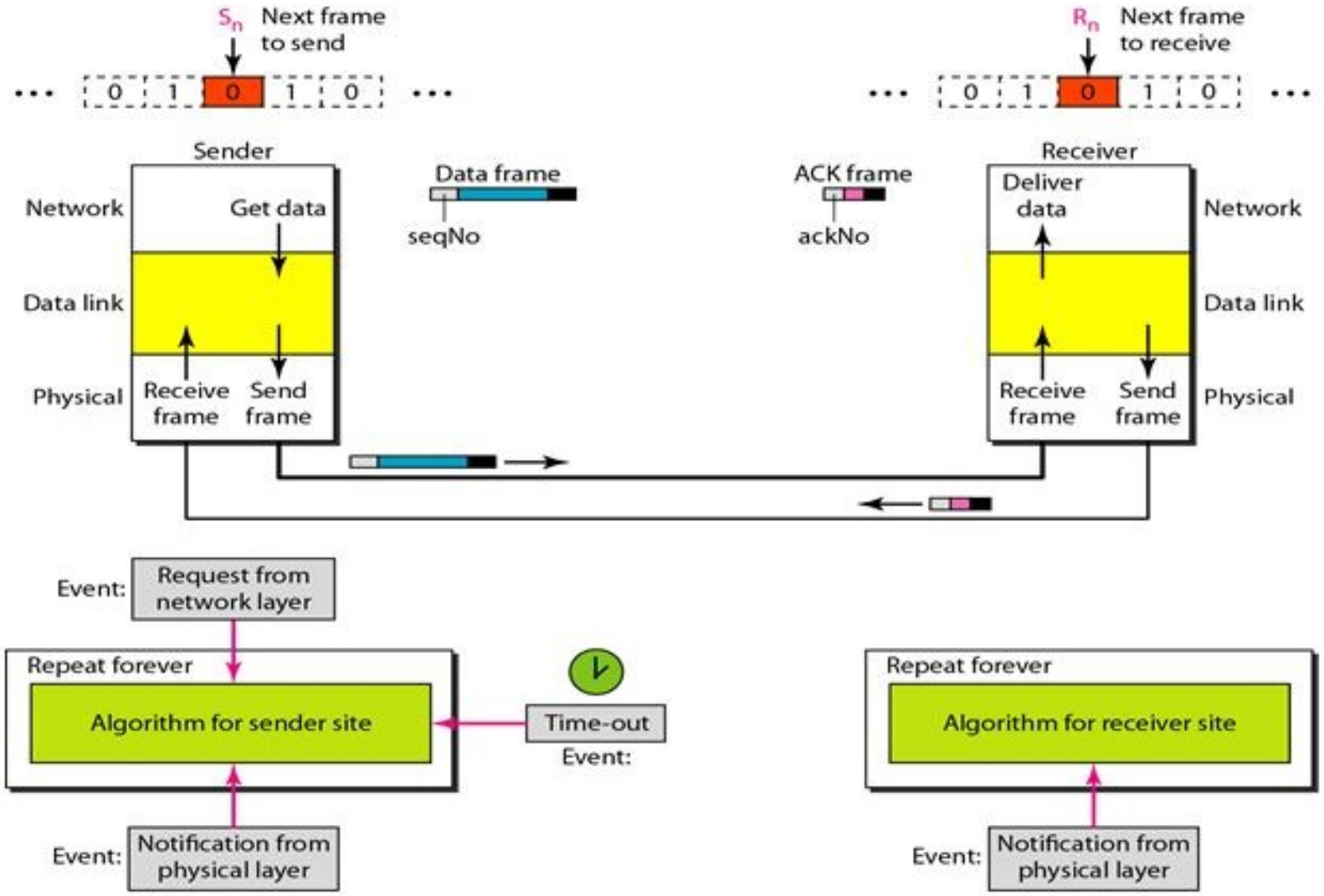


Figure 11.10 *Design of the Stop-and-Wait ARQ Protocol*

Noisy Channels

- **Stop-and-Wait Automatic Repeat Request**
- The sender 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).

Noisy Channels

- **Stop-and-Wait Automatic Repeat Request**
- The receiver has a control variable, R_n, that holds the number of the next frame expected.
- When a frame is sent, the value of S_n is incremented (modulo-2), which means if it is 0, it becomes 1 and vice versa.
- When a frame is received, the value of R_n is incremented (modulo-2), which means if it is 0, it becomes 1 and vice versa.
- Three events can happen at the sender site; one event can happen at the receiver site.

Noisy Channels

Algorithm 11.5 *Sender-site algorithm for Stop-and-Wait ARQ*

```
1  Sn = 0;                              // Frame 0 should be sent first
2  canSend = true;                      // Allow the first request to go
3  while(true)                          // Repeat forever
4  {
5    WaitForEvent();                    // Sleep until an event occurs
6    if(Event(RequestToSend) AND canSend)
7    {
8       GetData();
9       MakeFrame(Sn);                        //The seqNo is Sn
10      StoreFrame(Sn);                       //Keep copy
11      SendFrame(Sn);
12      StartTimer();
13      Sn = Sn + 1;
14      canSend = false;
15   }
16   WaitForEvent();                          // Sleep
```

(continued)

Noisy Channels

Algorithm 11.5 *Sender-site algorithm for Stop-and-Wait ARQ* *(continued)*

```
17      if(Event(ArrivalNotification)          // An ACK has arrived
18      {
19        ReceiveFrame(ackNo);                 //Receive the ACK frame
20        if(not corrupted AND ackNo == Sn)    //Valid ACK
21          {
22            Stoptimer();
23            PurgeFrame(Sn-1);                //Copy is not needed
24            canSend = true;
25          }
26       }
27
28      if(Event(TimeOut)                      // The timer expired
29      {
30       StartTimer();
31       ResendFrame(Sn-1);                    //Resend a copy check
32      }
33  }
```

Noisy Channels

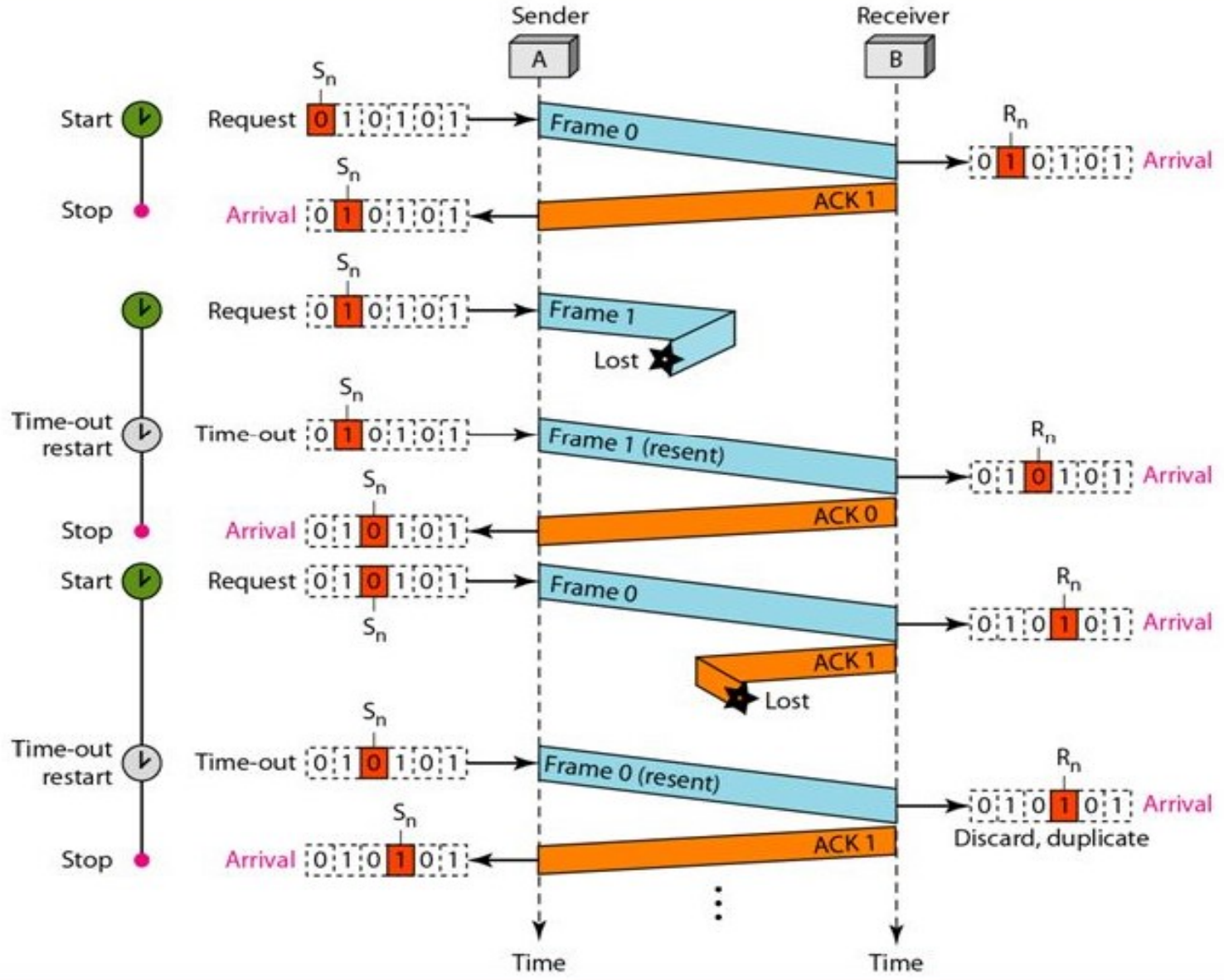
Algorithm 11.6 *Receiver-site algorithm for Stop-and-Wait ARQ Protocol*

```
1  Rn = 0;                          // Frame 0 expected to arrive first
2  while(true)
3  {
4    WaitForEvent();                // Sleep until an event occurs
5    if(Event(ArrivalNotification))  //Data frame arrives
6    {
7       ReceiveFrame();
8       if(corrupted(frame));
9          sleep();
10      if(seqNo == Rn)              //Valid data frame
11      {
12       ExtractData();
13        DeliverData();             //Deliver data
14        Rn = Rn + 1;
15      }
16       SendFrame(Rn);             //Send an ACK
17   }
18 }
```

Noisy Channels

- **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*



Noisy Channels

- **Efficiency**
- The Stop-and-Wait ARQ is very inefficient if our 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.

Noisy Channels

- **Go-Back-N Automatic Repeat Request**
- To improve the efficiency of transmission (filling the pipe), we need to let more than one frame be outstanding to keep the channel busy while the sender is waiting for acknowledgment.
- A protocol called Go-Back-N Automatic Repeat Request can be used to send several frames before receiving acknowledgments.

Noisy Channels

- **Go-Back-N Automatic Repeat Request**
- Sequence Numbers
- Frames from a sending station are numbered sequentially. 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, …
- In other words, the sequence numbers are modulo-2^m

Noisy Channels

- **Go-Back-N Automatic Repeat Request**
- Sliding Window
- The sliding window is an abstract concept that defines the range of sequence numbers that is the concern of the sender and receiver.
- 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.
- 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 $2^m - 1$

Noisy Channels

- **Go-Back-N Automatic Repeat Request**
- Sliding Window
- This protocol keeps size of the window fixed, but there are some protocols which may have a variable window size.
- The window at any time divides the possible sequence numbers into four regions.
- The first region, defines the sequence numbers belonging to frames that are already acknowledged. Sender does not save them as they are successfully received.
- The second region, has outstanding frames, that are the frames sent and have an unknown status. The sender needs to wait for their status.

Noisy Channels

- **Go-Back-N Automatic Repeat Request**
- The third region, 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.
- The fourth region defines sequence numbers that cannot be used until the window slides.

The send window is an abstract concept defining an imaginary box of size $2^m - 1$ with three variables: S_f, S_n, and $Ssize'$

- S_f(send window, the first outstanding frame), S_n (send window, the next frame to be sent), Ssize (send window, size)

Noisy Channels

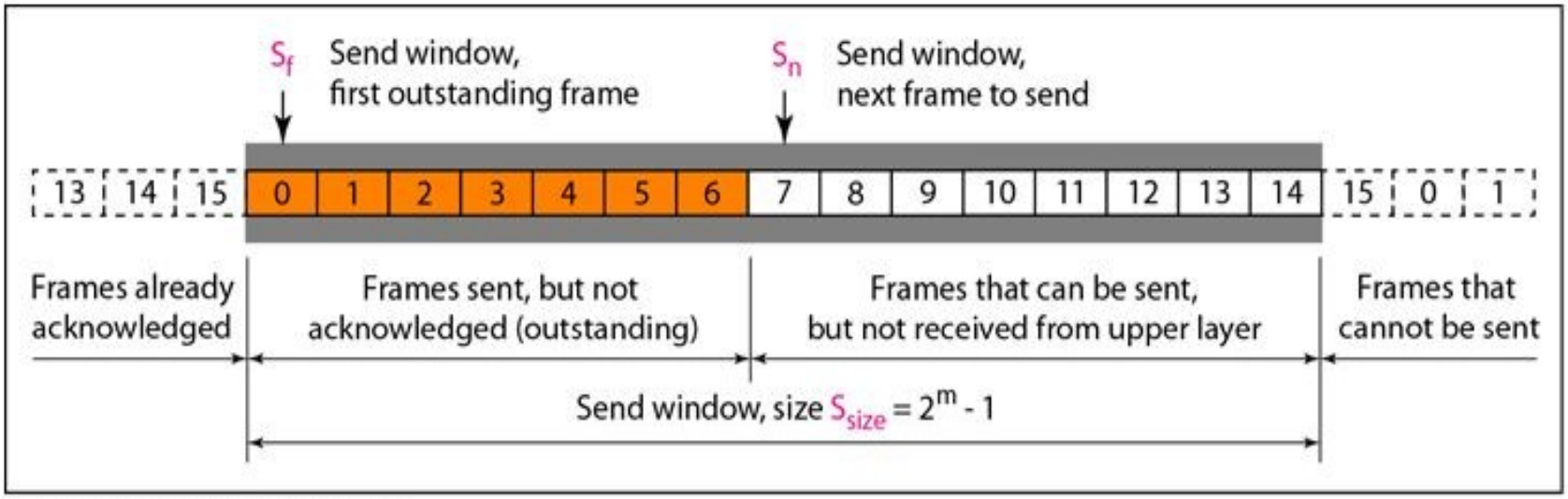
- **Go-Back-N Automatic Repeat Request**
- The third region, defines the range of sequence numbers for frames that can be sent; however, the corresponding data packets have not yet been received from the network layer.
- The fourth region defines sequence numbers that cannot be used until the window slides.

The send window is an abstract concept defining an imaginary box of size $2^m - 1$ with three variables: S_f, S_n, and $Ssize'$

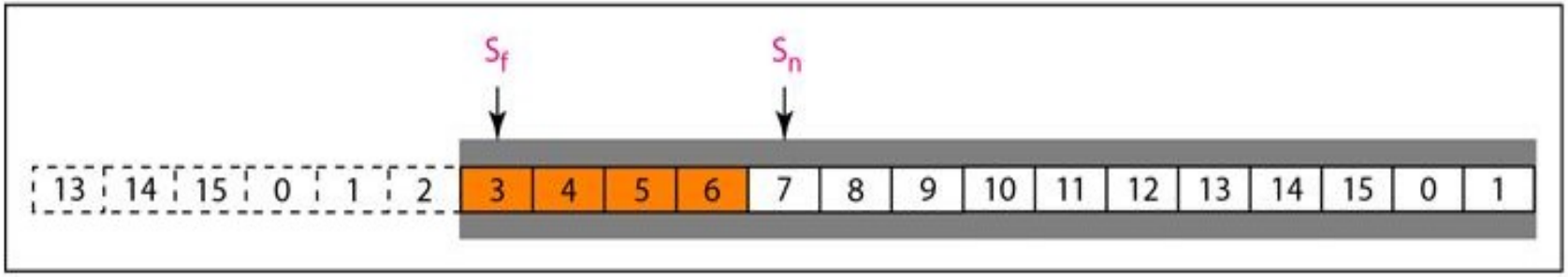
- S_f(send window, the first outstanding frame), S_n (send window, the next frame to be sent), Ssize (send window, size)

Noisy Channels

- **Go-Back-N Automatic Repeat Request**



a. Send window before sliding



b. Send window after sliding

Figure 11.12 ***Send window for Go-Back-NARQ***

Noisy Channels

- **Go-Back-N Automatic Repeat Request**
- Figure 11.12b shows how a send window can slide one or more slots to the right when an acknowledgment arrives from the other end.
- The acknowledgments in this protocol are cumulative, meaning that more than one frame can be acknowledged by an ACK frame.
- In Figure 11.12b, frames 0, I, and 2 are acknowledged, so the window has slid to the right three slots.

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

Noisy Channels

- **Go-Back-N Automatic Repeat Request**

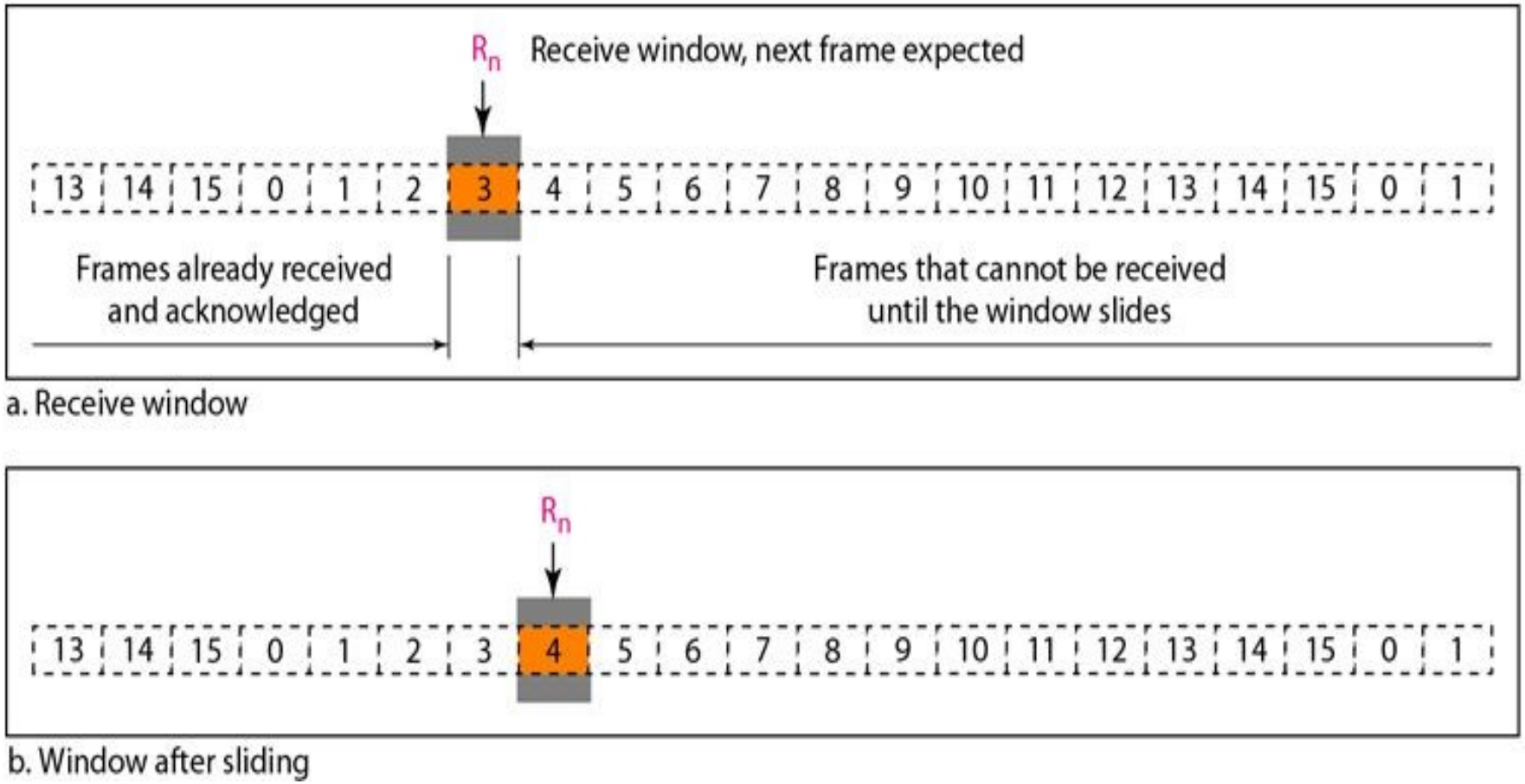


Figure 11.13 *Receive window for Go-Back-NARQ*

Noisy Channels

- **Go-Back-N Automatic Repeat Request**
- 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.
- 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.

Noisy Channels

- **Go-Back-N Automatic Repeat Request**
- Timers
- Although there can be a timer for each frame that is sent, in our protocol we use only one.
- The reason is that the timer for the first outstanding frame always expires first; we send all outstanding frames when this timer expires.

Noisy Channels

- **Go-Back-N Automatic Repeat Request**
- 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.

Noisy Channels

- **Go-Back-N Automatic Repeat Request**
- Acknowledgment
- 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.

Noisy Channels

- **Go-Back-N Automatic Repeat Request**
- Design
- Multiple frames can be in transit in the forward direction, and multiple acknowledgments in the reverse direction.
- 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.

Go-Back-N Automatic Repeat Request

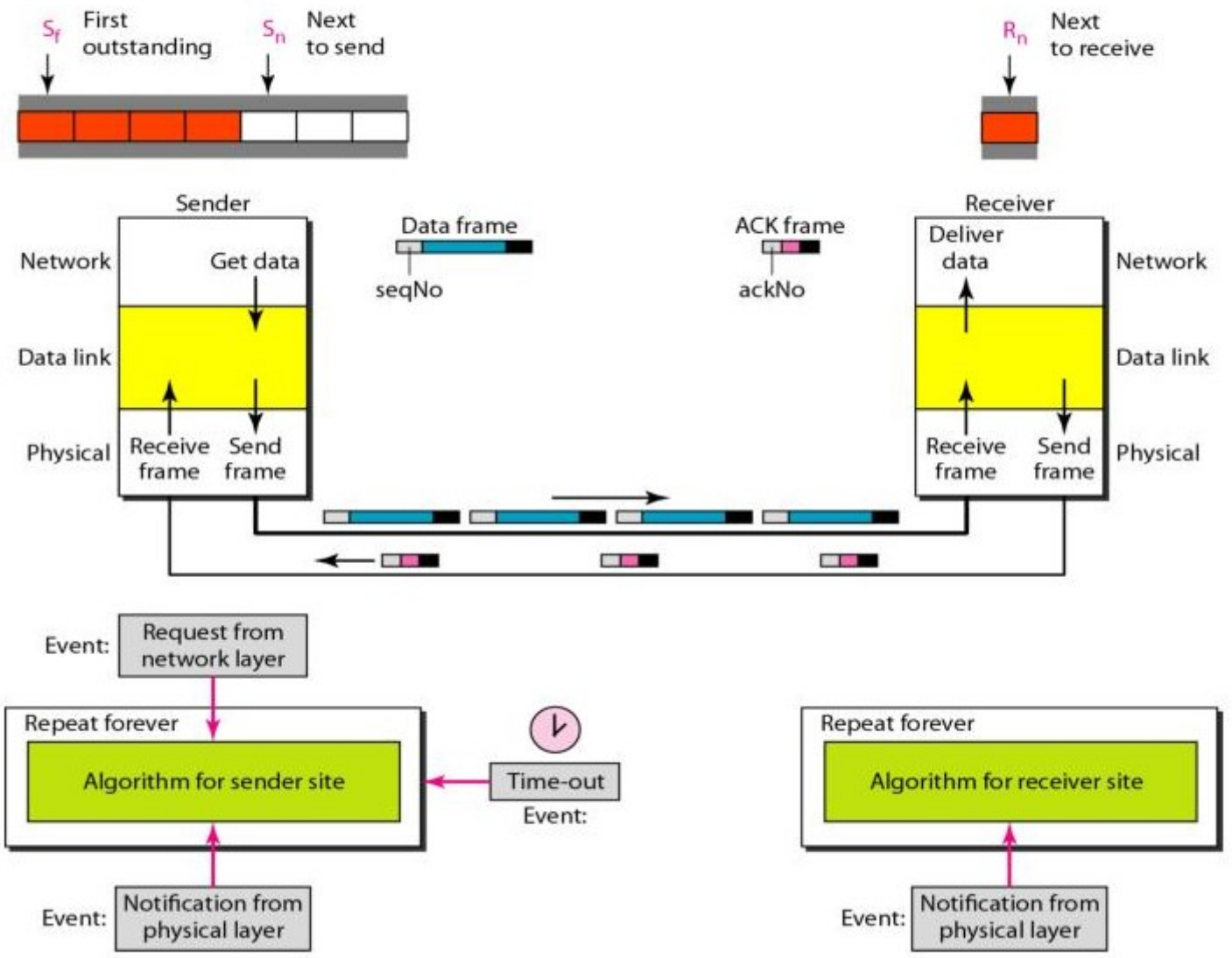
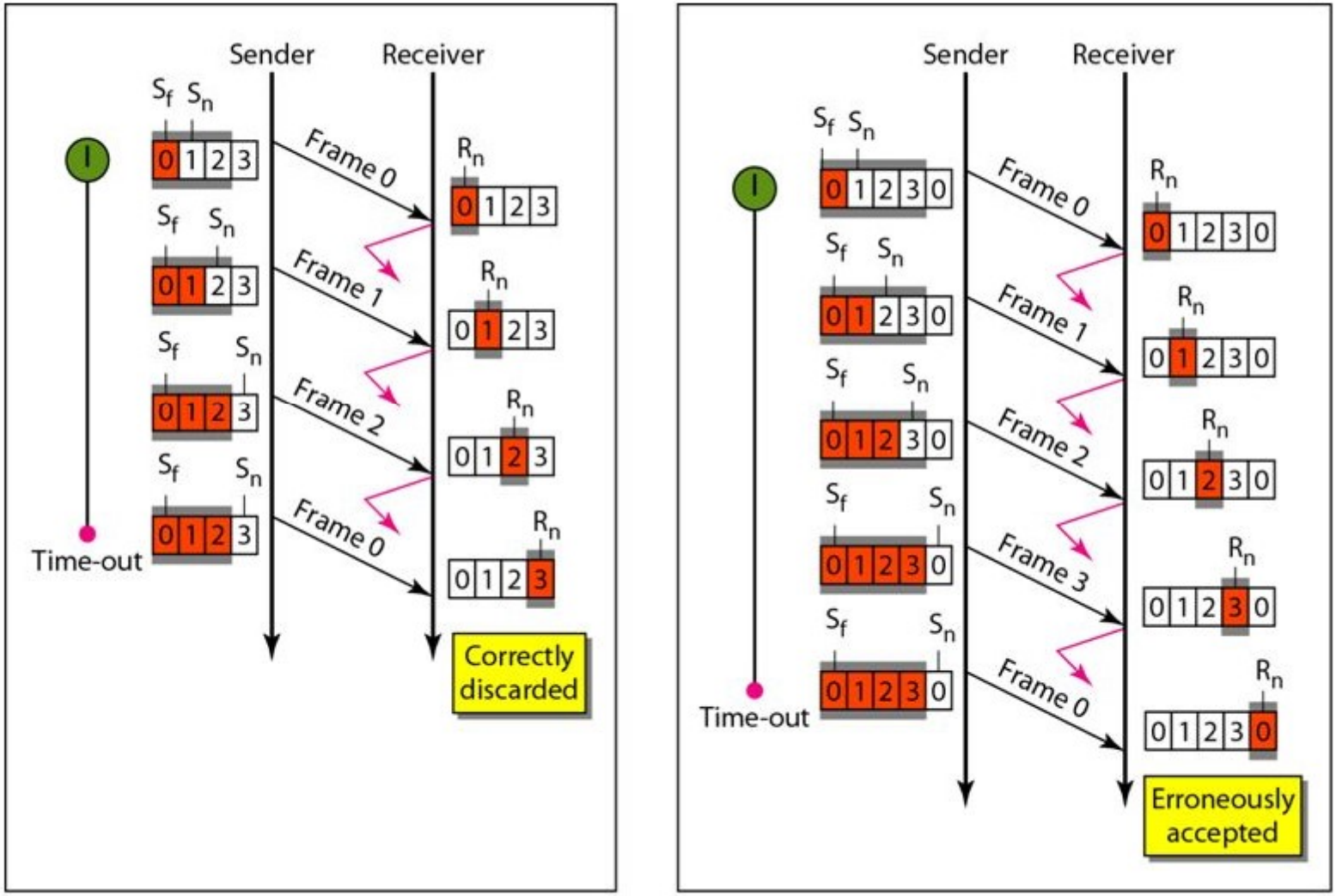


Figure **11.14** ***Design of Go-Back-N ARQ***

Noisy Channels

- **Go-Back-N Automatic Repeat Request**
- 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.
- M is the size of the window field in the header.

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



a. Window size < 2^m

b. Window size = 2^m

Algorithm 11.7 *Go-Back-N sender algorithm*

```
1  Sw = 2^m - 1;
2  Sf = 0;
3  Sn = 0;
4
5  while (true)                          //Repeat forever
6  {
7   WaitForEvent();
8    if(Event(RequestToSend))            //A packet to send
9    {
10      if(Sn-Sf >= Sw)                  //If window is full
11            Sleep();
12      GetData();
13      MakeFrame(Sn);
14      StoreFrame(Sn);
15      SendFrame(Sn);
16      Sn = Sn + 1;
17      if(timer not running)
18           StartTimer();
19   }
20
```

S_f(send window, the first outstanding frame), S_n (send window, the next frame to be sent), Ssize (send window, size),

(continued)

Algorithm 11.7 *Go-Back-N sender algorithm* (continued)

```
21   if(Event(ArrivalNotification)) //ACK arrives
22   {
23      Receive(ACK);
24      if(corrupted(ACK))
25           Sleep();
26      if((ackNo>Sf)&&(ackNo<=Sn)) //If a valid ACK
27      While(Sf <= ackNo)
28        {
29         PurgeFrame(Sf);
30         Sf = Sf + 1;
31        }
32        StopTimer();
33   }
34
35   if(Event(TimeOut))                    //The timer expires
36   {
37    StartTimer();
38    Temp = Sf;
39    while(Temp < Sn);
40      {
41       SendFrame(Sf);
42       Sf = Sf + 1;
43      }
44   }
45 }
```

Algorithm 11.8 *Go-Back-N receiver algorithm*

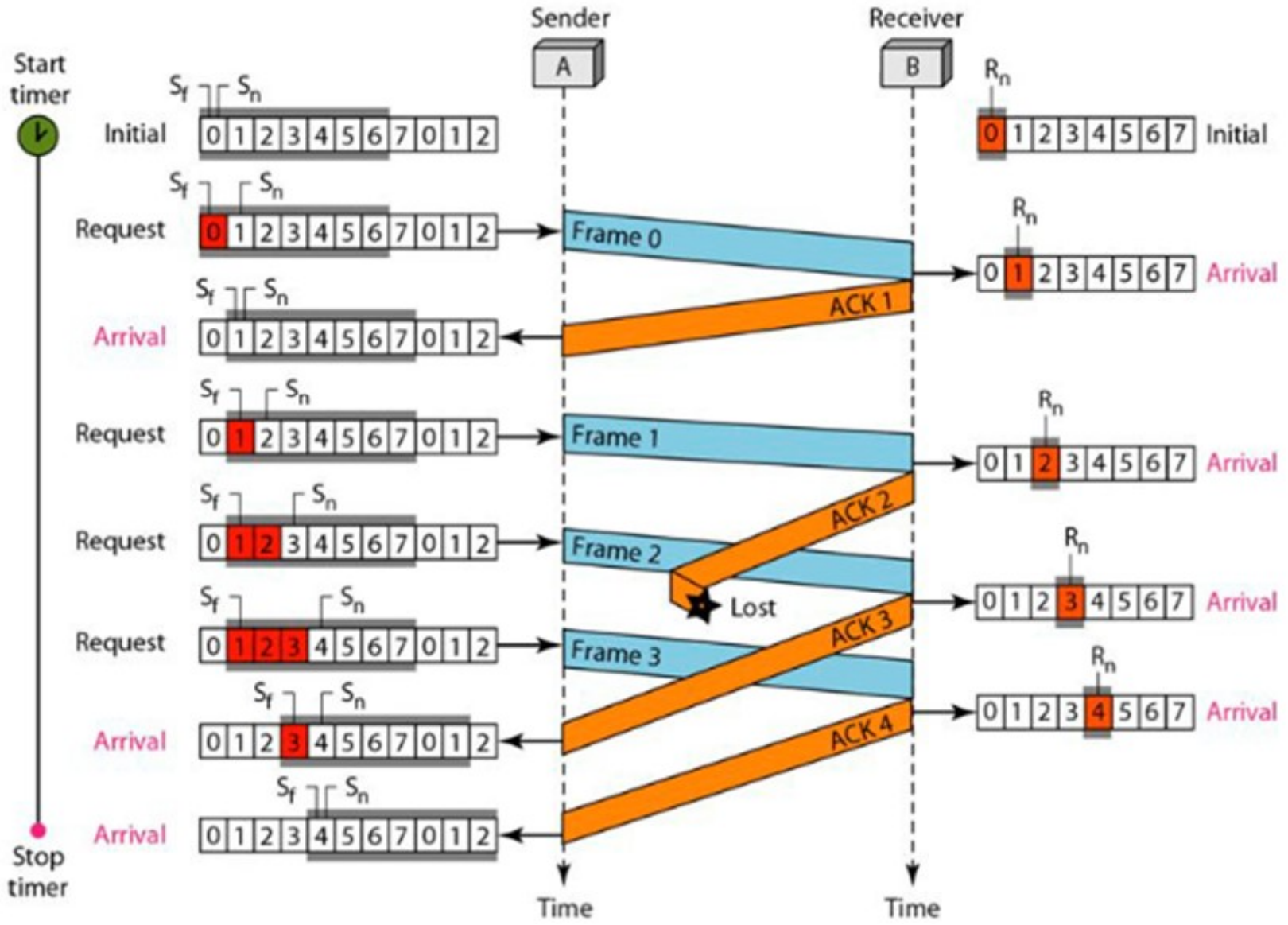
```
1  Rn = 0;
2
3  while (true)                          //Repeat forever
4  {
5    WaitForEvent();
6
7    if(Event(ArrivalNotification)) /Data frame arrives
8    {
9       Receive(Frame);
10      if(corrupted(Frame))
11           Sleep();
12      if(seqNo == Rn)                  //If expected frame
13      {
14        DeliverData();                 //Deliver data
15        Rn = Rn + 1;                   //Slide window
16        SendACK(Rn);
17      }
18   }
19 }
```

- R_n (Receive window, net frame expected)

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 (Continue)

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*