Module 2: Data Link and MAC Layer

Part-II Framing and Flow Control

Dr. Prasanna Shete
Dept. of Computer Engineering
K. J. Somaiya College of Engineering

Slide Source: B. A. Forauzan, Data Communications and Networking, McGraw-Hill Online Learning Centre
http://highered.mheducation.com/sites/0072967757/information_center_view0/index.html

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

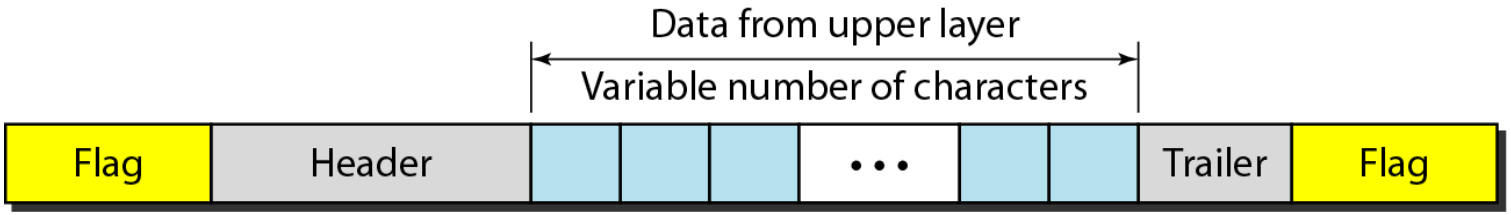
Framing in DLL

- Framing separates a message from one source to a destination, or from other message to other destinations
- Messages divided into smaller frames for efficient flow and error control
  - For large frame single-bit error would require retransmission of whole message
- Types: Fixed-size framing, Variable-size framing
- In Fixed-size framing no need for defining boundaries of the frame; size used as delimiter
- In variable-sized framing mechanism to define the end of the frame and the beginning of the next is required
  - Character oriented approach or bit-oriented approach

Character-Oriented Protocols

- Data to carried are 8-bit characters from a coding system e.g ASCII
  - Source and destination addresses and other control information carried in the Header or error detection /correction bits carried in the Trailer are multiples of 8-bits
- To separate one frame from the next, an 8-bit flag is added at the beginning and the end of a frame
- Flag composed of special character
- Any character not used for text-communication can be selected as flag

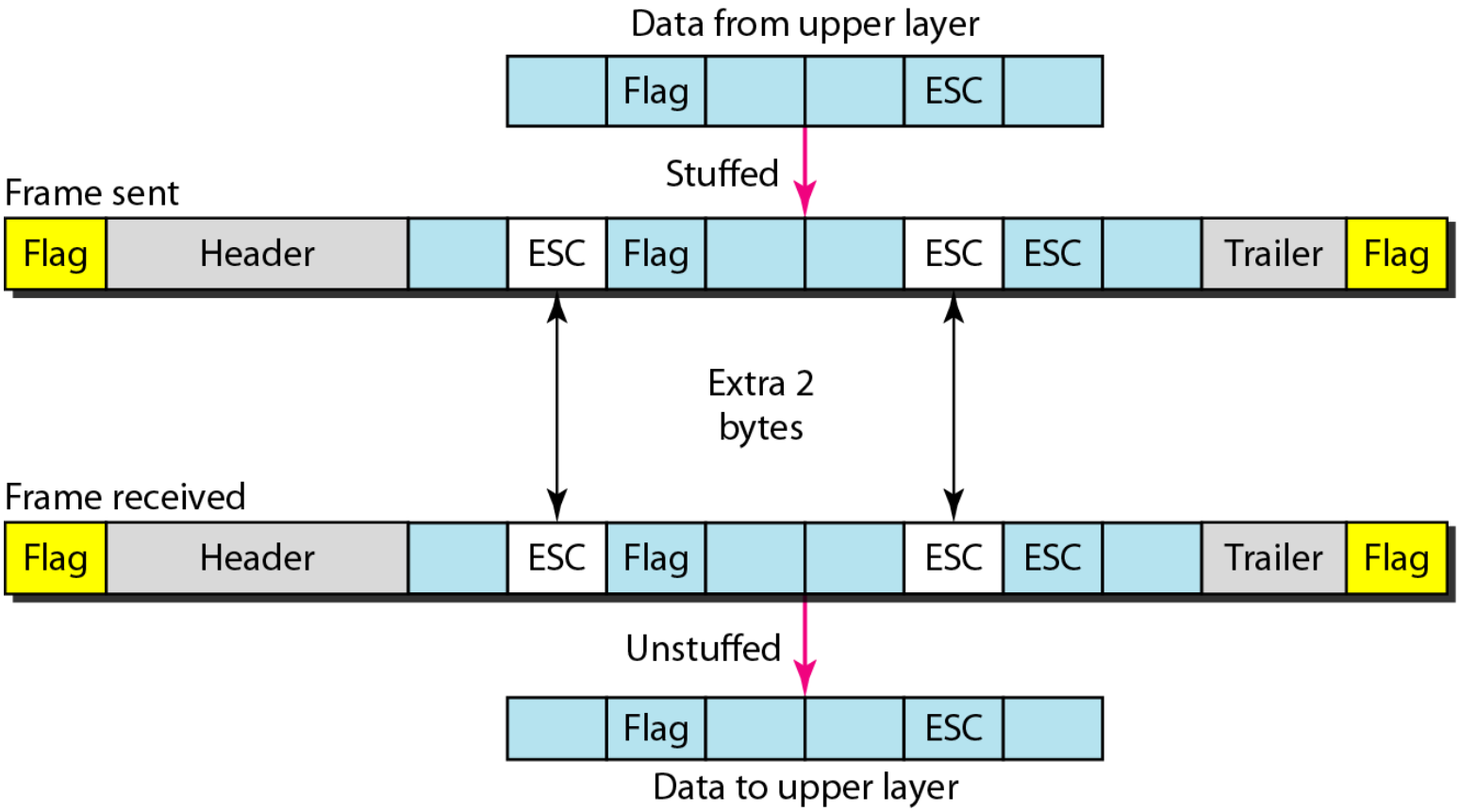
Figure 11.1 *A frame in a character-oriented protocol*



Character-Oriented Protocols contd…

- Problem: Pattern used for flag could be part of information (images, video or audio)
- On receiving such pattern in the middle of the data, Receiver thinks it has reached the end of the frame
- Solution: ***Byte-stuffing*** (or character stuffing)
- when there is a character with the same pattern as the flag, data section is stuffed with an extra byte- called escape character (**ESC),** having predefined bit pattern
- Receiver on encountering the ESC character, removes it from the data section and treats the next character as data, not as delimiting flag
- What if data consists bit pattern same as ESC?

Figure 11.2 *Byte stuffing and unstuffing*



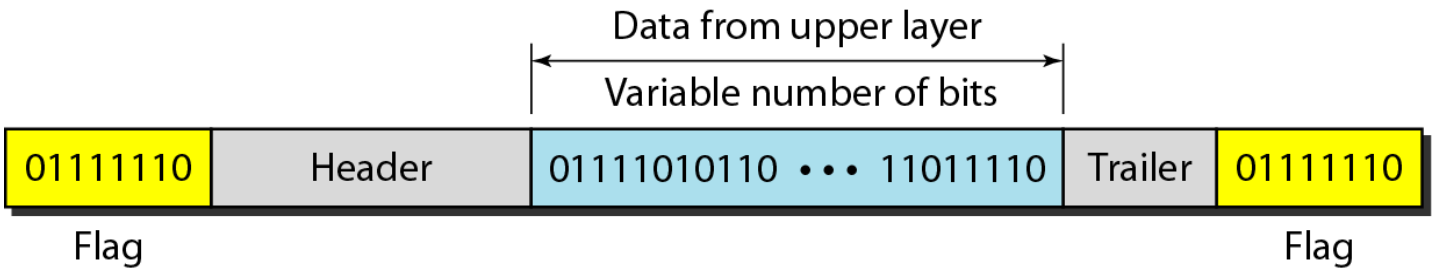
Note

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

Bit-Oriented Protocols

- Data section of a frame is a sequence of bits (representing text, graphic, audio, video etc.)
- To separate one frame from the other a delimiter is needed
- Special 8-bit pattern flag 01111110 used in most protocols
- If the flag pattern appears in the data, a mechanism is needed to inform the receiver that this is not the end of the frame
- Achieved by ***bit-stuffing***; a single bit is stuffed in the data
- If a 0 and five consecutive 1 bits are encountered, an extra 0 is added
  - Stuffed bit is removed from the data by the receiver

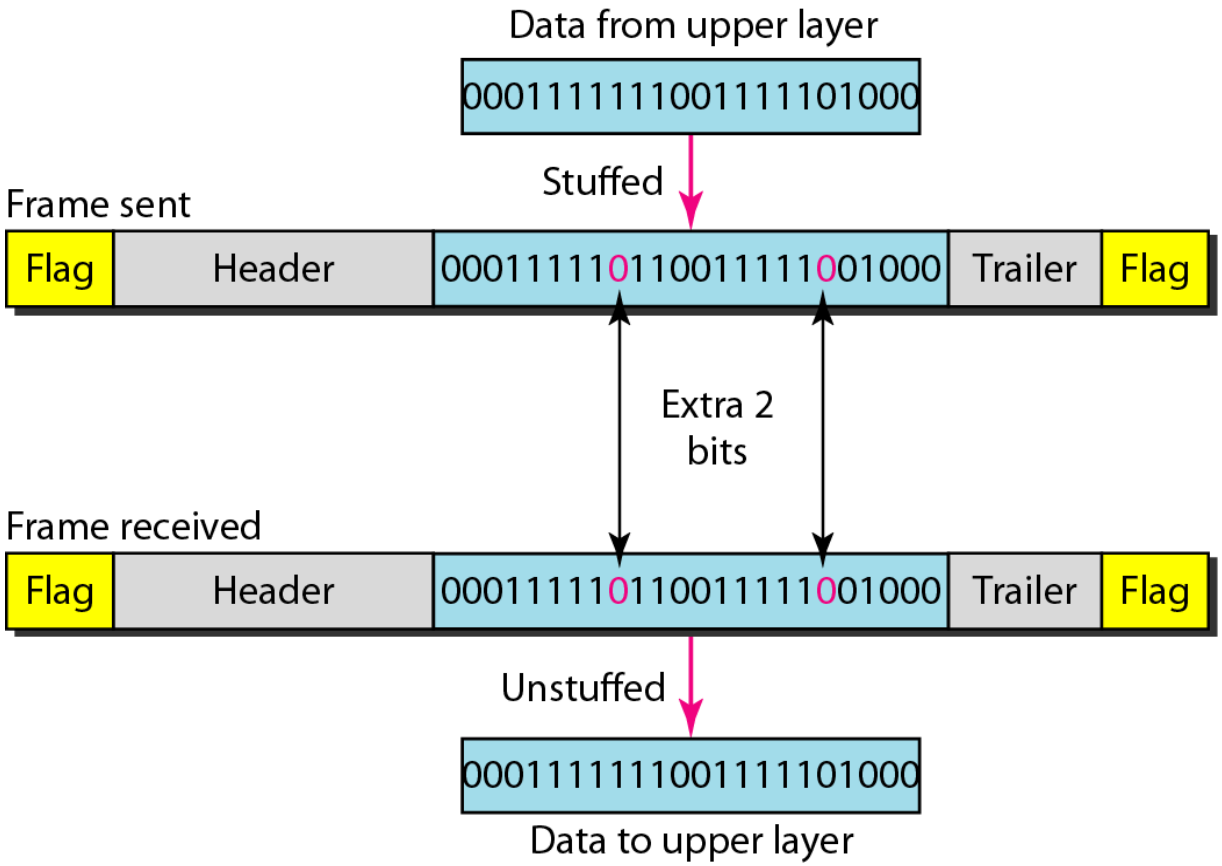
Figure 11.3 *A frame in a bit-oriented protocol*



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

Flow control ***coordinates the amount of data that can be sent before receiving an acknowledgement***

-Flow of data should not be allowed to overwhelm the receiver

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.

Error control* – *Error detection and correction

-When error is detected during exchange of data, specified frames are retransmitted

Note

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

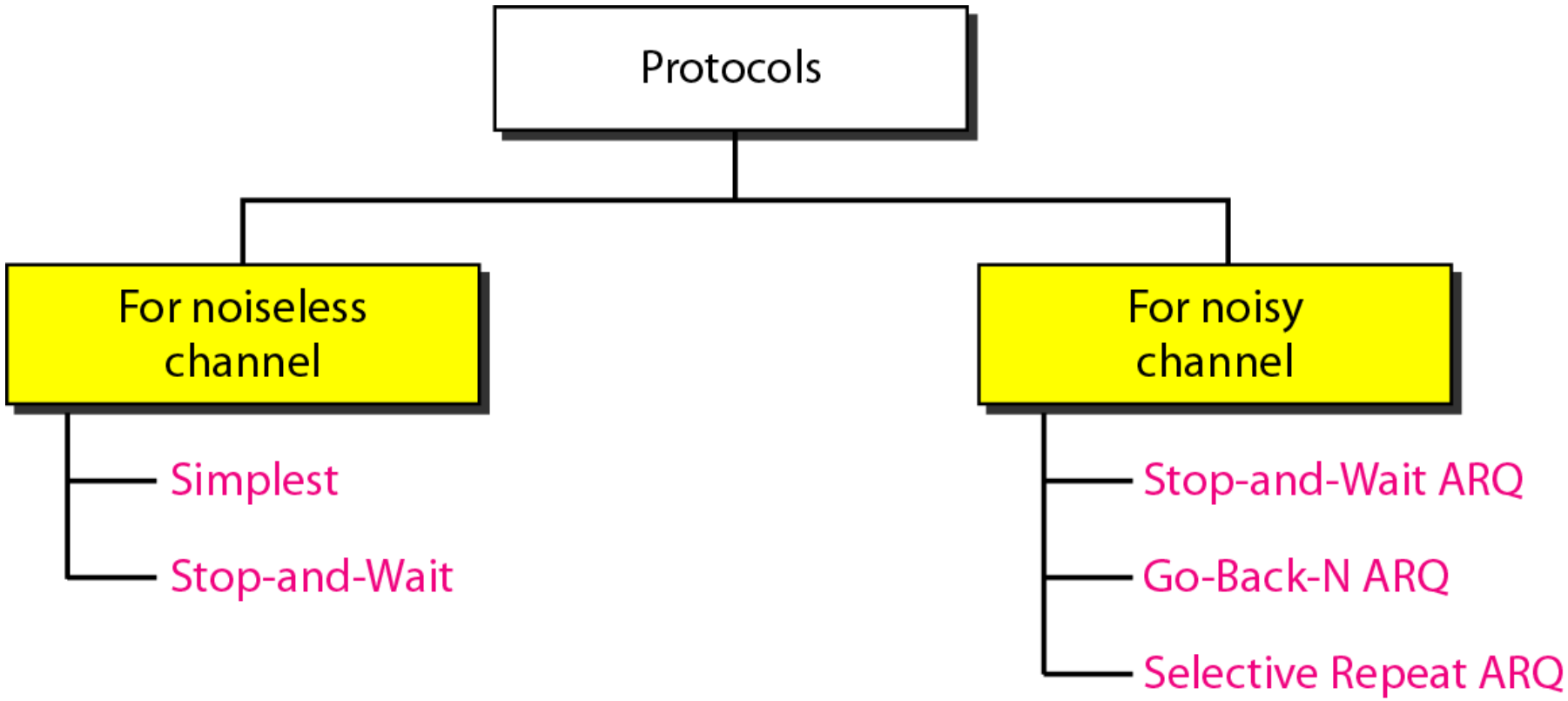
11-3 PROTOCOLS

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:
- Simplest protocol and Stop-and-Wait Protocol

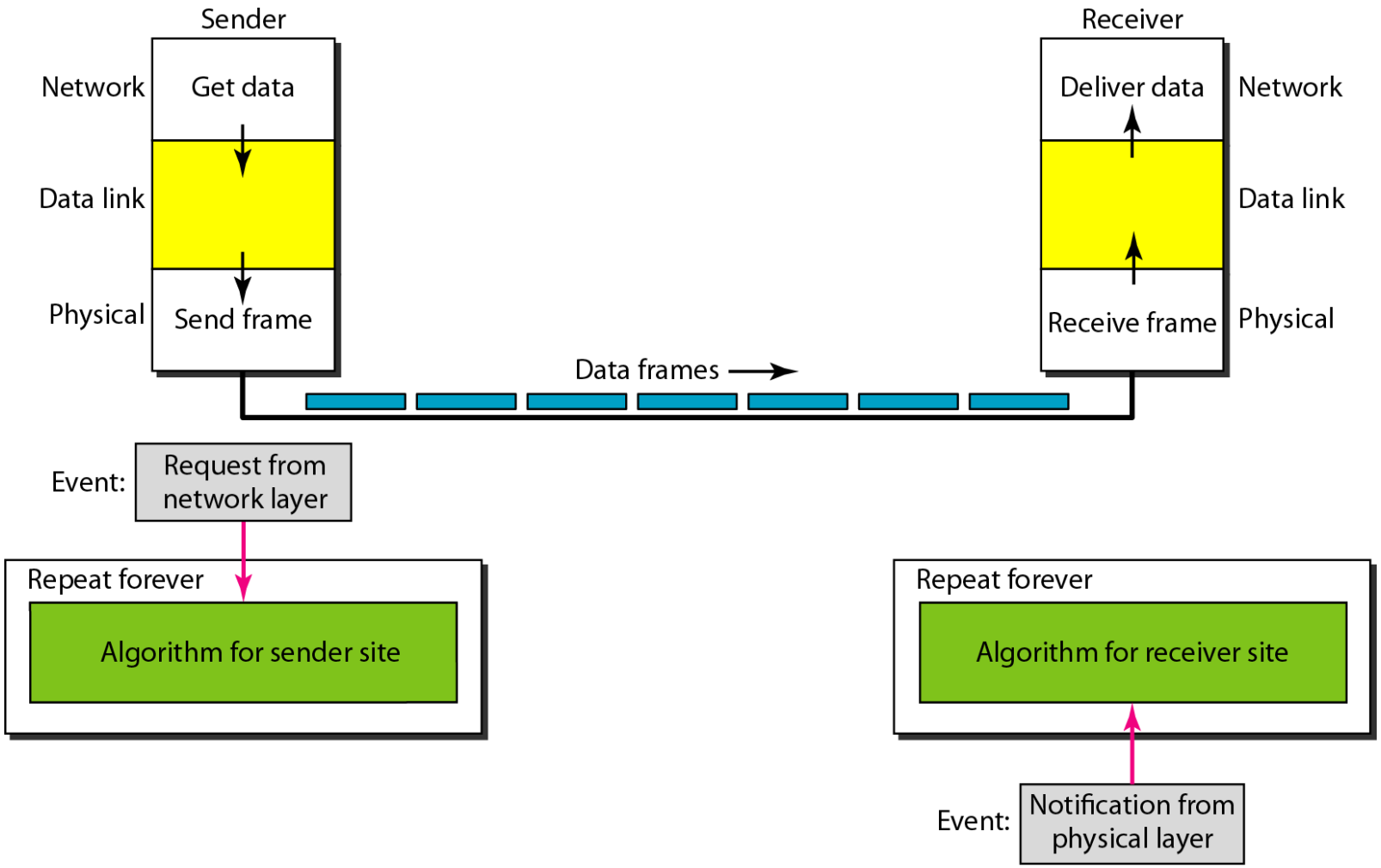
Topics discussed in this section:

Simplest Protocol
Stop-and-Wait Protocol

Simplest Protocol

- No flow control
- At Sender site: Data link layer gets data from Network layer, makes a frame out of the data, and sends it
- At Receiver : Data link layer receives a frame from its physical layer, extracts data, and delivers to the network layer.
- Sender cannot send a frame until its Network layer has data packet to send
- Receiver cannot deliver data packet to its Network layer until a frame arrives at PHY layer
  - Event driven protocol, implemented as procedures at sender and receiver (both running constantly, since they do not know when corresponding events will occur)

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



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 }
```

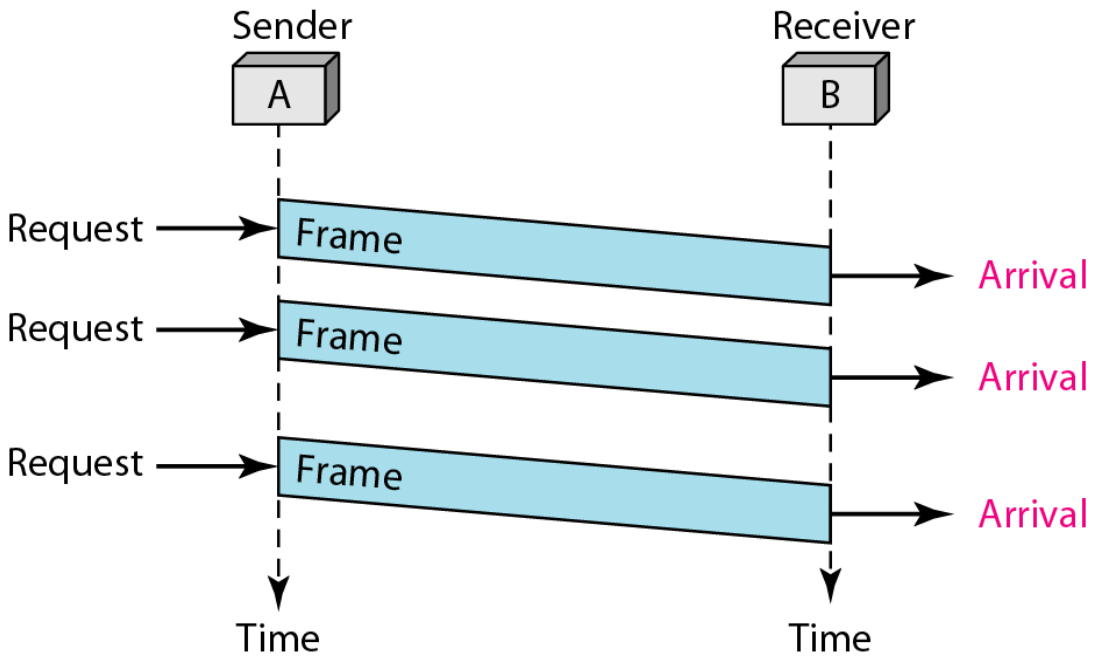
Example 11.1

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.

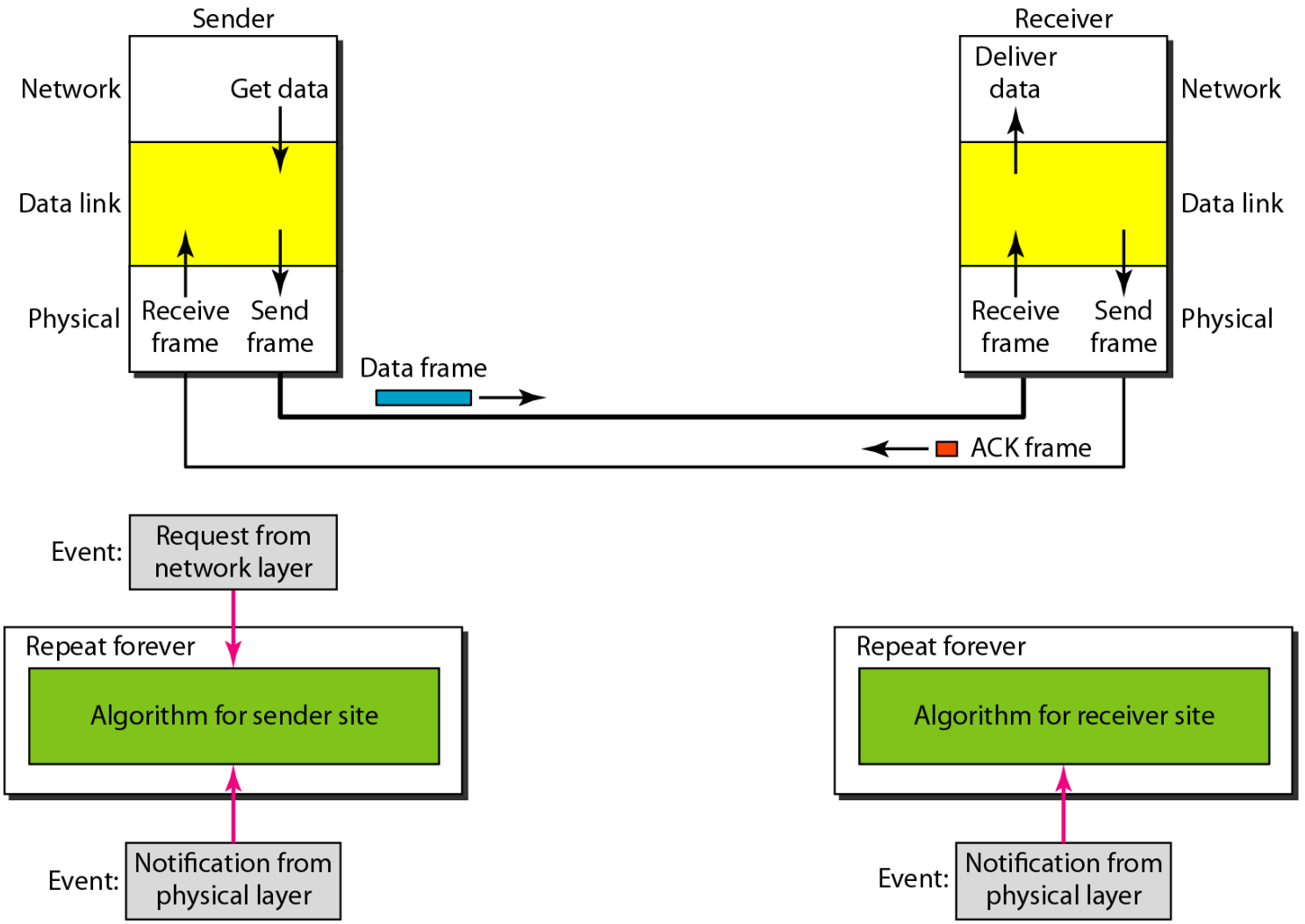
Figure 11.7 *Flow diagram for Example 11.1*



Stop-and-Wait Protocol

- If data frames arrive at the receiver faster than that can be processed, the receiver may discard (if no sufficient storage space available)
- To prevent overwhelming of the receiver, a mechanism needed to tell the sender to slow down→Stop-and-Wait:
- Sender sends one frame, stops until it receives confirmation from the Receiver to go ahead, and then sends the next frame
  - Thus flow control added to previous protocol
- Here, traffic in both directions; sender to receiver and vice-versa
  - either one data frame on forward channel or one ACK frame on the reverse channel → Half-Duplex links required

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



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 }
```

Algorithm 11.4 *Receiver-site algorithm 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 }
```

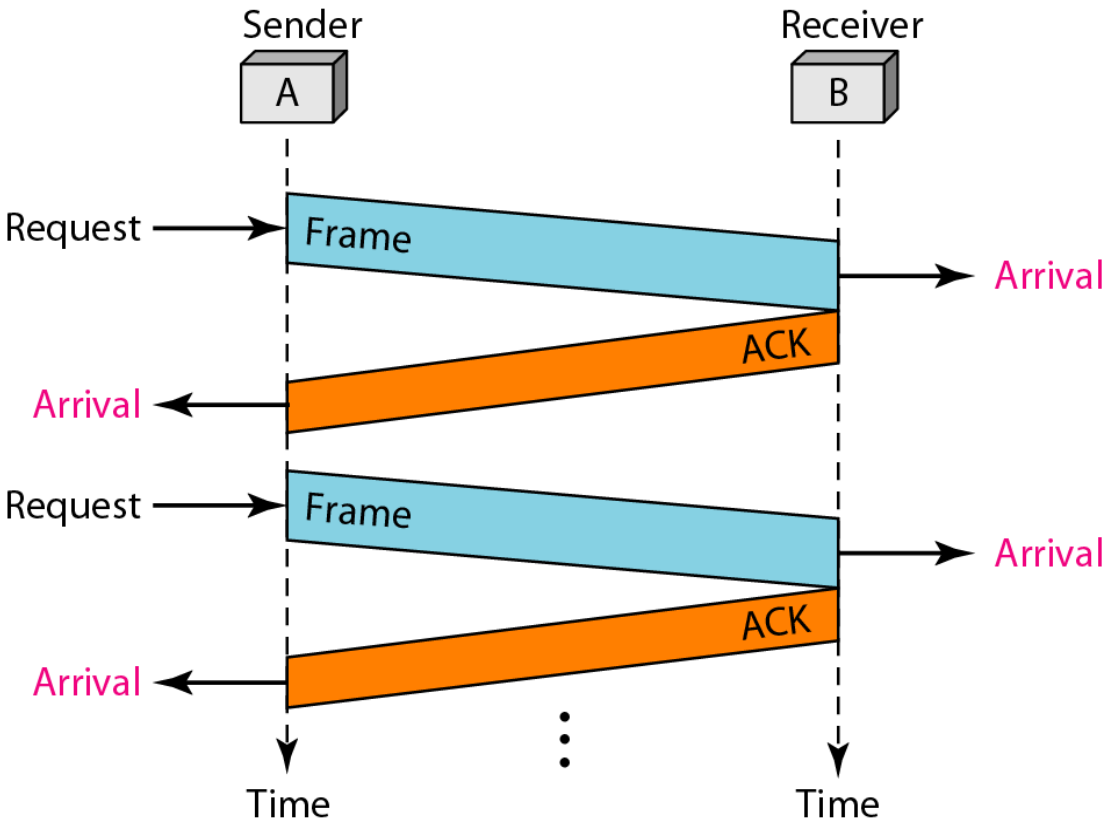
Example 11.2

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 ARQ

- Stop-and-Wait ARQ adds a simple error control mechanism to Stop-and-Wait protocol
- When the frame arrives at the Receiver, it is checked and if it is corrupted, it is silently discarded →not acknowledged
  - detection of errors through silence of the Receiver
- Handling of lost frame: Lost frames are difficult to handle than corrupted frames
  - Received frame could be correct one, or a duplicate, or out of order
- Solution is to number the frames (sequence number)
  - when the receiver receives an out of order frame, this means the frames were either lost or duplicated

Stop-and-Wait ARQ contd..

- Corrupted and lost frames need to be resent
  - How will sender know which frame to resend?
- Sender keeps a copy of the sent frame and starts the Timer
- If the timer expires and there is no ACK, the frame is resent; the copy is held, and the timer is restarted

- ACK frame can also be corrupted → needs redundancy bits, and sequence numbers
  - Sender simply discards a corrupted ACK frame or out-of-order ACK frame

Note

Error correction in Stop-and-Wait ARQ is done by keeping a copy of the sent frame and retransmitting of the frame when the timer expires.

Stop-and-Wait ARQ: Sequence / ACK Numbers

- Sequence number field added to data frame
- To keep the frame size minimal, Range of sequence numbers should be as small as possible; can wrap around
  - E.g. for sequence number of *m* bits, Range→0 to 2^m-1, and then repeated
- Receiver only needs to know correct receipt of data frame & ACK, and loss of data or ACK
- →Sequence numbers needed are *x* (current frame) and *x*+1 (next frame)
  - then if *x*=0, *x*+1= 1; i.e SeqNos will be 0,1,0,1…
- ACK numbers announce the sequence number of next frame expected by the receiver
  - E.g. if receiver receives frame 0 safe and sound, it sends ACK frame with seqNo. 1, meaning next frame 1 is expected

Note

In Stop-and-Wait ARQ, we use sequence numbers to number the frames. The sequence numbers are based on modulo-2 arithmetic.

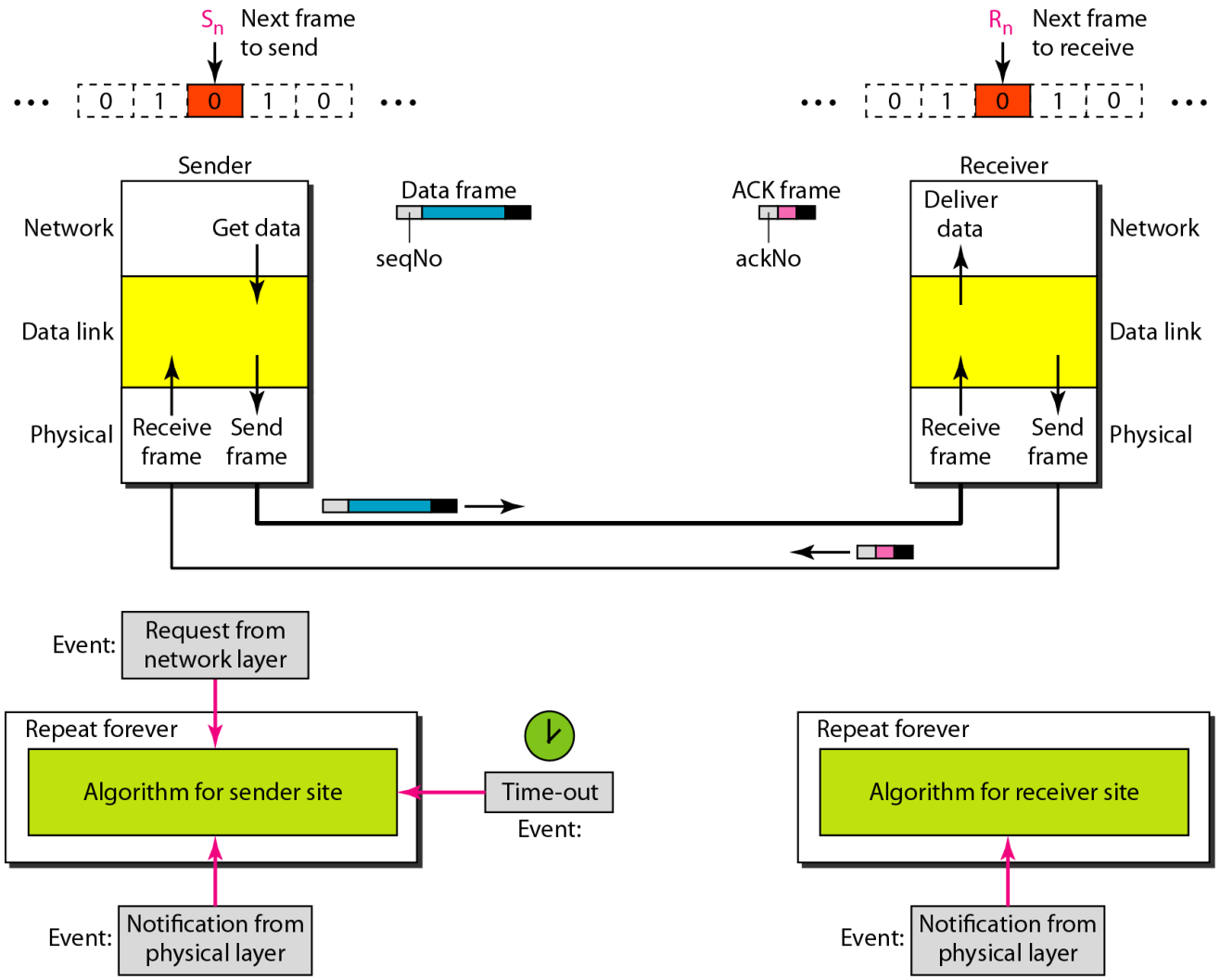
Note

In Stop-and-Wait ARQ, the acknowledgment number always announces in modulo-2 arithmetic the sequence number of the next frame expected.

Stop-and-Wait ARQ: Design

- Sender keeps a copy of the last frame transmitted until it receives an acknowledgement
- Data frame uses a seqNo; an ACK frame uses an ackNo.
- Two control variables S_n (sender next frame to send) and R_n (receiver next frame expected)
- Frame is sent- value of S_n *incremented* (modulo 2; if its 0, it will become 1 and vice-versa)
- Frame is received- value of R_n *incremented* (modulo2)
- 3 events can happen at sender site; 1 event can happen at receiver site
- S_n points to the slot that matches the sequence number of the frame that has been sent, but not acknowledged;
- R_n points to the slot that matches the seq. no. of the expected frame

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



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)

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  }
```

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 }
```

Example 11.3

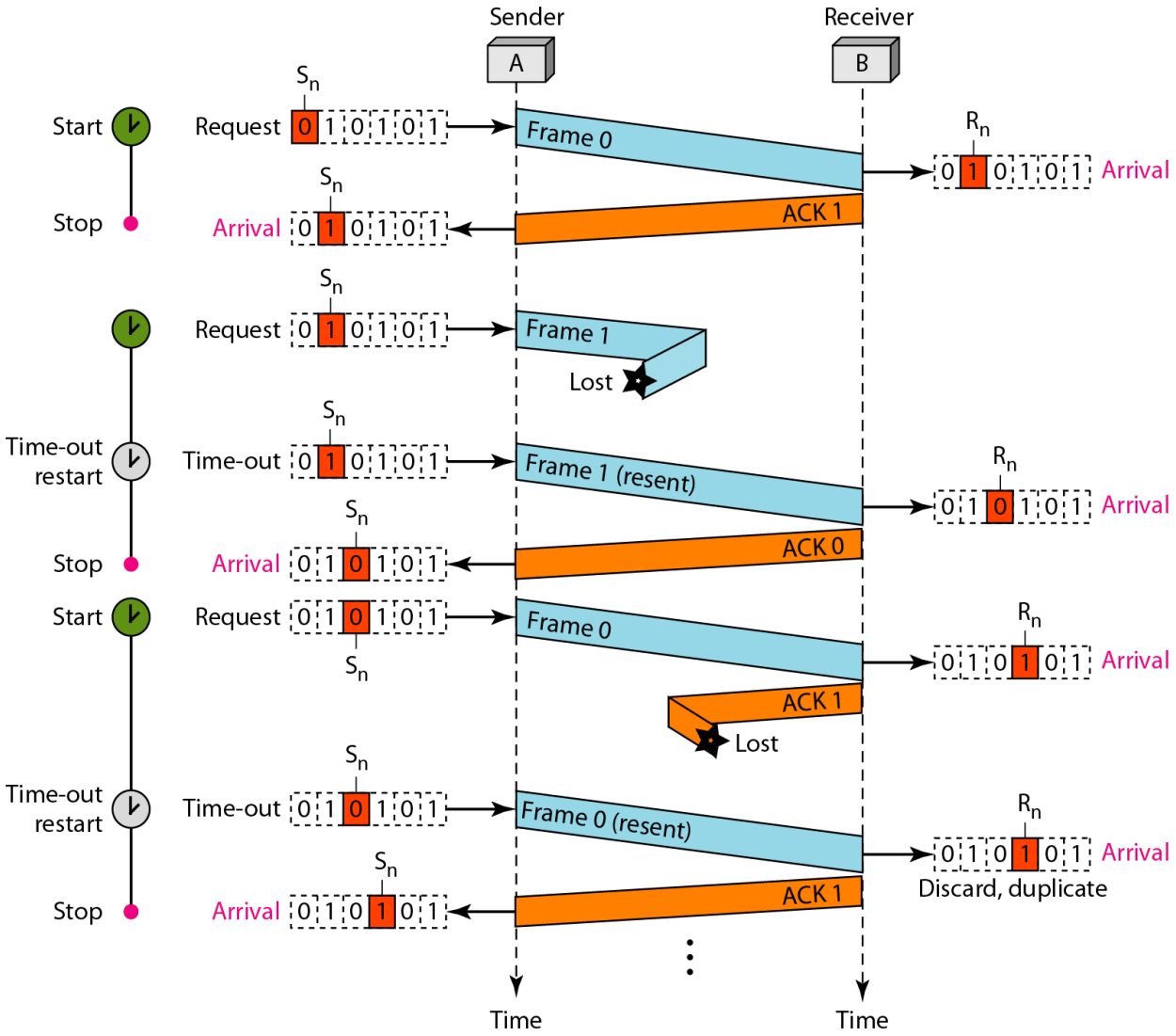
Figure 11.11 shows an example of Stop-and-Wait ARQ.

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*



Stop-and-Wait ARQ: *Efficiency*

- Stop-and-Wait ARQ is very in-efficient, when the channel has large Bandwidth and round-trip Delay (i.e ***Bandwidth-Delay*** product)

- *Bandwidth-Delay* product is the measure of the number of bits the sender can send out while waiting for feedback from the receiver

Example 11.4

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.

a) What is the bandwidth-delay product?

b) If the system data frames are 1000 bits in length, what is the utilization percentage of the link?

Solution

The bandwidth-delay product= BW (bps) * Delay (s)

$$(1 \times 10^6) \times (20 \times 10^{-3}) = 20{,}000 \text{ bits}$$

Example 11.4 (continued)

b) According to BW-Delay product, 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 11.5

What is the utilization percentage of the link in Example 11.4 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

- Pipelining means beginning a new task before the previous task has been completed
- No pipelining in Stop-and-Wait ARQ
  - its needed to wait for a frame to reach the destination and be acknowledged, before sending next frame
- With *pipelining,* several frames can be sent before receiving acknowledgement for previous frames
- →Pipelining improves the transmission efficiency, if the number of bits in transition is large w.r.t BW-delay product
  - Used in next schemes; Go-back-N ARQ and Selective Repeat ARQ

Go-Back-N ARQ

- To improve the efficiency of transmission, multiple frames must be in transition while waiting for acknowledgements

- Go-back-N can send multiple frames before receiving acknowledgements
- The sender keeps copy of sent frames until the ACKs arrive

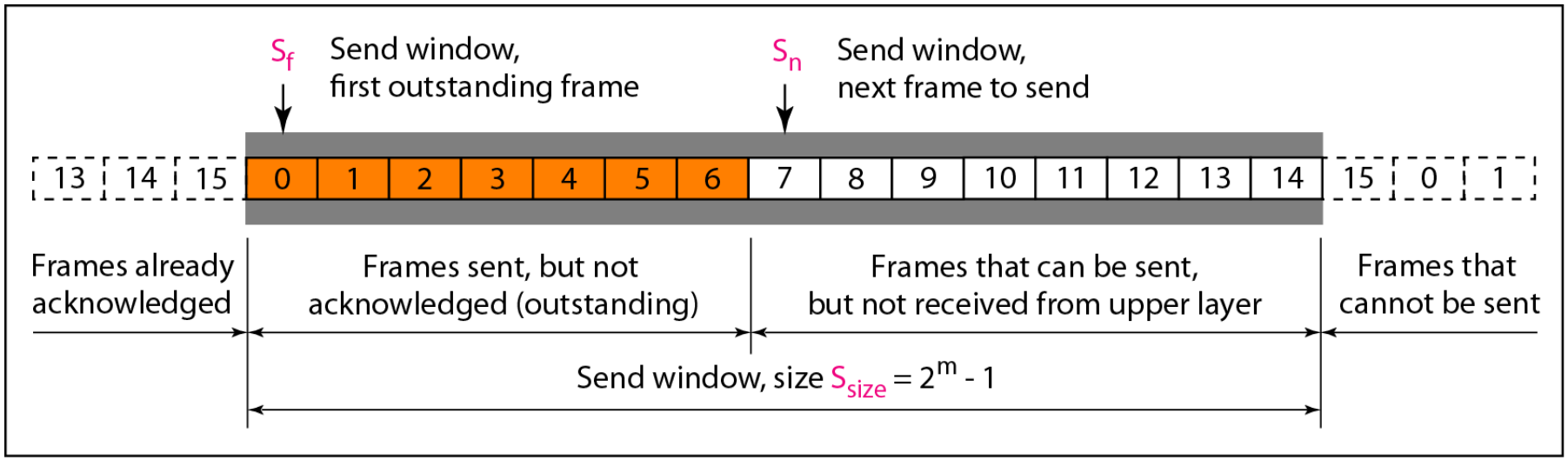
Go-Back-N ARQ: Sequence Numbers and Sliding window

- Frames from Sender are numbered sequentially
  - Limit set, to keep the frame header size minimal
- If *m* bit Sequence number allowed, the sequence numbers range from→0 to 2^m-1, and then repeated
  - For example if m=4, seqNos are: 0 to 15
  - i.e sequence numbers are modulo 2^m
- Sliding Window: Defines the range of sequence numbers that is the concern of the sender and receiver
  - send sliding window, and receive sliding window
- The send window is an imaginary box covering the sequence numbers of data frames which can be in transit
  - Maximum size of the window is 2^m-1
- Three variables define the size and location: S_f (first outstanding frame), S_n (next frame to be sent) and S_{size}

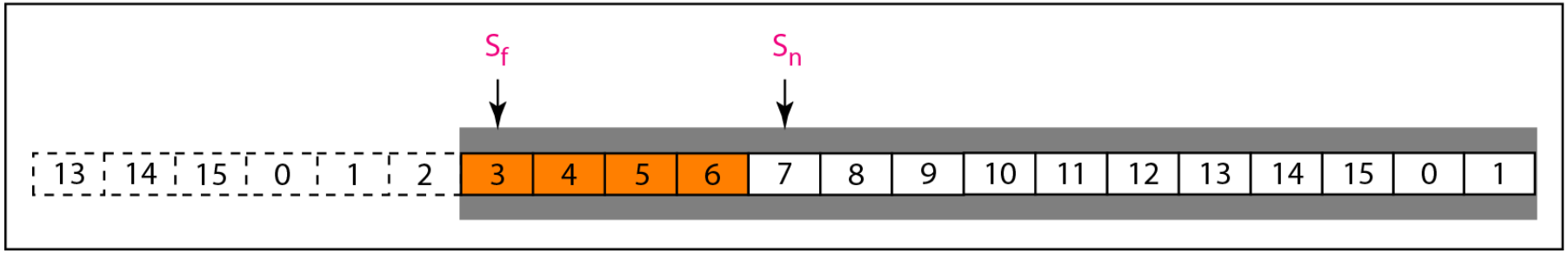
Note

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.

- The receive window ensures that correct frames are received and correct ACKs are sent
  - Any frame arriving out of order at the receiver, is discarded and needs to be resent

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

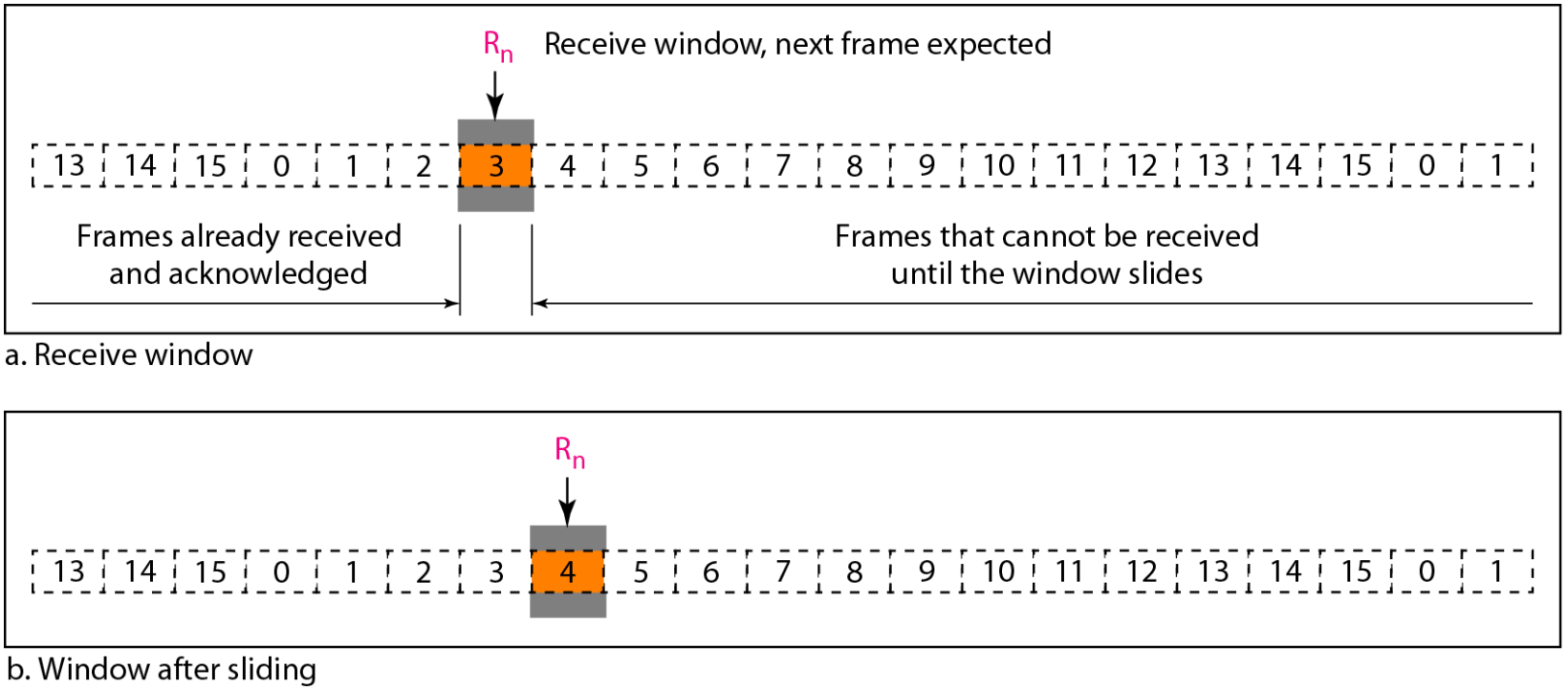
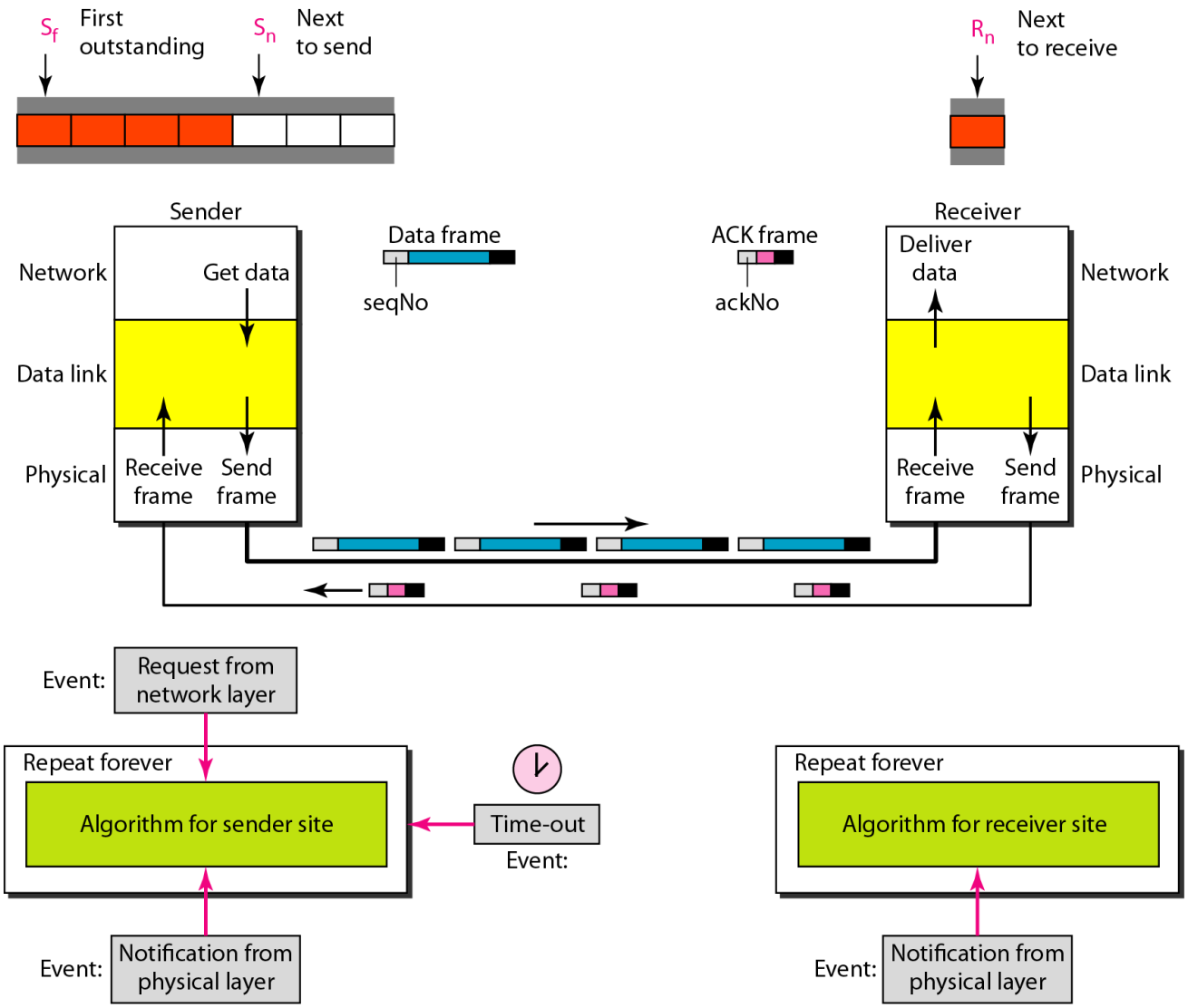


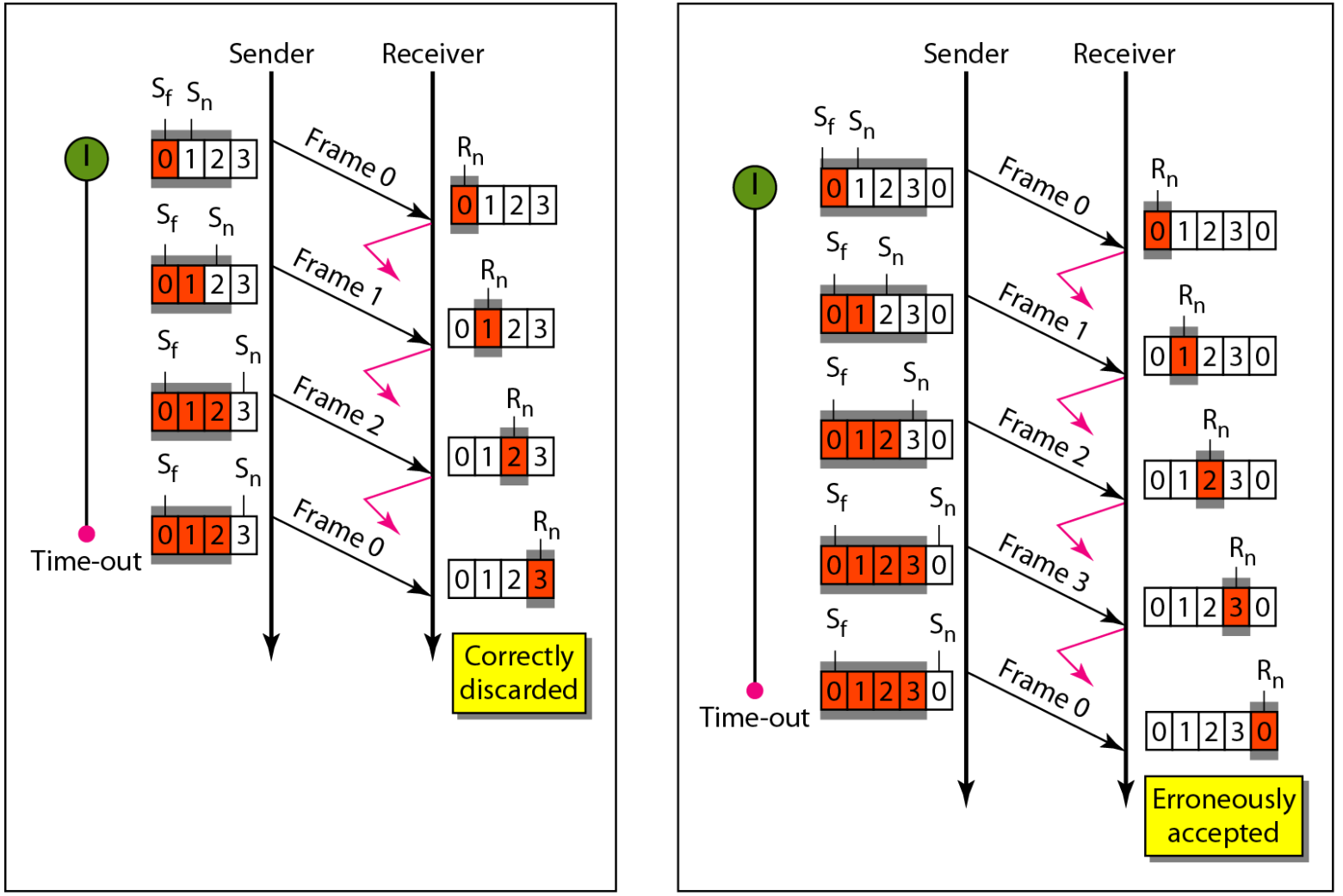
Figure 11.14 *Design of Go-Back-N ARQ*



Go-Back-N ARQ: Design

- Multiple frames can be in transit in the forward direction, and multiple acknowledgements in the reverse direction
  - Send window allows to have as many frames in transition as there are slots in it
- Timers: Only one timer used
- Acknowledgement: If a frame is damaged or received out of order, receiver discards all subsequent frames until it receives the one it is expecting
  - Need not have to acknowledge each frame received; can send one cumulative acknowledgement for several frames
- Resending a frame: When the timer expires, the sender resends all outstanding frames
  - E.g Sender sent 5 frames, but timer for frame 3 expires →sender goes back and sends frames 3, 4 and 5 again

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.

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
```

(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 }
```

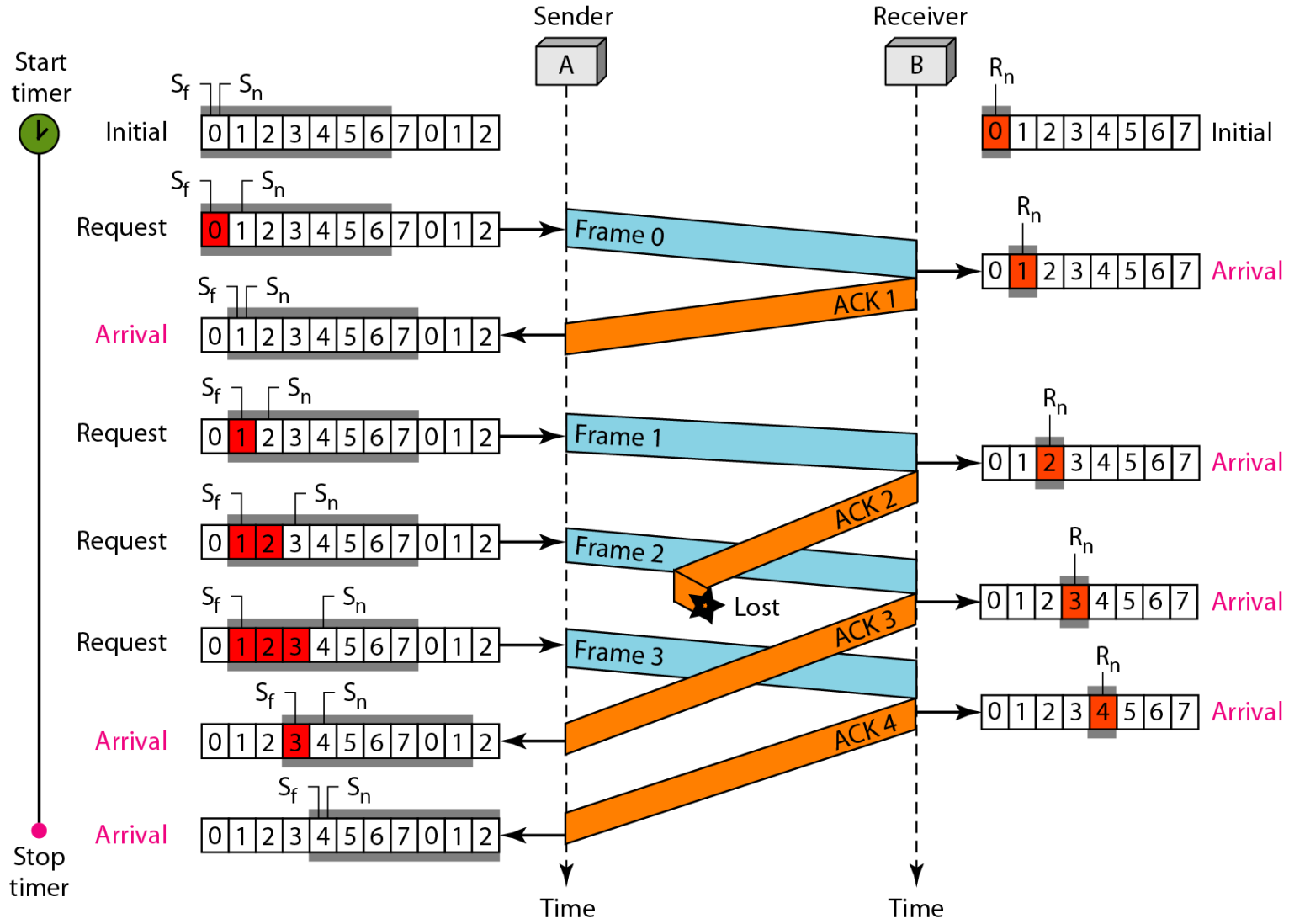
Algorithm 11.8 *Go-Back-N receiver algorithm*

```
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 }
```

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.

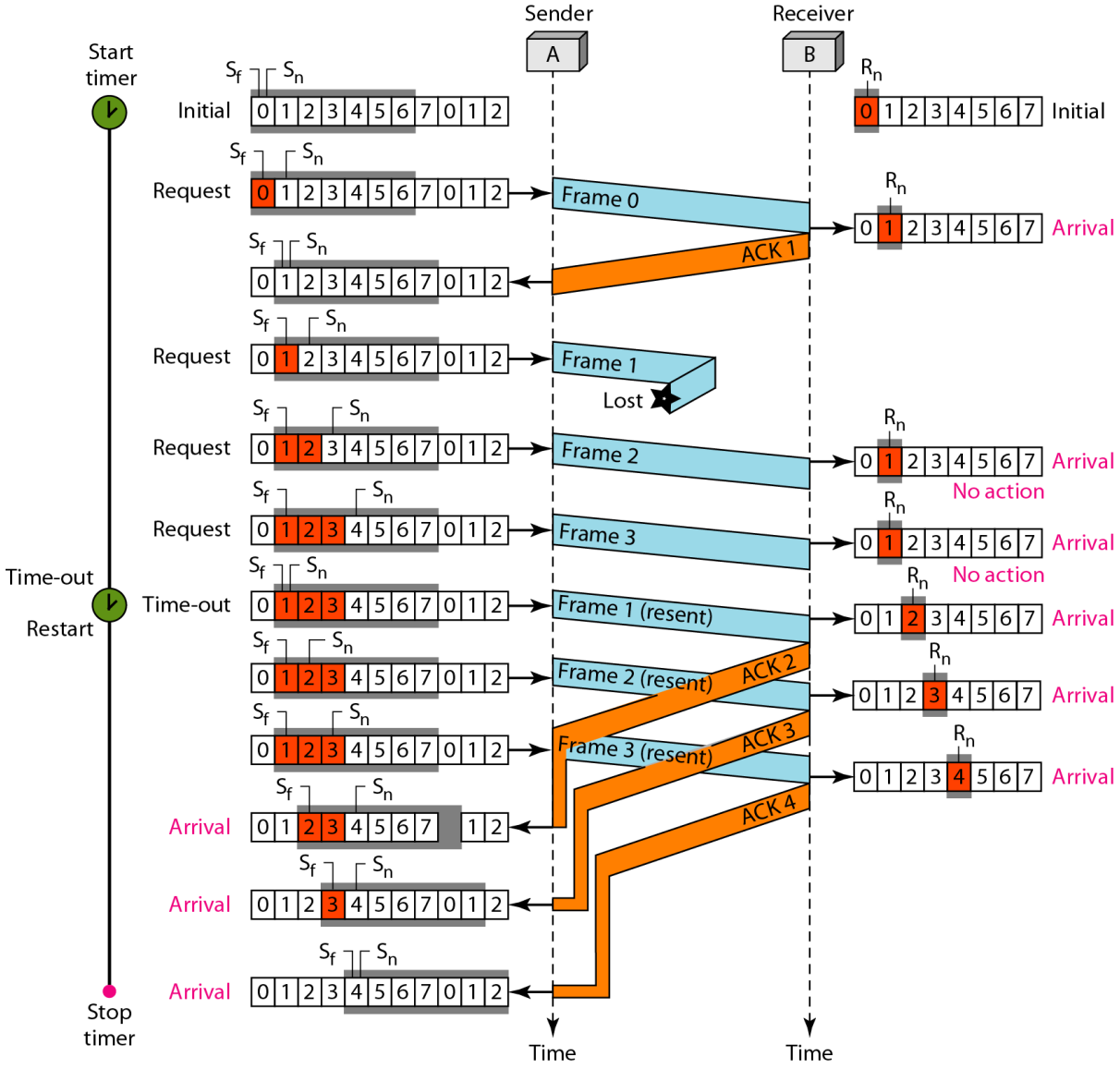
The sender receives no acknowledgment about frames 1, 2, or 3. Its timer finally expires. The sender sends all outstanding frames (1, 2, and 3) because it does not know what is wrong.

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

- Go-back-N ARQ sends multiple frames when just one frame is damaged→ is inefficient
- The receiver keeps track of only one variable; out of order frames are simply discarded
  - There is no buffer to hold out of order frames
- In noisy channel frames have higher probability of damage; hence more frames need to be resent
  → consumes more BW and slows down transmission
- Selective Repeat ARQ: only damaged frame is resent; does not resend N frames
- Receiver operation is complex as compared to Go-back-N ARQ

Selective Repeat ARQ: Sequence Numbers and Sliding window

- Uses two windows; send window, and receive window
- Size of send window is 2^{m-1} , much smaller as compared to Go-back-N
  - If m=4, the seqNos are 0 to 15, but size of window is 8
  - Smaller window size means less (pipeline) efficiency; compensated by fewer duplicate frames
- Receive window size is same as send window
- Selective repeat allows as many frames as the size of receive window to arrive out of order
- Out of order frames are stored in a buffer, until there is a set of in-order frames to be delivered to the network layer
  - Receiver never delivers packets out of order to the network layer

Figure 11.18 *Send window for Selective Repeat ARQ*

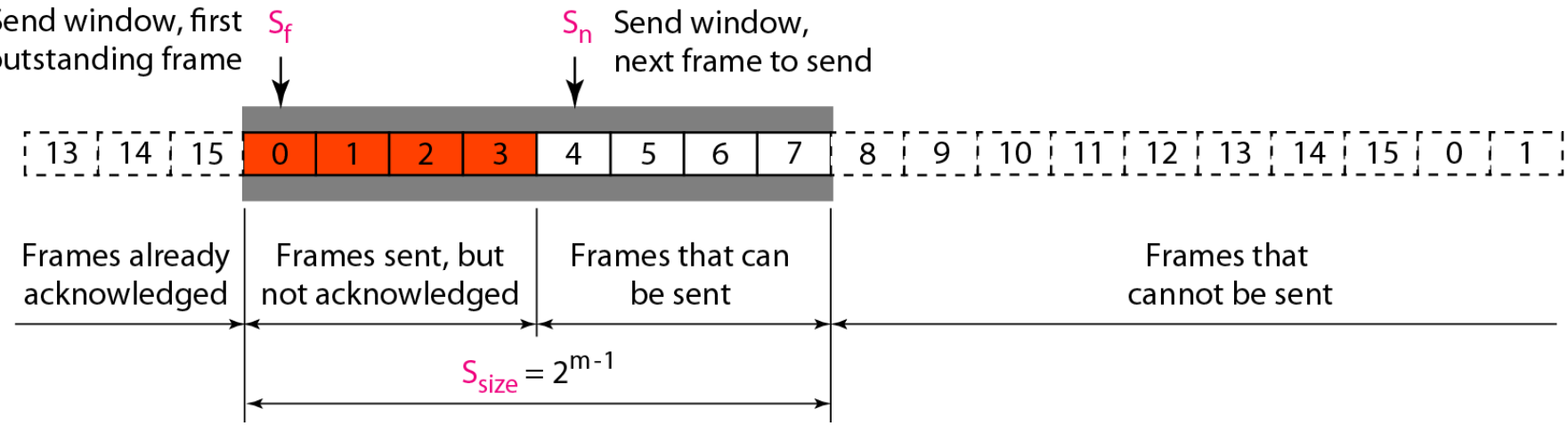
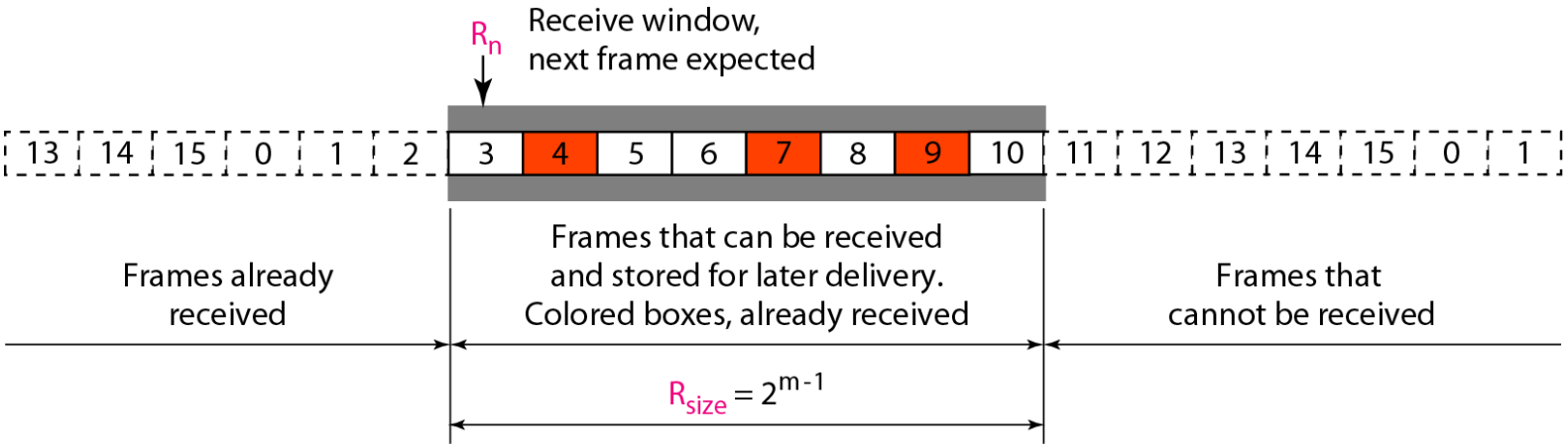


Figure 11.19 *Receive window for Selective Repeat ARQ*



Selective Repeat ARQ: Design

- Similar to Go-Back-N ARQ; Multiple frames can be in transit in the forward direction, and multiple acknowledgements in the reverse direction
- Timers: Multiple timers used; each per frame
- Acknowledgement: If a frame is received out of order, it is not discarded
- Resending a frame: When the timer for a frame expires, the sender resends that frame only
  - E.g Sender sent 5 frames, but timer for frame 3 expires →sender sends frame 3 again
  - Sender and Receiver window size must be at most one-half of 2^m

Figure 11.20 *Design of Selective Repeat ARQ*

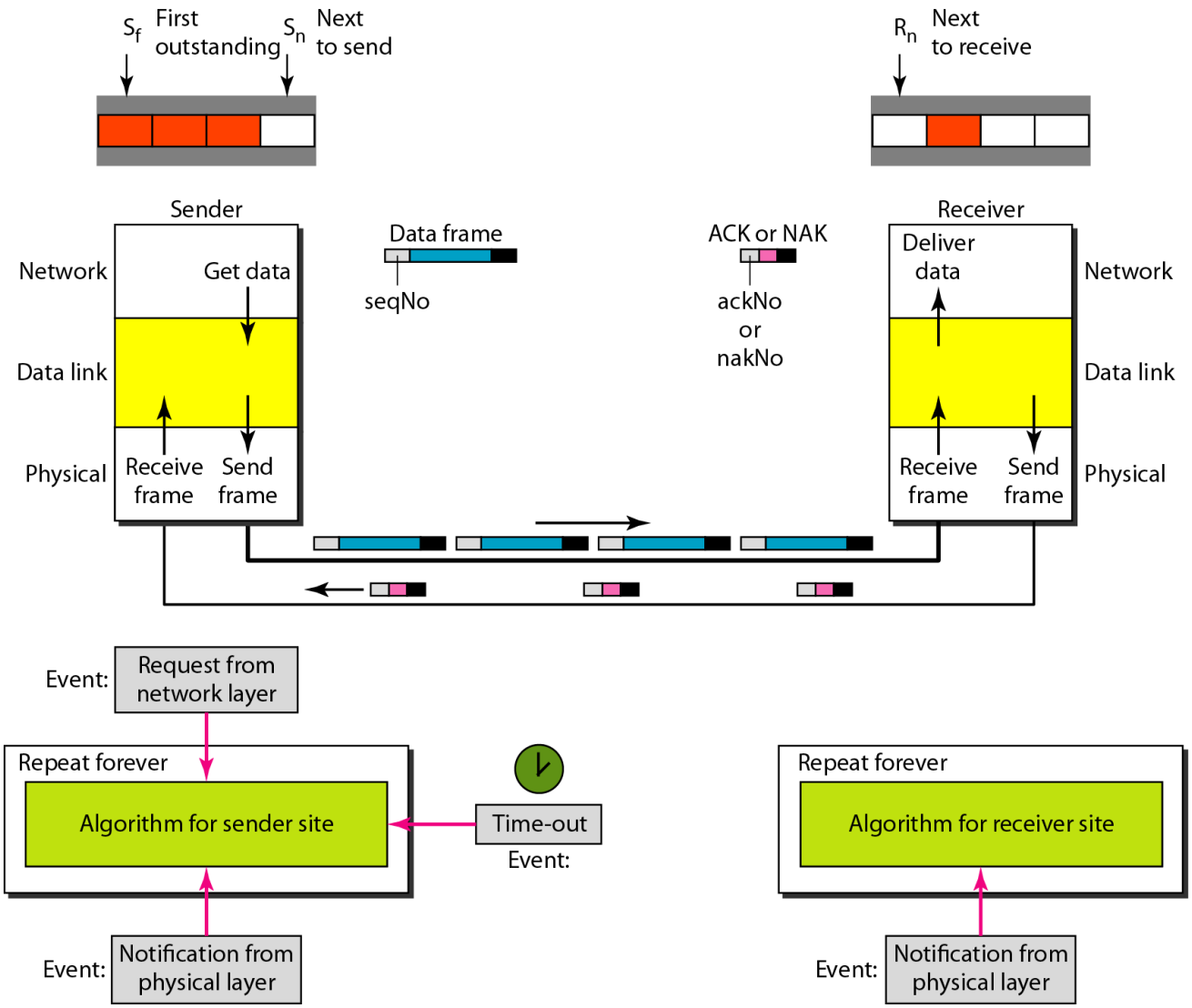
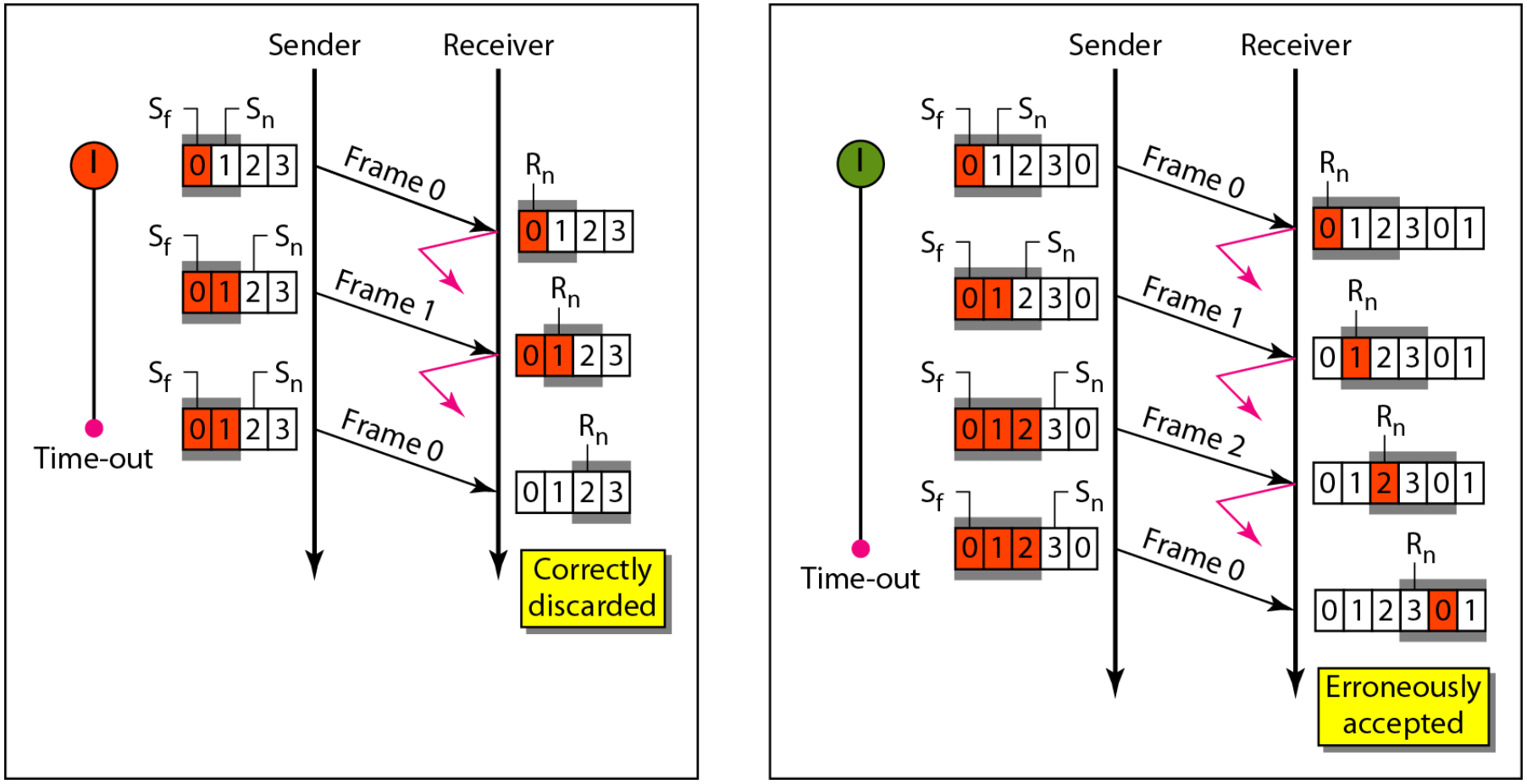


Figure 11.21 *Selective Repeat ARQ, window size*



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.

Algorithm 11.9 *Sender-site Selective Repeat 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))            //There is 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     StartTimer(Sn);
18   }
19
```

(continued)

Algorithm 11.9 *Sender-site Selective Repeat algorithm* (continued)

```
20  if(Event(ArrivalNotification)) //ACK arrives
21  {
22     Receive(frame);              //Receive ACK or NAK
23     if(corrupted(frame))
24          Sleep();
25     if (FrameType == NAK)
26        if (nakNo between Sf and Sn)
27        {
28         resend(nakNo);
29         StartTimer(nakNo);
30        }
31     if (FrameType == ACK)
32        if (ackNo between Sf and Sn)
33        {
34          while(sf < ackNo)
35          {
36           Purge(sf);
37           StopTimer(sf);
38           Sf = Sf + 1;
39          }
40        }
41  }
```

(continued)

Algorithm 11.9 *Sender-site Selective Repeat algorithm* (continued)

```
42
43    if(Event(TimeOut(t)))              //The timer expires
44    {
45     StartTimer(t);
46     SendFrame(t);
47    }
48  }
```

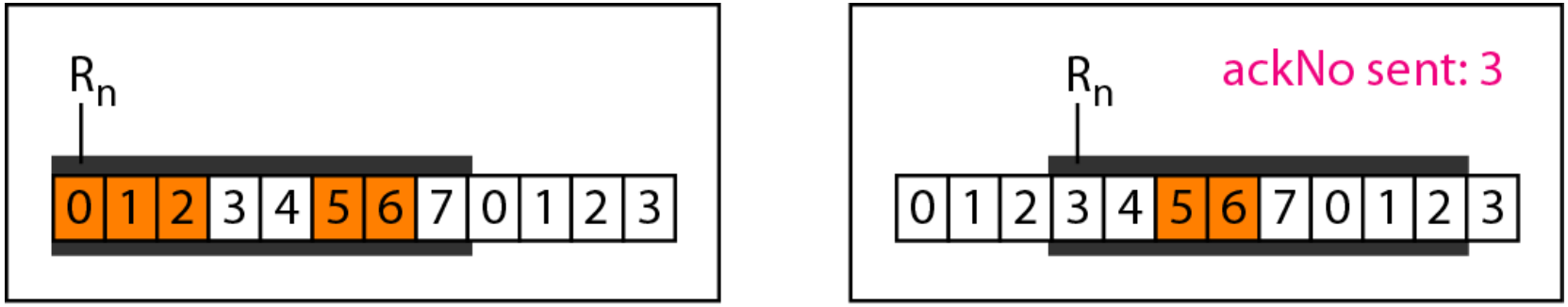
Algorithm 11.10 *Receiver-site Selective Repeat algorithm*

```
1  Rn = 0;
2  NakSent = false;
3  AckNeeded = false;
4  Repeat(for all slots)
5      Marked(slot) = false;
6
7  while (true)                                  //Repeat forever
8  {
9    WaitForEvent();
10
11   if(Event(ArrivalNotification))              /Data frame arrives
12   {
13      Receive(Frame);
14      if(corrupted(Frame))&& (NOT NakSent)
15      {
16       SendNAK(Rn);
17       NakSent = true;
18       Sleep();
19      }
20      if(seqNo <> Rn)&& (NOT NakSent)
21      {
22       SendNAK(Rn);
```

Algorithm 11.10 *Receiver-site Selective Repeat algorithm*

```
23          NakSent = true;
24          if ((seqNo in window)&&(!Marked(seqNo))
25          {
26           StoreFrame(seqNo)
27           Marked(seqNo)= true;
28           while(Marked(Rn))
29           {
30            DeliverData(Rn);
31            Purge(Rn);
32            Rn = Rn + 1;
33            AckNeeded = true;
34           }
35            if(AckNeeded);
36            {
37            SendAck(Rn);
38            AckNeeded = false;
39            NakSent = false;
40            }
41          }
42        }
43    }
44  }
```

Figure 11.22 *Delivery of data in Selective Repeat ARQ*



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.

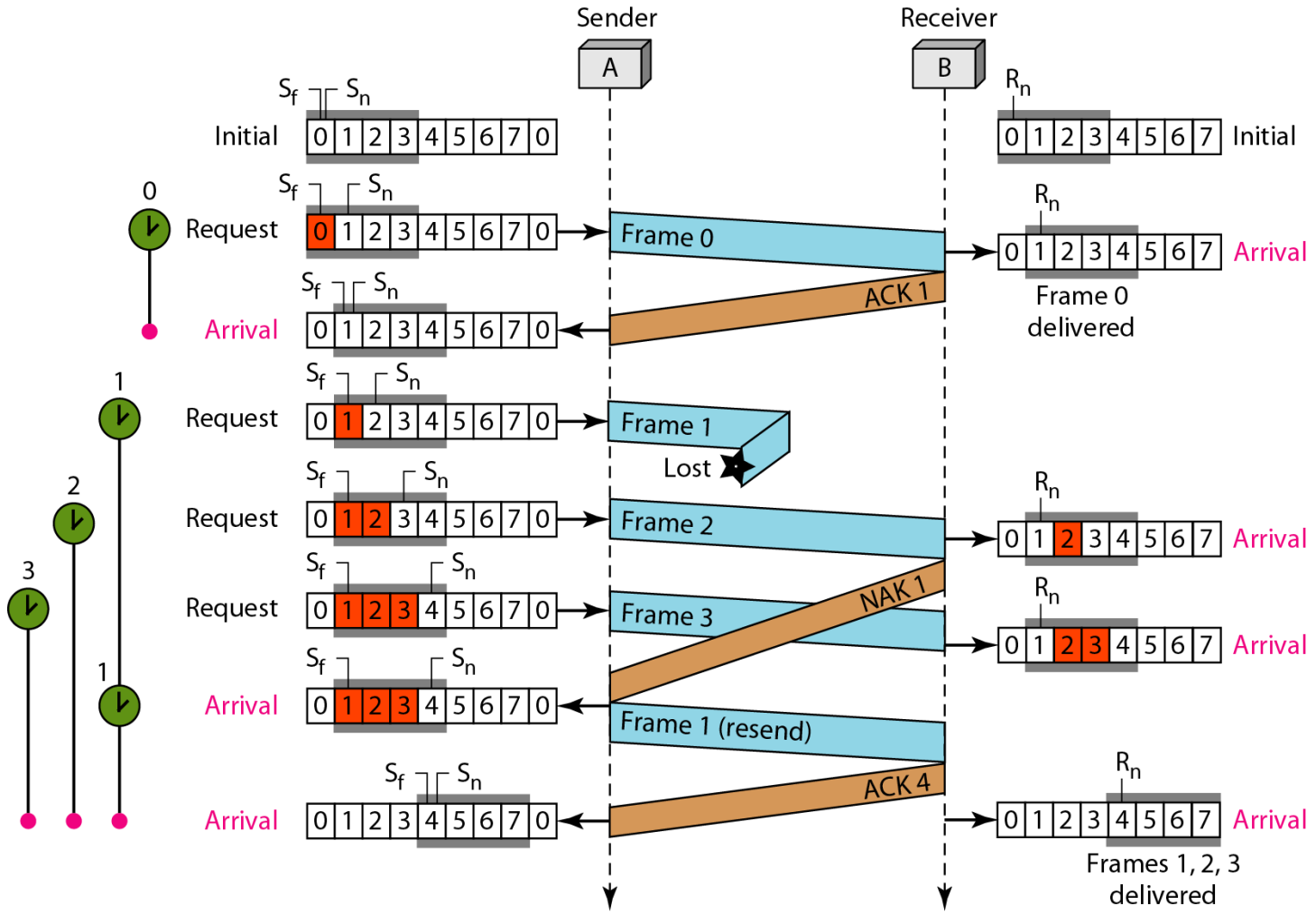
A NAK is sent once for each window position and defines the first slot in the window.

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*



Piggybacking

- In real life the data frames flow in both directions; so the control information also needs to travel in both directions
- 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 to A (also other way round from B to A)
- ***Go-Back-N ARQ using piggybacking***
- Each node has 2 windows: 1 send and 1 receive window
- Both are involved in 3 types of events: request, arrival and timeout
- Request event uses only the send window at each site; arrival event needs to use both windows (since when a frame arrives, node needs to handle control info as well as the frame itself)

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