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
11-4 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

```java
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
    }
}
```
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();
7.             ExtractData();
8.             DeliverData();       // Deliver data to network layer
9.         }
10. }

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 Protocol

```java
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(AssignmentNotification) // An ACK has arrived
            { 
                ReceiveFrame(); // Receive the ACK frame
                canSend = true;
            }
    }
```

11.20

11.21
Algorithm 11.4  Receiver-site algorithm for Stop-and-Wait Protocol

```c
1 while(true) { //Repeat forever
2   WaitForEvent(); // Sleep until an event occurs
3   if(Event(ArrivalNotification)) { //Data frame arrives
4       ReceiveFrame();
5       ExtractData();
6       Deliver(data); //Deliver data to network layer
7       SendFrame(); //Send an ACK frame
8   }
9 }
10 }
```

Figure 11.9  Flow diagram for Example 11.2
11.25

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

11.26

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

In Stop-and-Wait ARQ, the acknowledgment number always announces in modulo-2 arithmetic the sequence number of the next frame expected.
Algorithm 11.5  Sender-site algorithm for Stop-and-Wait ARQ

```c
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();           //The seqNo is Sn
9        MakeFrame(Sn);     //Keep copy
10       StoreFrame(Sn);     //Keep copy
11       SendFrame(Sn);      //Keep copy
12       StartTimer();       //Sleep
13       Sn = Sn + 1;
14       canSend = false;
15    }
16   WaitForEvent();          // Sleep
(continued)
```
Algorithm 11.5  Sender-site algorithm for Stop-and-Wait ARQ  (continued)

```plaintext
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
    }
```

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

```plaintext
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
            }
    }
```
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

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

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.
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

<table>
<thead>
<tr>
<th>$S_t$</th>
<th>$S_n$</th>
</tr>
</thead>
<tbody>
<tr>
<td>Send window, first outstanding frame</td>
<td>Send window, next frame to send</td>
</tr>
</tbody>
</table>

Frames already acknowledged | Frames sent, but not acknowledged (outstanding) | Frames that can be sent, but not received from upper layer | Frames that cannot be sent

Send window, size $S_{size} = 2^m - 1$

a. Send window before sliding

<table>
<thead>
<tr>
<th>$S_t$</th>
<th>$S_n$</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>1</td>
</tr>
</tbody>
</table>

b. Send window after sliding
Figure 11.13  Receive window for Go-Back-N ARQ

a. Receive window

b. Window after sliding

Figure 11.14  Design of Go-Back-N ARQ
Figure 11.15 Window size for Go-Back-N ARQ

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.

Note
Algorithm 11.7  Go-Back-N sender algorithm

(continued)

Algorithm 11.7  Go-Back-N sender algorithm

(continued)
Algorithm 11.8  Go-Back-N receiver algorithm

1. \( R_n = 0; \)
2. \[
\text{while (true) } //\text{Repeat forever} \\
\text{WaitForEvent();} \\
\]
3. \[
\text{if(Event(ArrivalNotification)) } //\text{Data frame arrives} \\
\text{Receive(Frame);} \\
\text{if(corrupted(Frame))} \\
\text{Sleep();} \\
\text{if(seqNo == R_n) } //\text{If expected frame} \\
\text{DeliverData(); } //\text{Deliver data} \\
\text{R_n = R_n + 1; } //\text{Slide window} \\
\text{SendACK(R_n);} \\
\]

Figure 11.16  Flow diagram for Example 11.6
Stop-and-Wait ARQ is a special case of Go-Back-N ARQ in which the size of the send window is 1.

**Note**
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

Figure 11.21  Selective Repeat ARQ, window size

11.58

11.59
In Selective Repeat ARQ, the size of the sender and receiver window must be at most one-half of $2^m$.
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(s_f < ackNo)
35                  {  
36                      Purge(s_f);
37                      StopTimer(s_f);
38                      S_f = S_f + 1;
39                  }
40          }
41 }

(continued)

Algorithm 11.9  Sender-site Selective Repeat algorithm  (continued)

42 if(Event(TimeOut(t)))  //The timer expires
43 {  
44    StartTimer(t);
45    SendFrame(t);
46 }
47
(continued)
Algorithm 11.10  Receiver-site Selective Repeat algorithm

1. $R_n = 0$;
2. NakSent = false;
3. AckNeeded = false;
4. Repeat(for all slots)
   5. Marked(slot) = false;
   6. while (true) //Repeat forever
      7. WaitForEvent();
      8. if(Event(ArrivalNotification)) /Data frame arrives
      9.   Receive(Frame);
     10.   if(corrupted(Frame))&& (NOT NakSent)
     11.      { SendNAK($R_n$);
     12.         NakSent = true;
     13.         Sleep();
     14.      }
     15.     if(seqNo <> $R_n$)& (NOT NakSent)
     16.     { SendNAK($R_n$);
     17.     }

Algorithm 11.10  Receiver-site Selective Repeat algorithm

23. NakSent = true;
24. if (((seqNo in window) && (!Marked(seqNo))
25.      { StoreFrame(seqNo)
26.        Marked(seqNo) = true;
27.        while(Marked($R_n$))
28.            { DeliverData($R_n$);
29.               Purge($R_n$);
30.               $R_n$ = $R_n$ + 1;
31.               AckNeeded = true;
32.            }
33.        if(AckNeeded);
34.            { SendAck($R_n$);
35.               AckNeeded = false;
36.               NakSent = false;
37.            }
38.        }
39.     }
40. }
Figure 11.22  Delivery of data in Selective Repeat ARQ

a. Before delivery

b. After delivery

Figure 11.23  Flow diagram for Example 11.8
Figure 11.24  Design of piggybacking in Go-Back-N ARQ