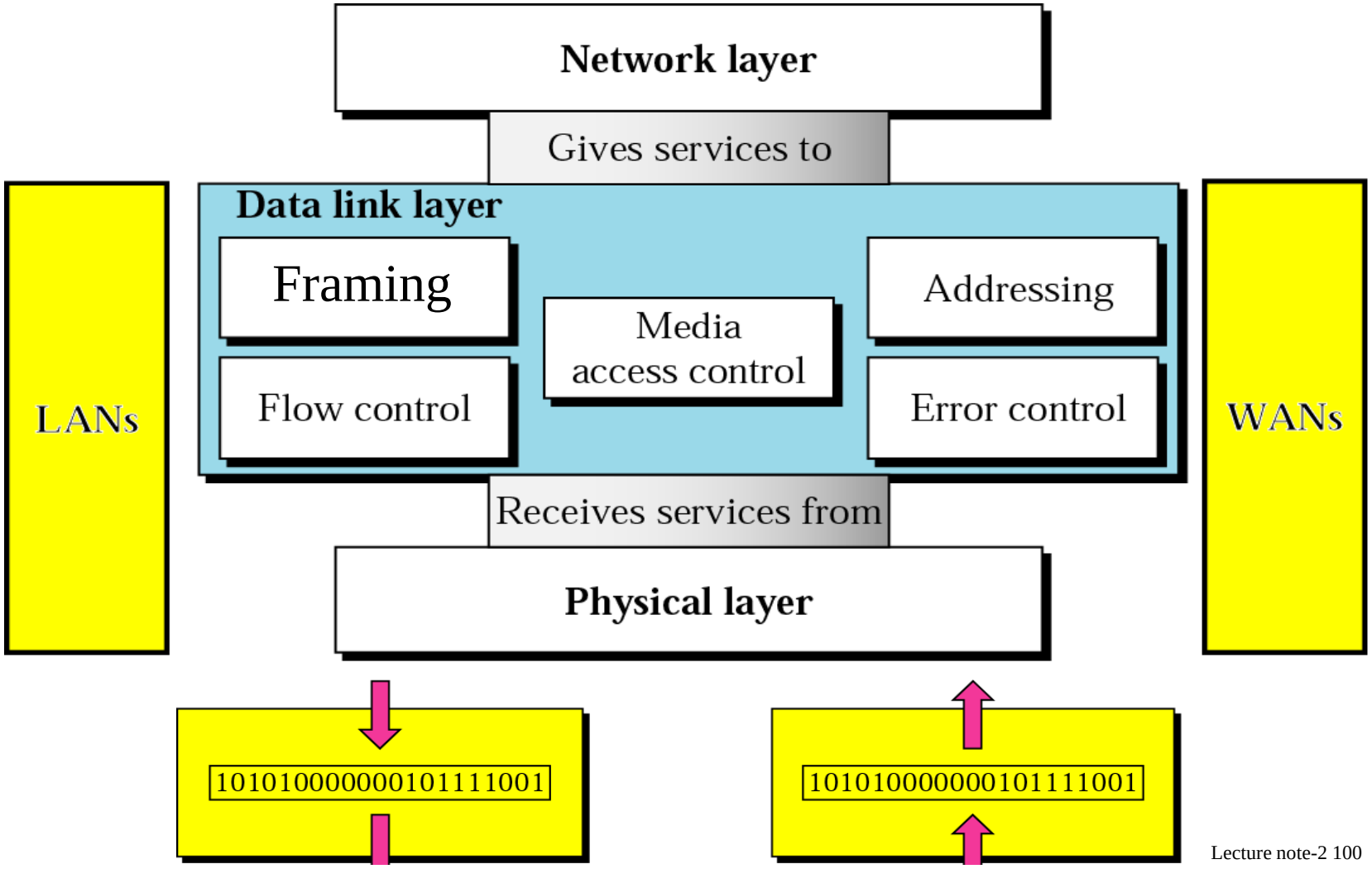
Data Link Layer

Chapter 10 & 11 of ***Data Communications and Networking, 4th Edition, Behrouz A. Forouzan (ISBN: 0-07-251584-8)***

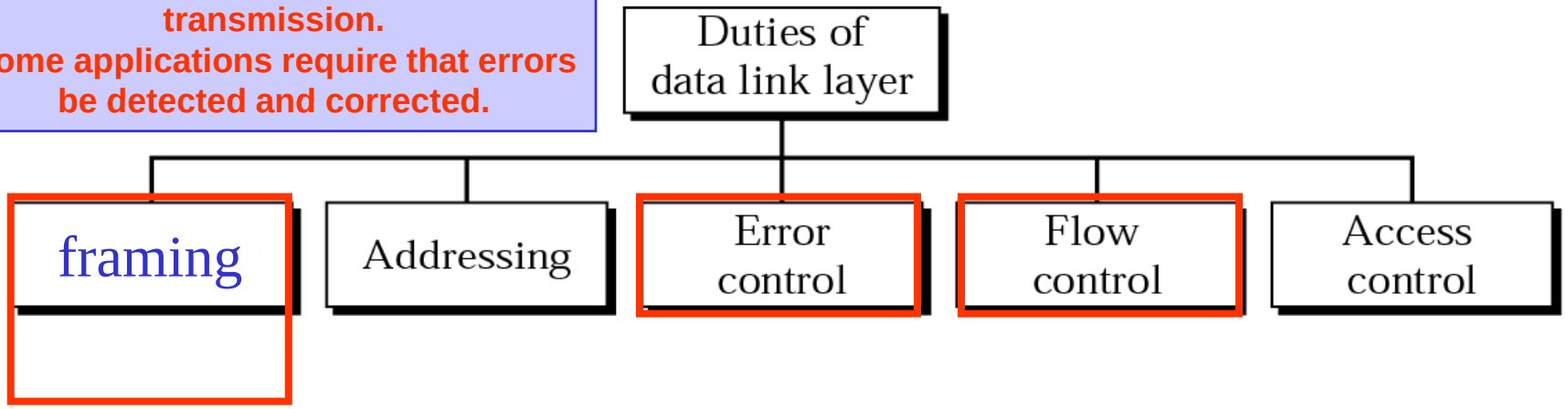
Jin Seek Choi
jinseek@hanyang.ac.kr

Review Position of the data-link layer



Review of Data link layer duties

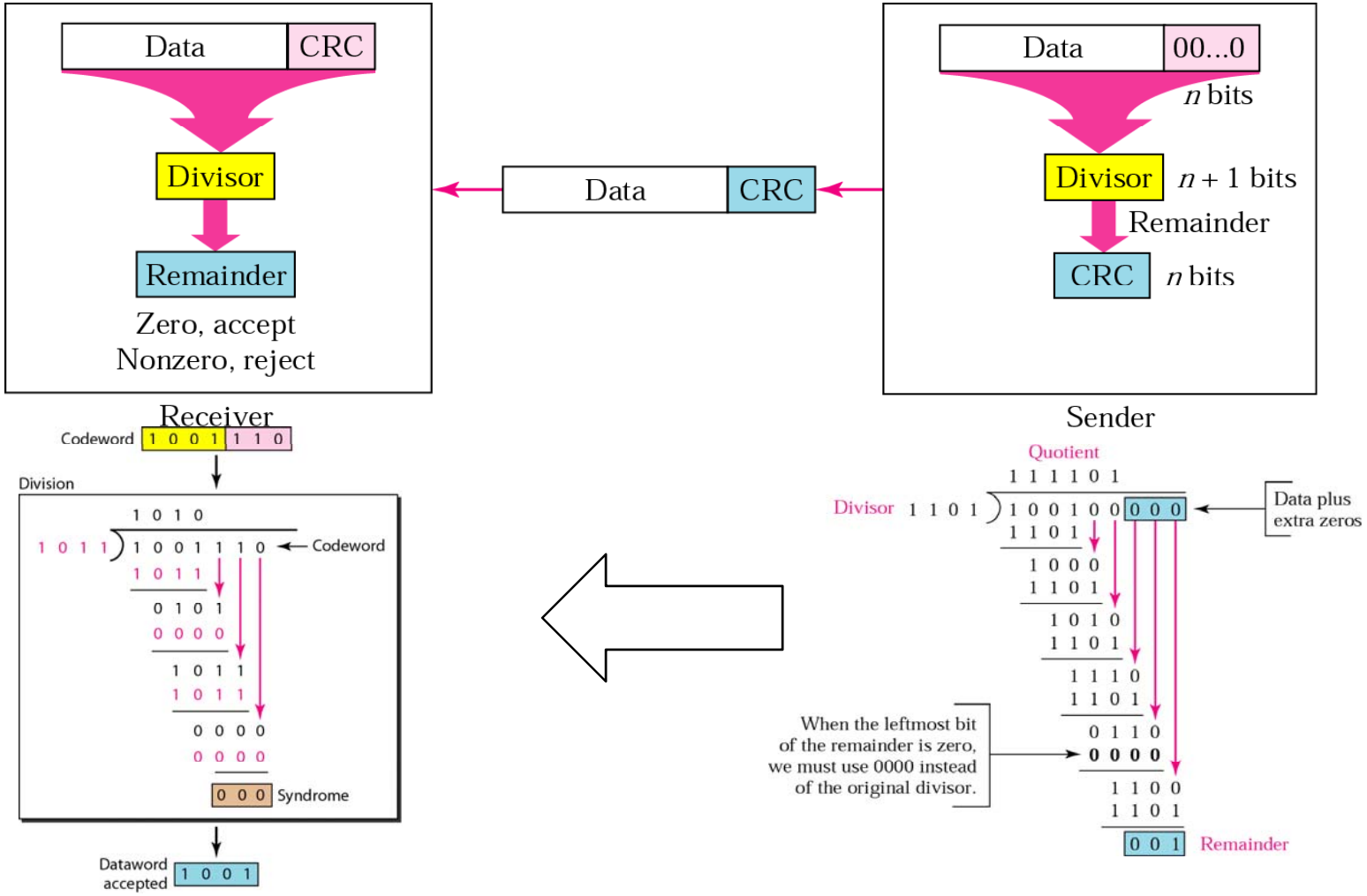
Data can be corrupted during transmission.
Some applications require that errors be detected and corrected.



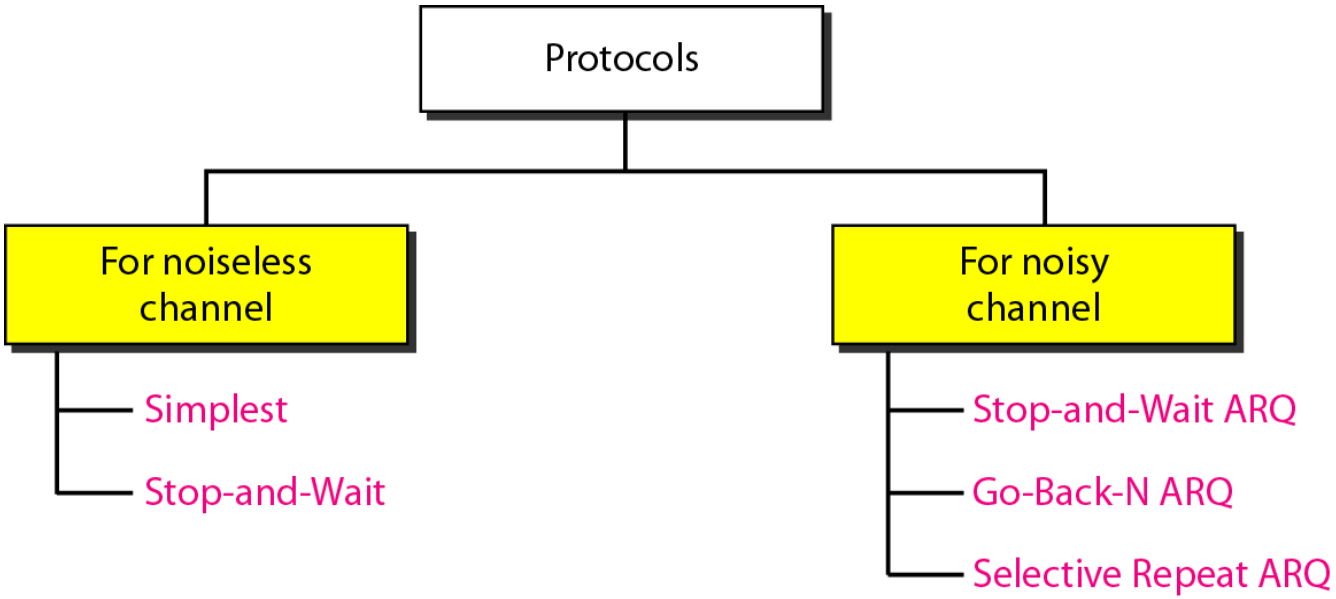
- **Data link protocols have three functions:**
  - Error Control**: Detecting and correcting transmission errors. (Error & flow)**
  - Media Access Control**: Controlling when computers transmit. Who should send now(Access control)**
  - Message Delineation**: Identifying the beginning and end of a message. (Framing & Addressing)**

Review of Cyclic Redundancy Check (CRC)

- **Most powerful of redundancy checking techniques based on modular 2 division**
- **CRC-16 (99.969% effective) and CRC-32 (99.99%) are in common use today**



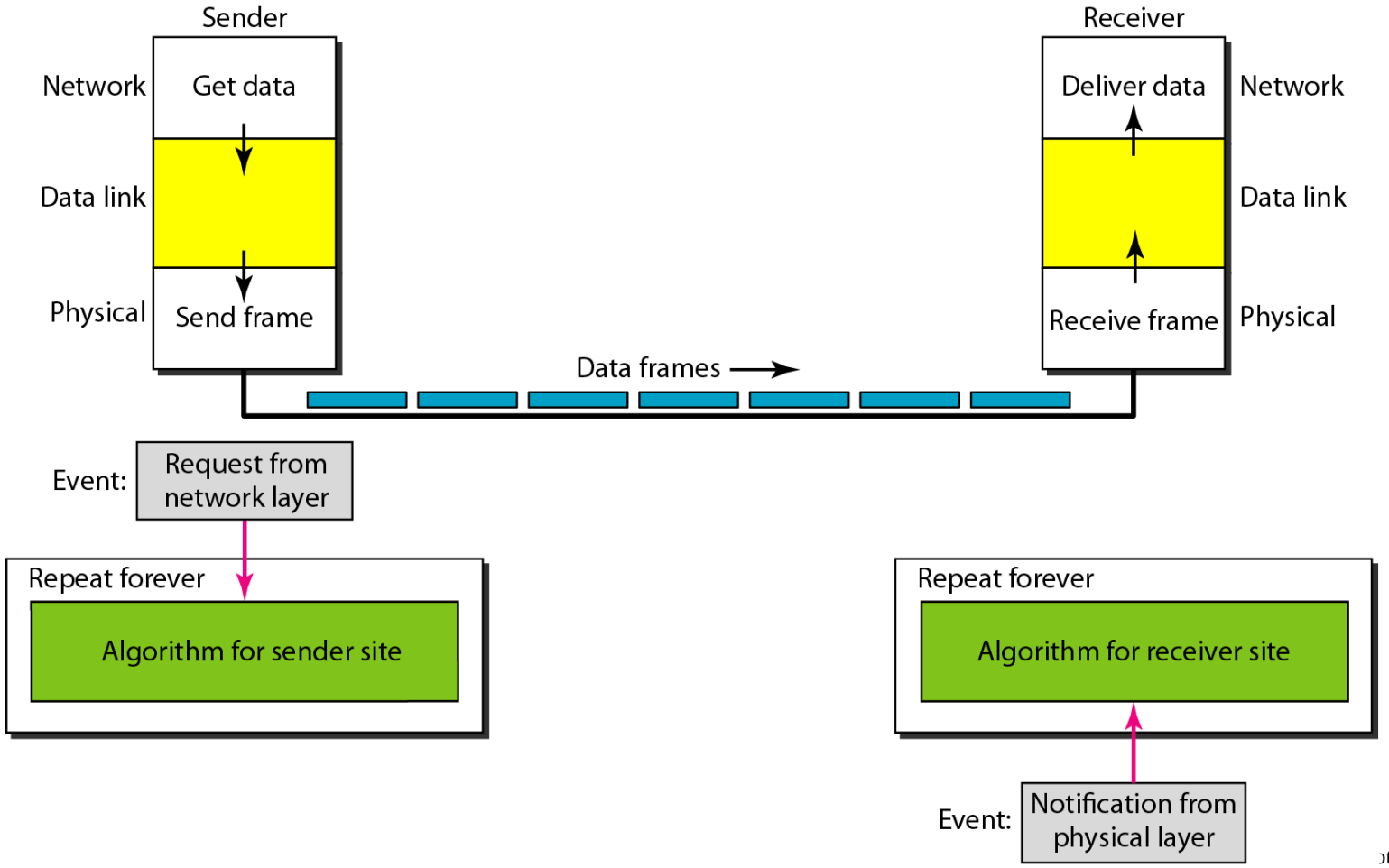
Data Link Control
- Correcting Error



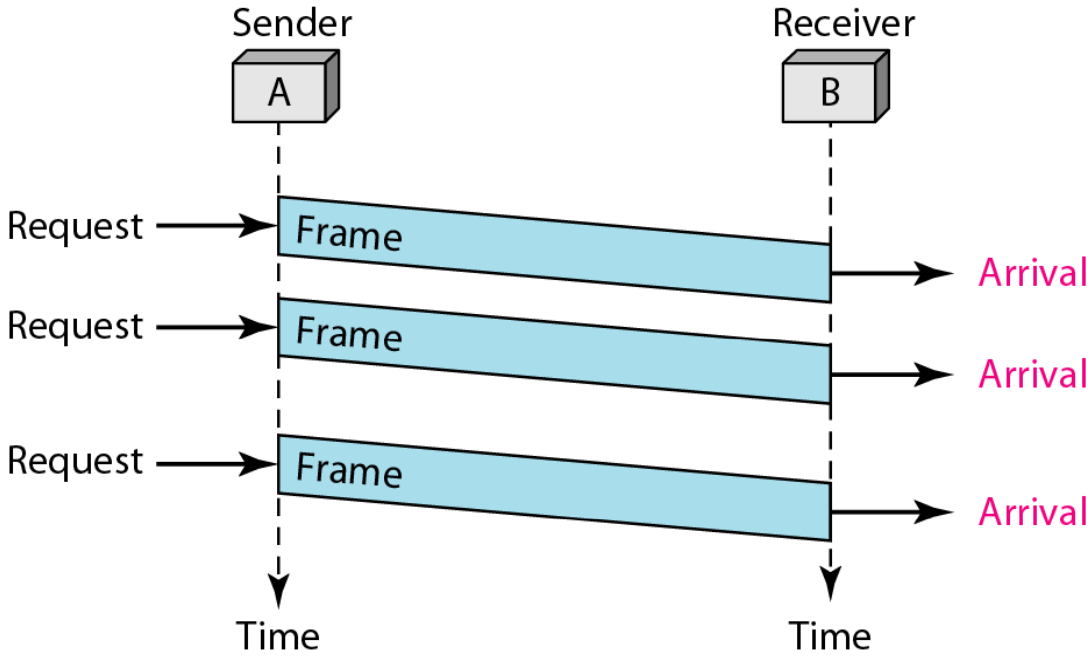
Chapter 11 of Data Communications and Networking, 4th Edition, Behrouz A. Forouzan (ISBN: 0-07-251584-8)

11-4 NOISELESS CHANNELS

Simplest Protocol (with no error correction) vs. Stop-and-Wait Protocol



Example 11.1


Sender
Receiver
A
B
Request
Frame
Arrival
Request
Frame
Arrival
Request
Frame
Arrival
Time
Time

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();                      // Receive from high layer protocol
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();                 // Receive from physical layer protocol
7       ExtractData();
8       DeliverData();                  //Deliver data to network layer
9    }
10 }
```

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

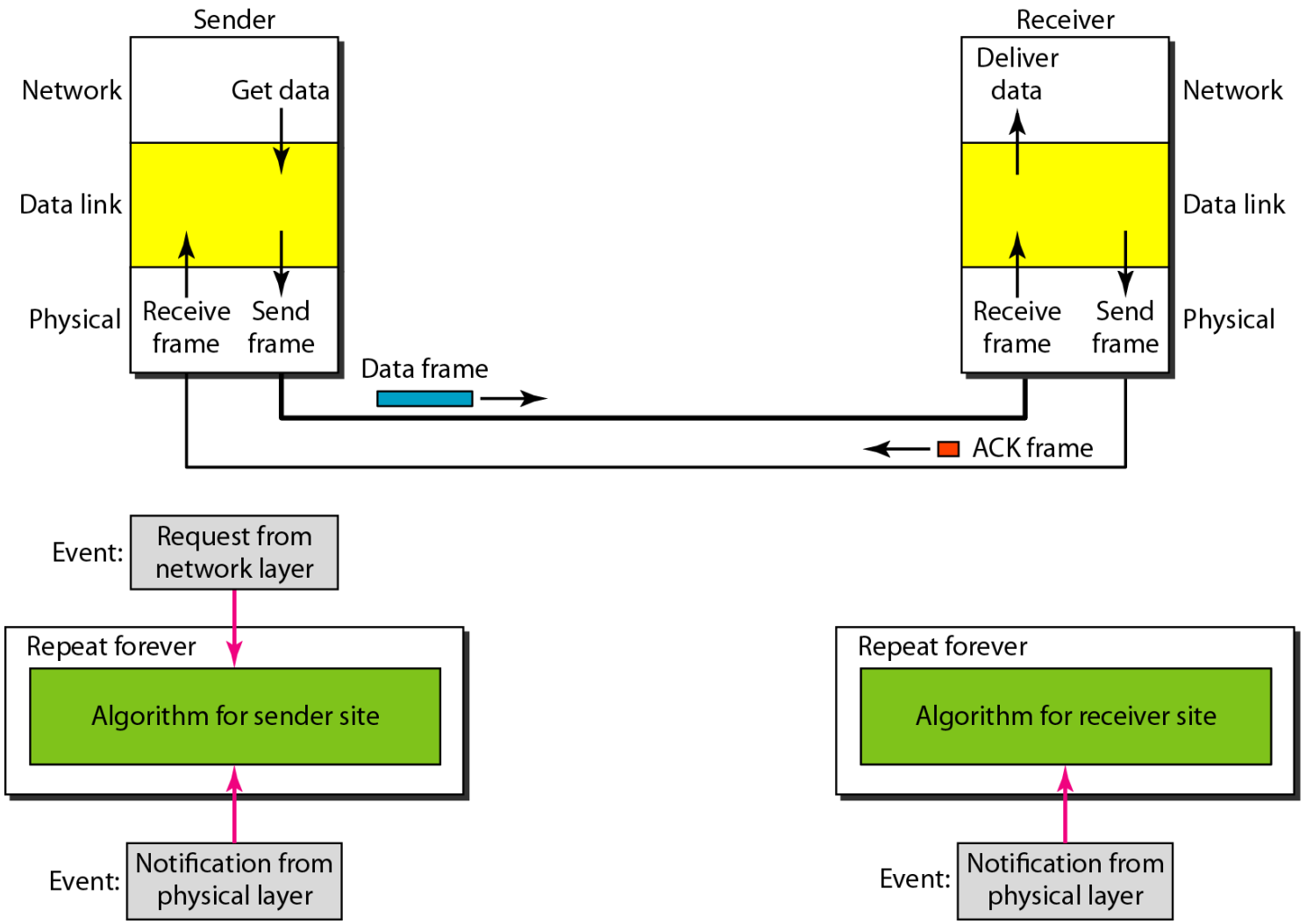
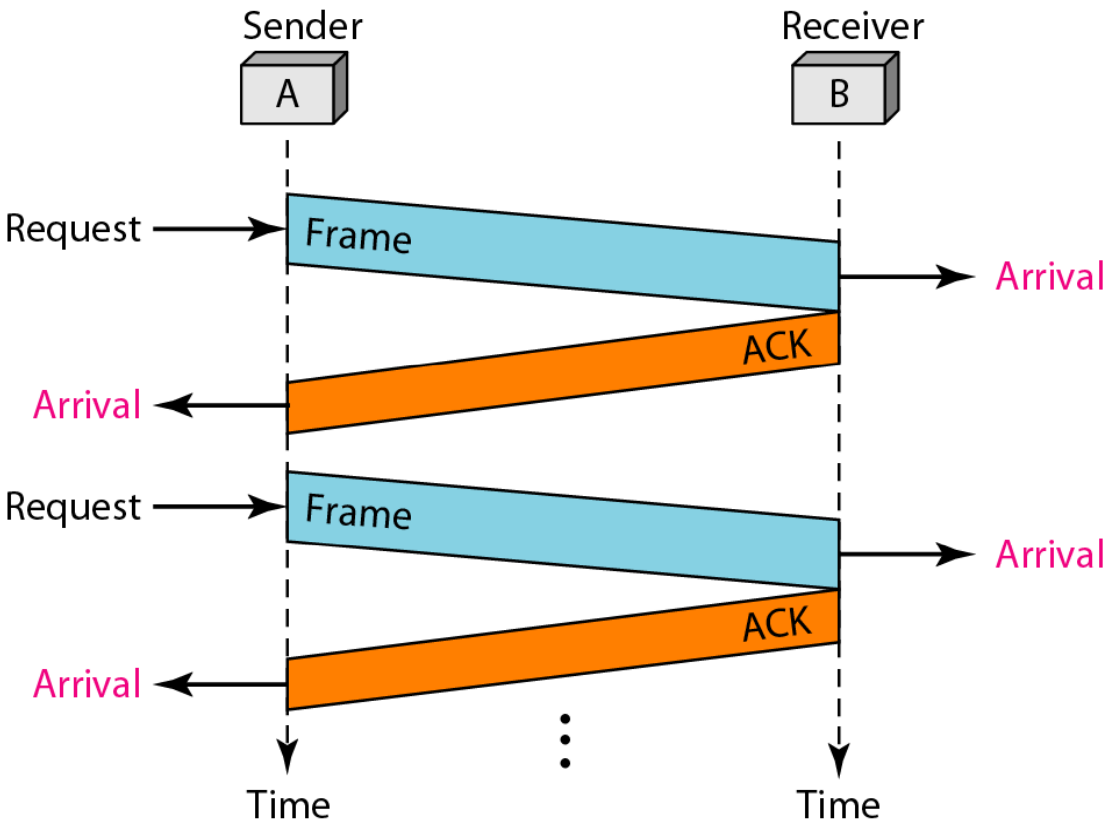


Figure 11.9 *Flow diagram for Example 11.2*



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    {                                  //Before sending next packet, waiting for ACK
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 }
```

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 based on Retransmission Techniques.

Topics discussed in this section:

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

Retransmission - Backward Error Correction

- **Based on automatic repeat request (ARQ)**
- **There are two categories:**
  - **Stop-and wait ARQ: wait ACK to be sent back for no error and NAK for error detected (see Figures 11.??)**
  - Sliding window ARQ**:**
    - **Go-back-*n***:if one frame is lost or damaged, all frames sent since the last frame acknowledgement are retransmitted
    - **Selective-reject**:only specific damaged or lost frame is retransmitted

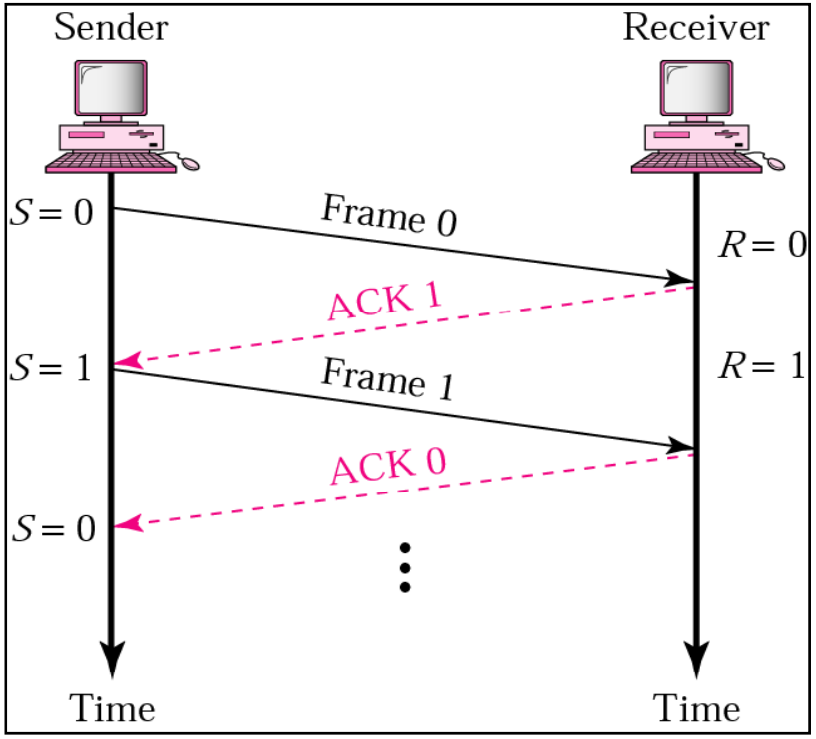
- **Correction is harder**
  - **Retransmission is simplest approach**

STOP-AND-WAIT ARQ

Note: *To avoid duplication, data frames are alternately labeled with 0 or 1, thus acknowledgement (ACK) frames are also numbered 0 and 1 alternately.*

Normal Operation

- ***Sender sends frame 0 and waits to receive ACK 1***
- ***When ACK 1 is received, it sends frame 1 and then waits to receive ACK0***
- ***repeat from step 1 and 2 and so on.***

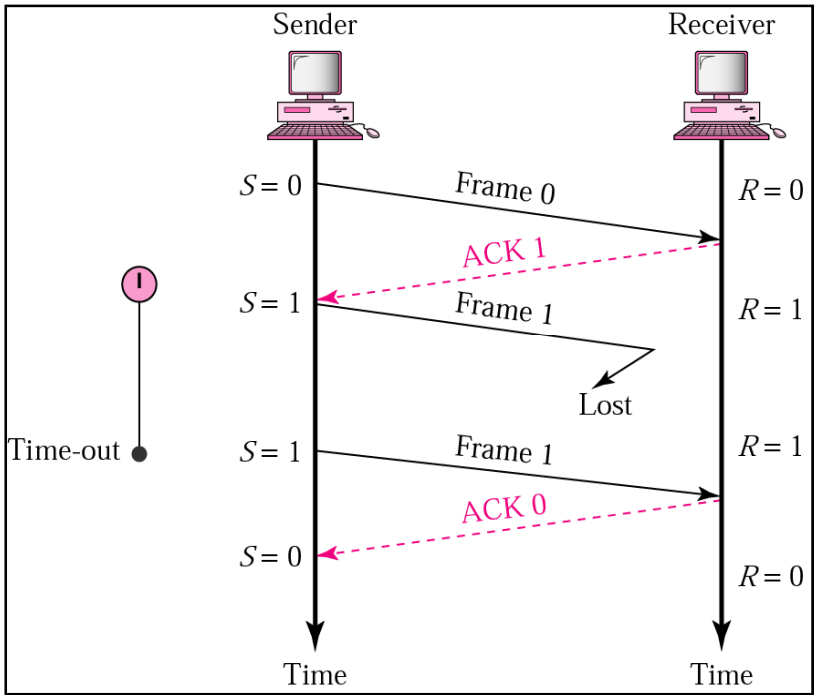


STOP-AND-WAIT ARQ (CONT)

Note: *The ACK must be received before the time set for each frame expires (Time-out).*

Lost or Damaged Frame

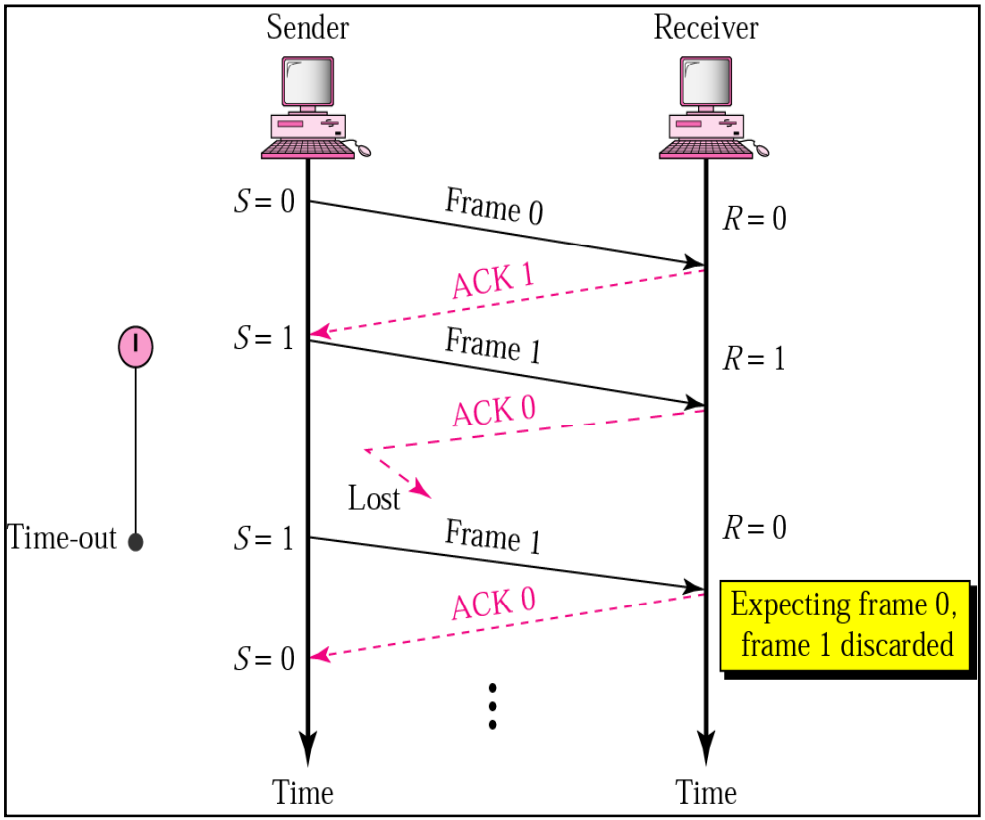
- *A lost or damaged frame is handled in the same way by the receiver*
- *When the receiver receives a damaged frame, it discards it*
- *The receiver remains silent about a lost frame and keeps its value of R*
- *After the timer at the sender site expires, another copy of frame 1 is sent -*



STOP-AND-WAIT ARQ (CONT)

Lost or Damaged Acknowledgement

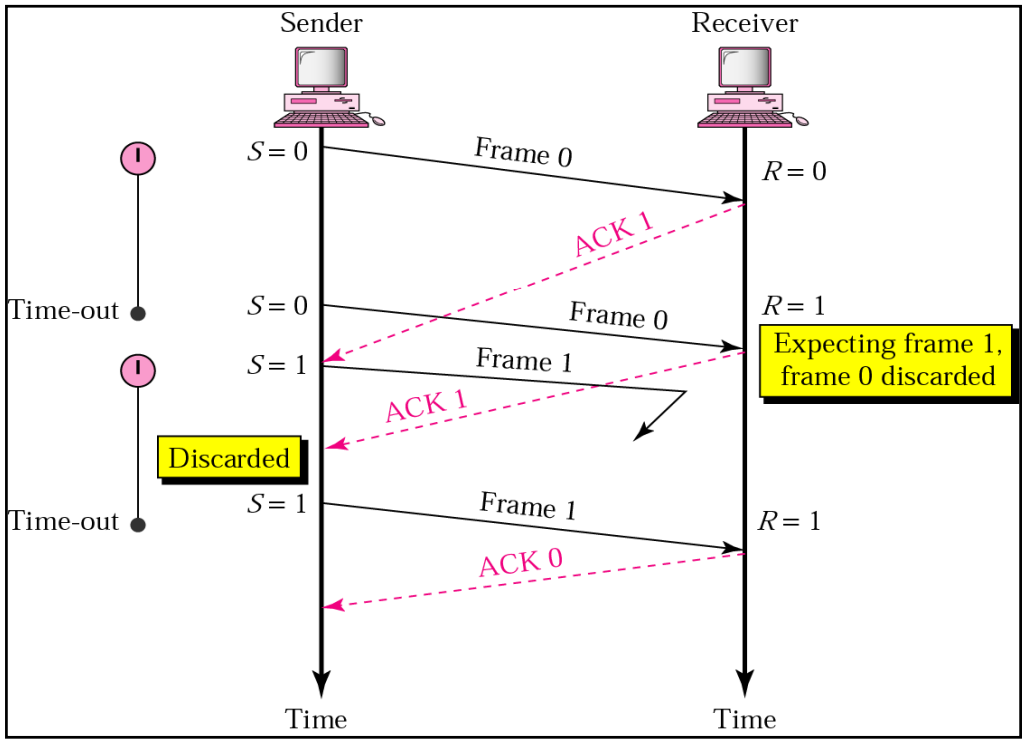
- ***If the sender receives a damaged ACK, it discards it.***
- ***When the timer for a frame expires, the sender retransmits the frame.***



STOP-AND-WAIT ARQ (CONT)

Delayed Acknowledgement

- ***An acknowledge can be delayed at the receiver or by some problem with the link***
- ***If receiver receives a duplicated frame, it discards the duplicated frame***



STOP-AND-WAIT ARQ (CONT)

Piggybacking

- ***A method to combine a data frame with an acknowledgement in case both sender and the receiver have data to send.***

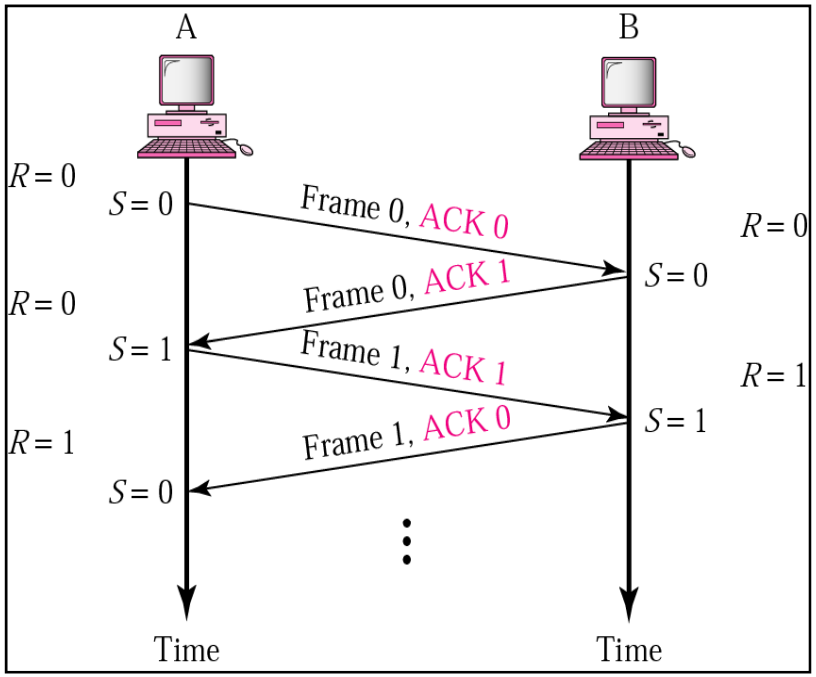
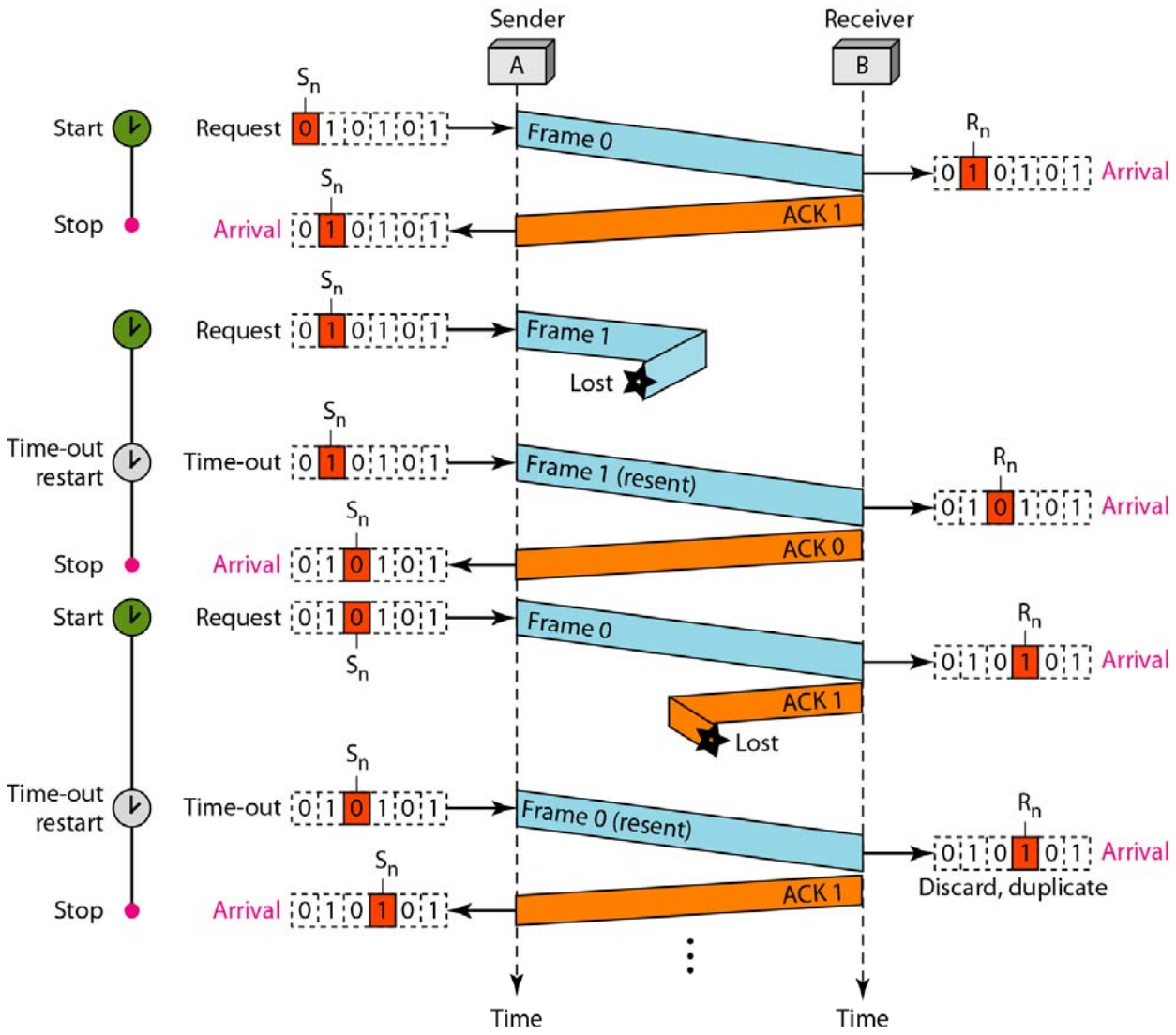
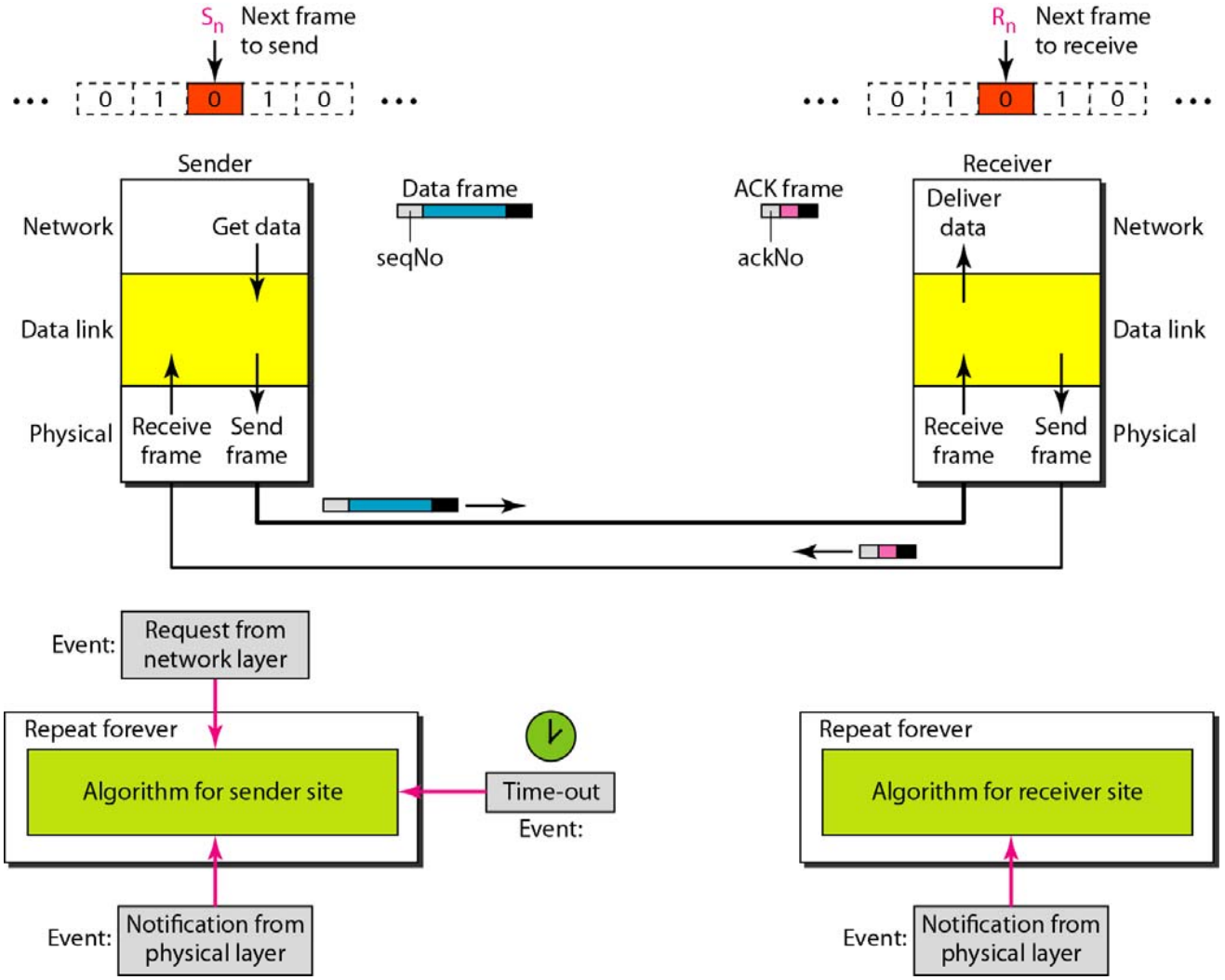


Figure 11.11 *Flow diagram for Example 11.3*



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



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
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 11.4

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

Solution

The bandwidth-delay product is

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

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

Example 11.5

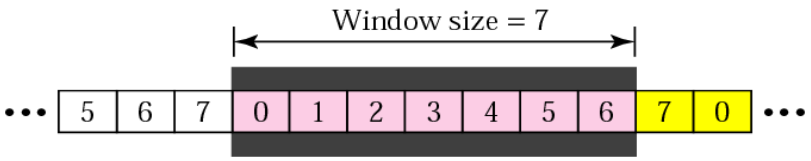
What is the utilization percentage of the link in Example 11.4 if we have a protocol that can send up to 15 frames before stopping and worrying about the acknowledgments?

Solution

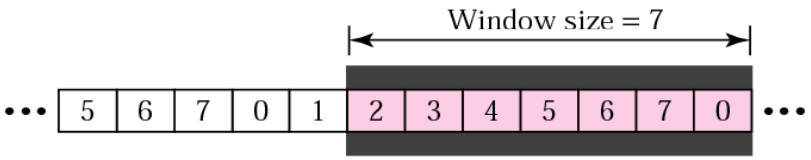
The bandwidth-delay product is still 20,000 bits. The system can send up to 15 frames or 15,000 bits during a round trip. This means the utilization is 15,000/20,000, or 75 percent. Of course, if there are damaged frames, the utilization percentage is much less because frames have to be resent.

GO-BACK-N ARQ

- **Sequence Numbers**
  - Frame from a sending station are numbered sequentially. If the header of the frame allows m bits fro the sequence number, the sequence numbers range from 0 ~ $2^m - 1$
- **Sender Sliding Window**
  - Sender uses *window* to hold the outstanding frames until they are acknowledged
  - Frames to the left of the window are those that have already been acknowledged and can be purged
  - Frames to the right of the window can not be sent until the window slides them



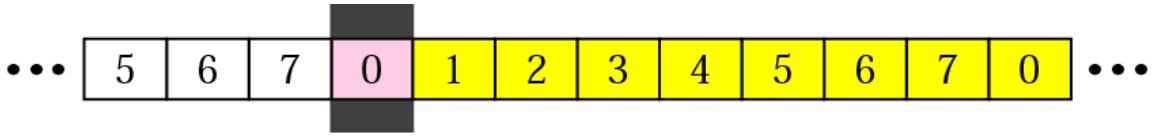
a. Before sliding



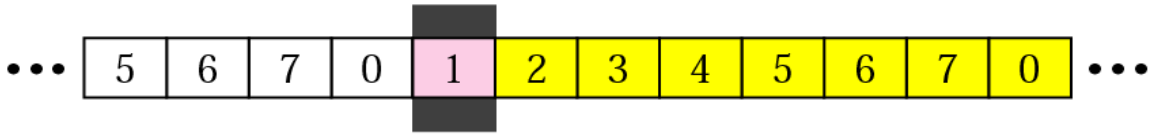
b. After sliding two frames

GO-BACK-N ARQ (CONT)

- **Receiver Sliding Window**
  - **Size of the window is always 1**
  - **The receiver is always looking for a specific frame to arrive in a specific order**



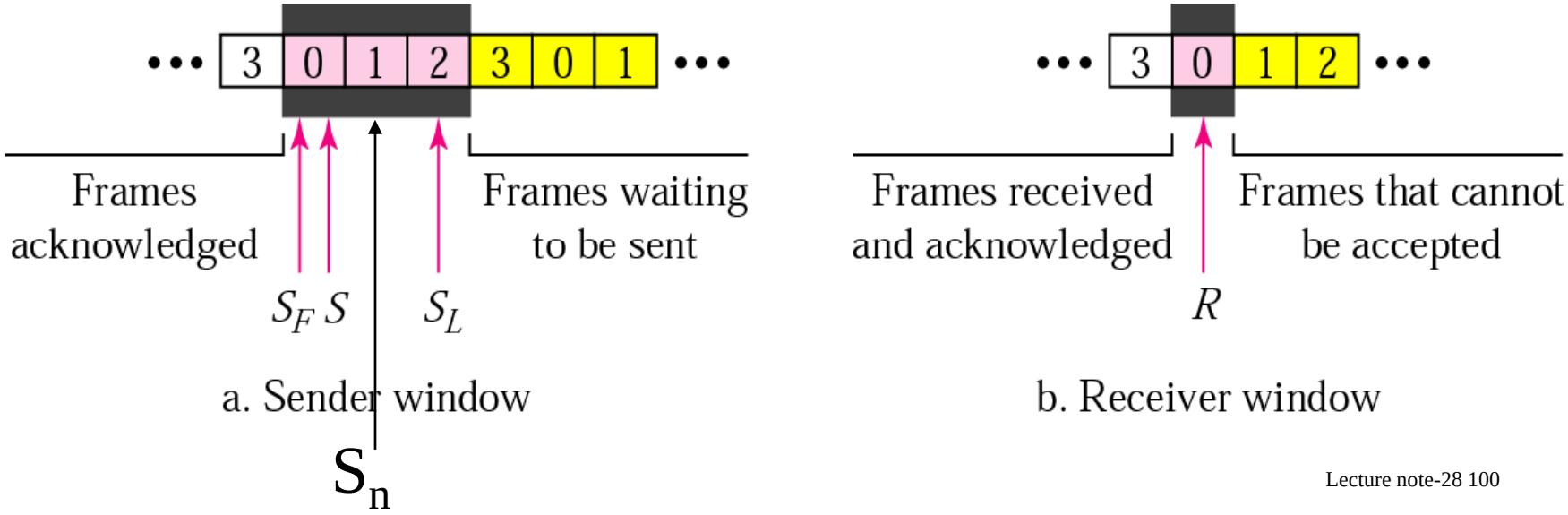
a. Before sliding



b. After sliding

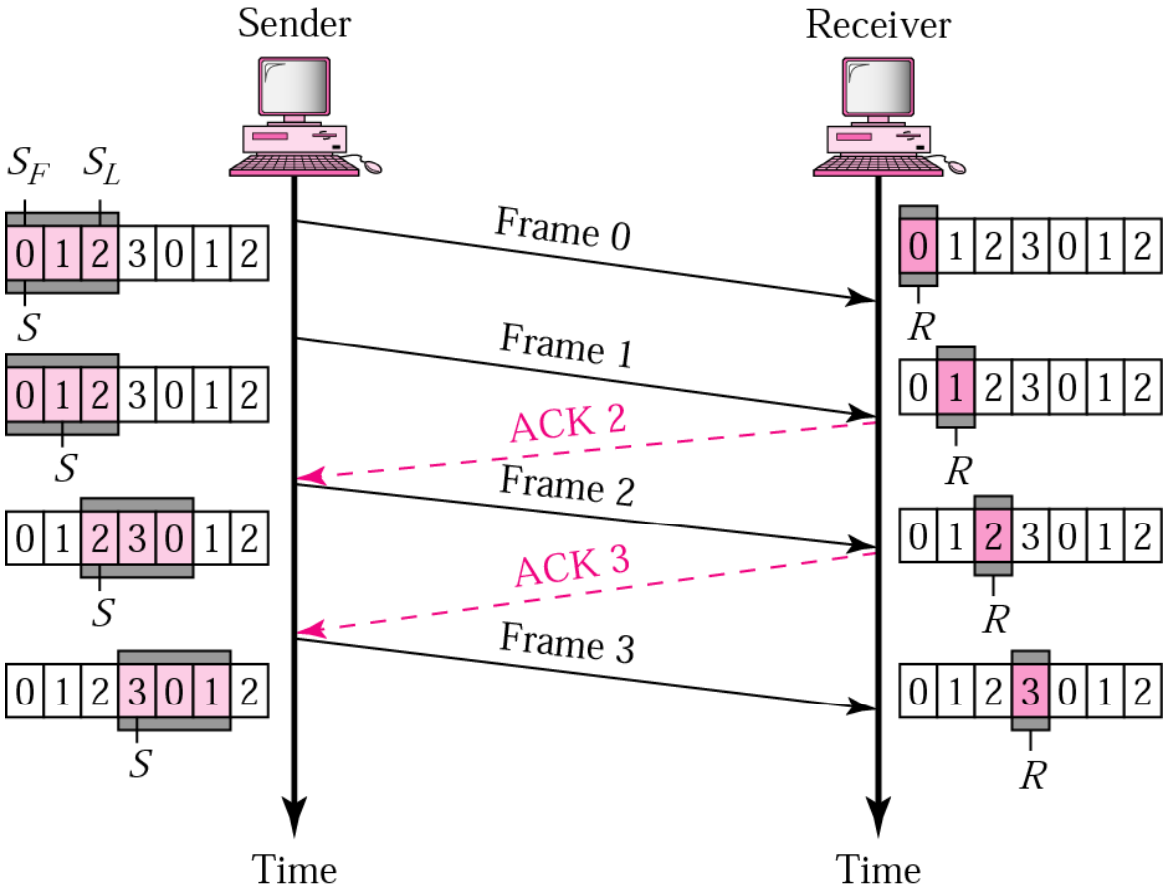
GO-BACK-N ARQ (CONT)

- **Control Variables**
  - **The sender has three variables**
    - ***S***: holds the sequence number of the recently sent frame
    - S_L : holds the sequence number of the last frame in the window
    - S_F : Holds the sequence number of the first frame in the window
    - Size of the window: $W = S_L - S_F + 1$
  - **The receiver only has one variable: R, that holds the sequence number of the frame it expects to receive**



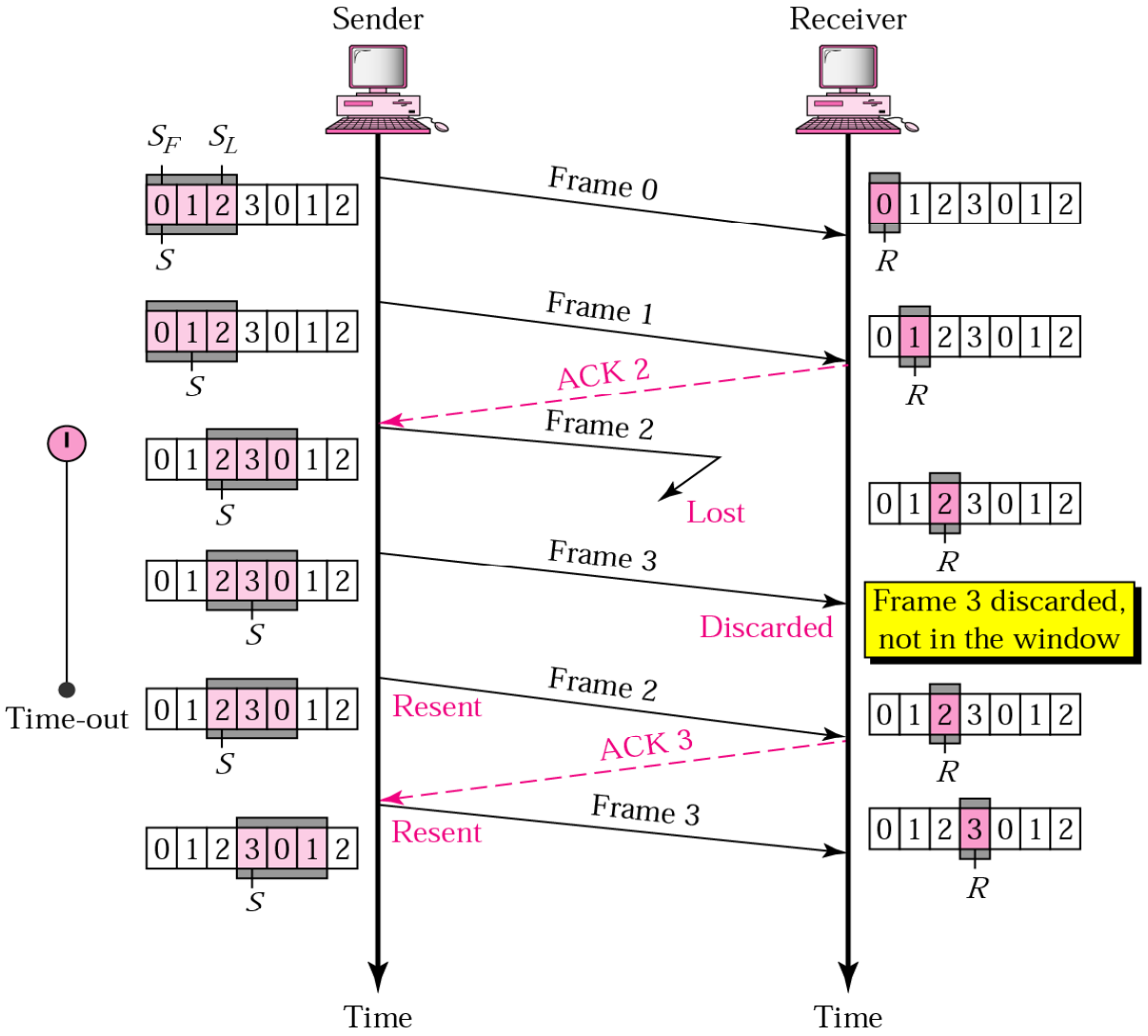
GO-BACK-N ARQ (CONT)

Normal Operation



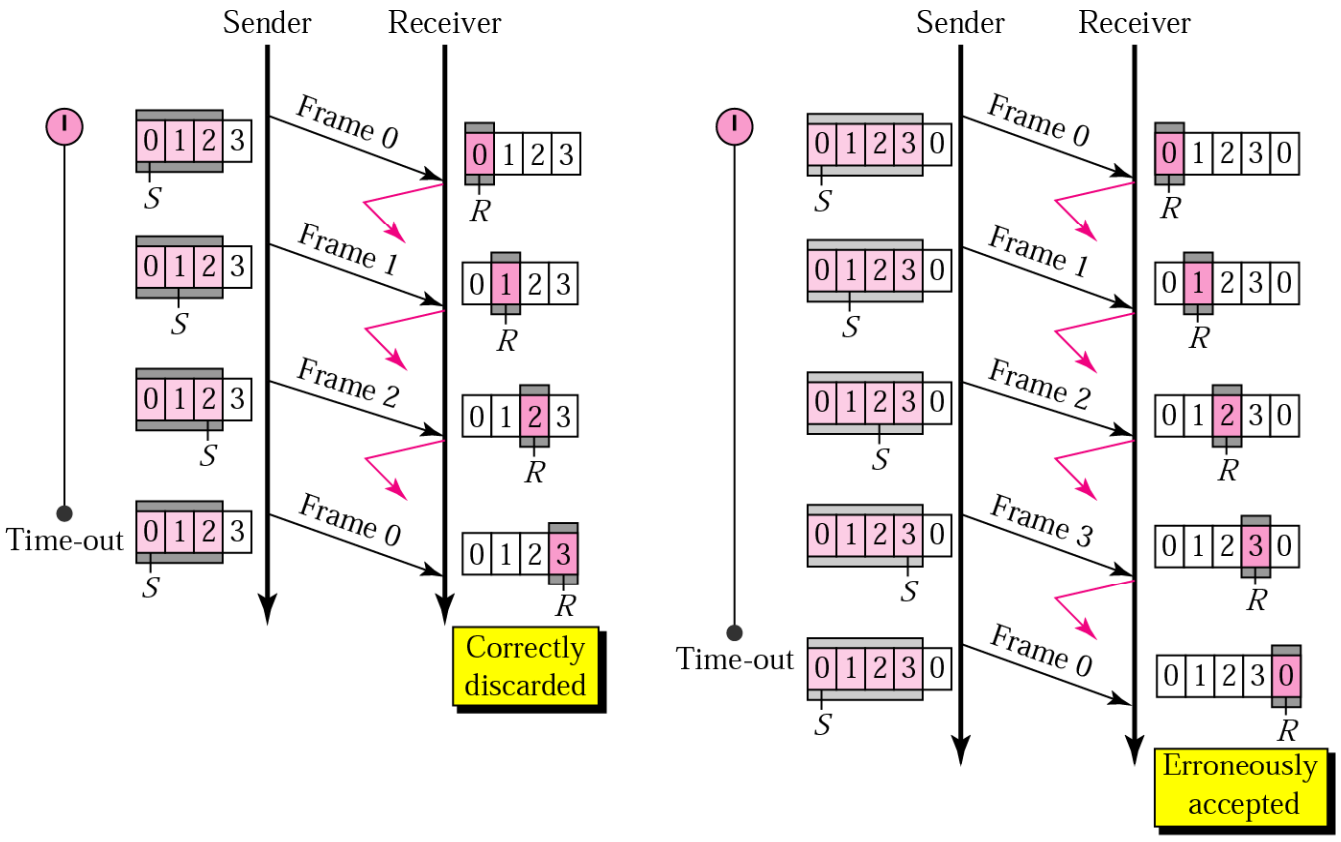
GO-BACK-N ARQ (CONT)

Damaged or Lost Frame



GO-BACK-N ARQ(CONT)

Sender window size



a. Window size < 2^m

b. Window size = 2^m

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.

Figure 11.16 *Flow diagram for Example 11.6*

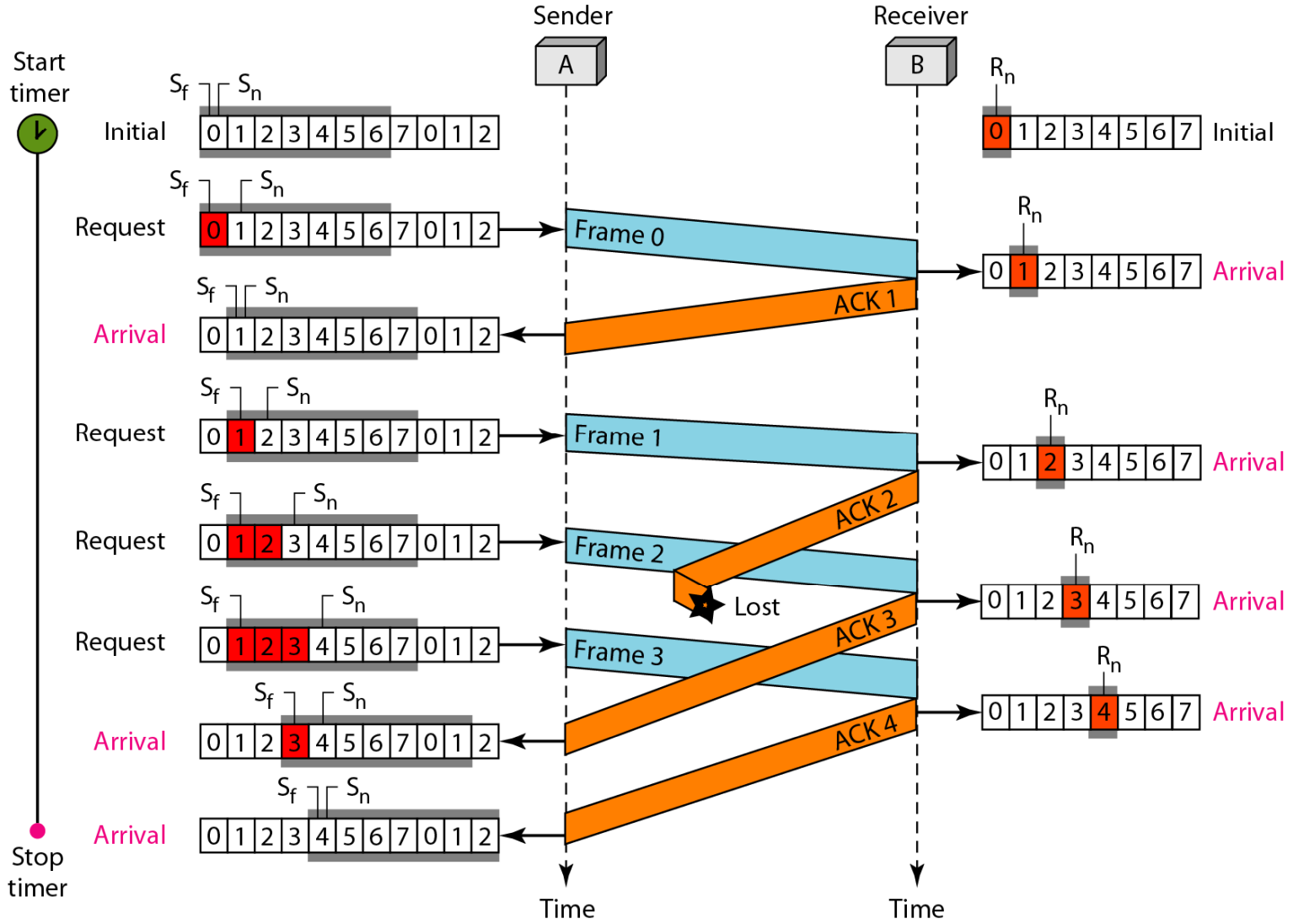
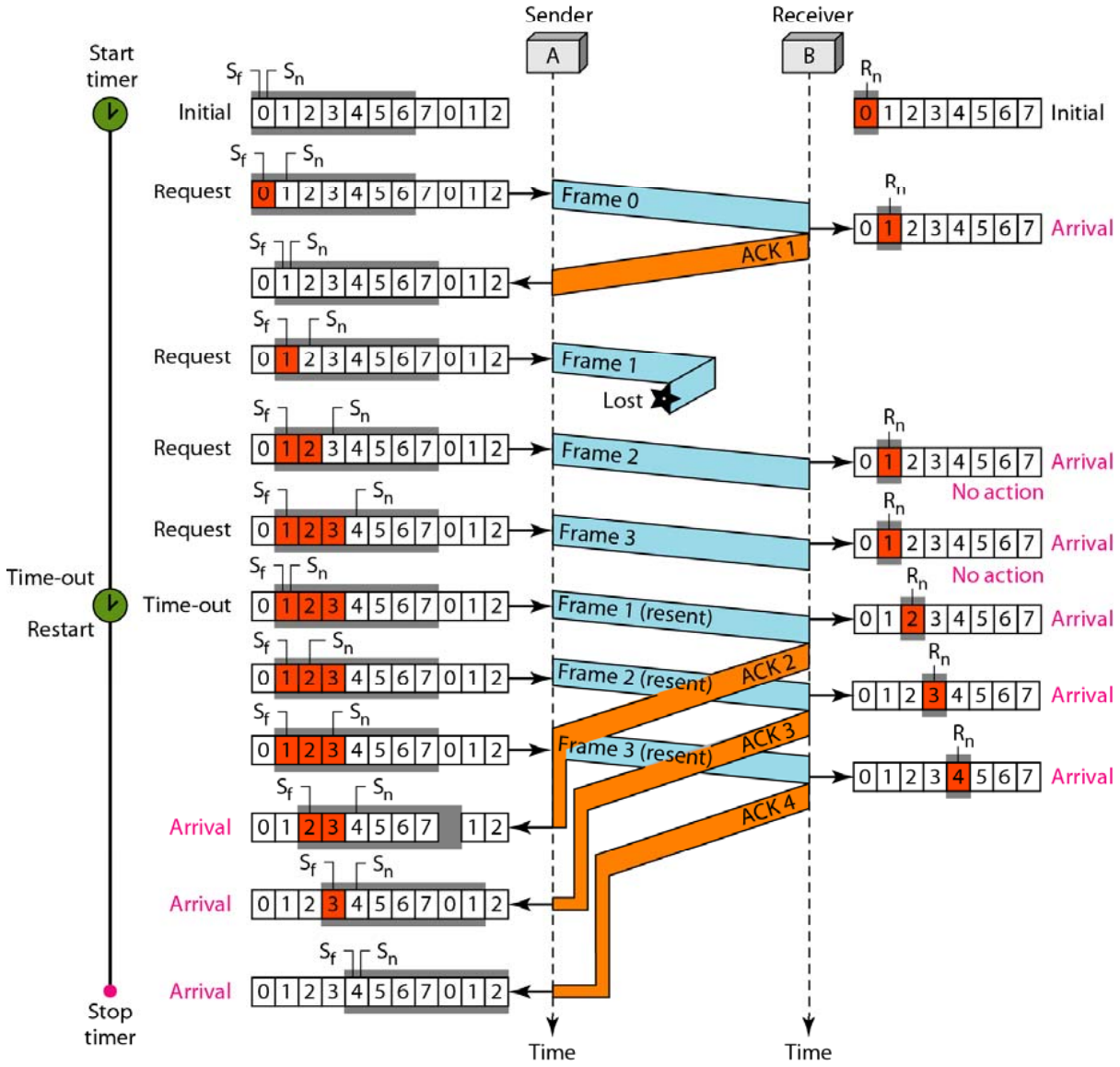
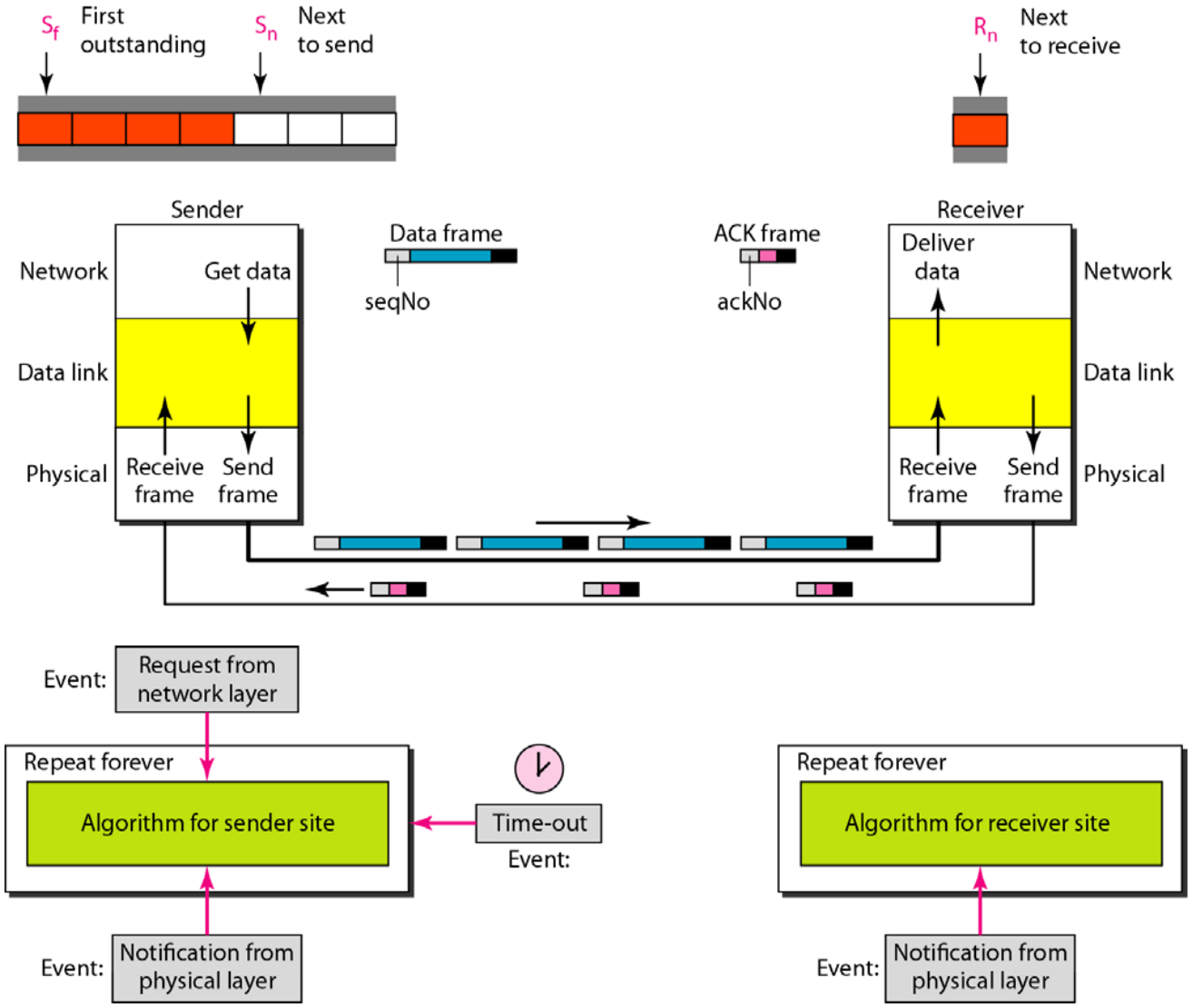


Figure 11.17 *Flow diagram for Example 11.7*



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



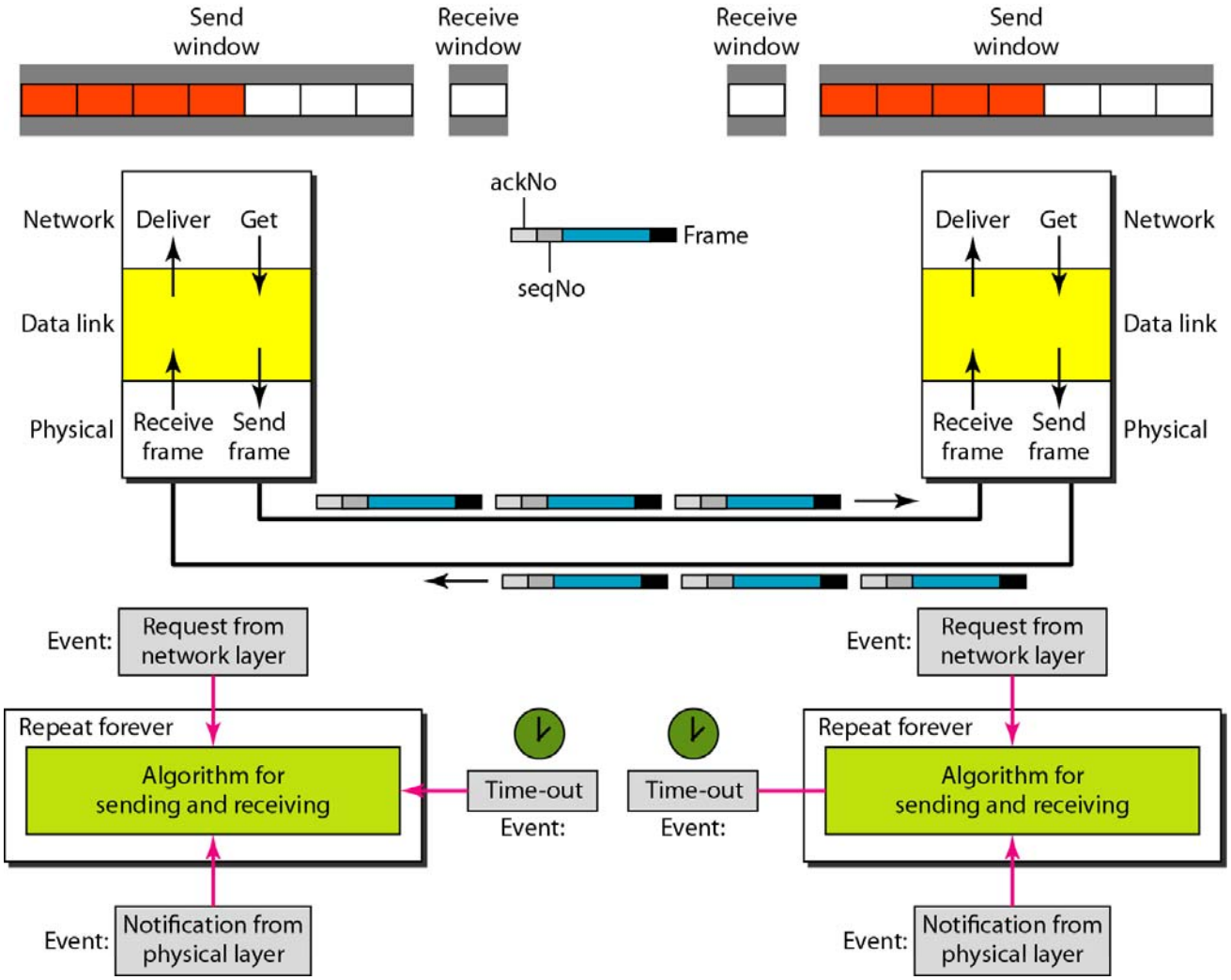
Algorithm 11.7 *Go-Back-N sender algorithm*

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

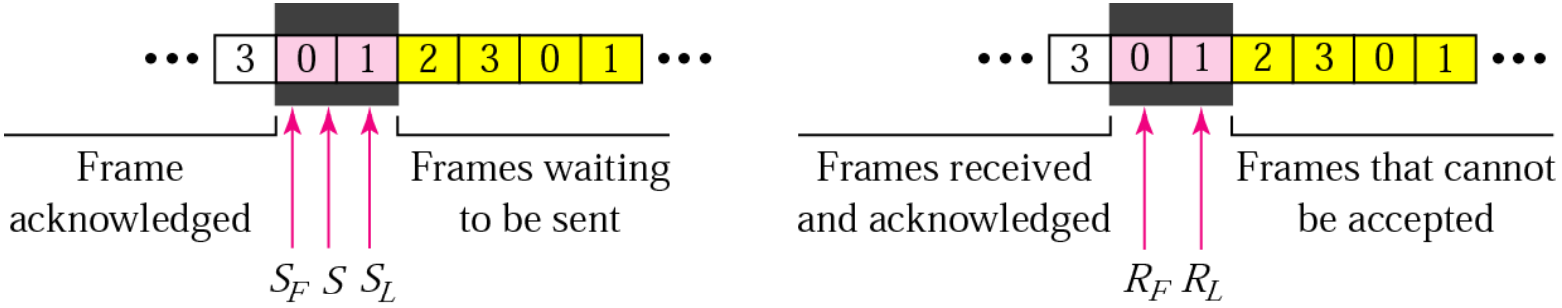


GO-BACK-N ARQ(CONT)

- **Problem**
  - **No out-of-order frames**
  - **Not efficient**
    - Multiple frame may be retransmitted

SELECTIVE REPEAT ARQ

- **Only damaged frame is resent**
- **Sender and receiver windows**
  - **Size of the sender's window must be one half of 2^m**
  - **Size of the receiver's window must be same as the sender's**
  - **Negative acknowledgement (NACK) is used to define damaged frame**

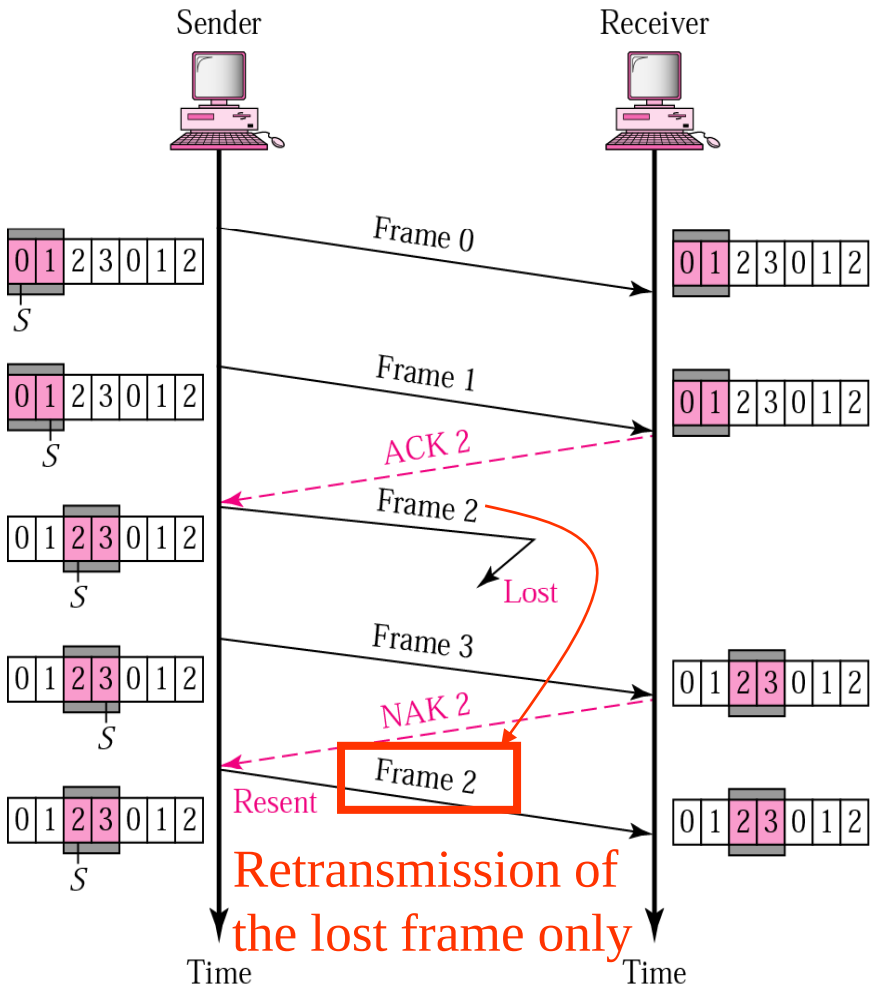


a. Sender window

b. Receiver window

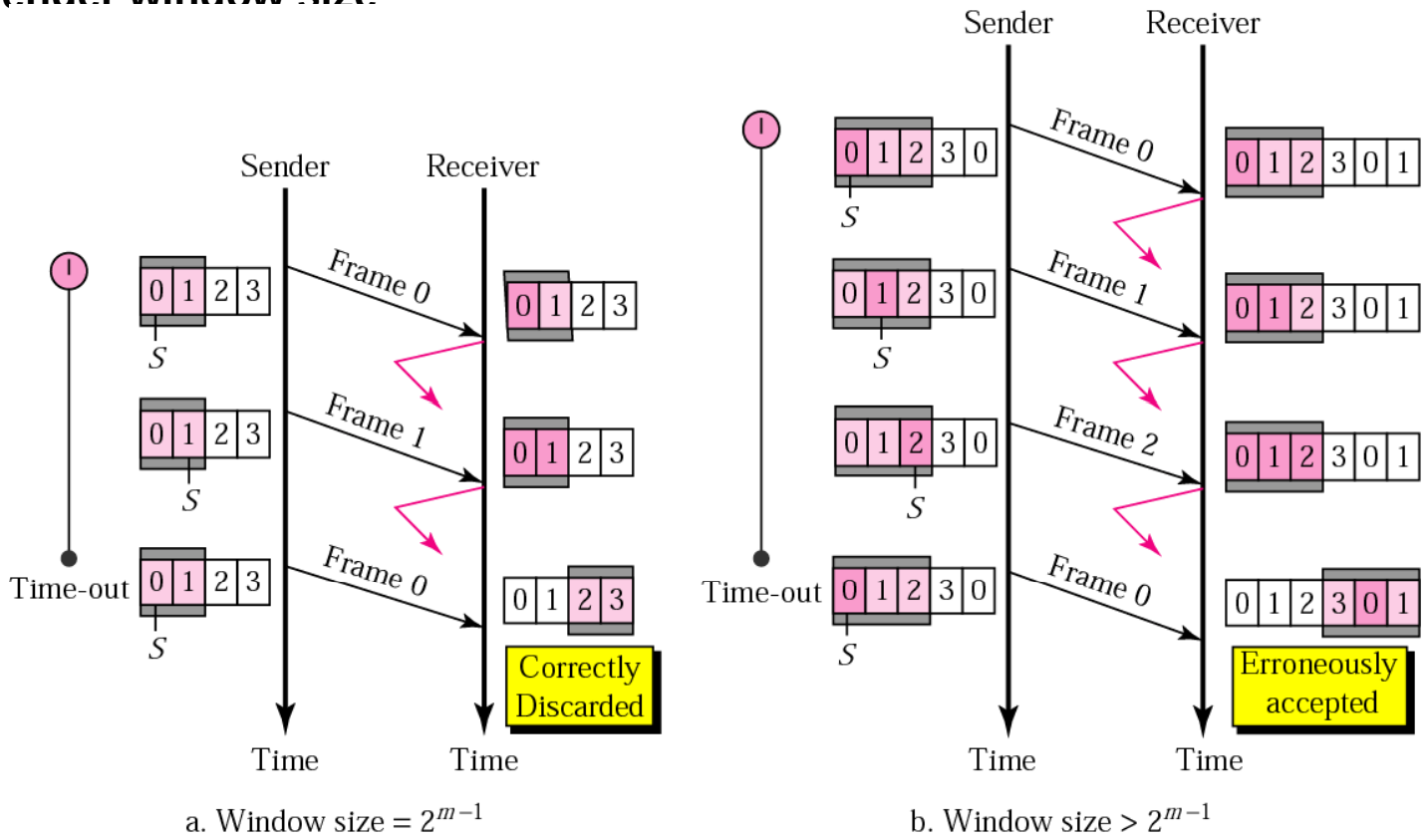
SELECTIVE REPEAT ARQ (CONT)

Lost Frame



SELECTIVE REPEAT ARQ (CONT)

Sender window size



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

Figure 11.23 *Flow diagram for Example 11.8*

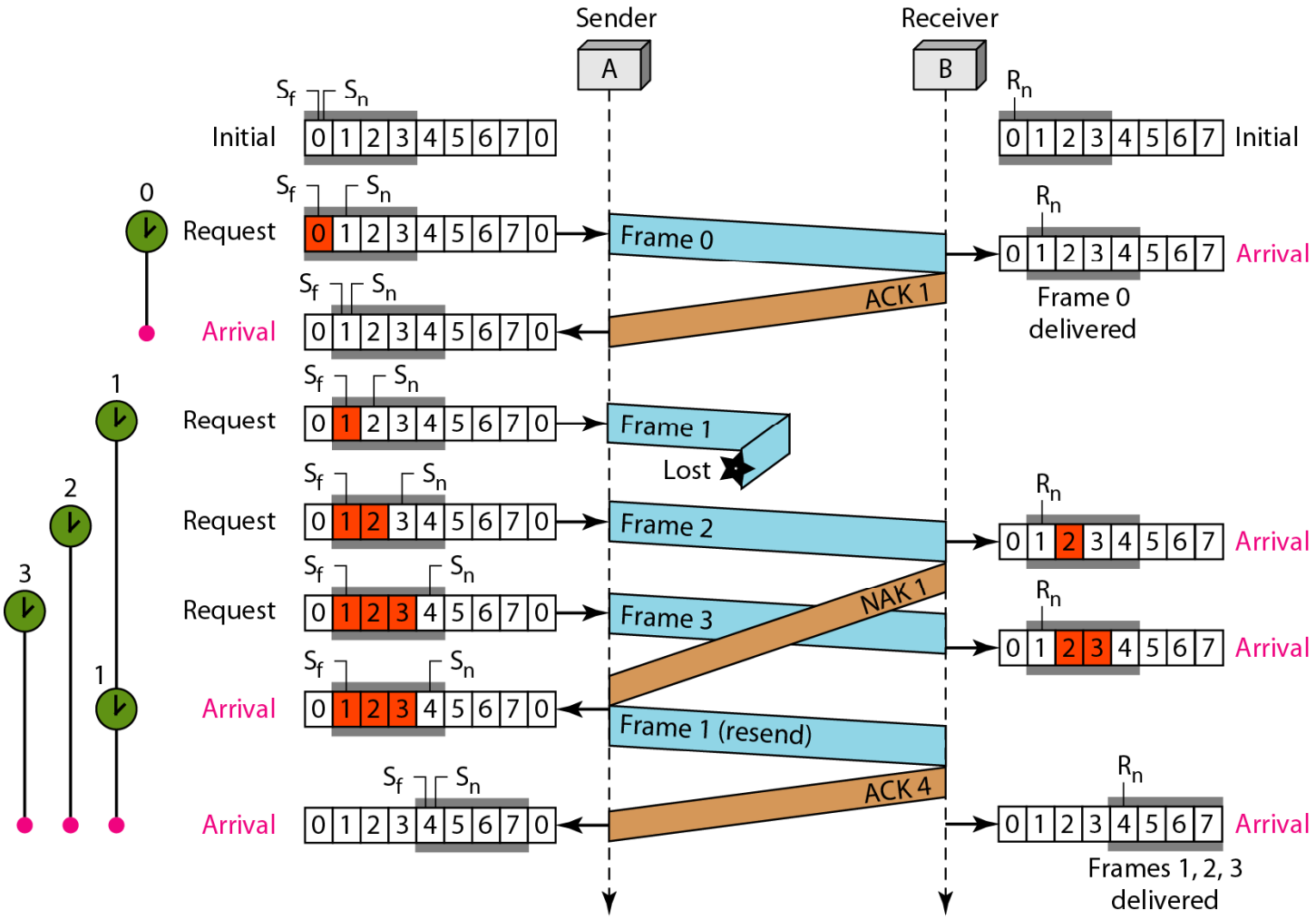
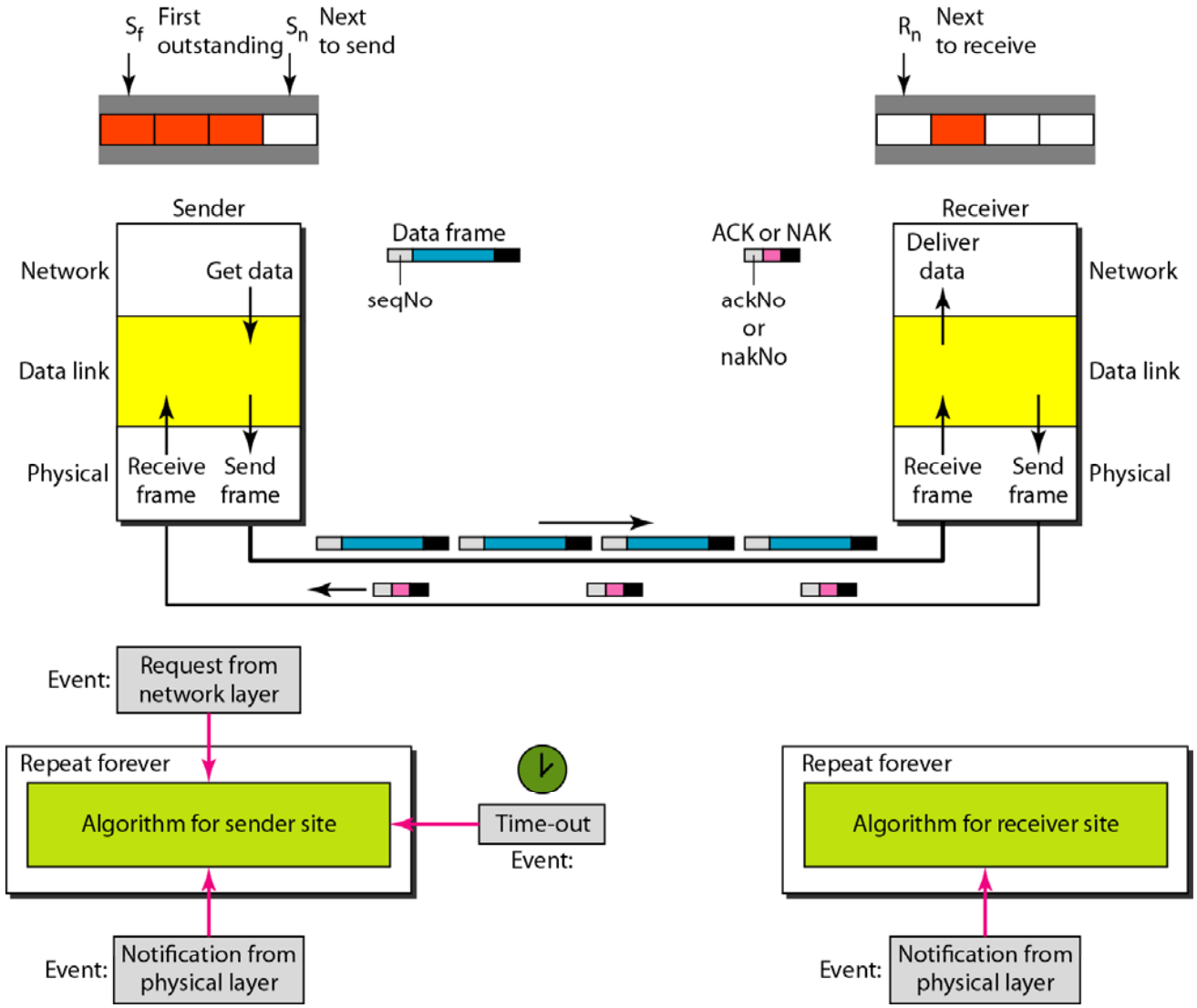


Figure 11.20 *Design of Selective Repeat ARQ*



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  }
```

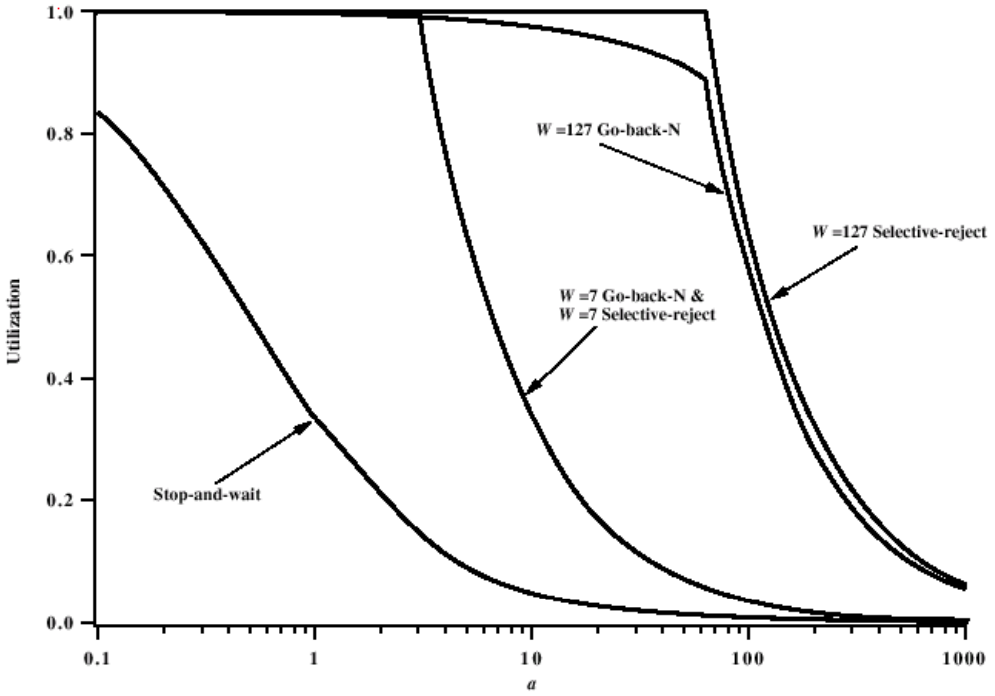
ARQ Performance

- **Stop and Wait:**

$$U = \frac{1-P}{1+2a}$$

- **Selective Reject**

$$U = \begin{cases} 1 & W \geq 2a+1 \\ \dfrac{W(1-P)}{(2a+1)} & W < 2a+1 \end{cases}$$



- **Go back N**

$$U = \begin{cases} \dfrac{1-P}{1+2aP} & W \geq 2a+1 \\ \dfrac{W(1-P)}{(2a+1)(1-P+WP)} & W < 2a+1 \end{cases}$$

$a = t_{prop}/t_{frame}$

P=error probability of a frame

W= window size

ERROR DETECTION / CORRECTION SUMMARY

- **Error detection & correction designed to preserve the INTEGRITY of transmitted data -- not SECURITY**
- **Detection is relatively easy**
- **Correction is harder**
  - **Retransmission is simplest approach**
- **Both sender and receiver must use same detection/correction method**
  - **Type of method and algorithms specified by protocol**

DATA LINK ERROR CONTROL (CONT)

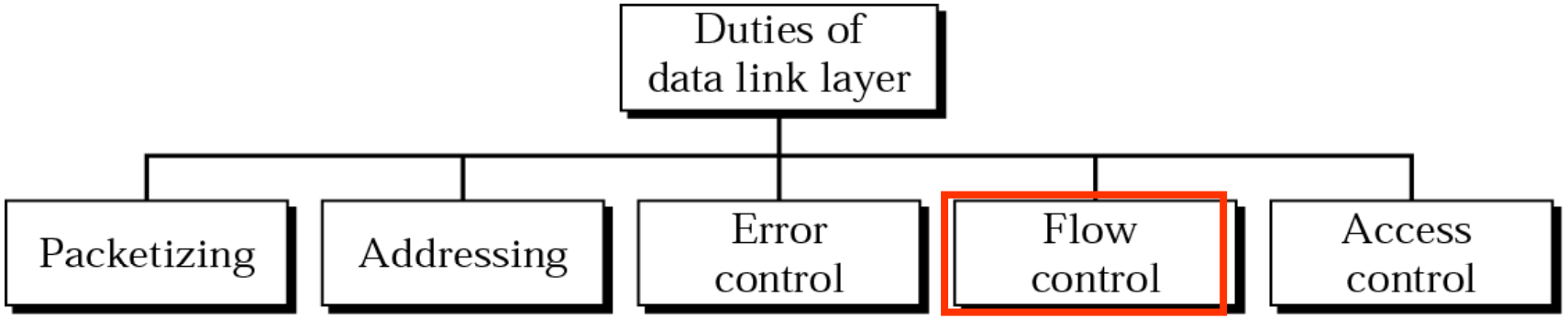
- **Error control**
  - **Based on automatic repeat request (ARQ), which is the retransmission of data**
- **Type of error handled**
  - **Lost frame**
  - **Damaged frame**
- **Techniques for error control**
  - **Error detection**
  - **Positive acknowledgement**
  - **Retransmission after timeout**
  - **Negative acknowledgement and retransmission**

Note: **data link layer divides the stream of bits received into manageable data units called *frames*.**

DATA LINK ERROR CONTROL (CONT)

- **Stop & Wait ARQ (Half Duplex)**
  - **Receiver responds to each block (ACK / NAK / WACK)**
  - **Sender waits for response before sending further blocks**
- **Continuous ARQ (Full Duplex)**
  - **Blocks sent continuously without waiting for confirmation**
  - **If error message returned to sender,**
  - **Types of continuous ARQ**
    - Go-Back-N: restart transmission at the Nth frame
    - Selective Repeat: retransmit the selected frame only

Data link layer duties



framing

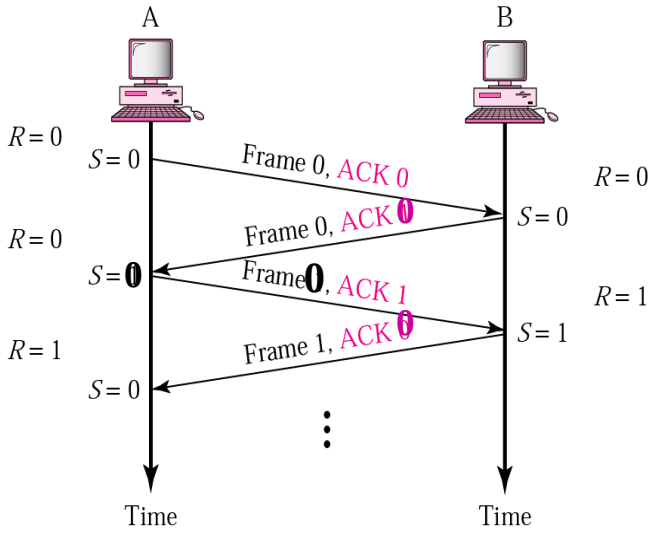
- **Data link protocols have three functions:**
  - Error Control**: Detecting and correcting transmission errors. (Error & flow)**
  - Media Access Control**: Controlling when computers transmit. Who should send now(Access control)**
  - Message Delineation**: Identifying the beginning and end of a message. (Packetizing & Addressing)**

Flow Control

- **Is *a set of procedures* that tells the sender how much data it can transmit before it must wait for an acknowledgement from the receiver**
- **This is due to the limited speed at which the receiver can process incoming data and limited amount of memory**
- **The data received is normally store in a block of memory (**buffer**) before they are checked and processed**
- **2 main methods used flow control:**
  - Stop-and-wait: **send one frame at a time**
  - Sliding window: **send several frames at a time**

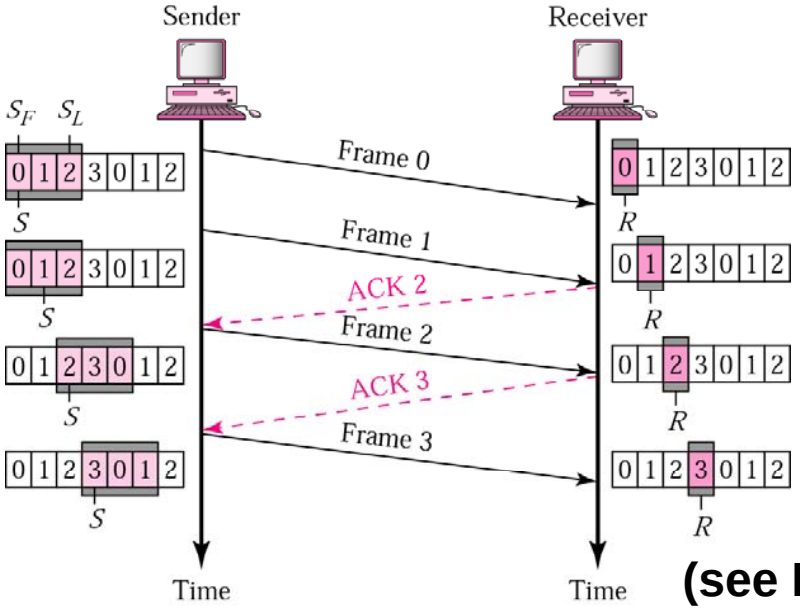
Stop-and-wait Flow Control

- The sender waits for an acknowledgement after every frame (see Figure 11.1)
- The process repeats until a end of transmission (EOT) frame is received
- The advantage is simplicity as each frame is checked and acknowledged before the next frame is sent
- The disadvantage is inefficiency as each frame must travel all the way to the receiver and an acknowledgement must travel all the way back before the next frame can be sent.



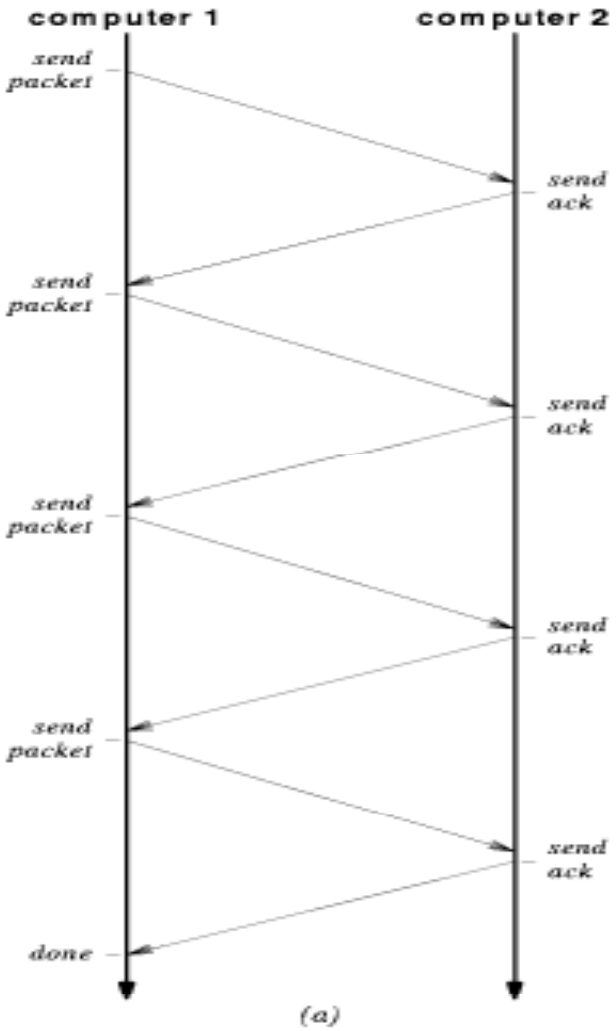
Sliding Windows Flow Control

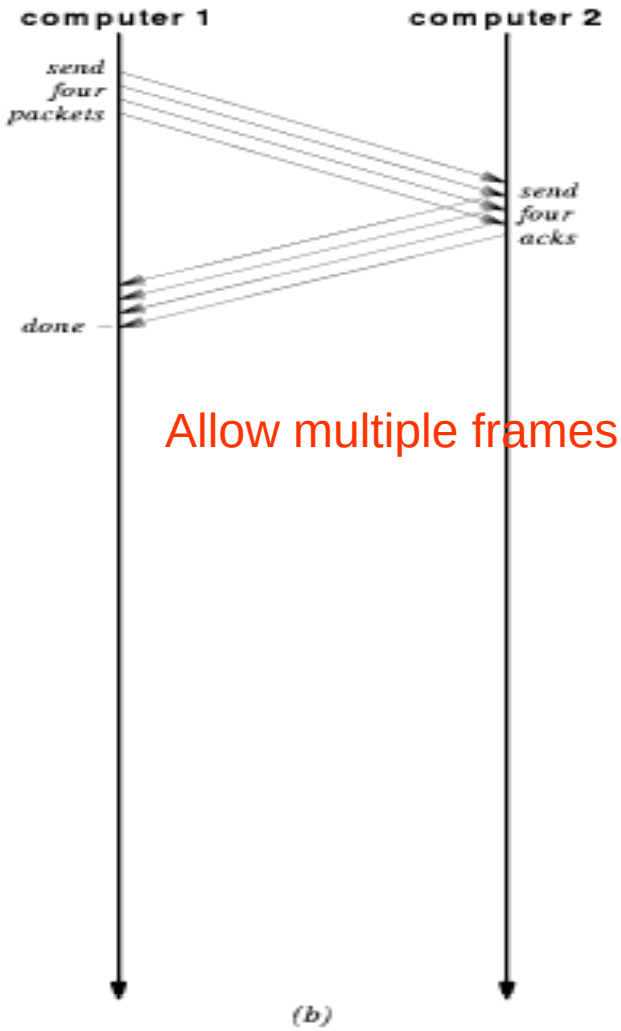
- **Allows sender to** transmit multiple packets before receiving an acknowledgment
- Number of packets that can be sent **is defined by the protocol and called the** window
- **As acknowledgments arrive from the receiver, the window is moved along the data packets; hence ``sliding window''**



(see Figure 11.8)

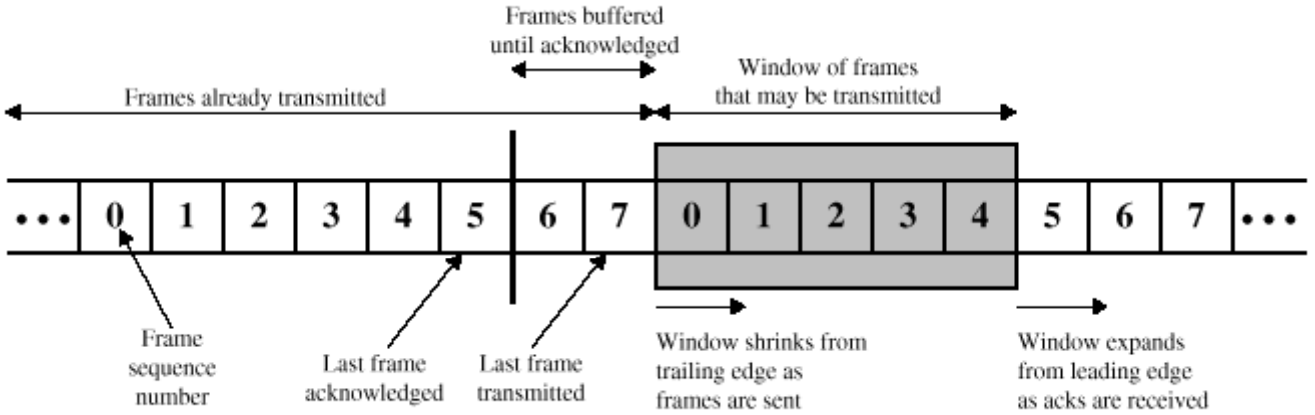
stop-and-go vs sliding window



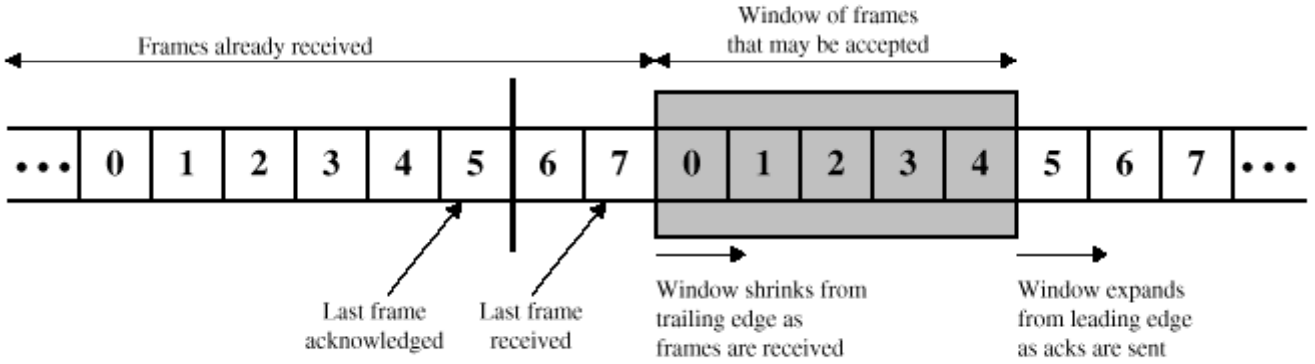


Allow multiple frames to be in transit

Example of Sliding Window



(a) Sender's perspective



(b) Receiver's perspective

Figure 9.5 Sliding-Window Depiction

Example

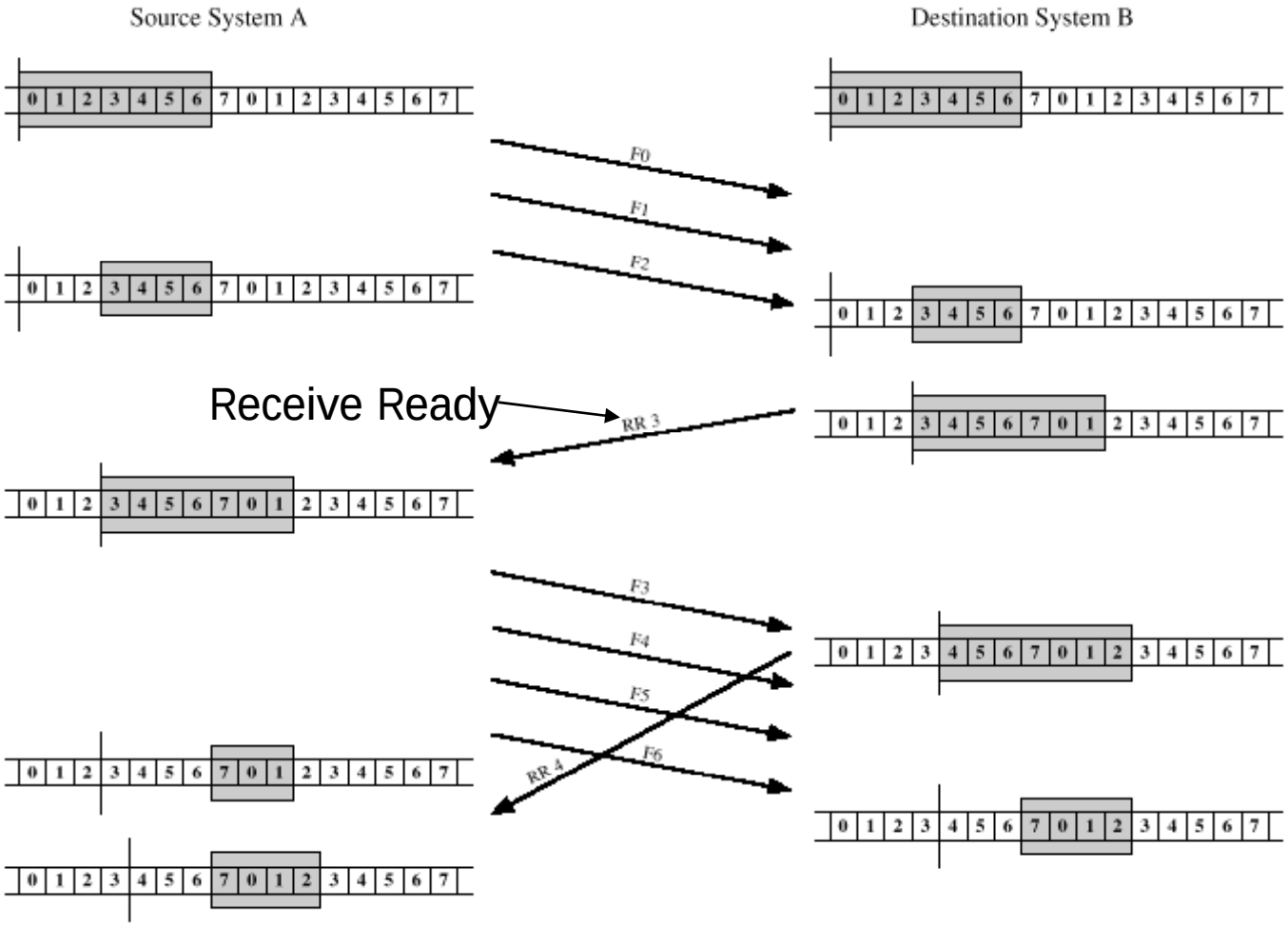
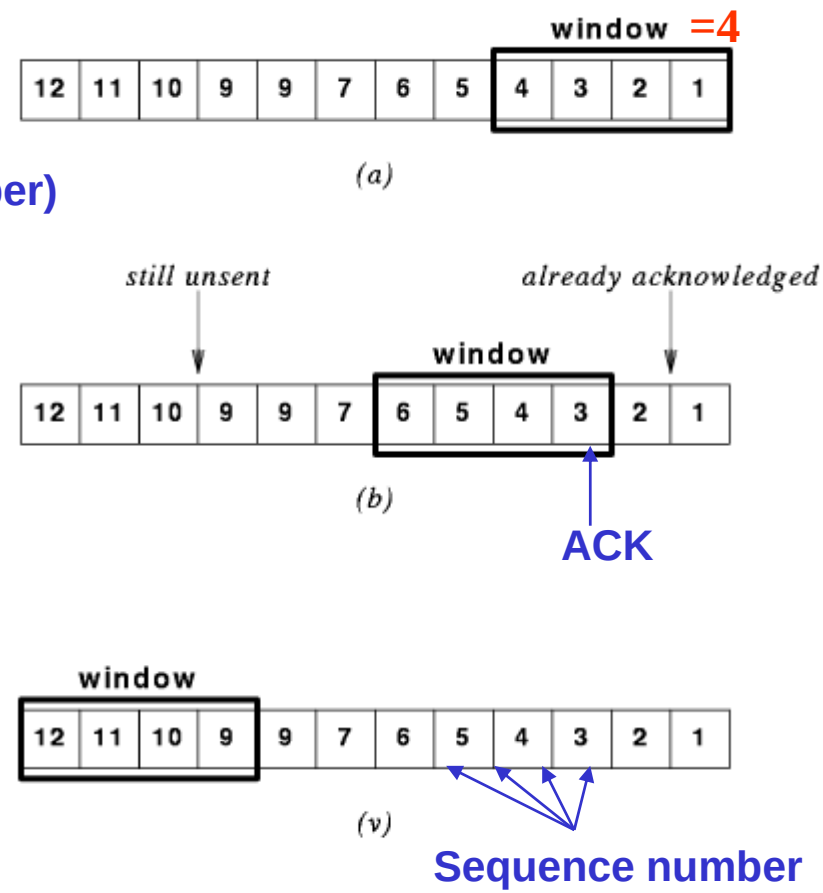


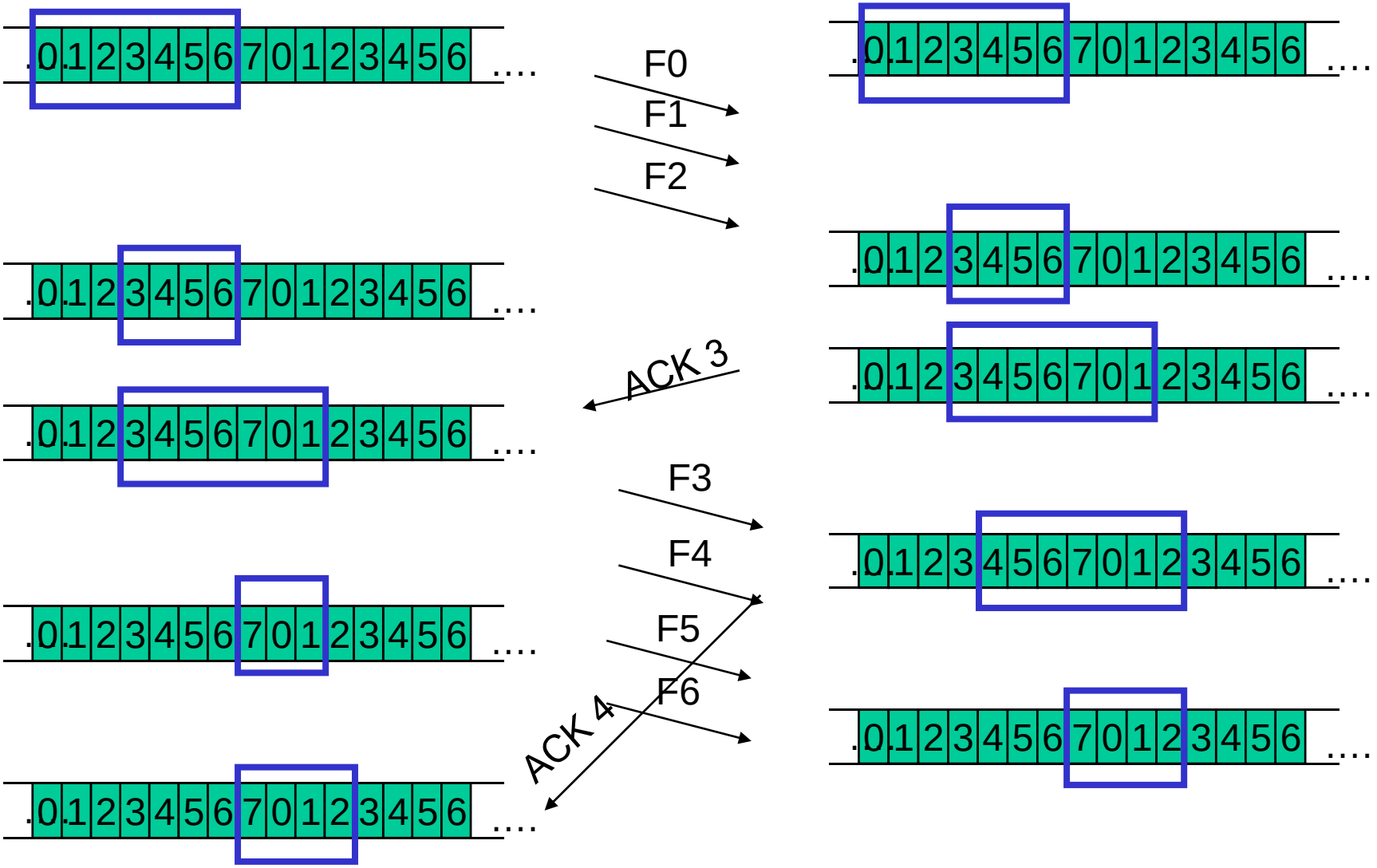
Figure 9.6 Example of a Sliding-Window Protocol

Example of sliding window

- Receiver has buffer W long
- Transmitter can send up to W frames without ACK
- Each frame is numbered (Sequence number)
- ACK includes number of next frame expected
- Sequence number bounded by k bits
  - frames are numbered modulo 2^k



Sliding Window: Example



Summary- Data Link Protocols

- **Is a set of specifications used to implement the data link layer – HDLC or Ethernet and LLC**
- **The remaining problem is that Data link protocols differ by message delineation, frame length, and frame field structure.**
- **Another fundamental difference is between asynchronous and synchronous transmission data link protocols.**