Data Communications and Networking Fourth Edition

Forouzan

Chapter 11

Data Link Control

11.1

11-2 FLOW AND ERROR CONTROL

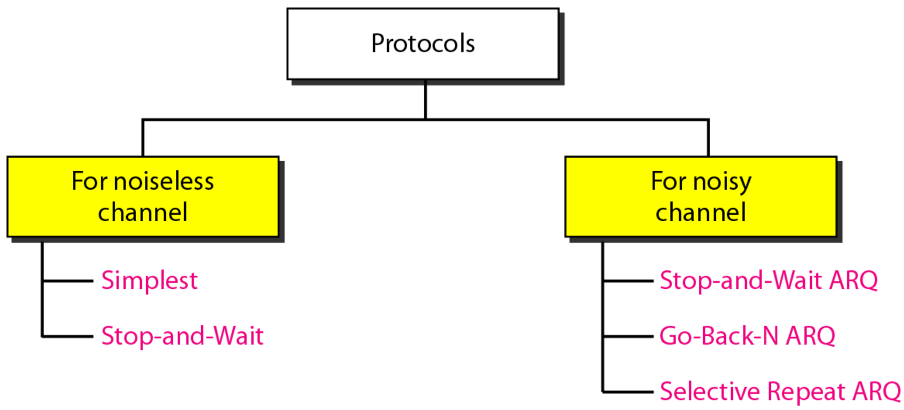
The most important responsibilities of the data link layer are flow control and error control. Collectively, these functions are known as data link control.

Topics discussed in this section:

Flow Control
Error Control

11.9

Figure 11.5 *Taxonomy of protocols discussed in this chapter*



11.13

11-4 NOISELESS CHANNELS

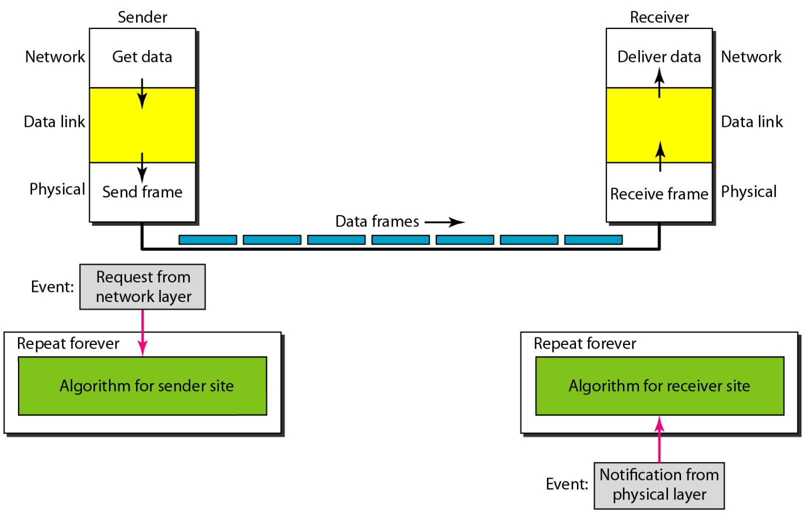
Let us first assume we have an ideal channel in which no frames are lost, duplicated, or corrupted. We introduce two protocols for this type of channel.

Topics discussed in this section:

Simplest Protocol
Stop-and-Wait Protocol

11.14

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



11.15

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

```
while(true)                        // Repeat forever
{
  WaitForEvent();                  // Sleep until an event occurs
  if(Event(RequestToSend))         //There is a packet to send
  {
     GetData();
     MakeFrame();
     SendFrame();                  //Send the frame
  }
}
```

11.16

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

```
while(true)                          // Repeat forever
{
  WaitForEvent();                    // Sleep until an event occurs
  if(Event(ArrivalNotification))     //Data frame arrived
  {
     ReceiveFrame();
     ExtractData();
     DeliverData();                  //Deliver data to network layer
  }
}
```


Figure 11.7 *Flow diagram for Example 11.1*

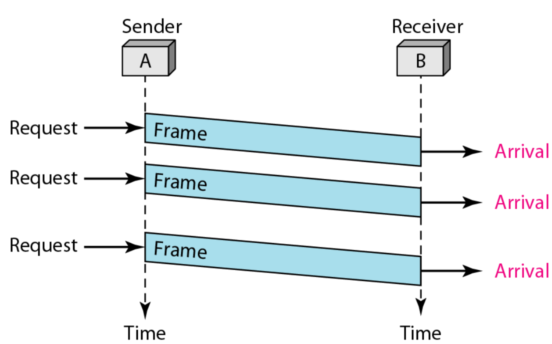
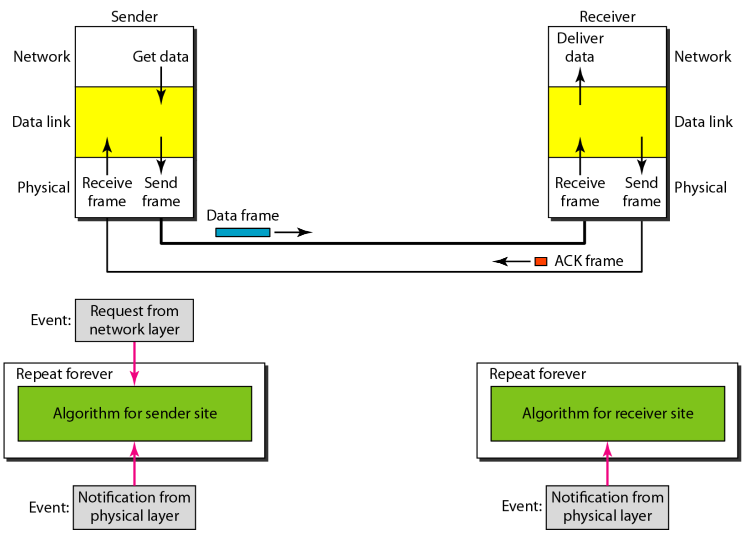


Figure 11.8 *Design of Stop-and-Wait Protocol*



11.20

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

```
while(true)                           //Repeat forever
canSend = true                        //Allow the first frame to go
{
  WaitForEvent();                     // Sleep until an event occurs
  if(Event(RequestToSend) AND canSend)
  {
     GetData();
     MakeFrame();
     SendFrame();                     //Send the data frame
     canSend = false;                 //Cannot send until ACK arrives
  }
  WaitForEvent();                     // Sleep until an event occurs
  if(Event(ArrivalNotification)       // An ACK has arrived
   {
     ReceiveFrame();                  //Receive the ACK frame
     canSend = true;
   }
}
```

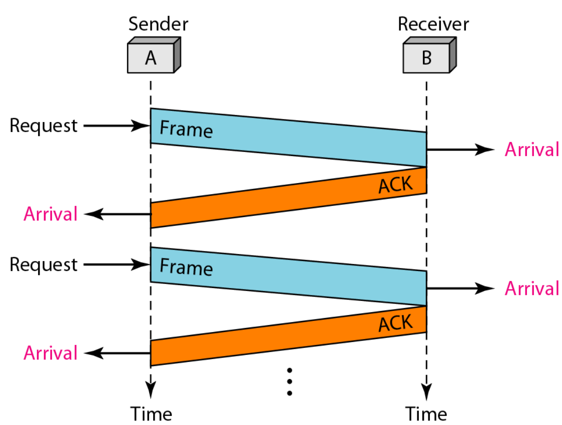
11.21

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

```
while(true)                          //Repeat forever
{
  WaitForEvent();                    // Sleep until an event occurs
  if(Event(ArrivalNotification))     //Data frame arrives
  {
     ReceiveFrame();
     ExtractData();
     Deliver(data);                  //Deliver data to network layer
     SendFrame();                    //Send an ACK frame
  }
}
```

11.22

Figure 11.9 *Flow diagram for Example 11.2*



11.24

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

11.25

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.

11.26

Note

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

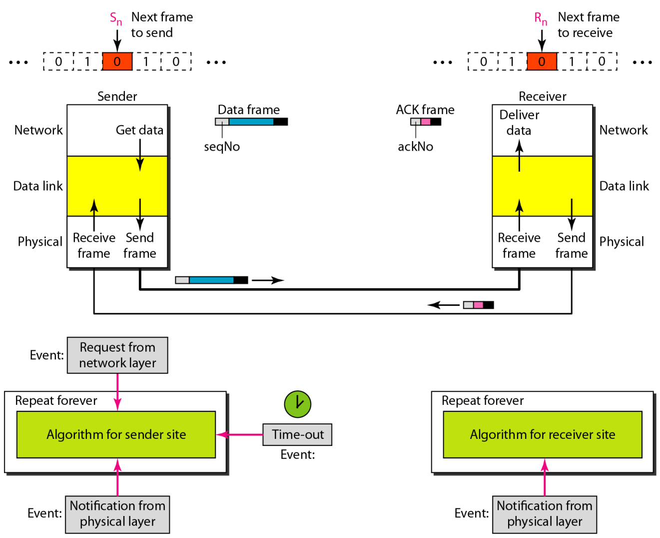
11.27

Note

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

11.28

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



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

```
S_n = 0;                          // Frame 0 should be sent first
canSend = true;                   // Allow the first request to go
while(true)                       // Repeat forever
{
  WaitForEvent();                 // Sleep until an event occurs
  if(Event(RequestToSend) AND canSend)
  {
     GetData();
     MakeFrame(S_n);                  //The seqNo is S_n
     StoreFrame(S_n);                 //Keep copy
     SendFrame(S_n);
     StartTimer();
     S_n = S_n + 1;
     canSend = false;
  }
  WaitForEvent();                     // Sleep
```

(continued)

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

```
    if(Event(ArrivalNotification)      // An ACK has arrived
    {
      ReceiveFrame(ackNo);             //Receive the ACK frame
      if(not corrupted AND ackNo == S_n) //Valid ACK
        {
          Stoptimer();
          PurgeFrame(S_{n-1});         //Copy is not needed
          canSend = true;
        }
     }

    if(Event(TimeOut)                  // The timer expired
    {
     StartTimer();
     ResendFrame(S_{n-1});             //Resend a copy check
    }
}
```

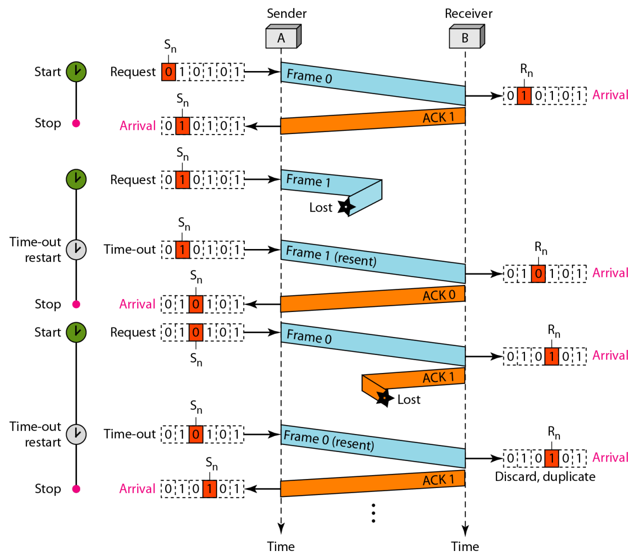
11.31

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

```
R_n = 0;                          // Frame 0 expected to arrive first
while(true)
{
  WaitForEvent();                 // Sleep until an event occurs
  if(Event(ArrivalNotification))  //Data frame arrives
  {
     ReceiveFrame();
     if(corrupted(frame));
        sleep();
     if(seqNo == R_n)             //Valid data frame
     {
      ExtractData();
       DeliverData();             //Deliver data
       R_n = R_n + 1;
     }
      SendFrame(R_n);             //Send an ACK
  }
}
```

11.32

Figure 11.11 *Flow diagram for Example 11.3*



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

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

Example 11.4 (continued)

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.

11.36

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.

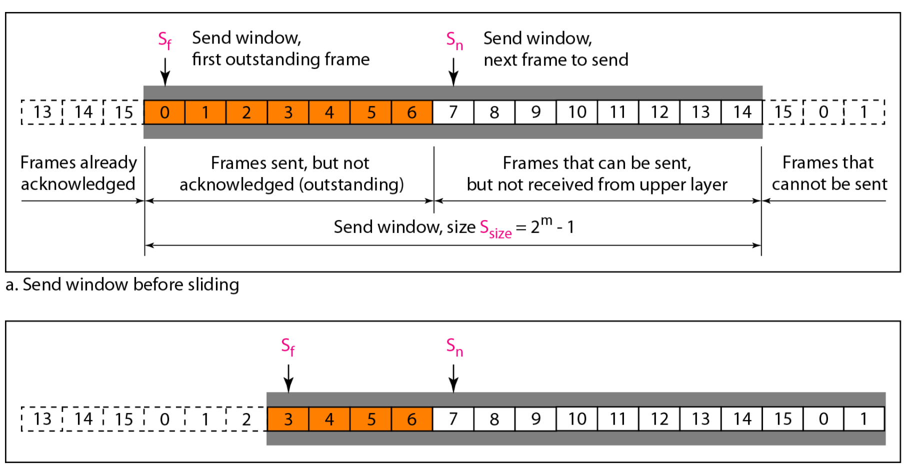
11.37

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.

11.38

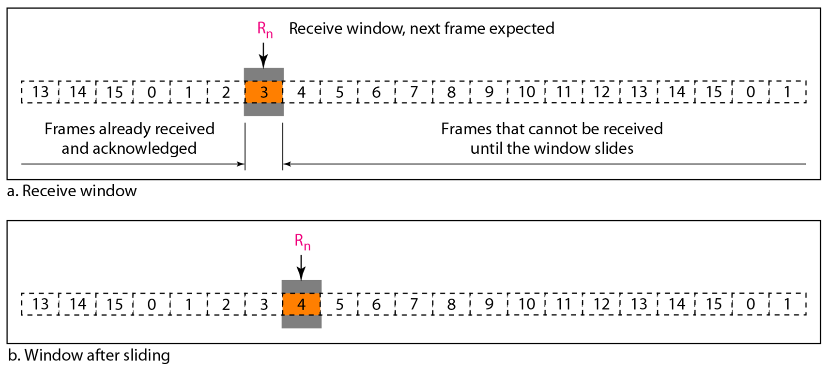
Figure 11.12 *Send window for Go-Back-N ARQ*



b. Send window after sliding

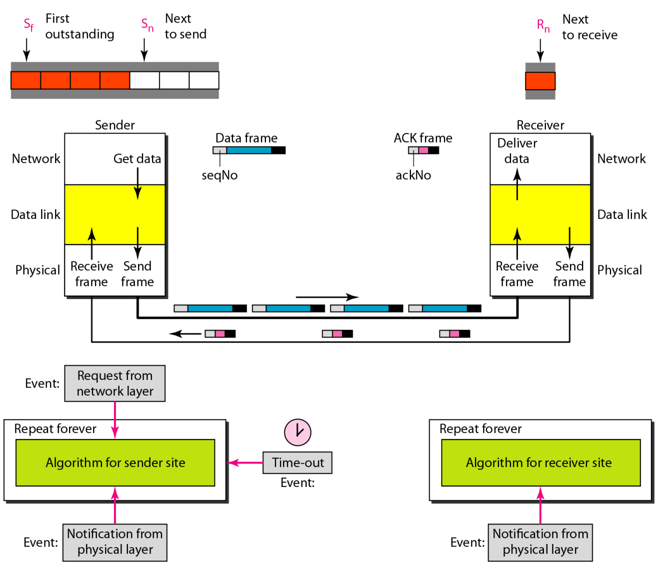
11.39

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



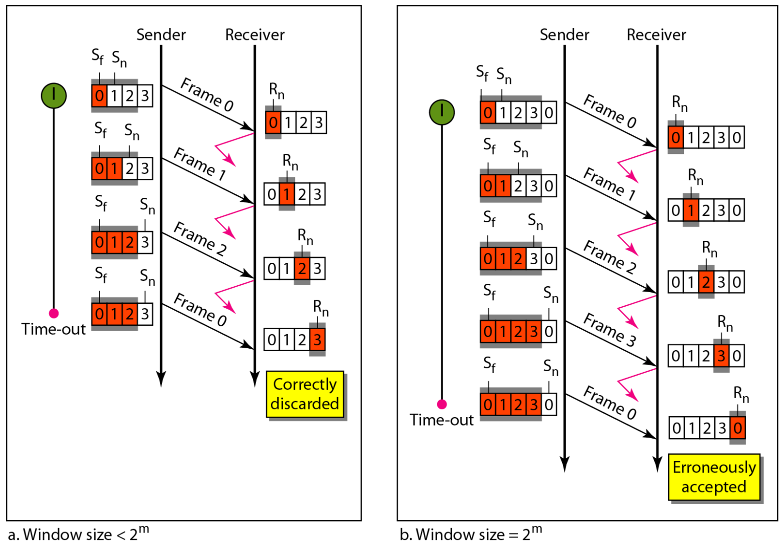
11.42

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



11.44

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



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*

```
S_w = 2^m - 1;
S_f = 0;
S_n = 0;

while (true)                             //Repeat forever
{
 WaitForEvent();
  if(Event(RequestToSend))               //A packet to send
  {
     if(S_n-S_f >= S_w)                  //If window is full
           Sleep();
     GetData();
     MakeFrame(S_n);
     StoreFrame(S_n);
     SendFrame(S_n);
     S_n = S_n + 1;
     if(timer not running)
          StartTimer();
  }

```

(continued)

11.47

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

```
  if(Event(ArrivalNotification))  //ACK arrives
  {
     Receive(ACK);
     if(corrupted(ACK))
          Sleep();
     if((ackNo>S_f)&&(ackNo<=S_n))  //If a valid ACK
     While(S_f <= ackNo)
      {
       PurgeFrame(S_f);
       S_f = S_f + 1;
      }
      StopTimer();
  }

  if(Event(TimeOut))                //The timer expires
  {
   StartTimer();
   Temp = S_f;
   while(Temp < S_n);
    {
     SendFrame(S_f);
     S_f = S_f + 1;
    }
  }
}
```

11.48

Algorithm 11.8 *Go-Back-N receiver algorithm*

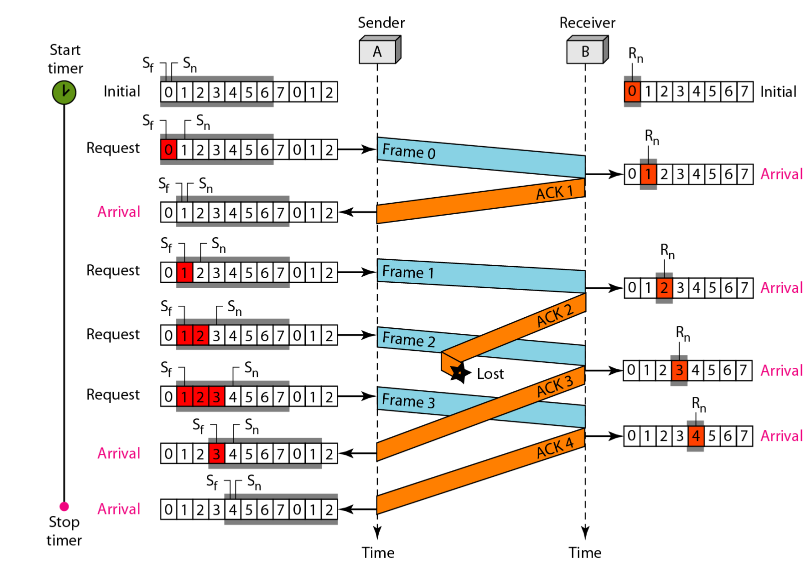
```
R_n = 0;

while (true)                          //Repeat forever
{
  WaitForEvent();

  if(Event(ArrivalNotification))  /Data frame arrives
  {
     Receive(Frame);
     if(corrupted(Frame))
          Sleep();
     if(seqNo == R_n)                  //If expected frame
     {
       DeliverData();                 //Deliver data
       R_n = R_n + 1;                 //Slide window
       SendACK(R_n);
     }
  }
}
```

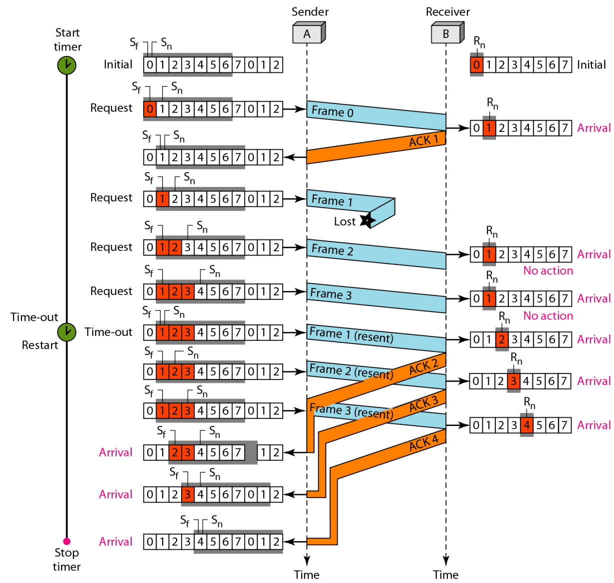
11.49

Figure 11.16 *Flow diagram for Example 11.6*



11.51

Figure 11.17 *Flow diagram for Example 11.7*



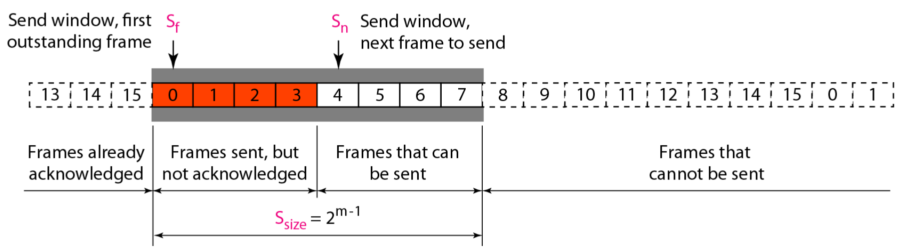
11.54

Note

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

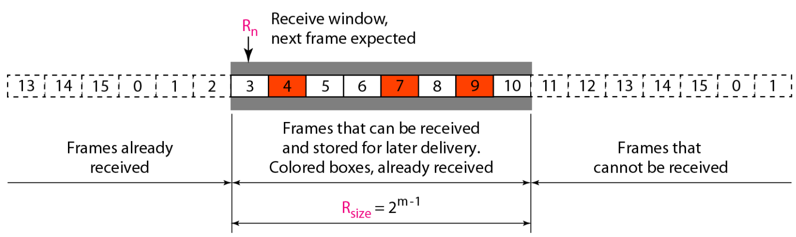
11.55

Figure 11.18 *Send window for Selective Repeat ARQ*



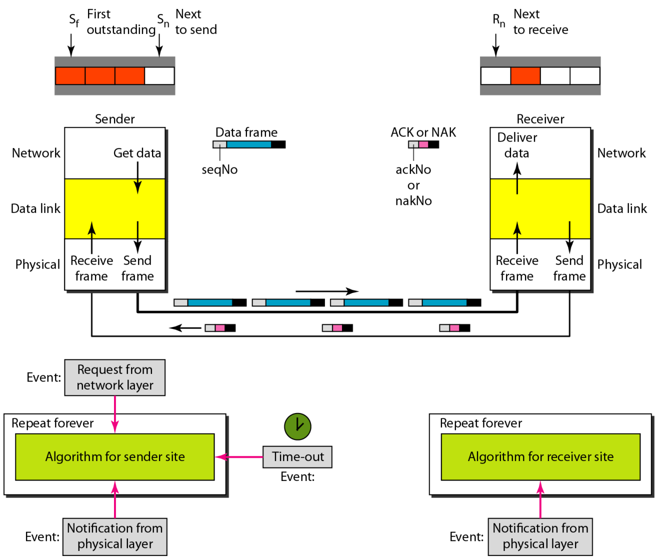
11.56

Figure 11.19 *Receive window for Selective Repeat ARQ*



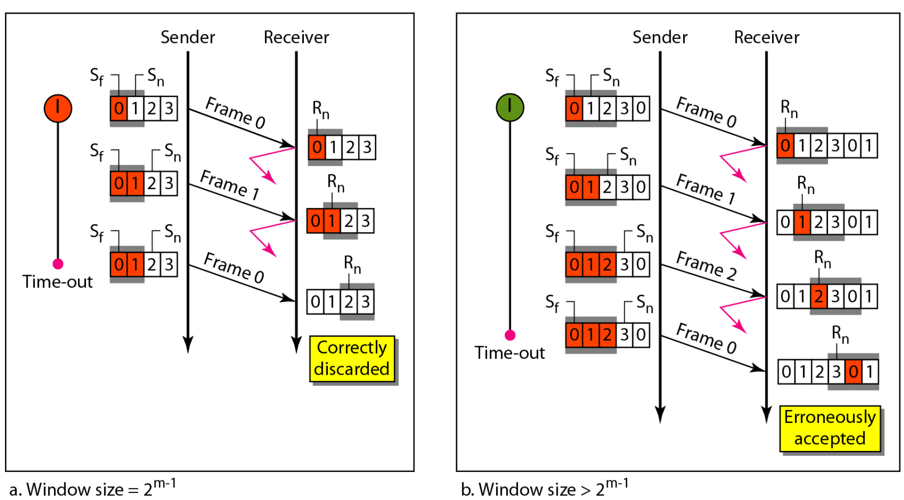
11.57

Figure 11.20 *Design of Selective Repeat ARQ*



11.58

Figure 11.21 *Selective Repeat ARQ, window size*



11.59

Note

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

11.60

Algorithm 11.9 *Sender-site Selective Repeat algorithm*

```
Sw = 2^(m-1) ;
Sf = 0;
Sn = 0;

while (true)                          //Repeat forever
{
  WaitForEvent();
  if(Event(RequestToSend))            //There is a packet to send
  {
     if(Sn-Sf >= Sw)                  //If window is full
            Sleep();
     GetData();
     MakeFrame(Sn);
     StoreFrame(Sn);
     SendFrame(Sn);
     Sn = Sn + 1;
     StartTimer(Sn);
  }

```

(continued)

11.61

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

```
   if(Event(ArrivalNotification)) //ACK arrives
   {
      Receive(frame);             //Receive ACK or NAK
      if(corrupted(frame))
           Sleep();
      if (FrameType == NAK)
         if (nakNo between Sf and Sn)
         {
          resend(nakNo);
          StartTimer(nakNo);
         }
      if (FrameType == ACK)
         if (ackNo between Sf and Sn)
         {
           while(sf < ackNo)
           {
            Purge(sf);
            StopTimer(sf);
            Sf = Sf + 1;
           }
         }
   }
```

(continued)

11.62

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

```

  if(Event(TimeOut(t)))          //The timer expires
  {
   StartTimer(t);
   SendFrame(t);
  }
}
```

11.63

Algorithm 11.10 *Receiver-site Selective Repeat algorithm*

```
Rn = 0;
NakSent = false;
AckNeeded = false;
Repeat(for all slots)
    Marked(slot) = false;

while (true)                                   //Repeat forever
{
  WaitForEvent();

  if(Event(ArrivalNotification))               /Data frame arrives
  {
     Receive(Frame);
     if(corrupted(Frame))&& (NOT NakSent)
     {
      SendNAK(Rn);
      NakSent = true;
      Sleep();
     }
     if(seqNo <> Rn)&& (NOT NakSent)
     {
      SendNAK(Rn);
```

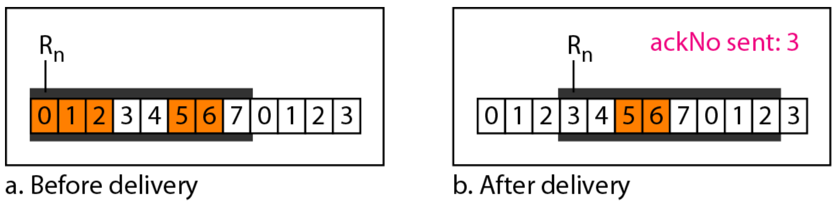
11.64

Algorithm 11.10 *Receiver-site Selective Repeat algorithm*

```
      NakSent = true;
      if ((seqNo in window)&&(!Marked(seqNo))
      {
       StoreFrame(seqNo)
       Marked(seqNo)= true;
       while(Marked(Rn))
       {
        DeliverData(Rn);
        Purge(Rn);
        Rn = Rn + 1;
        AckNeeded = true;
       }
        if(AckNeeded);
        {
        SendAck(Rn);
        AckNeeded = false;
        NakSent = false;
        }
      }
     }
  }
}
```

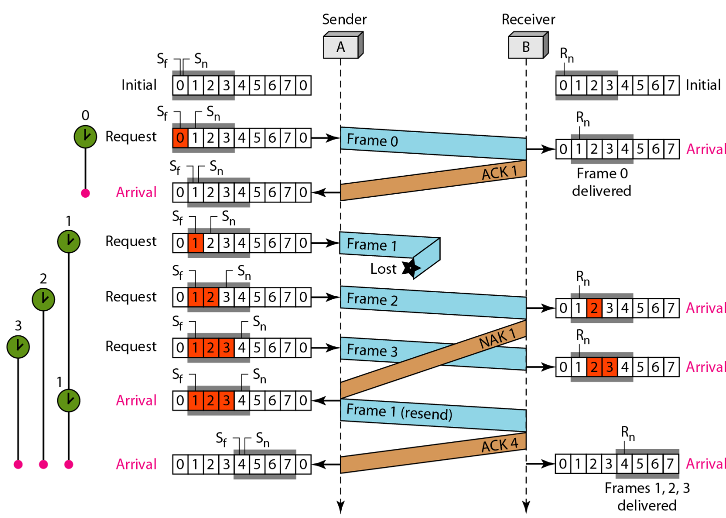
11.65

Figure 11.22 *Delivery of data in Selective Repeat ARQ*



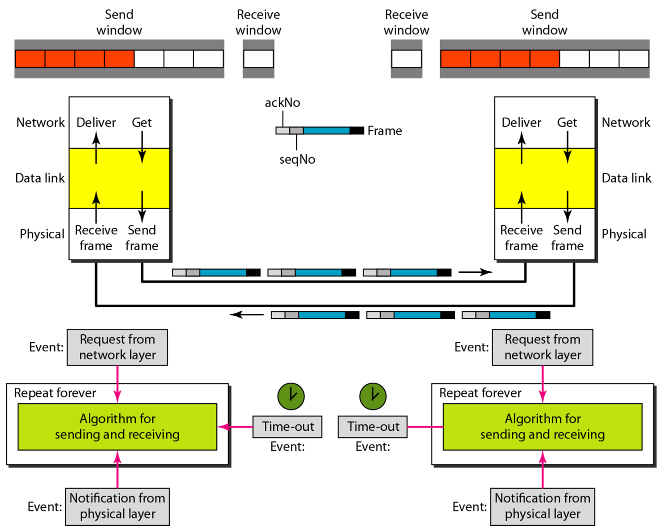
11.66

Figure 11.23 *Flow diagram for Example 11.8*



11.71

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



11.72