CHAPTER 11 (corrisponde al cap. 10 italiano)

Data Link Control

Solutions to 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.
2. The data link layer needs to pack bits into ***frames***. Framing divides a message into smaller entities to make flow and error control more manageable.
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.
4. Character-oriented protocols use ***byte-stuffing*** to be able to carry an 8-bit pattern that is the same as the flag. Byte-stuffing adds an extra character to the data section of the frame to escape the flag-like pattern. Bit-oriented protocols use ***bit-stuffing*** to be able to carry patterns similar to the flag. Bit-stuffing adds an extra bit to the data section of the frame whenever a sequence of bits is similar to the flag.
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.
6. In this chapter, we discussed two protocols for noiseless channels: the ***Simplest*** and the ***Stop-and-Wait***.
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***.
8. Go-Back-N ARQ is more efficient than Stop-and-Wait ARQ. The second uses ***pipelining***, the first does not. In the first, we need to wait for an acknowledgment for each frame before sending the next one. In the second we can send several frames before receiving an acknowledgment.

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 $2^m - 1$; 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 2^{m-1}.
10. ***HDLC*** is a *bit-oriented protocol* for communication over point-to-point and multipoint links. ***PPP*** is a byte-oriented protocol used for point-to-point links.
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.
12. Only ***Go-Back-N*** and ***Selective-Repeat*** protocols use ***pipelining***.

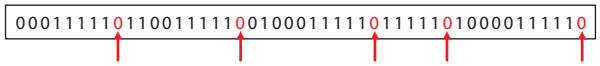
Exercises

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*

14. Figure 11.2 shows data to be encapsulated in the frame.

Figure 11.2 *Solution to exercise 14*



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)
{
```

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

```
    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
```

16. We write two very simple algorithms. We assume that a frame is made of an 8-bit flag (01111110), variable-length data (possibly bit-stuffed), and an 8-bit ending flag (0111110); we ignore header and trailer. We also assume that there is no error during the transmission.

    a. Algorithm 11.3 can be used at the sender site.

Algorithm 11.3 *Sender's site solution to Exercise 16*

```
InsertFrame (8-bit flag);   // Insert beginning flag
counter = 0;
while (more bits in data buffer)
{
    ExtractBuffer (bit);
    InsertFrame (bit);
    if (bit = = 1)      counter = counter + 1;
    else                counter = 0;

    if (counter = = 5)
    {
        InsertFrame (bit 0);   // Bit stuff
        counter = 0;
    }
}
InsertFrame (8-bit flag);   // Insert ending flag
```

b. Algorithm 11.4 can be used at the receiver's site. Note that when the algorithm exits from the loop, there are six bits of the ending flag in the buffer, which need to be removed after the loop.

Algorithm 11.4 *Receiver's site solution to Exercise 16*

```
ExtractFrame (8 bits);    // Extract beginning flag
counter = 0;
while (more bits in frame)
{
    ExtractFrame (bit);
    if (counter = = 5)
    {
        if (bit is 0)    Discard (bit);      counter = 0;   // Un-stuff
        if (bit is 1)    exit ();    // Flag is encountered
    }

    if (counter < 5)
    {
        if (bit is 0)   counter = 0;
        if (bit is 1)   counter = counter + 1;
        InsertBuffer (bit);
    }
}
ExtractBuffer (last 6 bits);    // These bits are part of flag
Discard (6 bits);
```

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**.

18.

Stop-And-Wait ARQ	send window = **1**	receive window = **1**
Go-Back-N ARQ	send window = $2^5 - 1$ = **31**	receive window = **1**
Selective-Repeat ARQ	send window = 2^4 = **16**	receive window = **16**

19. See Algorithm 11.5. Note that we have assumed that both events (request and arrival) have the same priority.

Algorithm 11.5 *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 ();
```

Algorithm 11.5 *Algorithm for bidirectional Simplest Protocol*

```
        ExtractData ();
        DeliverData ();     // Deliver data to network layer
    }
}  // End Repeat forever
```

20. See Algorithm 11.6. Note that in this algorithm, we assume that the arrival of a frame by a site also means the acknowledgment of the previous frame sent by the same site.

Algorithm 11.6

```
while (true)     // Repeat forever
{
    canSend = true;
    WaitForEvent ();    // Sleep until an event occurs
    if (Event (RequestToSend) AND canSend)    // A packet can be sent
    {
        GetData ();
        MakeFrame ();
        SendFrame ();     // Send the frame
        canSend = false;
    }

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

21. Algorithm 11.7 shows one design. This is a very simple implementation in which we assume that both sites always have data to send.

Algorithm 11.7 *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;
    }
```

Algorithm 11.7 *A bidirectional algorithm for Stop-And-Wait ARQ*

```
    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
        {
            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
```

22. Algorithm 11.8 shows one design. This is a very simple implementation in which we assume that both sites always have data to send.

Algorithm 11.8 *Bidirectional algorithm for Go-Back-And-N algorithm*

```
Sw = 2m – 1;
Sf = 0;
Sn = 0;
Rn = 0;
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;
        if (timer not running)    StartTimer ();
    }

    if (Event (ArrivalNotification))
    {
        Receive (Frame);
        if (corrupted (ACK))      Sleep();
        if ((ackNo > Sf) AND (ackNo <= Sn))   // If a valid ACK
        {
```

Algorithm 11.8 *Bidirectional algorithm for Go-Back-And-N algorithm*

```
            while (Sf <= ackNo)
            {
                PurgeFrame (Sf);
                Sf = Sf + 1;
            }
            StopTimer ();
        }
        if (seqNo = = Rn)   // If expected frame
        {
            DeliverData ();    // Deliver data
            Rn = Rn + 1;    // Slide window one slot
            SendACK (Rn);
        }
    }

    if (Event (TimeOut))    // The timer expires
    {
        StartTimer ();
        Temp = Sf;
        while (Temp < Sn);
        {
            SendFrame (Sf);
            Sf = Sf + 1;
        }
    }
}   // End Repeat forever
```

23. Algorithm 11.9 shows one design. This is a very simple implementation in which we assume that both sites always have data to send.

Algorithm 11.9 *A bidirectional algorithm for Selective-Repeat ARQ*

```
Sw = 2m–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);
    }
```

Algorithm 11.9 *A bidirectional algorithm for Selective-Repeat ARQ*

```
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);
        }
    }

    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
```

Algorithm 11.9 *A bidirectional algorithm for Selective-Repeat ARQ*

```
    {
        StartTimer (t);
        SendFrame (t);
    }
} // End Repeat forever
```

24. State $S_n = 0$ means the sender has sent Frame 1, but is waiting for the acknowledgment. State $S_n = 1$ means the sender has sent Frame 0, but is waiting for the acknowledgment. We can then say

Event A:	**Sender Site:**	ACK 0 received.
Event B:	**Sender Site:**	ACK 1 received.

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

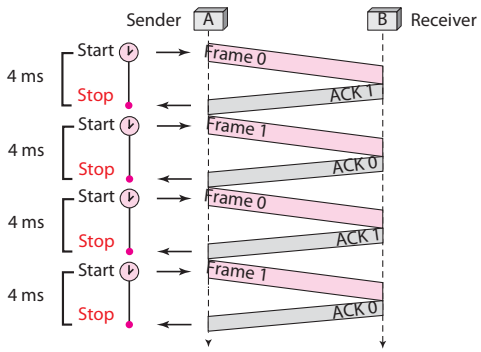
Event A:	**Receiver Site:**	Frame 0 received.
Event B:	**Receiver Site:**	Frame 1 received.

26. We can say that in this case, each state defines that a frame or an acknowledgment in transit. In other words,
$(1 , 0) \rightarrow$ Frame 0 is in transit $\quad (1 , 1) \rightarrow$ ACK 1 is in transit
$(0 , 1) \rightarrow$ Frame 1 is in transit $\quad (0 , 0) \rightarrow$ ACK 0 is in transit

Event A:	**Receiver Site:**	Frame 0 arrives and ACK 1 is sent.
Event B:	**Sender Site:**	ACK 1 arrives and Frame 1 is sent.
Event C:	**Receiver Site:**	Frame 1 arrives and ACK 0 is sent.
Event D:	**Sender Site:**	ACK 0 arrives and Frame 0 is sent.

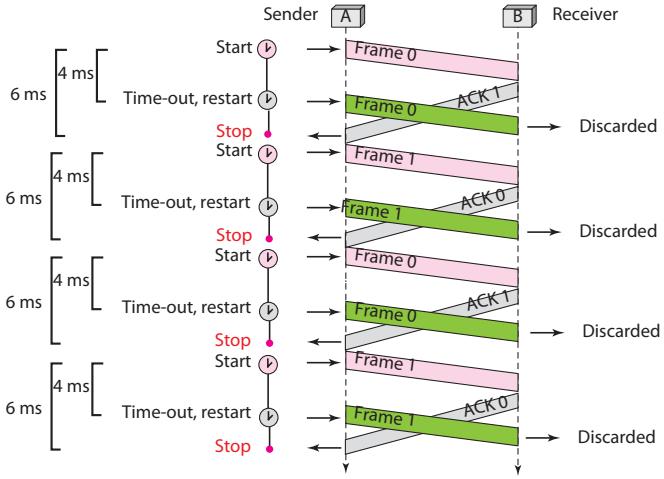
27. Figure 11.3 shows the situation. Since there are no lost or damaged frames and the round trip time is less than the time-out, each frame is sent only once.

Figure 11.3 *Solution to Exercise 27*



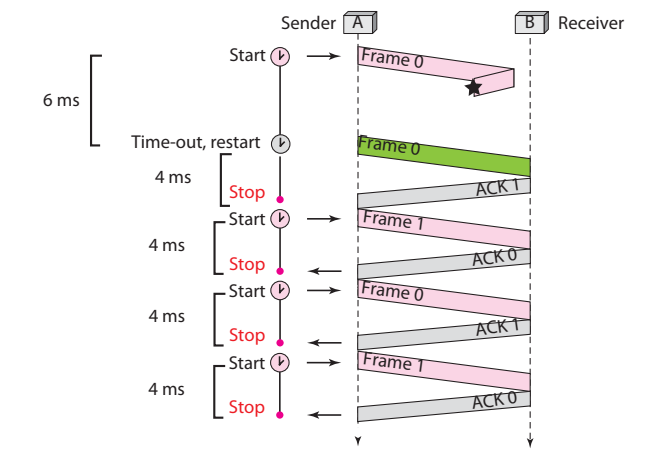
28. Figure 11.4 shows the situation. Here, we have a special situation. Although no frame is damaged or lost, the sender sends each frame twice. The reason is that the the acknowledgement for each frame reaches the sender after its timer expires. The sender thinks that the frame is lost.

Figure 11.4 *Solution to Exercise 28*



29. Figure 11.5 shows the situation. In this case, only the first frame is resent; the acknowledgment for other frames arrived on time.

Figure 11.5 *Solution to Exercise 29*



30. We need to send 1000 frames. We ignore the overhead due to the header and trailer.

    Data frame Transmission time = 1000 bits / 1,000,000 bits = 1 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 frame = 1 + 25 + 25 = 51 ms.
    Total delay = 1000 × 51 = 51 s

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**

32. In the worst case, we send the a full window of size 4 and then wait for the acknowledgment of the whole window. We need to send 1000/4 = 250 windows. We ignore the overhead due to the header and trailer.

    Transmission time for one window = 4000 bits / 1,000,000 bits = 4 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 = 4 + 25 + 25 = 54 ms.
    Total delay = 250 × 54 ms = **13.5 s**