CHAPTER 11

Data Link Control

Solutions to Odd-Numbered Review Questions and Exercises

Review Questions

1. The two main functions of the data link layer are ***data link control*** and ***media access control***. Data link control deals with the design and procedures for communication between two adjacent nodes: node-to-node communication. Media access control deals with procedures for sharing the link.

3. In a ***byte-oriented protocol***, data to be carried are 8-bit characters from a coding system. Character-oriented protocols were popular when only text was exchanged by the data link layers. In a ***bit-oriented protocol***, the data section of a frame is a sequence of bits. Bit-oriented protocols are more popular today because we need to send text, graphic, audio, and video which can be better represented by a bit pattern than a sequence of characters.

5. ***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*** refers to a set of procedures used to detect and correct errors.

7. In this chapter, we discussed three protocols for noisy channels: the ***Stop-and-Wait ARQ***, the ***Go-Back-N ARQ***, and the ***Selective-Repeat ARQ***.

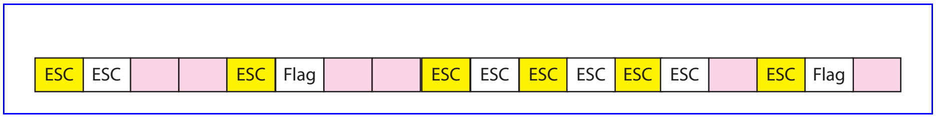
9. In the ***Go-Back-N ARQ Protocol***, we can send several frames before receiving acknowledgments. If a frame is lost or damaged, all outstanding frames sent before that frame are resent. In the ***Selective- Repeat ARQ protocol*** we avoid unnecessary transmission by sending only the frames that are corrupted or missing. Both Go-Back-*N* and Selective-Repeat Protocols use ***sliding windows***. In Go-Back-*N* ARQ, if m is the number of bits for the sequence number, then the size of the send window must be at most $\mathbf{2^m - 1}$; the size of the receiver window is always 1. In Selective-Repeat ARQ, the size of the sender and receiver window must be at most $\mathbf{2^{m-1}}$.

11. ***Piggybacking*** is used to improve the efficiency of bidirectional transmission. When a frame is carrying data from A to B, it can also carry control information about frames from B; when a frame is carrying data from B to A, it can also carry control information about frames from A.

Exercises

13. We give a very simple solution. Every time we encounter an ESC or flag character, we insert an extra ESC character in the data part of the frame (see Figure 11.1).

Figure 11.1 *Solution to Exercise 13*



15. We write two very simple algorithms. We assume that a frame is made of a one-byte beginning flag, variable-length data (possibly byte-stuffed), and a one-byte ending flag; we ignore the header and trailer. We also assume that there is no error during the transmission.
    a. Algorithm 11.1 can be used at the sender site. It inserts one ESC character whenever a flag or ESC character is encountered.

Algorithm 11.1 *Sender's site solution to Exercise 15*

```
InsertFrame (one-byte flag);        // Insert beginning flag
while (more characters in data buffer)
{
    ExtractBuffer (character);
    if (character is flag or ESC)    InsertFrame (ESC);     // Byte stuff
    InsertFrame (character);
}
InsertFrame (one-byte flag);      // Insert ending flag
```

    b. Algorithm 11.2 can be used at the receiver site.

Algorithm 11.2 *Receiver's site solution to Exercise 15*

```
ExtractFrame (character);     // Extract beginning flag
Discard (character);          // Discard beginning flag
while (more characters in the frame)
{
    ExtractFrame (character);
    if (character = = flag)   exit();          // Ending flag is extracted

    if (character = = ESC)
    {
        Discard (character);        // Un-stuff
        ExtractFrame (character);        // Extract flag or ESC as data
    }
    InsertBuffer (character);
}
Discard (character);          // Discard ending flag
```

17. A five-bit sequence number can create sequence numbers from 0 to 31. The sequence number in the Nth packet is (N mod 32). This means that the 101th packet has the sequence number (101 mod 32) or **5**.

19. See Algorithm 11.3. Note that we have assumed that both events (request and arrival) have the same priority.

Algorithm 11.3 *Algorithm for bidirectional 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
    }

    if (Event (ArrivalNotification))     // Data frame arrived
    {
        ReceiveFrame ();
        ExtractData ();
        DeliverData ();     // Deliver data to network layer
    }
}  // End Repeat forever
```

21. Algorithm 11.4 shows one design. This is a very simple implementation in which we assume that both sites always have data to send.

Algorithm 11.4 *A bidirectional algorithm for Stop-And-Wait ARQ*

```
Sn = 0;    // Frame 0 should be sent first
Rn = 0;    // Frame 0 expected to arrive first
canSend = true;    // Allow the first request to go
while (true)     // Repeat forever
{
    WaitForEvent ();     // Sleep until an event occurs
    if (Event (RequestToSend) AND canSend)    // Packet to send
    {
        GetData ();
        MakeFrame (Sn , Rn);   // The seqNo of frame is Sn
        StoreFrame (Sn , Rn);    //Keep copy for possible resending
        SendFrame (Sn , Rn);
        StartTimer ();
        Sn = (Sn + 1) mod 2;
        canSend = false;
    }

    if (Event (ArrivalNotification))     // Data frame arrives
    {
        ReceiveFrame ();
        if (corrupted (frame))    sleep();
        if (seqNo = = Rn)     // Valid data frame
        {
            ExtractData ();
            DeliverData ();     // Deliver data
            Rn = (Rn + 1) mod 2;
        }
        if (ackNo = = Sn)     // Valid ACK
```

Algorithm 11.4 *A bidirectional algorithm for Stop-And-Wait ARQ*

```
        {
            StopTimer ();
            PurgeFrame (Sn−1 , Rn−1);    //Copy is not needed
            canSend = true;
        }
    }

   if (Event(TimeOut))   //  The timer expired
   {
     StartTimer ();
     ResendFrame (Sn-1 , Rn-1);     // Resend a copy
   }
}    // End Repeat forever
```

23. Algorithm 11.5 shows one design. This is a very simple implementation in which we assume that both sites always have data to send.

Algorithm 11.5 *A bidirectional algorithm for Selective-Repeat ARQ*

```
Sw = 2^(m−1);
Sf = 0;
Sn = 0;
Rn = 0;
NakSent = false;
AckNeeded = false;
Repeat (for all slots);
Marked (slot) = false;
while (true)    // Repeat forever
{
    WaitForEvent ();
    if (Event (RequestToSend))     // There is a packet to send
    {
        if (Sn–Sf >= Sw)    Sleep ();    // If window is full
        GetData ();
        MakeFrame (Sn , Rn);
        StoreFrame (Sn , Rn);
        SendFrame (Sn , Rn);
        Sn = Sn + 1;
        StartTimer (Sn);
    }

    if (Event (ArrivalNotification))
    {
        Receive (frame);   // Receive Data or NAK
        if (FrameType is NAK)
        {
            if (corrupted (frame))      Sleep();
            if (nakNo between Sf and Sn)
            {
                resend (nakNo);
                StartTimer (nakNo);
            }
        }
```

Algorithm 11.5 *A bidirectional algorithm for Selective-Repeat ARQ*

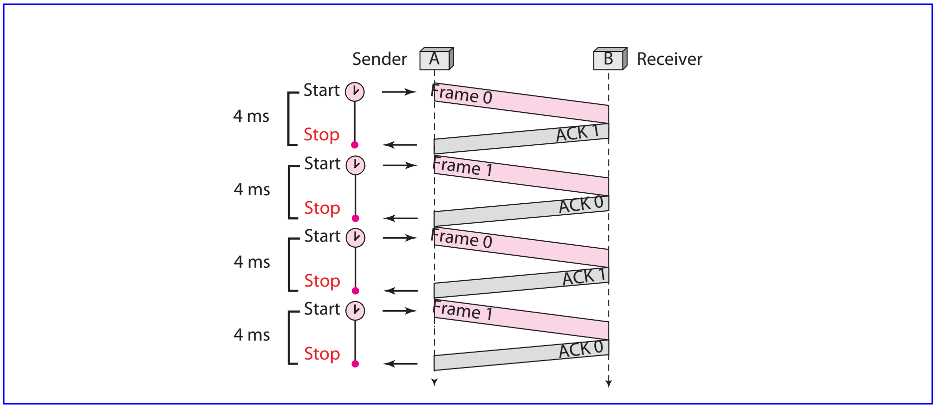
```
        if (FrameType is Data)
        {
            if (corrupted (Frame)) AND (NOT NakSent)
            {
                SendNAK (Rn);
                NakSent = true;
                Sleep();
            }

            if (ackNo between Sf and Sn)
            {
                while (Sf < ackNo)
                {
                    Purge (Sf);
                    StopTimer (Sf);
                    Sf = Sf + 1;
                }
            }

            if ((seqNo <> Rn) AND (NOT NakSent))
            {
                SendNAK (Rn);
                NakSent = true;
            }

            if ((seqNo in window) AND (NOT Marked (seqNo))
            {
                StoreFrame (seqNo);
                Marked (seqNo) = true;
                while (Marked (Rn))
                {
                    DeliverData (Rn);
                    Purge (Rn);
                    Rn = Rn + 1;
                   AckNeeded = true;
                }
            }
        }   // End if (FrameType is Data)
      }   // End if (arrival event)

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

25. State $R_n = 0$ means the receiver is waiting for Frame 0. State $R_n = 1$ means the receiver is waiting for Frame 1. We can then say

Event A:	**Receiver Site:**	Frame 0 received.
Event B:	**Receiver Site:**	Frame 1 received.

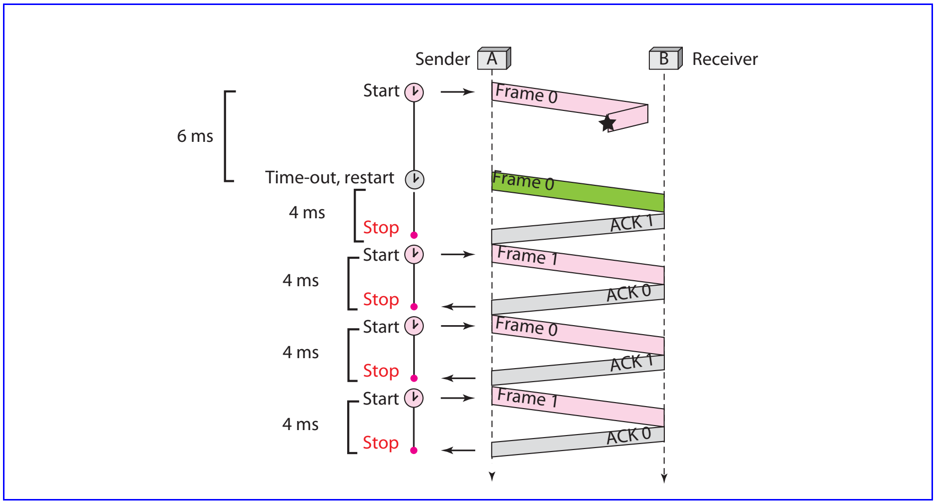
27. Figure 11.2 shows the situation. Since there are no lost or damaged frames and the round trip time is less than the time-out, each frame is sent only once.

Figure 11.2 *Solution to Exercise 27*



29. Figure 11.3 shows the situation. In this case, only the first frame is resent; the acknowledgment for other frames arrived on time.

Figure 11.3 *Solution to Exercise 29*



31. In the worst case, we send the a full window of size 7 and then wait for the acknowledgment of the whole window. We need to send 1000/7 ≈ 143 windows. We ignore the overhead due to the header and trailer.

Transmission time for one window = 7000 bits / 1,000,000 bits = 7 ms
Data frame trip time = 5000 km / 200,000 km = 25 ms
ACK transmission time = 0 (It is usually negligible)
ACK trip time = 5000 km / 200,000 km = 25 ms

Delay for 1 window = 7 + 25 + 25 = 57 ms.
Total delay = 143 × 57 ms = **8.151 s**