Data Link Layer Protocol

Eva María Urbano González

November 8, 2016

Contents

List of Figures

List of Tables

Chapter 1

DLL

1.1 Functions of the DLL

The Data-Link layer is the protocol layer in a program that handles the moving of data in and out across a physical link in a network. The Data-Link layer is layer 2 in the Open Systems Interconnect (OSI) model for a set of telecommunication protocols. According to the IEEE-802 LAN standards, the DLL can be divided into two sublayers:

- Logical Link Control (LLC): Deals with protocols, flow-control, and error control.
- Media Access Control (MAC): Deals with actual control of the media.

The DLL is responsible for converting data stream to signals bit by bit and to sent that over the underlying hardware.At the receiving end, DLL picks up data from hardware which are in the form of electrical signals, assembles them in a recognizable frame format, and hands over to upper layer. The DLL also ensures that an initial connection has been set up, divides output data into data frames, and handles the acknowledgements from a receiver that the data arrived successfully. It also ensures that incoming data has been received successfully by analyzing bit patterns at special places in the frames. The specific functions of the DLL are explained in the following lines.

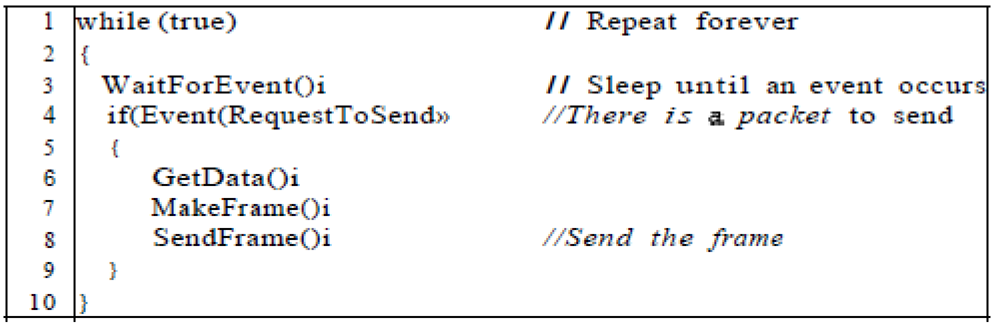
- **Framing**: Data-link layer takes packets from Network Layer and encapsulates them into Frames.Then, it sends each frame bit-by-bit on the hardware. At receiver end, data link layer picks up signals from hardware and assembles them into frames.
- **Adressing**: Each device on a network has a unique number, usually called a hardware address or MAC address, that is used by the data link layer protocol to ensure that data intended for a specific machine gets to it properly.
- **Synchronization**: When data frames are sent on the link, both machines must be synchronized in order to transfer to take place.
- **Error control**: Sometimes signals may have encountered problem in transition and the bits are flipped. These errors are detected and attempted to recover actual data bits.
- **Flow control**: Stations on same link may have different speed or capacity. Data-link layer ensures flow control that enables both machine to exchange data on same speed.

1.2 Protocols

In the previous section, the functions of the DLL have been determined. Now, the way it is achieved will be exposed. To do so, a list of possible protocols from the simplest one to the more complex will explained. With all the protocols explained, the one that best fits the needs of the missions will be selected. All the images have been extracted from [1].

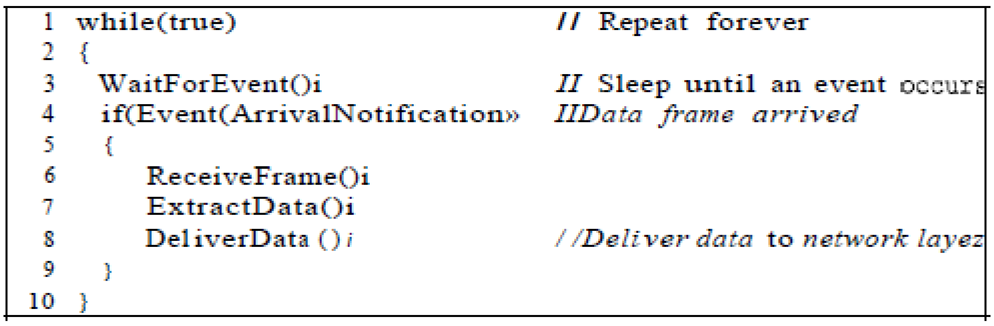
1.2.1 Simplest Protocol

This protocol has no error or flow control. It is supposed that the frames are traveling only in one direction, from the sender to the receiver. It is also supposed that the receiver can immediately handle the frames received, so there is no overwhelming. The DLL of the sender site gets data from its network layer, makes a frame out of the data and sends it. The DLL at the receiver site receives a frame from its physical layer, extracts data from the frame and delivers the data to its network layer. The problem here is that the sender site cannot send a frame until its network layer has a data packet to send and the receiver site cannot deliver a data packet to its network layer until a frame arrives. There is the need to introduce the idea of events in the protocol. The procedure at the sender site is constantly running; there is no action until there is a request from the network layer. The procedure at the receiver site is also constantly running, but there is no action until notification from the physical layer arrives.



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

Figure 1.1: Sender algorithm for the simplest protocol.

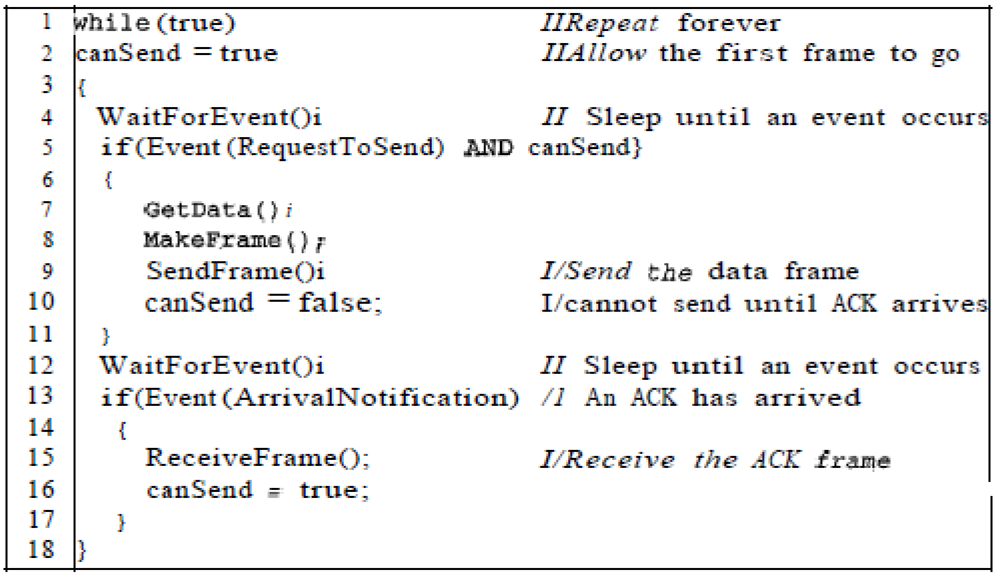


```
while(true)                         // Repeat forever
{
  WaitForEvent()i                   II Sleep until an event occurs
  if(Event(ArrivalNotification»     IIData frame arrived
  {
      ReceiveFrame()i
      ExtractData()i
      DeliverData ()i               //Deliver data to network layez
  }
}
```

Figure 1.2: Receiver algorithm for the simplest protocol.

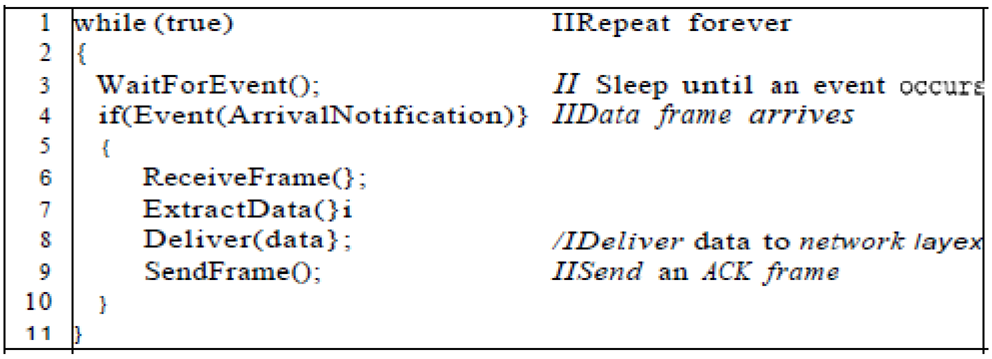
1.2.2 Stop-and-Wait Protocol

If data frames arrive at the receiver site faster than they can be processed, the frames must be stored until their use. Normally, the receiver does not have enough storage space, especially if it is receiving data from many sources. This may result in either the discarding of frames or denial of service. To prevent the receiver from becoming overwhelmed with frames,we somehow need to tell the sender to slow down. There must be feedback from the receiver to the sender. In the Stop-and-Wait Protocol the sends one frame, stops until it receives confirmation from the receiver and then sends the next frame. We still have unidirectional communication for data frames, but auxiliary ACK frames (simple tokens of acknowledgment) travel from the other direction. We add flow control to our previous protocol. In this case the algorithms of the sender and the receiver are the following ones.



```
while(true)                              IIRepeat forever
canSend = true                           IIAllow the first frame to go
{
 WaitForEvent()i                         II Sleep until an event occurs
 if(Event(RequestToSend) AND canSend}
 {
    GetData()i
    MakeFrame();
    SendFrame()i                         I/Send the data frame
    canSend = false;                     I/cannot send until ACK arrives
 }
 WaitForEvent()i                         II Sleep until an event occurs
 if(Event(ArrivalNotification)           /1 An ACK has arrived
  {
    ReceiveFrame();                      I/Receive the ACK frame
    canSend = true;
  }
}
```

Figure 1.3: Sender algorithm for the Stop-and-Wait Protocol.



```
while(true)                              IIRepeat forever
{
 WaitForEvent();                         II Sleep until an event occurs
 if(Event(ArrivalNotification)}          IIData frame arrives
 {
    ReceiveFrame();
    ExtractData()i
    Deliver(data};                       /IDeliver data to network layex
    SendFrame();                         IISend an ACK frame
 }
}
```

Figure 1.4: Receiver algorithm for the Stop-and-Wait Protocol.

The two protocols explained are protocols that can be suitable for noiseless channels. However, noiseless channels are nonexistent. There is a need to add error control to the protocol. Three protocols are discussed with the aim of doing so.

1.2.3 Stop-and-Wait Automatic Repeat Request

The Stop-and Wait ARQ adds a simple error control mechanism to the Stop-and-Wait Protocol. To detect and correct corrupted frames, we need to add redundancy bits to our data frame. When the frame arrives at the receiver site, it is checked and if it is corrupted, it is silently discarded. The detection of errors in this protocol is manifested by the silence of the receiver. Frames are also numbered so if the receiver receives a data frame that is out of order, this means that frames were either lost or duplicated. What is done to solve the error is that when the sender sends a frame, it keeps a copy of the sent frame. At the same time, it starts a timer. If the timer expires and there is no ACK for the sent frame, the frame is resent, the copy is held, and the timer is restarted. Since the protocol uses the stop-and-wait mechanism, there is only one specific frame that needs an ACK even though several copies of the same frame can be in the network. Since an ACK frame can also be corrupted and lost, it too needs redundancy bits and a sequence number. In the following figure is possible to see more clearly what is going on with this protocol.

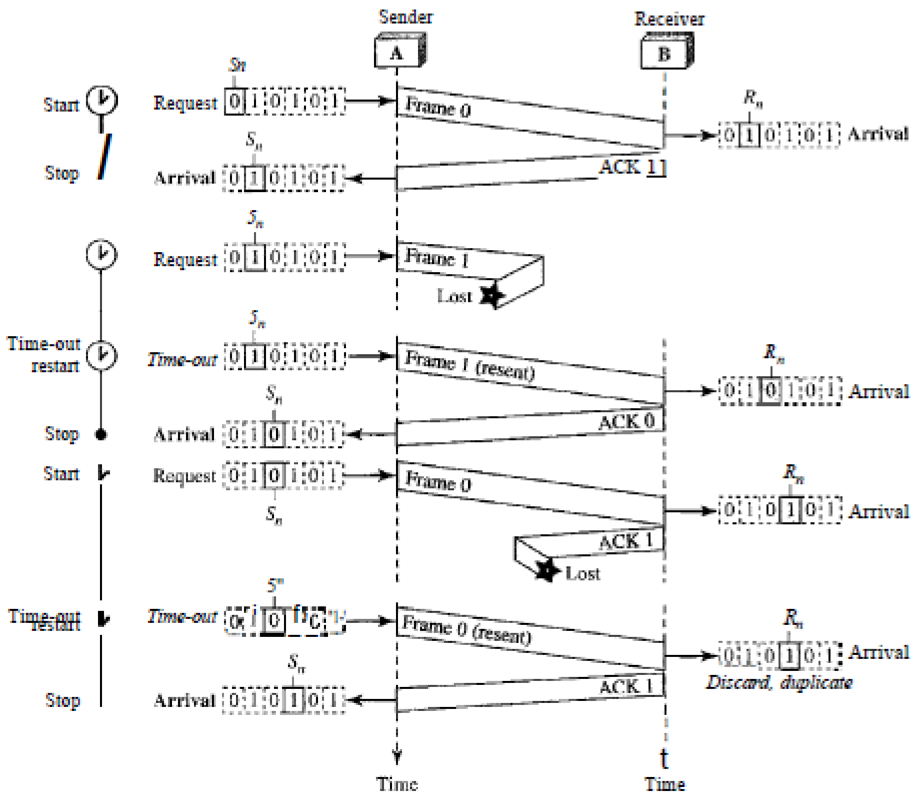


Figure 1.5: Flow diagram of the Stop-and Wait ARQ.

The main problem of this protocol is its efficiency. The Stop-and-Wait ARQ is very inneficinet if our channel is thick and long. The product of thickness and longitud is called the bandwidth-delay product. We can think of the channel as a pipe. The bandwidth-delay product then is the volume of the pipe in bits. The pipe is always there. If we do not use it, we are inefficient.

1.2.4 Go-Back-N Automatic Repeat Request

To improve the efficiency of transmission (filling the pipe), multiple frames must be in transition while waiting for acknowledgment. In other words, we need to let more than one frame be outstanding to keep the channel busy while the sender is waiting for acknowledgment. In the Go-Back-N Automatic Repeat Request the sender sends several frames before receiving acknowledgments. It also keeps a copy of these frames until the acknowledgments arrive. Although there can be a timer for each frame that is sent, in this protocol only one is used.

The reason is that the timer for the first outstanding frame always expires first and then all outstanding frames when this timer expires are sent again. The receiver sends a positive acknowledgment if a frame has arrived safe and sound and in order. If a frame is damaged or is received out of order, the receiver is silent and will discard all subsequent frames until it receives the one it is expecting. The silence of the receiver causes the timer of the unacknowledged frame at the sender site to expire. This, in turn, causes the sender to go back and resend all frames, beginning with the one with the expired timer. The receiver does not have to acknowledge each frame received. It can send one cumulative acknowledgment for several frames. That is the reason why the protocol is called Go-Back-N. The flow diagram and the and algorithms of the sender and the receiver are shown next.

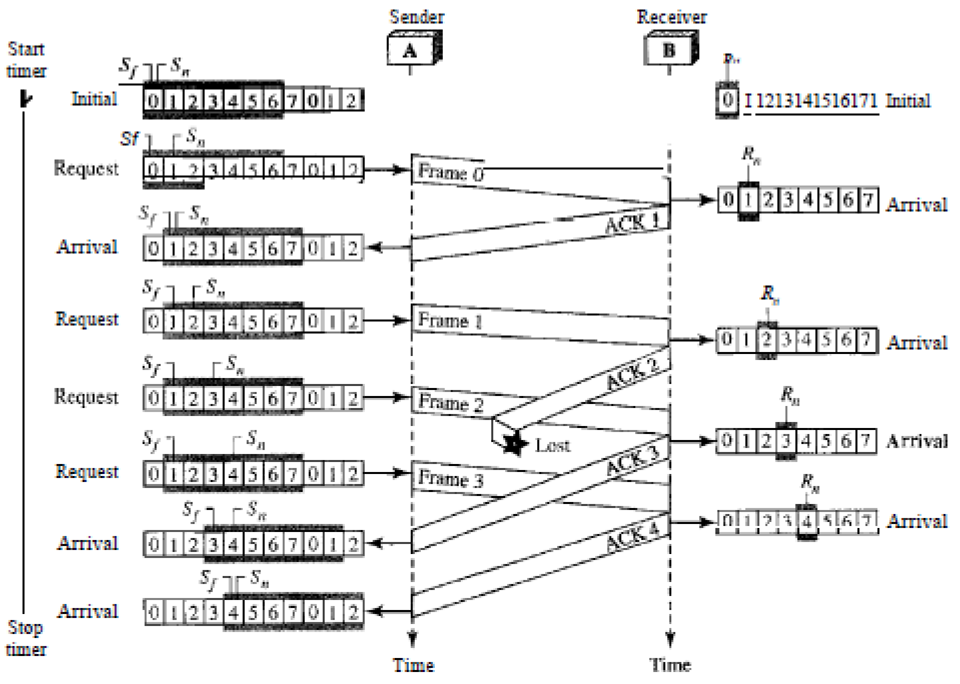
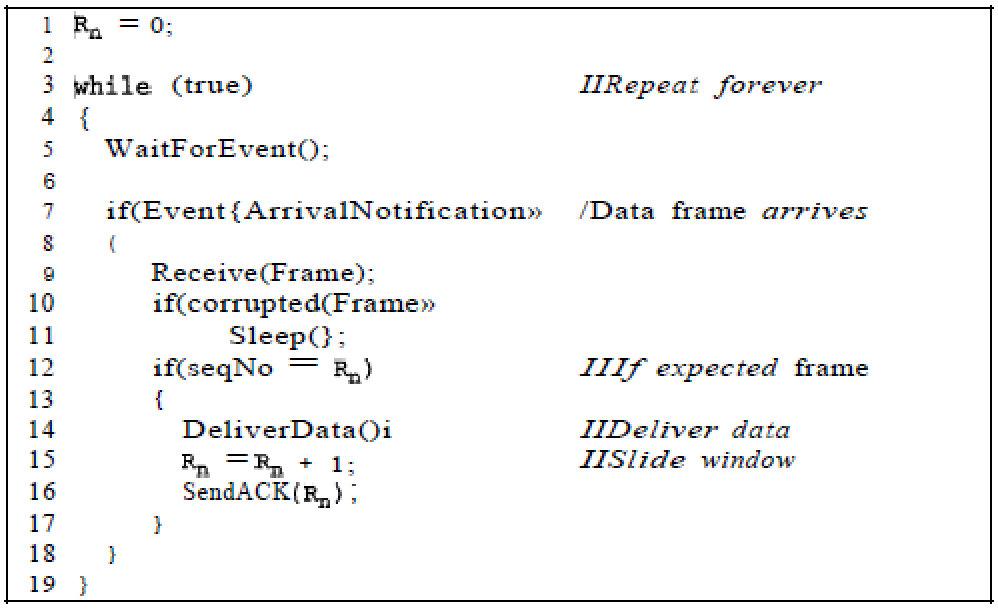


Figure 1.6: Flow diagram of the Go-Back-N ARQ.

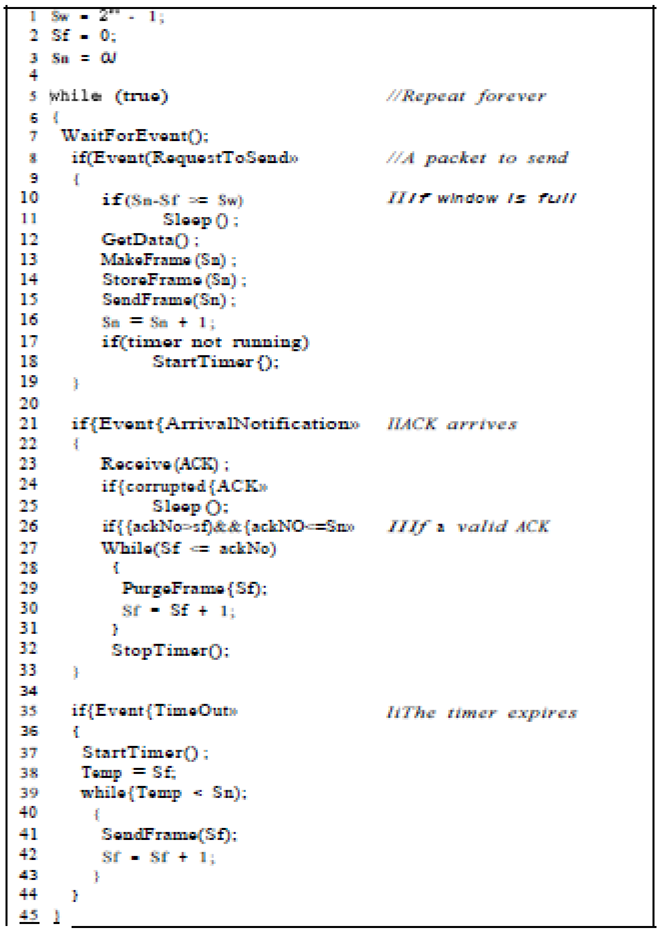


```
Rn = 0;

while (true)                          IIRepeat forever
{
  WaitForEvent();

  if(Event{ArrivalNotification»       /Data frame arrives
  (
     Receive(Frame);
     if(corrupted(Frame»
          Sleep();
     if(seqNo = Rn)                   IIIf expected frame
     {
        DeliverData()i                IIDeliver data
        Rn =Rn + 1;                   IISlide window
        SendACK(Rn);
     }
  }
}
```

Figure 1.7: Receiver algorithm for the Go-Back-N ARQ.



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

while (true)                          //Repeat forever
{
 WaitForEvent();
  if(Event(RequestToSend»             //A packet to send
  {
     if(Sn-Sf >= Sw)                  IIIf window is full
            Sleep();
     GetData();
     MakeFrame(Sn);
     StoreFrame(Sn);
     SendFrame(Sn);
     Sn = Sn + 1;
     if(timer not running)
          StartTimer();
  }

  if{Event{ArrivalNotification»       IIACK arrives
  {
     Receive(ACK);
     if{corrupted{ACK»
          Sleep();
     if{{ackNo>sf)&&{ackNO<=Sn»       IIIf a valid ACK
     While(Sf <= ackNo)
      {
       PurgeFrame(Sf);
       Sf = Sf + 1;
      }
      StopTimer();
  }

  if{Event{TimeOut»                   IiThe timer expires
  {
   StartTimer();
   Temp = Sf;
   while{Temp < Sn);
    {
     SendFrame(Sf);
     Sf = Sf + 1;
    }
  }
}
```

Figure 1.8: Sender algorithm for the Go-Back-N ARQ.

1.2.5 Selective Repeat Automatic Repeat Request

Go-Back-N ARQ simplifies the process at the receiver site. The receiver keeps track of only one variable, and there is no need to buffer out-of-order frames; they are simply discarded. However, this protocol is very inefficient for a noisy link. In a noisy link a frame has a higher probability of damage, which means the resending of multiple frames. In the case of these protocol, the Selective Repeat ARQ, the processing at the receiver is more complex but is more efficient for noisy links. The Selective Repeat Protocol allows a number of frames to arrive out of order and be kept until there is a set of in-order frames to be delivered to the network layer. The handling of the request event is similar to that of the previous protocol except that one timer is started for each frame sent. The arrival event is more complicated here. An ACK or a NAK frame may arrive. If a valid NAK frame arrives, the corresponding frame is resent. If a valid ACK arrives the corresponding timer stops. When the time for a frame has expire, only this frame is resent.

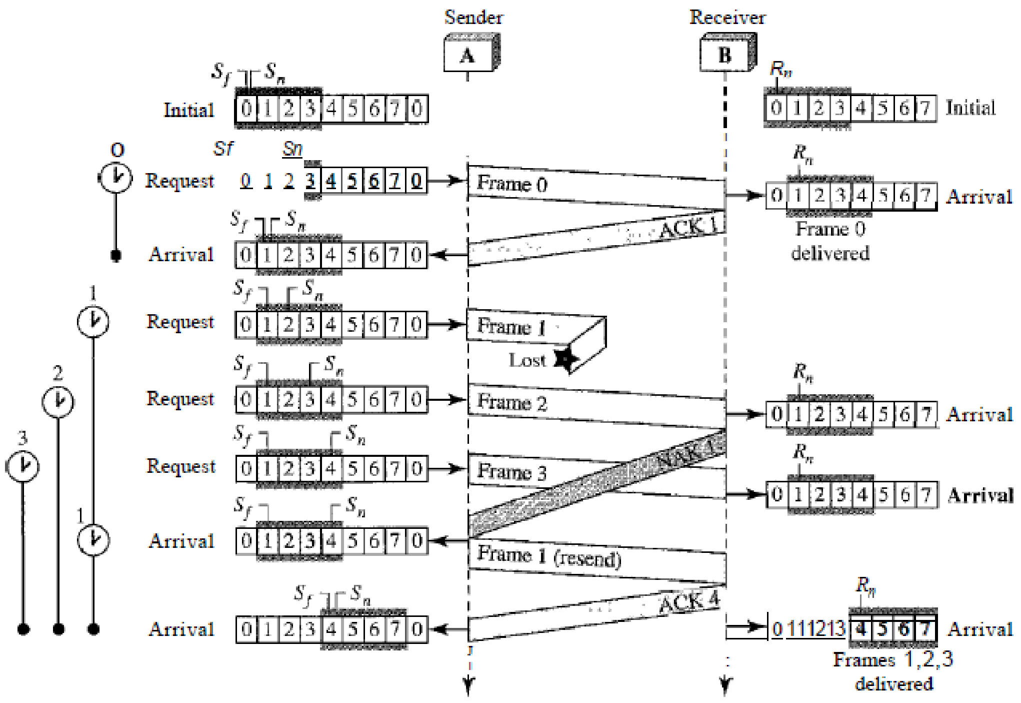
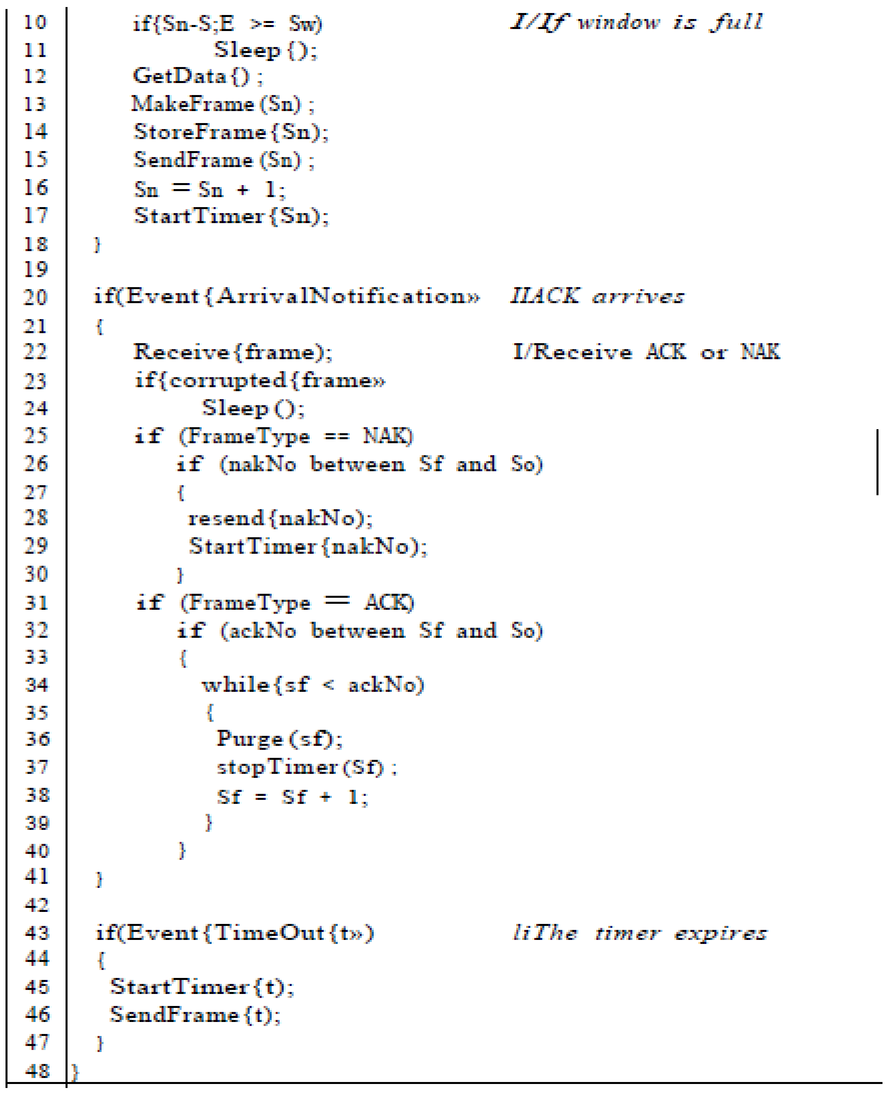


Figure 1.9: Flow diagram of the Selective Repeat ARQ.

```
     = 2m. 1  i
     = Oi
     = Oi

 hile  (true)                          //Repeat forever
{
  WaitForEvent()i
  if(Event(RequestToSend)}             //There is a packet to sen
  {
```

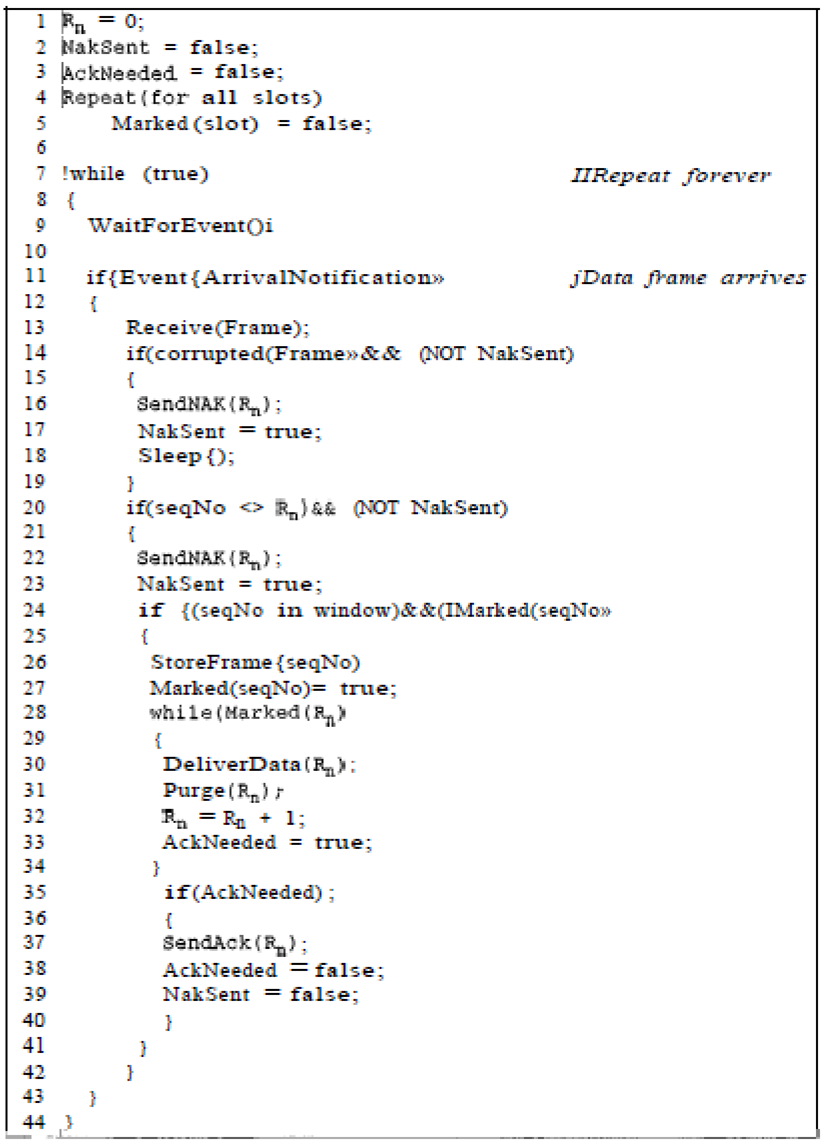


```
        if{Sn-S;E  >=  Sw)                  I/If window is full
              Sleep ();
        GetData() ;
        MakeFrame(Sn) ;
        StoreFrame{Sn);
        SendFrame(Sn) ;
        Sn  = Sn  +  1;
        StartTimer{Sn);
    }

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

    if(Event{TimeOut{t»)                 liThe timer expires
    {
      StartTimer{t);
      SendFrame{t);
    }
}
```

Figure 1.10: Sender algorithm for the Selective Repeat ARQ.



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

!while (true)                              IIRepeat forever
{
  WaitForEvent()i

  if{Event{ArrivalNotification»            jData frame arrives
  {
     Receive(Frame);
     if(corrupted(Frame»&& (NOT NakSent)
     {
      SendNAK($R_n$);
      NakSent = true;
      Sleep{();
     }
     if(seqNo <> $R_n$)&& (NOT NakSent)
     {
      SendNAK($R_n$);
      NakSent = true;
      if {(seqNo in window)&&(IMarked(seqNo»
      {
       StoreFrame{seqNo)
       Marked(seqNo)= true;
       while(Marked($R_n$)
       {
        DeliverData($R_n$);
        Purge($R_n$);
        $R_n$ = $R_n$ + 1;
        AckNeeded = true;
       }
        if(AckNeeded);
        {
        SendAck($R_n$);
        AckNeeded =false;
        NakSent = false;
        }
      }
     }
  }
}
```

Figure 1.11: Receiver algorithm for the Selective Repeat ARQ.

1.3 Bidirecional links: Piggybacking

Piggybacking is not a protocol, is a technique. All que protocols explained until now are all unidirectional: data frames flow in only one direction although control information such as ACK and NAK frames can travel in the other direction. In real life, data frames are normally flowing in both directions: from node A to node B and from node B to node A. This means that the control information also needs to flow in both directions. Piggybacking is usedto improve the efficiency of the bidirectional protocols. When a frame is carrying data from A to B, it can also carry control information about arrived (or lost) frames from B; when a frame is carrying data from B to A, it can also carry control information about the arrived (or lost) frames from A.

1.4 Protocol selection

Now its time to choose the protocol that best fits the needs of the mission. To do so, an OWA (Ordered Weighted Average) will be used. The criteria to consider is the following one:

- Efficiency: This fact deals with how the channel is being used. Protocols will be classified as non-efficient or efficient.
- Time: This fact deals about the time needed to transmit the data satisfactory.
- Error correction: Deals about whether a protocol can correct an error of transmission or not.

It is important also to take into account that the protocol to use should have a flow control, that is, should know if the receiver is available or not to receive the data. For this reason the Simplest Protocol is rejected and won't be studied in the OWA. Regarding the factors of the OWA, all of them will be rated from 0 to 1. In this project the fact of transmitting the data without errors is more important than transmitting it fast, as is possible to appreciate un the project charter (the latency can be relative high, but incorrect information is useless). The efficiency of the protocol is very important too, because the less the efficiency the less power provided by the CubeSat is being used. Since the CubeSat has limited space, ideally al the power it can gives for transmittion will be used for it. Then, the weights of the different factors are the following ones:

- Efficiency: 40
- Time: 30
- Error correction: 60

In the following table the rating of each protocol together with the corresponding OWA is shown.

Protocol	Efficiency	Time	Error correction	OWA
Stop-and-Wait Protocol	0	0	0	0
Stop-and-Wait ARQ	0	0	1	0,46
Go-Back-N ARQ	1	0	1	0.69
Selective Repeat ARQ	1	1	1	1

Table 1.1: OWA of the DLL protocols.

The selected protocol is the **Selective Repeat ARQ**. In order to make it more efficient in bidirectional link as the one is used in the mission, piggybacking technique will be used.

Bibliography

[1] Behrouz a. Forouzan. *Data Communications and Networking - Global Edition.* 2012.