Data Communications and Networking Fourth Edition

Forouzan

# Chapter 11

# Data Link Control

# 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.**

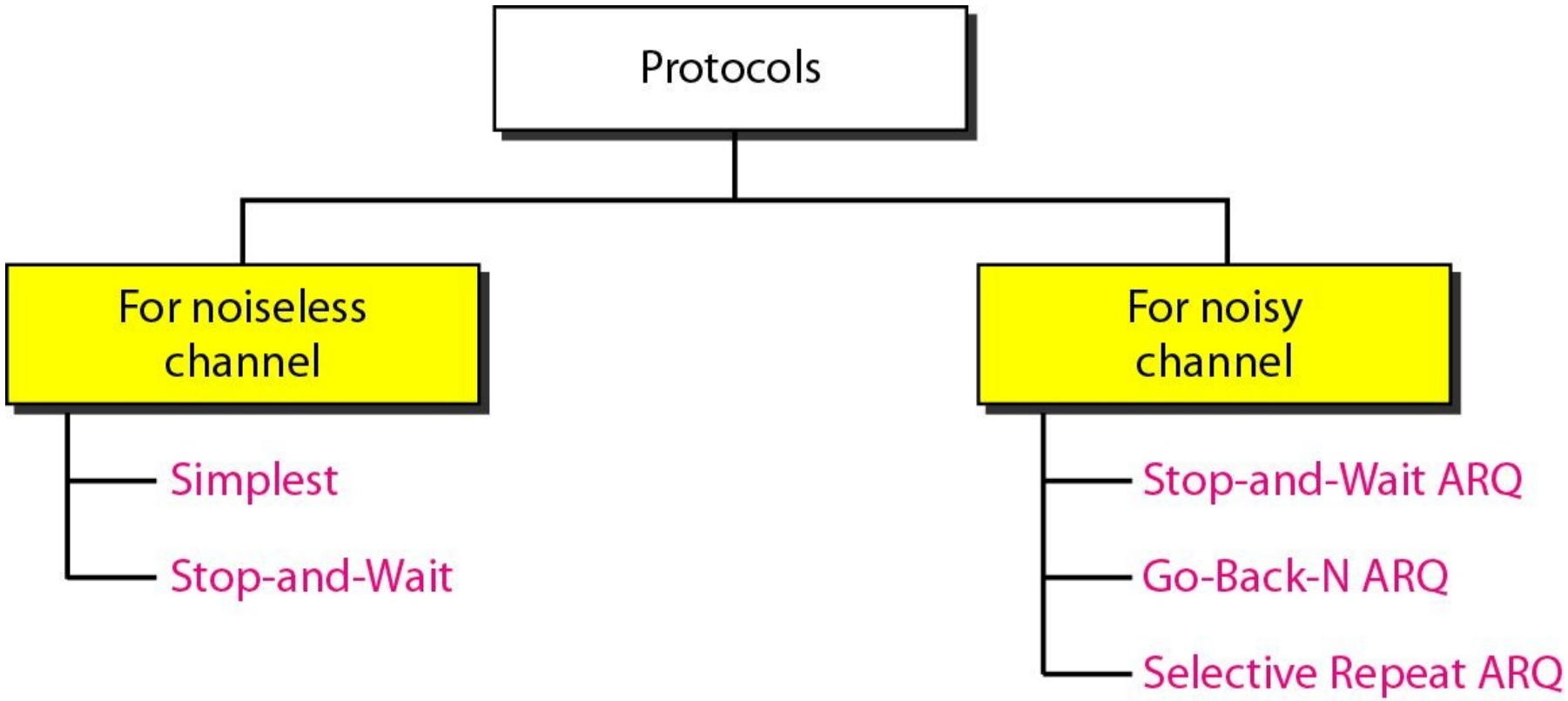
*Note*

**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

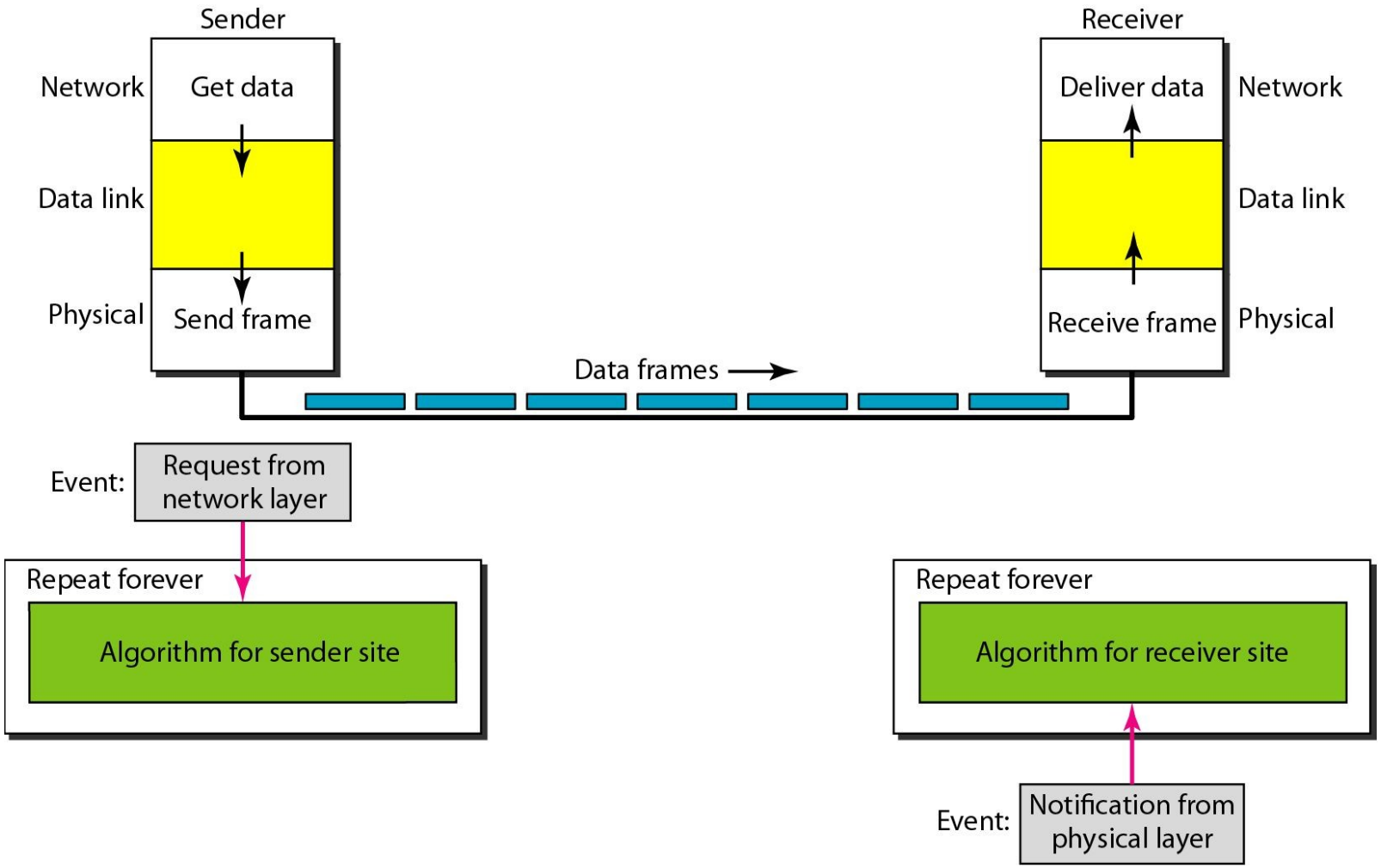
***Let us first assume we have an ideal channel in which no frames are lost, duplicated, or corrupted. We introduce two protocols for this type of channel.***

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

**Simplest Protocol**

**Stop-and-Wait Protocol**

**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 }
```

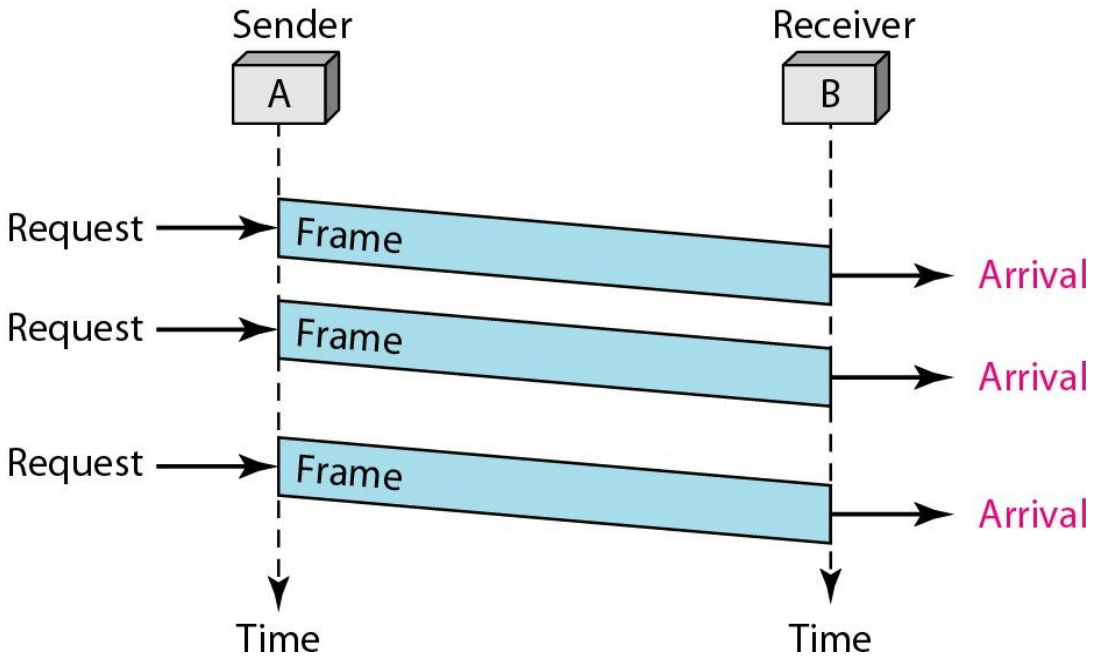
## Algorithm 11.2 *Receiver-site algorithm for the simplest*

```
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 }
```

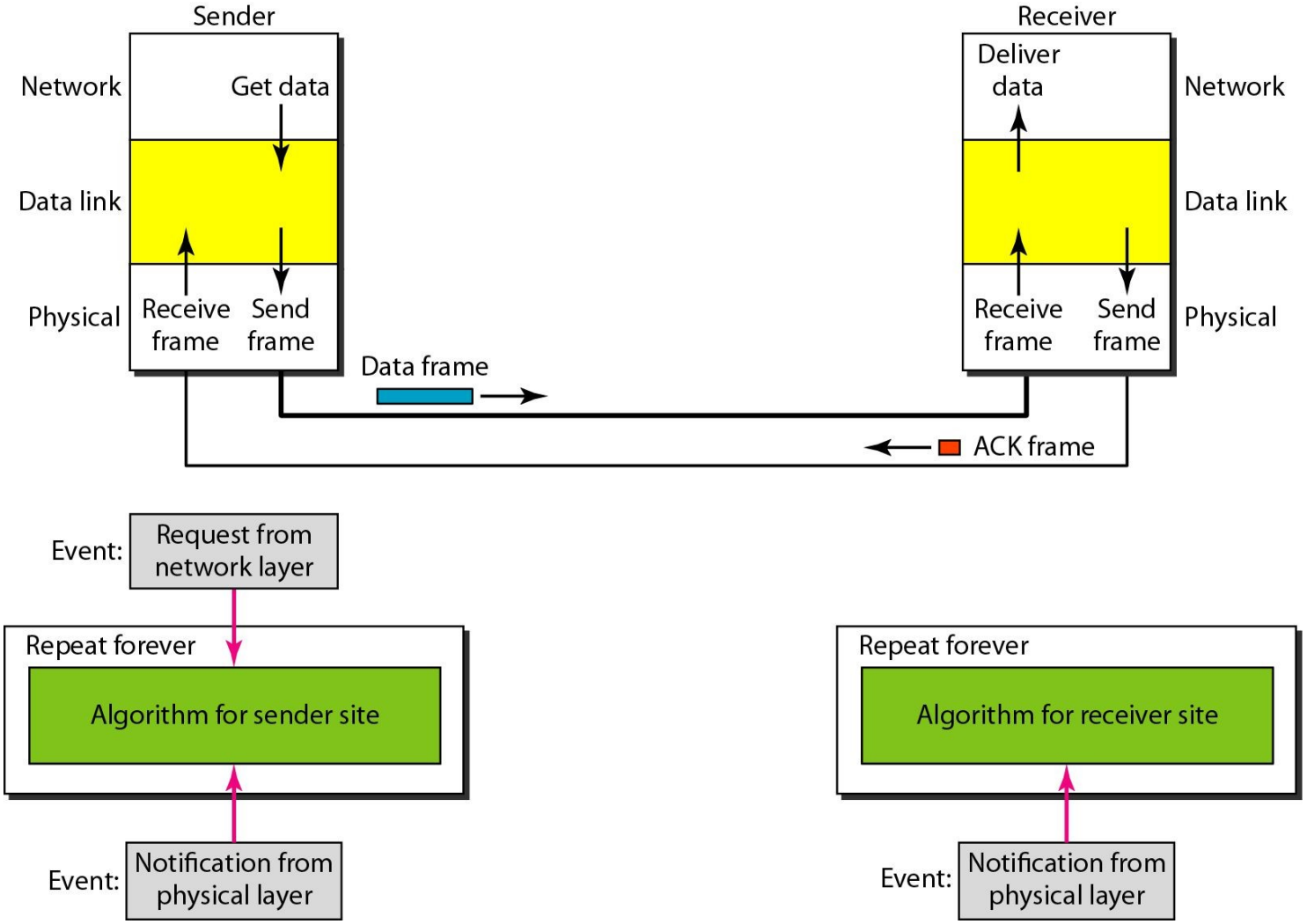
# Example 11.1

*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*



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



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

```
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 }
```

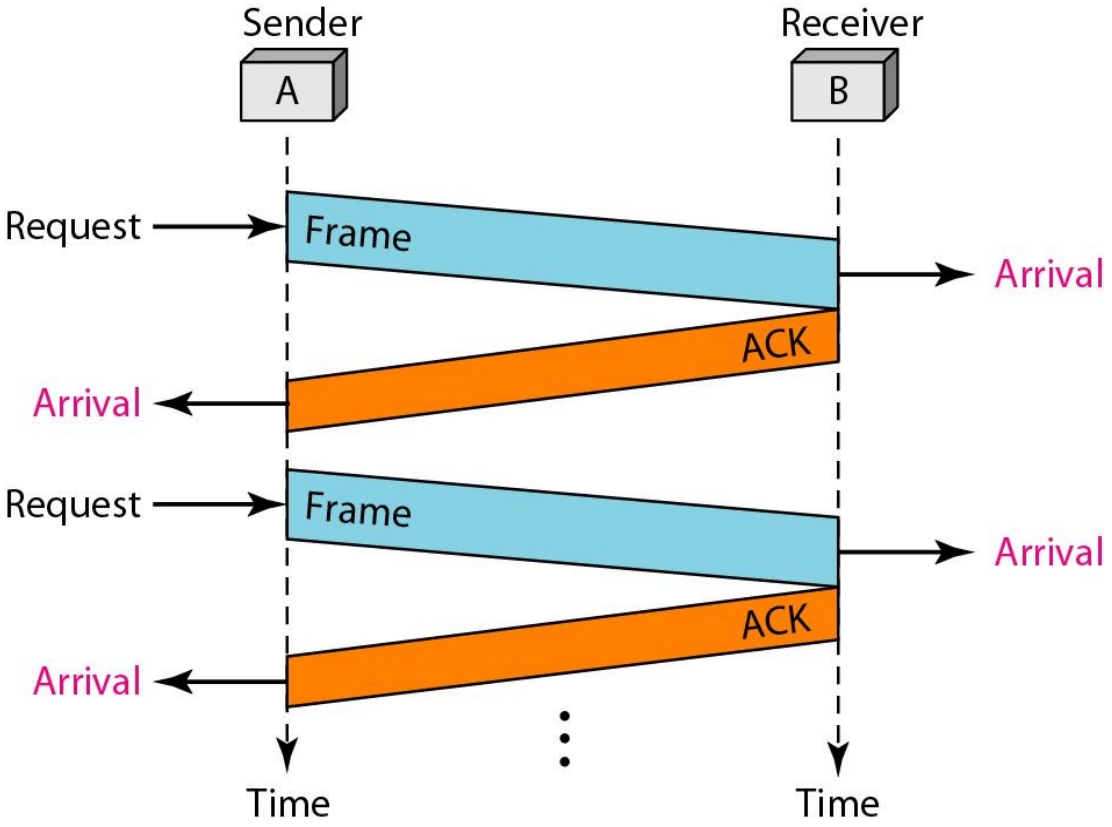
## Algorithm 11.4 *Receiver-site algorithm for Stop-and-Wait*

```
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**

*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.**

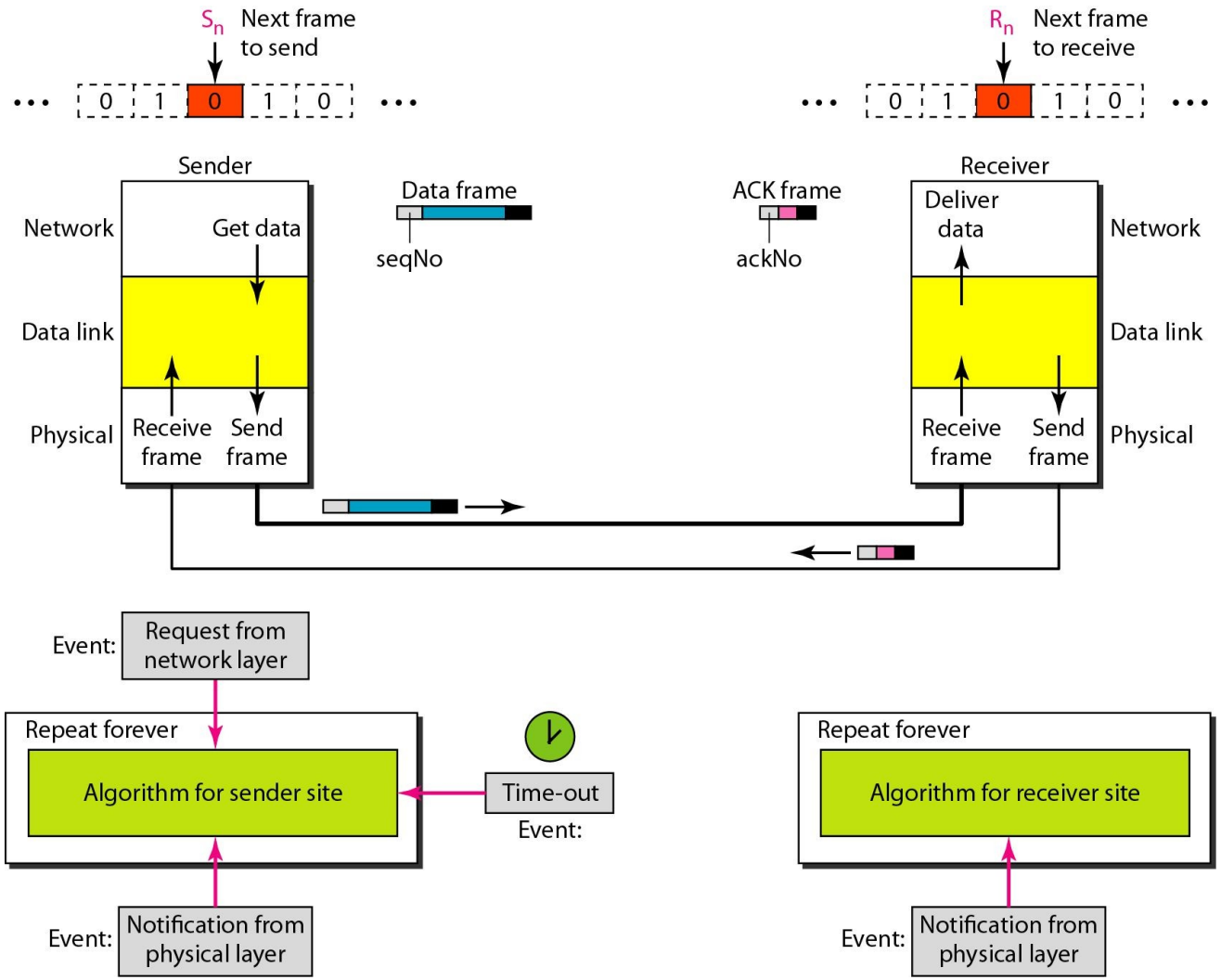
*Note*

**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.**

# 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  }
```

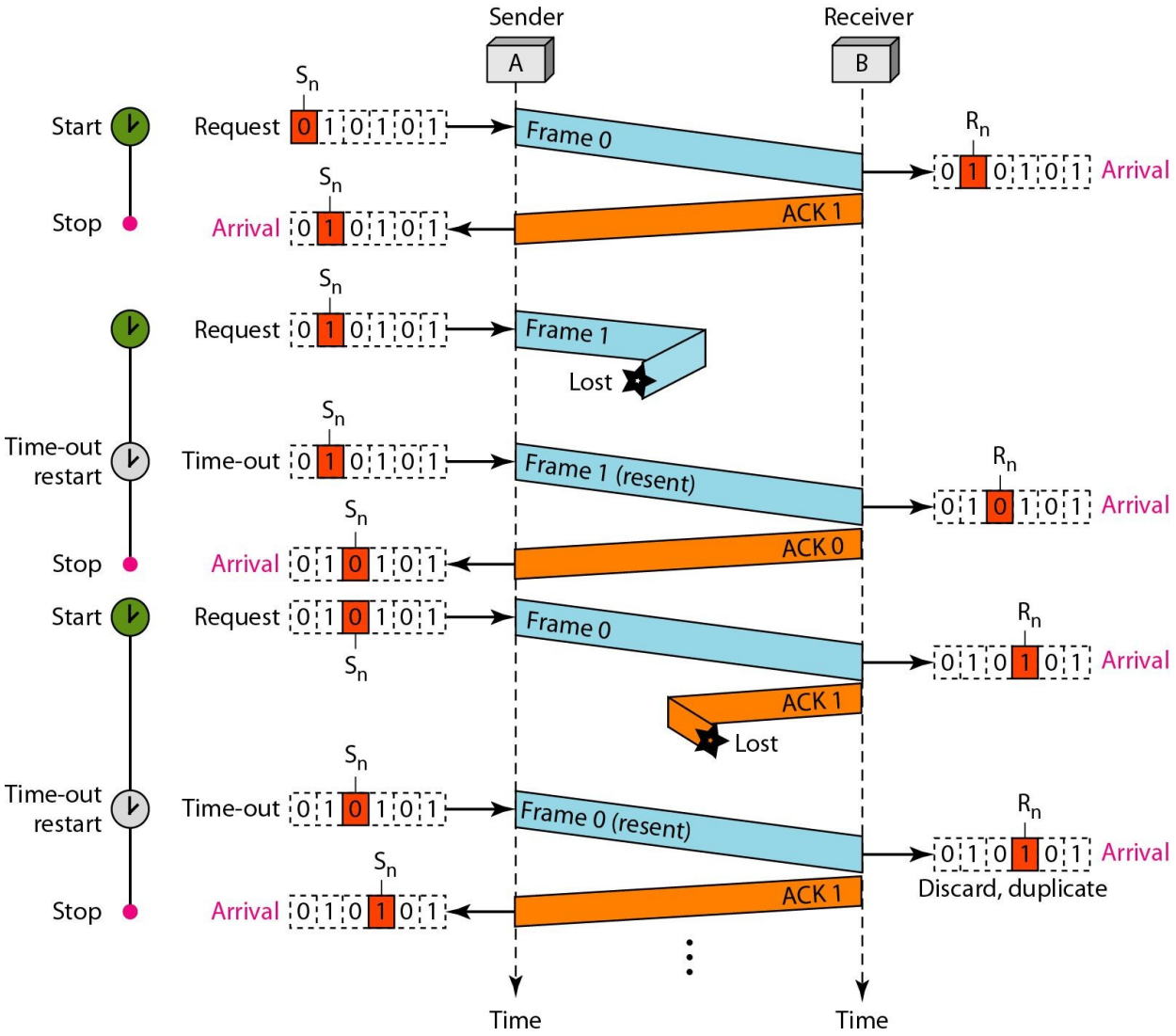
## Algorithm 11.6 *Receiver-site algorithm for Stop-and-Wait ARQ*

```
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.*

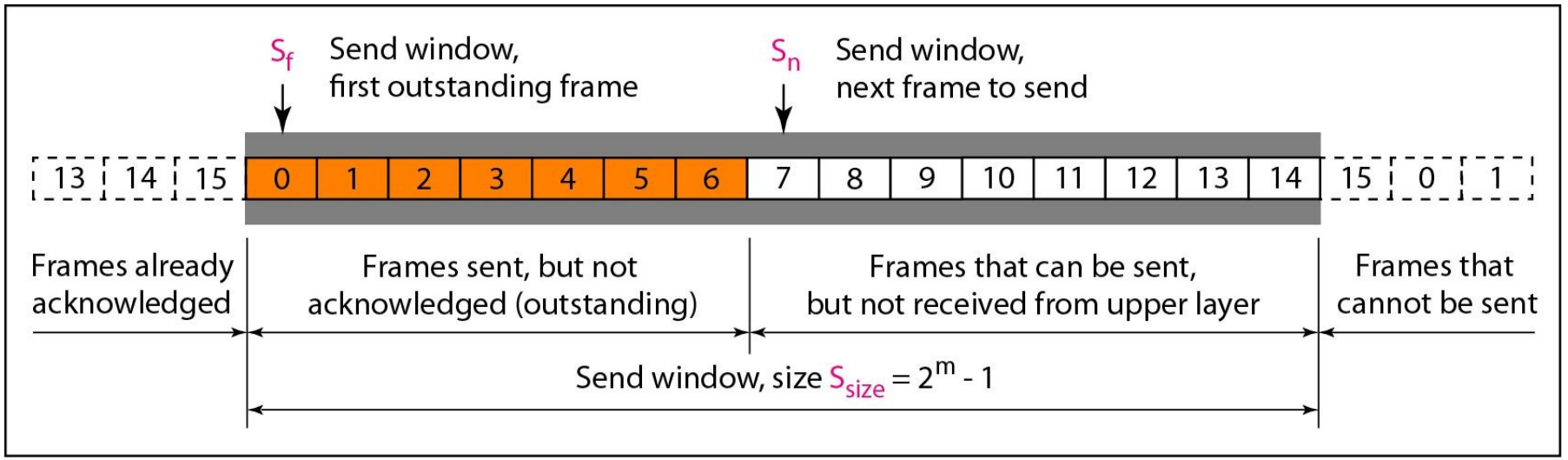
## Figure 11.11 *Flow diagram for Example 11.3*



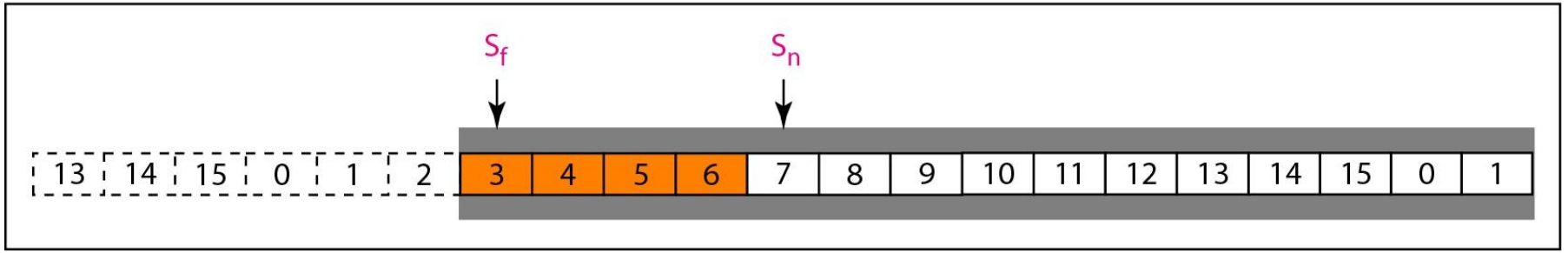
*Note*

**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

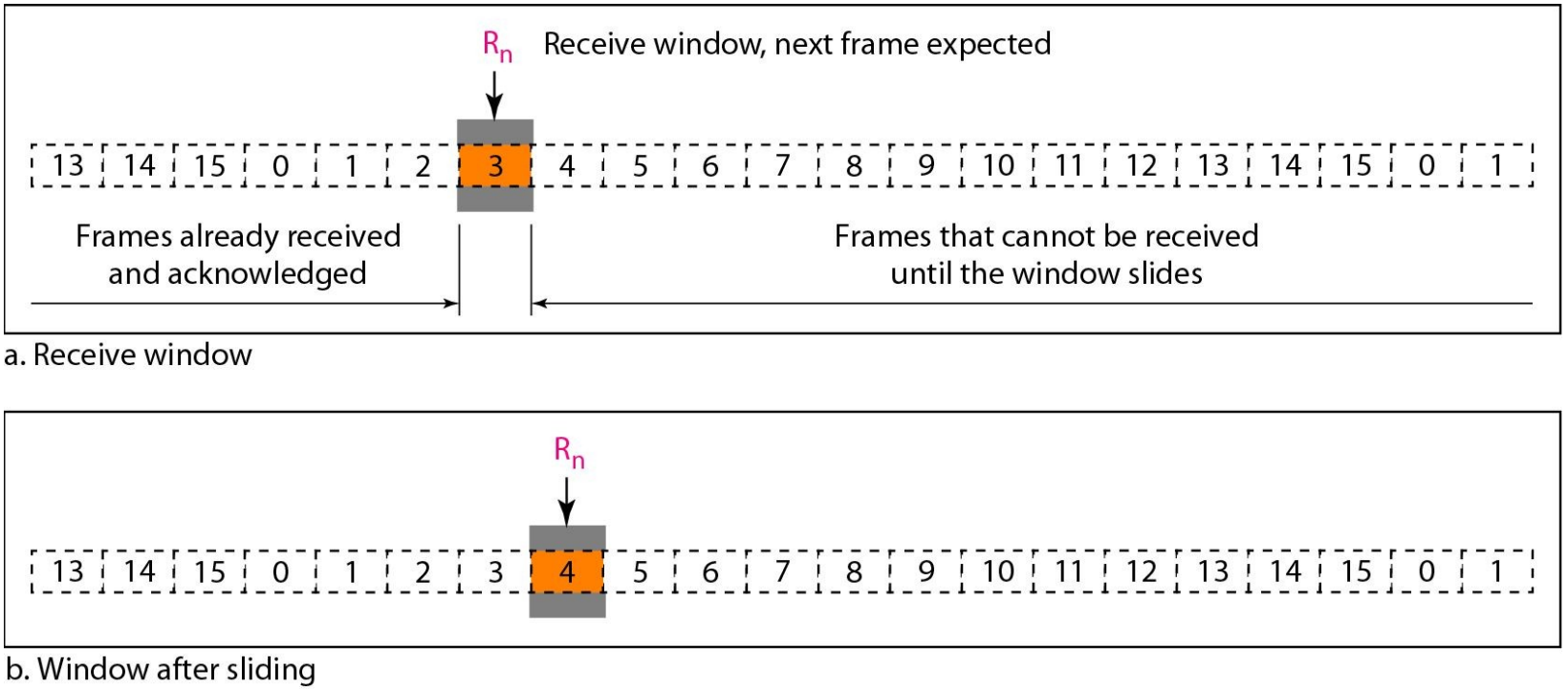
*Note*

**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.**

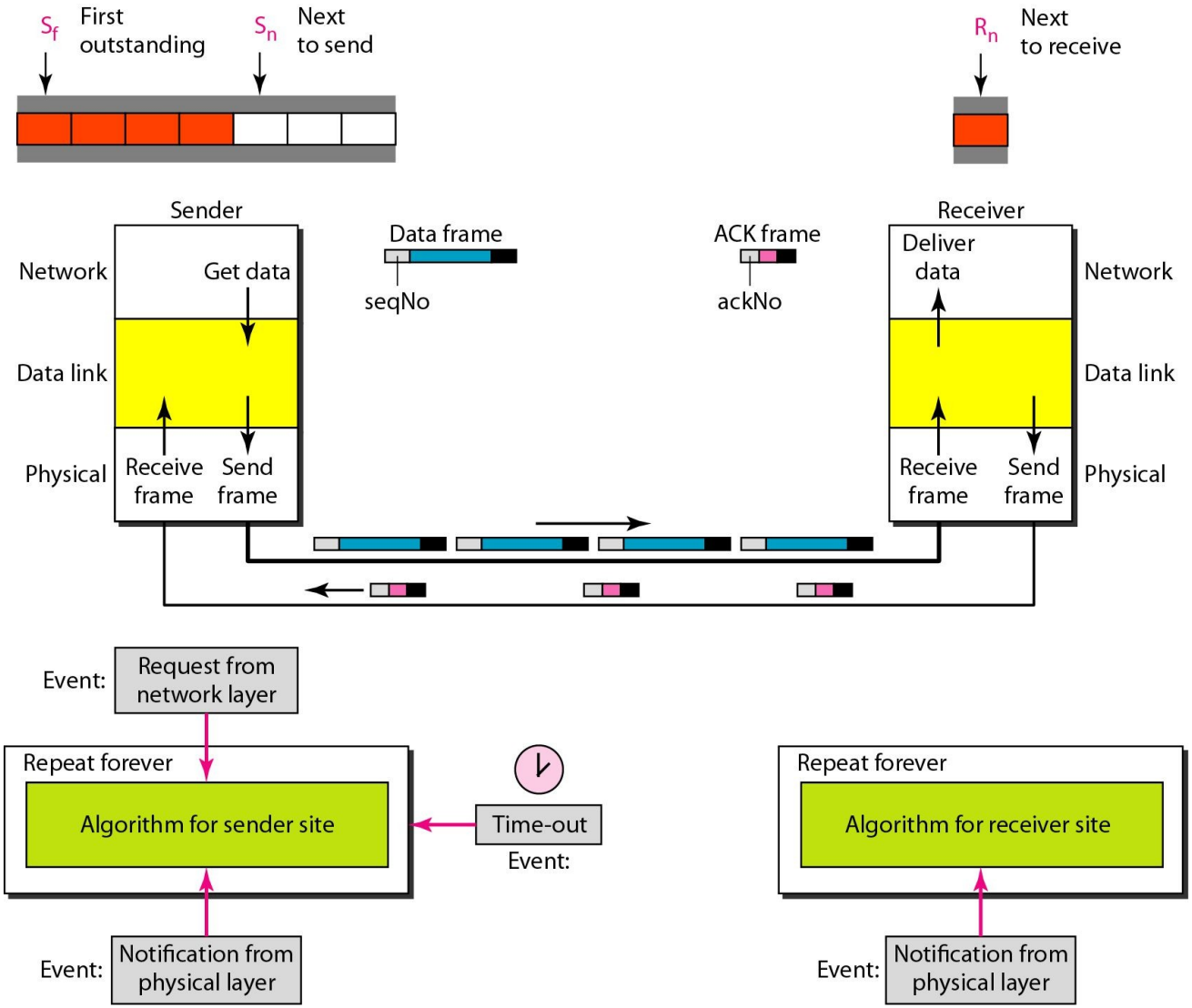
# Figure 11.13 *Receive window for Go-Back-N ARQ*



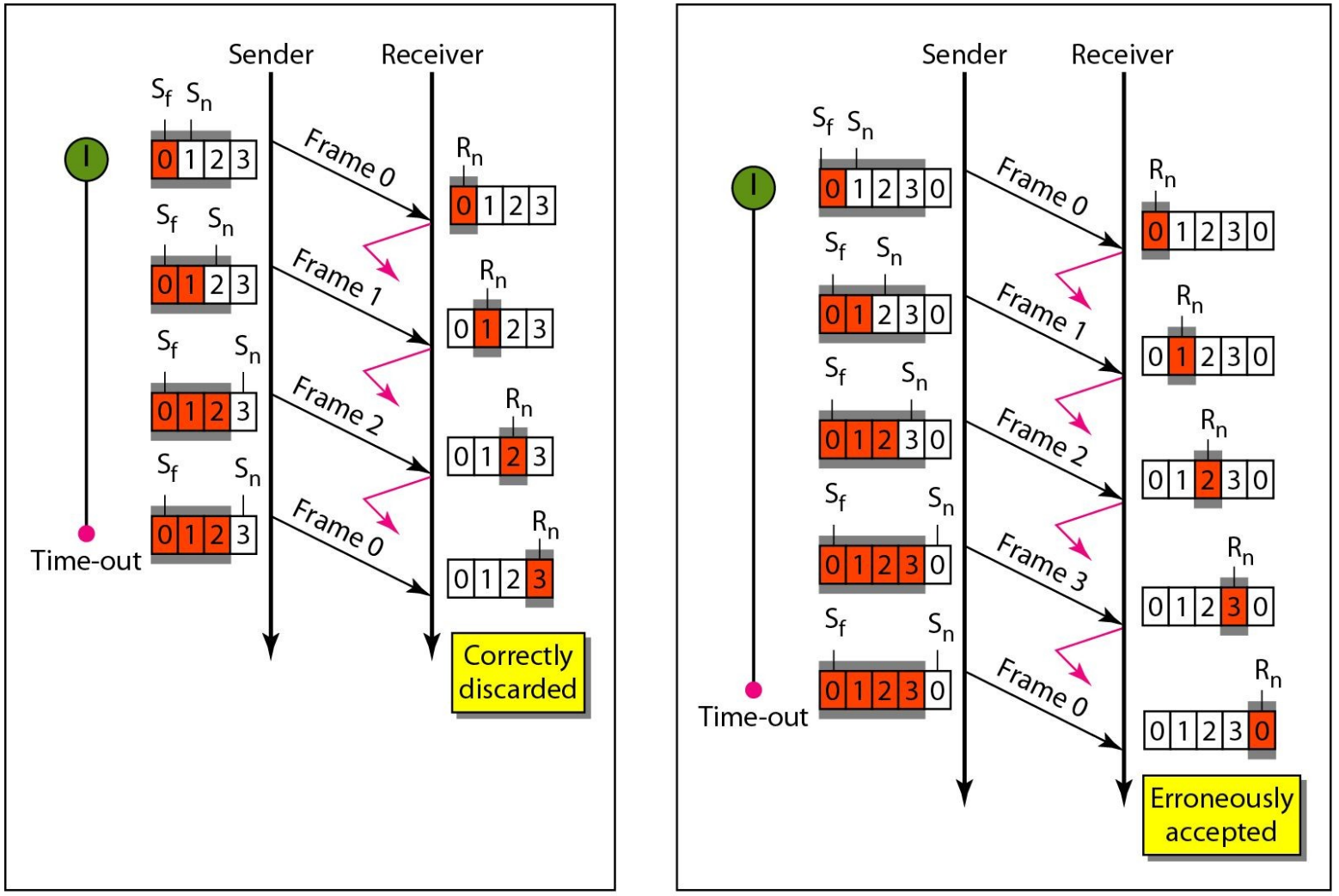
*Note*

**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.**

## Algorithm 11.7 *Go-Back-N sender*

```
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*

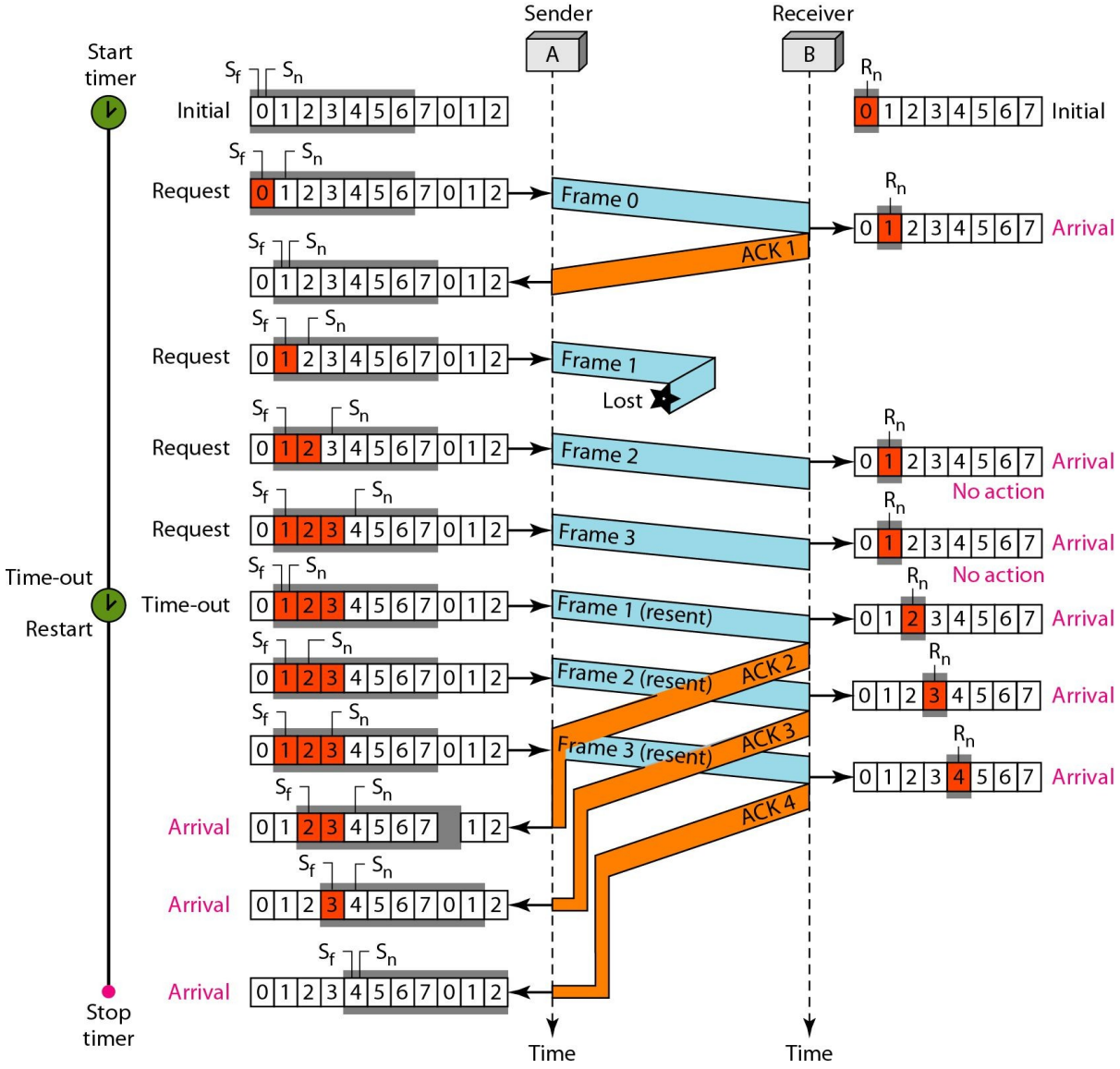
## Algorithm 11.7 *Go-Back-N sender algorithm* (continued)

```
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 }
```

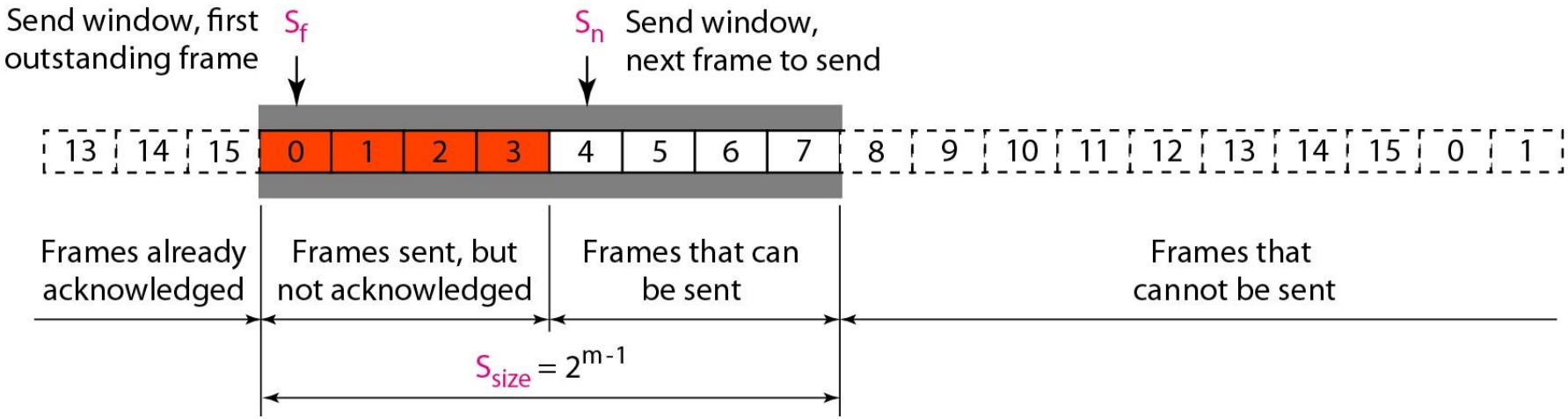
## Algorithm 11.8 *Go-Back-N receiver algorithm*

```
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 }
```

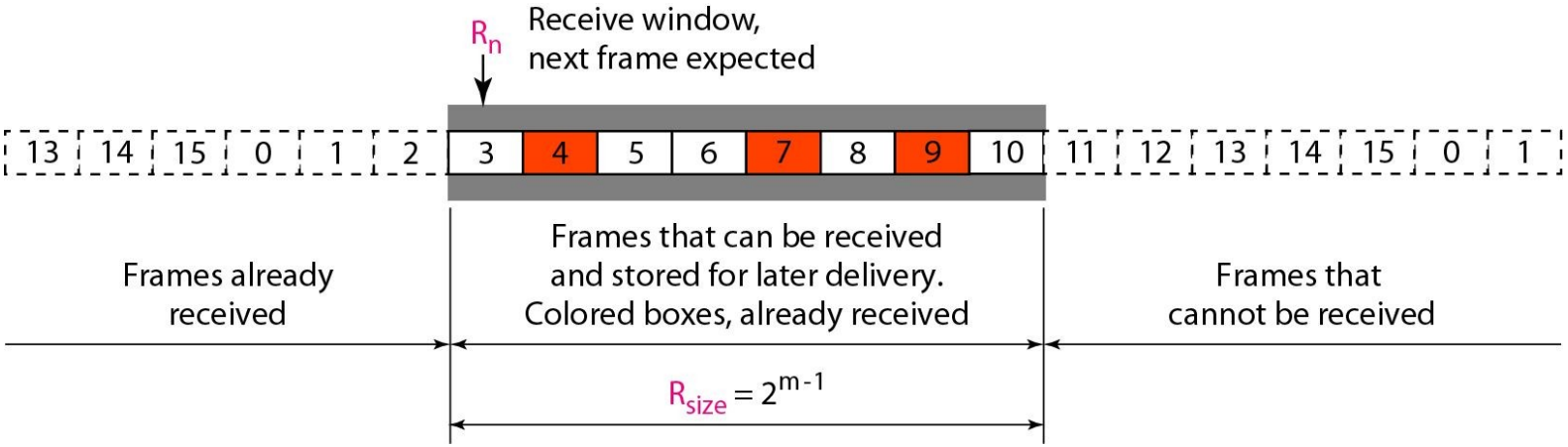
# Figure 11.17 *Flow diagram for Example 11.7*



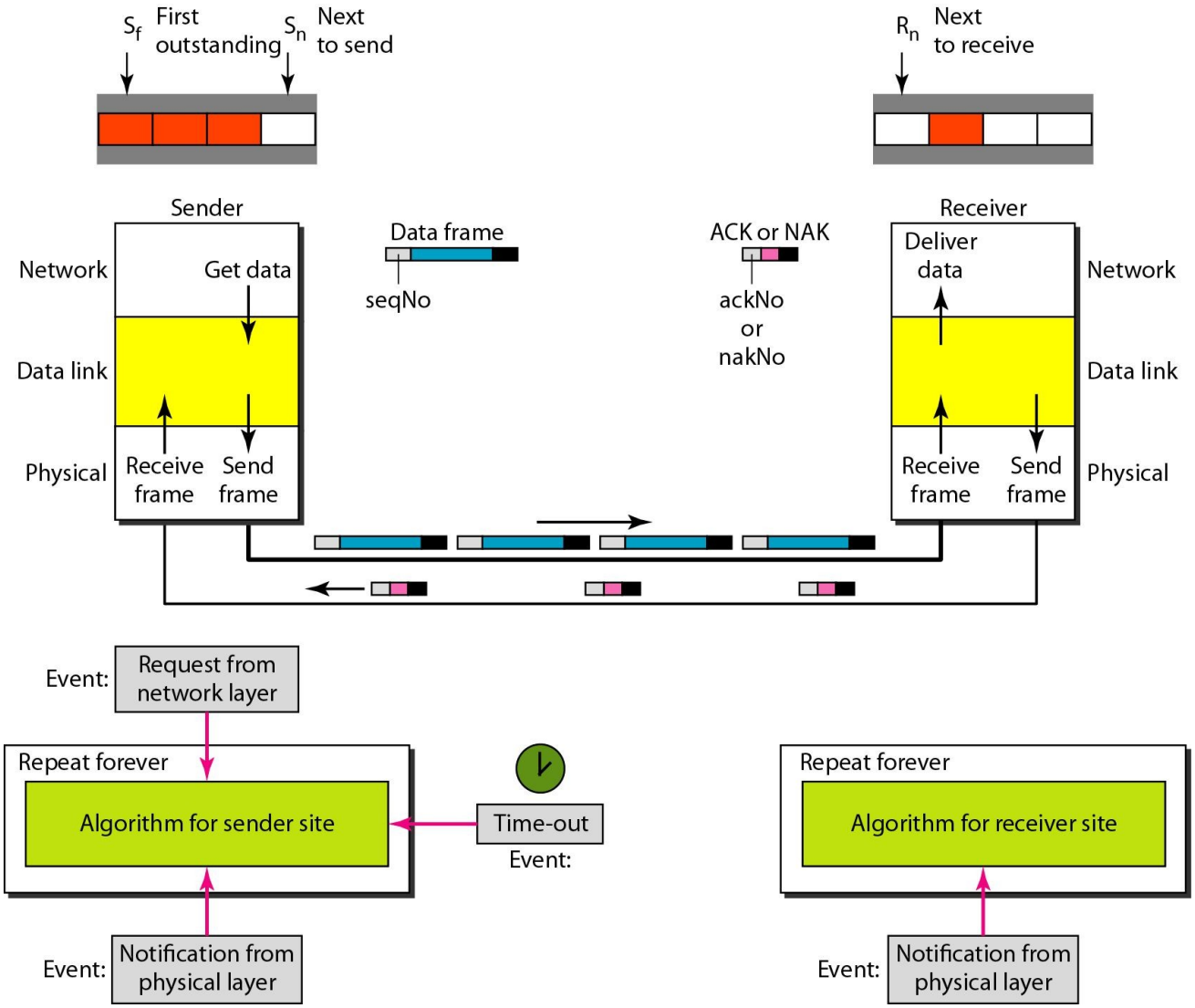
## Figure 11.18 *Send window for Selective Repeat ARQ*



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



# Figure 11.20 *Design of Selective Repeat ARQ*



## Algorithm 11.9 *Sender-site Selective Repeat*

```
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)*



*(continued)*

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

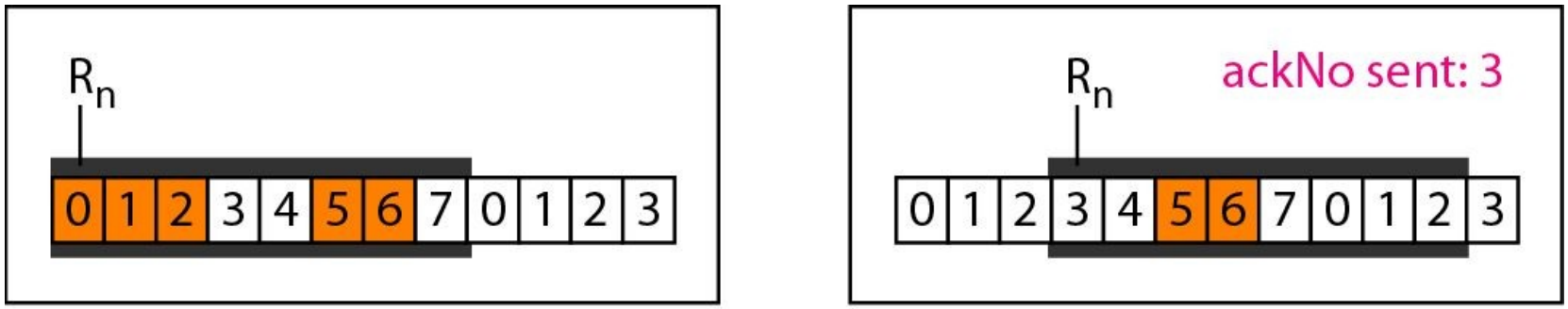
```
42
43    if(Event(TimeOut(t)))                    //The timer expires
44    {
45     StartTimer(t);
46     SendFrame(t);
47    }
48  }
```

## Algorithm 11.10 *Receiver-site Selective Repeat algorithm*

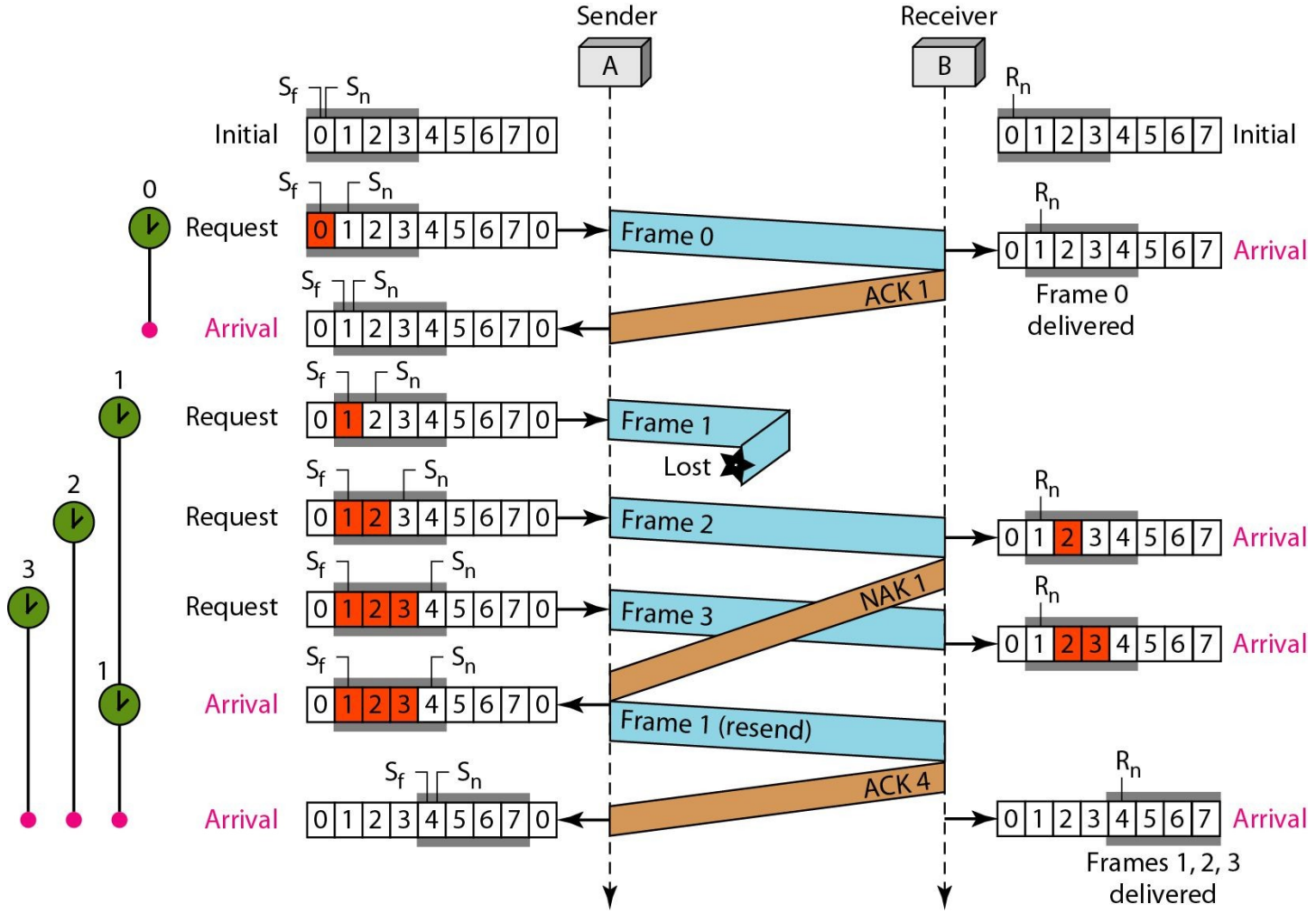
```
1  Rn = 0;
2  NakSent = false;
3  AckNeeded = false;
4  Repeat(for all slots)
5      Marked(slot) = false;
6
7  while (true)                                  //Repeat forever
8  {
9    WaitForEvent();
10
11   if(Event(ArrivalNotification))              /Data frame arrives
12   {
13       Receive(Frame);
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*

## Figure 11.22 *Delivery of data in Selective Repeat ARQ*



## Figure 11.23 *Flow diagram for Example 11.8*



# 11-4 Point to Point Protocol (PPP)

- One of the most common protocols for point-to-point access is the Point-to-Point Protocol (PPP).

- Today, millions of Internet users who need to connect their home computers to the server of an Internet service provider use PPP.

- To control and manage the transfer of data, there is a need for a point-to-point protocol at the data-link layer. PPP is by far the most common.

# Services

- The designers of PPP have included several services to make it suitable for a point-to-point protocol, but have ignored some traditional services to make it simple.

# Services provided by Point-point protocol

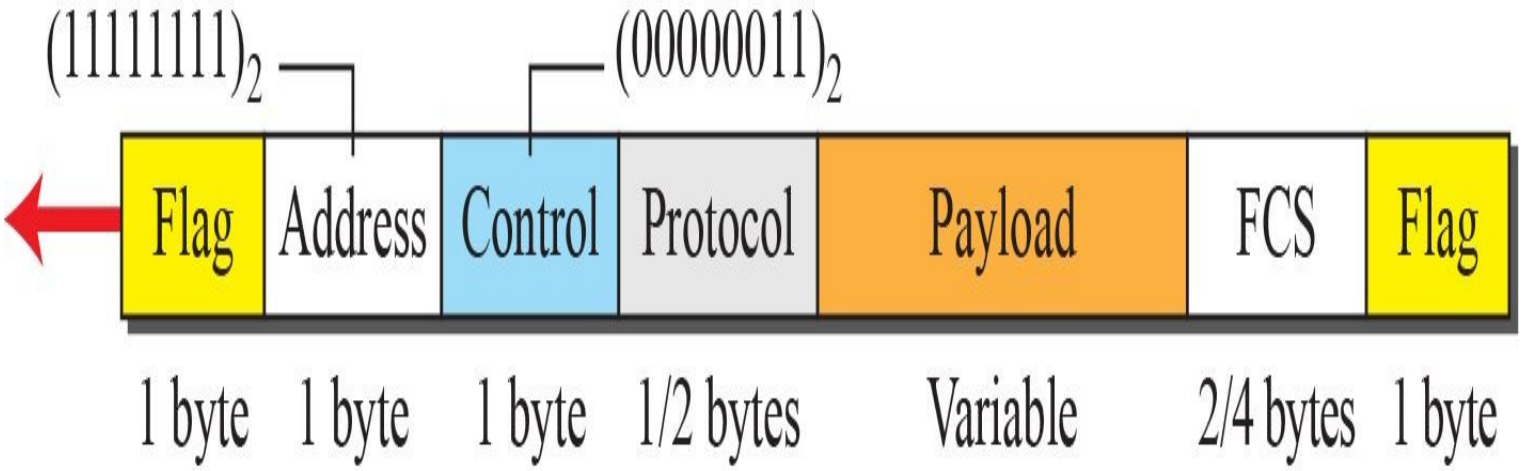
## Defines

1. format of the frame to be exchanged between the devices.
2. How two devices can negotiate the establishment of the link and the exchange of data.
3. How N/W layer data are encapsulated in the data link frame.
4. How two devices can authenticate each other.
5. Provides Multiple N/W services supporting a variety of N/W Layer protocols.
6. Connection over multiple links is provided by new version of PPP.
7. N/W address configuration. Useful when a home user needs temporary N/W address to connect to Internet

# Services not provided by ppp

- PPP does not provide flow control. Sender can send frames one after another w/o concerning about overwhelming receivers buffer.
- Simple mechanism for error control. A CRC field is used to detect errors. If frame is corrupted , frame is silently discarded. Upper Layer protocol need to take care of this.
- Does not provide addressing mechanism to handle frames in a multipoint configuration.

# Framing




***PPP frame format***

# Framing

- PPP uses a character-oriented (or byte-oriented) frame. Figure below shows the format of a PPP frame. PPP is a byte oriented protocol

Flag:□ A PPP frame starts and ends with a one byte flag with the bit pattern 0111110. The flag is treated as a byte.

Address:□ The address field is constant value and set to 11111111(broadcast address)

Control□ The field is set to a constant value 11000000.

PPP doe not provide FC and limited EC. Two parties can

# Framing

Protocol□ defines what is being carried in the data field either user data or other information.

Default size is 2 bytes, two parties can negotiate to use only 1 byte.

Payload field□ carries use data or other info.

Data field is a sequence of bytes with the default of maximum of 1500 bytes, But can be changed during negotiation.

The data field is byte stuffed if the flag byte pattern appears in the field. Because there is no field defining the size of the data field, padding is needed if the size is less than the

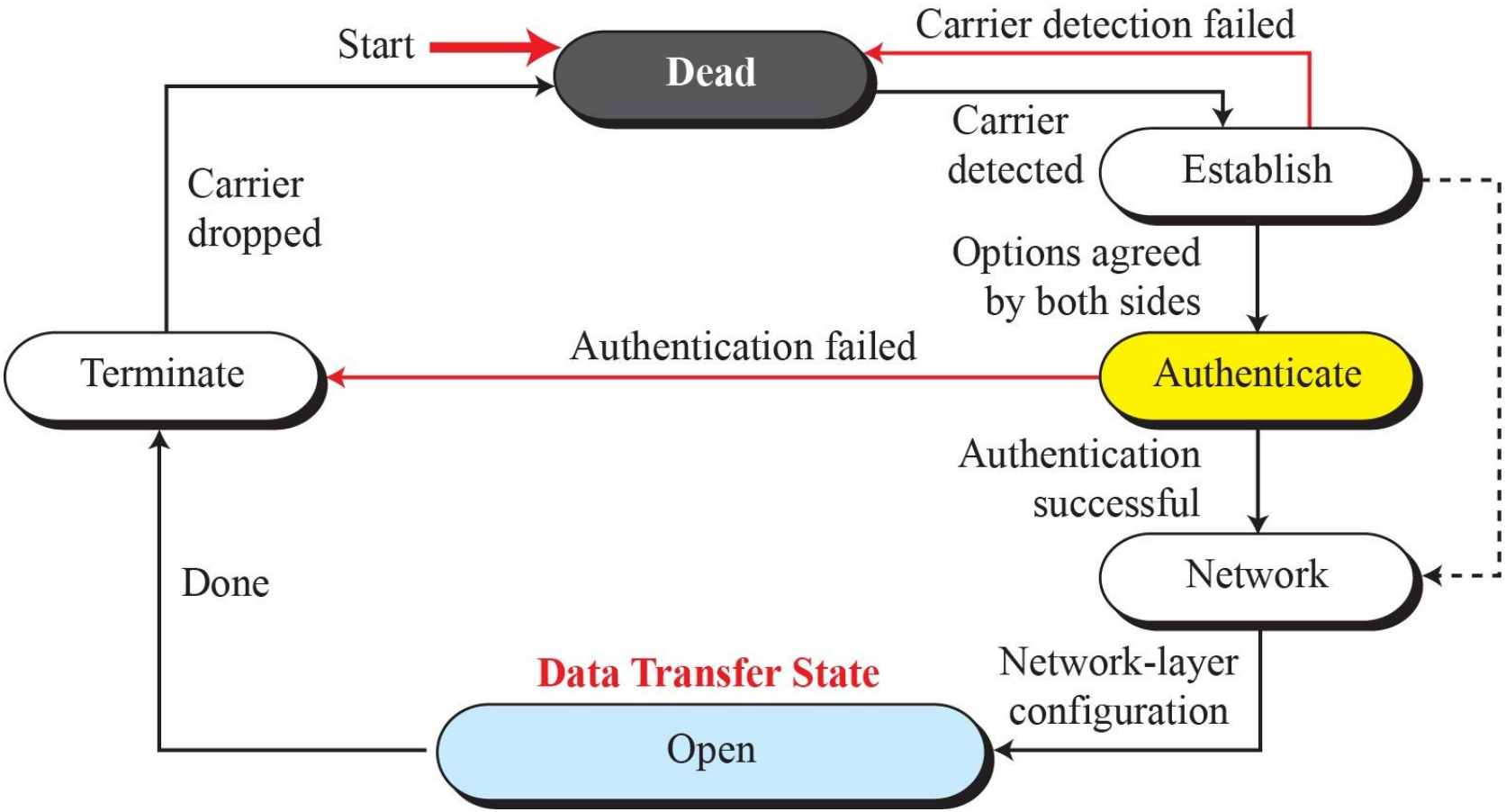
# Byte stuffing

- In PPP flag is byte. The flag byte need to be escaped whenever it appears in the data section of the frame.
- The escape byte is 01111101 , which means that every time the flag like pattern appears in the data, this extra byte is stuffed to tell the receiver the next byte is not a flag.
- PPP is a byte-oriented protocol using byte stuffing with the escape byte 01111101.

# Transition Phases

- A PPP connection goes through phases which can be shown in a transition phase diagram.

# Transition phases



# Transition phases

Dead: In this phase link is not used. There is no active carrier and line is quiet.

Establish: When one of the node starts communication, connection goes into establish phase. Options are negotiated between two parties.

If negotiation is successful, system goes to authentication phase or directly to networking phase. Link control packet are used in this phase.

Authenticate: Two nodes can negotiate and skip this phase as it is optional. If not to skip, authentication packets are used.

# Transition phases

Network: Negotiation for the network layer protocols takes place. PPP specifies that two nodes establish a N/W agreement before data at the N/w can be exchanged.

PPP supports multi protocols at N/W Layer.

If a node is running multi protocol simultaneously at the N/W Layer, receiving node needs to know which protocol will receive the data.

# Transition phases

Open: Data transfer takes place.

When a connection reaches this phase, exchange of data packets takes place

Connections remains in this phase until one of the endpoints wants to terminate the connection

Terminate: Connection is terminated. Several packets are exchanged b/w the two ends for house cleaning and closing of the link