UNIT 2 DATA LINK LAYER

Data Link Layer: Design issues, Data Link Control and Protocols: Flow and Error Control, Stop-andwait ARQ. Sliding window protocol, Go-Back-N ARQ, Selective Repeat ARQ, HDLC, Point-to – Point Access: PPP Point –to- Point Protocol, PPP Stack

Design issues

The data link layer in the OSI (Open System Interconnections) Model is in between the physical layer and the network layer. This layer converts the raw transmission facility provided by the physical layer to a reliable and error-free link.

Data-link layer is the second layer after the physical layer. The data link layer is responsible for maintaining the data link between two hosts or nodes. Before going through the design issues in the data link layer. Some of its sub-layers and their functions are as following below.
The data link layer is divided into two sub-layers:

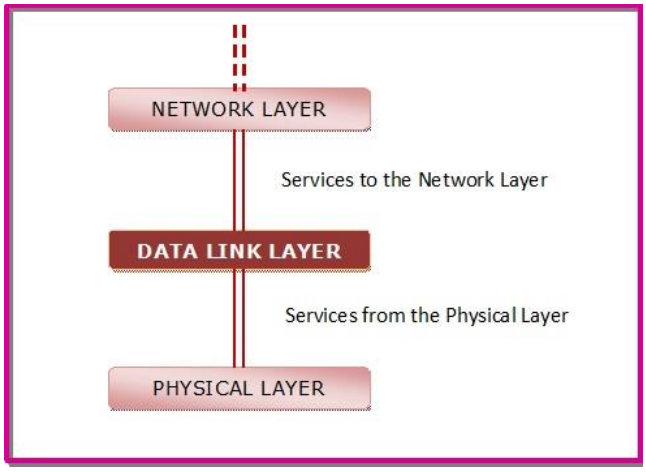
1. **Logical Link Control Sub-layer (LLC) –**
   Provides the logic for the data link, Thus it controls the synchronization, flow control, and error checking functions of the data link layer. Functions are –
   - **(i)** Error Recovery.
   - **(ii)** It performs the flow control operations.
   - **(iii)** User addressing.

2. **Media Access Control Sub-layer (MAC) –**
   It is the second sub-layer of data-link layer. It controls the flow and multiplexing for transmission medium. Transmission of data packets is controlled by this layer. This layer is responsible for sending the data over the network interface card.
   Functions are –
   - **(i)** To perform the control of access to media.
   - **(ii)** It performs the unique addressing to stations directly connected to LAN.
   - **(iii)** Detection of errors.

The main functions and the design issues of the data link layer are

- **Providing services to the network layer**
- **Framing**
- **Error Control**
- **Flow Control**

Services to the Network Layer



In the OSI Model, each layer uses the services of the layer below it and provides services to the layer above it. The data link layer uses the services offered by the physical layer. The primary function of this layer is to provide a well-defined service interface to network layer above it.

The types of services provided can be of three types −

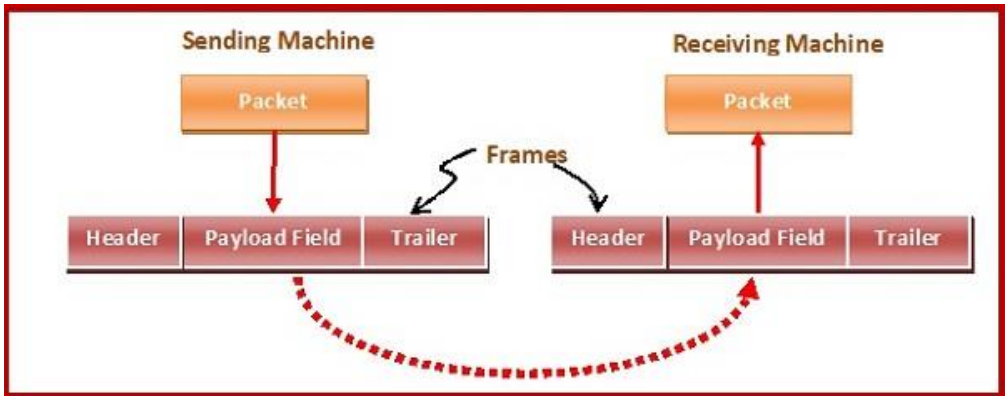
- Unacknowledged connectionless service
- Acknowledged connectionless service
- Acknowledged connection - oriented service

Framing

The data link layer encapsulates each data packet from the network layer into frames that are then transmitted.

A frame has three parts, namely −

- Frame Header
- Payload field that contains the data packet from network layer
- Trailer



Error Control

The data link layer ensures error free link for data transmission. The issues it caters to with respect to error control are −

- Dealing with transmission errors
- Sending acknowledgement frames in reliable connections
- Retransmitting lost frames
- Identifying duplicate frames and deleting them
- Controlling access to shared channels in case of broadcasting

Flow Control

The data link layer regulates flow control so that a fast sender does not drown a slow receiver. When the sender sends frames at very high speeds, a slow receiver may not be able to handle it. There will be frame losses even if the transmission is error-free. The two common approaches for flow control are −

- Feedback based flow control
- Rate based flow control

Data Link Control and Protocols

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

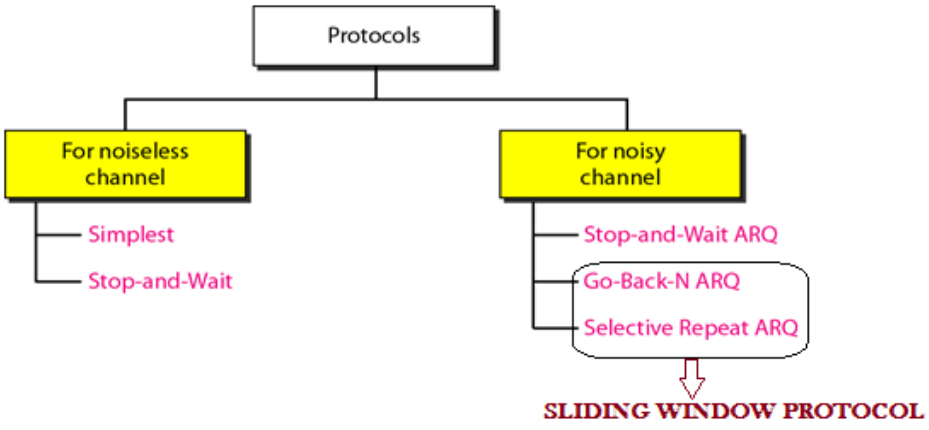
Data link control functions include framing, flow and error control, and softwareimplemented protocols that provide smooth and reliable transmission of frames between nodes. To implement data link control, we need protocols. Each protocol is a set of rules that need to be implemented in software and run by the two nodes involved in data exchange at the data link layer. We discuss five protocols: two for noiseless (ideal) channels and three for noisy (real) channels.

Flow Control

Flow control coordinates the amount of data that can be sent before receiving an acknowledgment and is one of the most important duties of the data link layer. In most protocols, flow control is a set of procedures that tells the sender how much data it can transmit before it must wait for an acknowledgment from the receiver. The flow of data must not be allowed to overwhelm the receiver. Any receiving device has a limited speed at which it can process incoming data and a limited amount of memory in which to store incoming data. The receiving device must be able to inform the sending device before those limits are reached and to request that the transmitting device send fewer frames or stop temporarily. Incoming data must be checked and processed before they can be used. The rate of such processing is often slower than the rate of transmission. For this reason, each receiving device has a block of memory, called a buffer, reserved for storing incoming data until they are processed. If the buffer begins to fill up, the receiver must be able to tell the sender to halt transmission until it is once again able to receive

Error Control

Error control is both error detection and error correction. It allows the receiver to inform the sender of any frames lost or damaged in transmission and coordinates the retransmission of those frames by the sender. In the data link layer, the term error control refers primarily to methods of error detection and retransmission. Error control in the data link layer is often implemented simply: Any time an error is detected in an exchange, specified frames are retransmitted. This process is called automatic repeat request (ARQ).



NOISELESS CHANNELS (FOR FLOW CONTROL)

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. The first is a protocol that does not use flow control; the second is the one that does. Of course, neither has error control because we have assumed that the channel is a perfect noiseless channel.

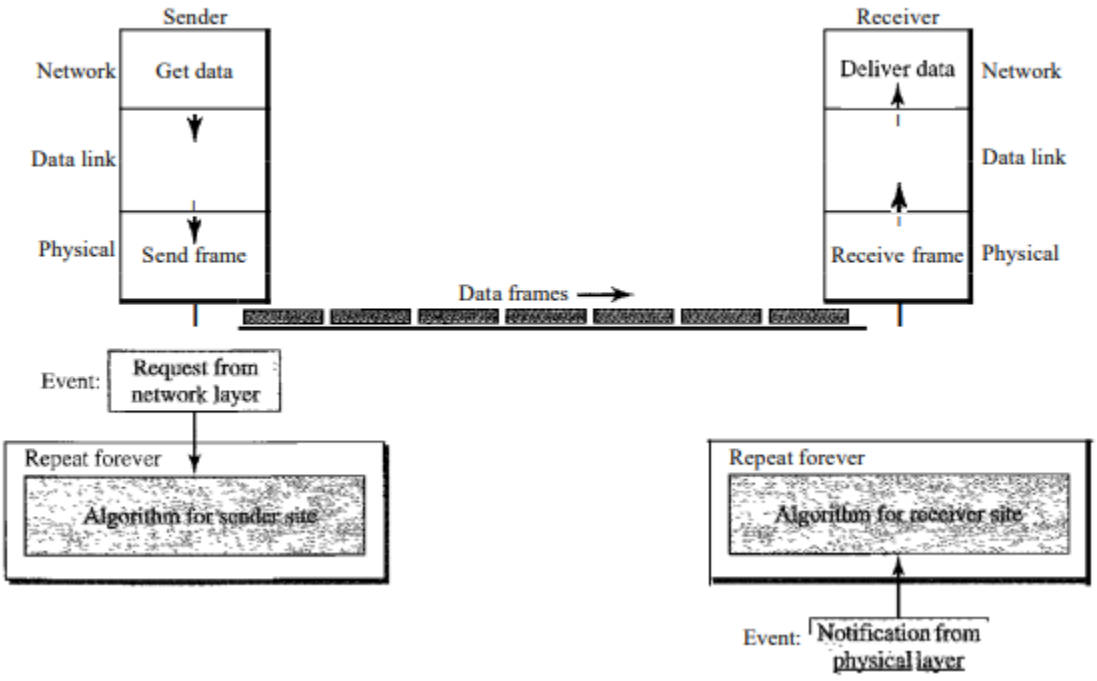
Simplest Protocol

Our first protocol, which we call the Simplest Protocol for lack of any other name, is one that has no flow or error control, it is a unidirectional protocol in which data frames are traveling in only one direction-from the sender to receiver. We assume that the receiver can immediately handle any frame it receives with a processing time that is small enough to be negligible. The data link layer of the receiver immediately removes the header from the frame and hands the data packet to its network layer, which can also accept the packet immediately. In other words, the receiver can never be overwhelmed with incoming frames.

Design

There is no need for flow control in this scheme. The data link layer at the sender site gets data from its network layer, makes a frame out of the data, and sends it. The data link layer 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 data link layers of the sender and receiver provide transmission services for their network layers. The data link layers use the services provided by their physical layers (such as signaling, multiplexing, and so on) for the physical transmission of bits. Figure below shows a design.

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



We need to elaborate on the procedure used by both data link layers. The sender site cannot send a frame until its network layer has a data packet to send. The receiver site cannot

deliver a data packet to its network layer until a frame arrives. If the protocol is implemented as a procedure, we 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 rulming, but there is no action until notification from the physical layer arrives. Both procedures are constantly running because they do not know when the corresponding events will occur.

Sender site Algorithm

Algorithm *Sender-site algorithm for the simplest protocol*

```
1  while (true)                    // Repeat forever
2  {
3    WaitForEvent()i               // Sleep until an event occurs
4    if(Event(RequestToSend»       //There is a packet to send
5    {
6        GetData()i
7        MakeFrame()i
8        SendFrame()i              //Send the frame
9    }
10 }
```

Analysis

The algorithm has an infinite loop, which means lines 3 to 9 are repeated forever once the program starts. The algorithm is an event-driven one, which means that it sleeps (line 3) until an event wakes it up (line 4). This means that there may be an undefined span of time between the execution of line 3 and line 4; there is a gap between these actions. When the event, a request from the network layer, occurs, lines 6 though 8 are executed. The program then repeats the loop and again sleeps at line 3 until the next occurrence of the event. We have written pseudocode for the main process. We do not show any details for the modules GetData, MakeFrame, and SendFrame. GetData takes a data packet from the network layer, MakeFrame adds a header and delimiter flags to the data packet to make a frame, and SendFrame delivers the frame to the physical layer for transmission.

Receiver site Algorithm

Algorithm *Receiver-site algorithm for the simplest protocol*

```
1  while(true)                     // Repeat forever
2  {
3    WaitForEvent()i               // Sleep until an event occurs
4    if(Event(ArrivalNotification» //Data frame arrived
5    {
6        ReceiveFrame()i
7        ExtractData()i
8        DeliverData ()i           //Deliver data to network layez
9    }
10 }
```

Analysis

This algorithm has the same format as Algorithm at sender site, except that the direction of the frames and data is upward. The event here is the arrival of a data frame. After the event

occurs, the data link layer receives the frame from the physical layer using the ReceiveFrame process, extracts the data from the frame using the ExtractData process, and delivers the data to the network layer using the DeliverData process. Here, we also have an event-driven algorithm because the algorithm never knows when the data frame will arrive.

Stop-and-Wait Protocol

The protocol we discuss now is called the Stop-and-Wait Protocol because the sender sends one frame, stops until it receives confirmation from the receiver (okay to go ahead), 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.

Design

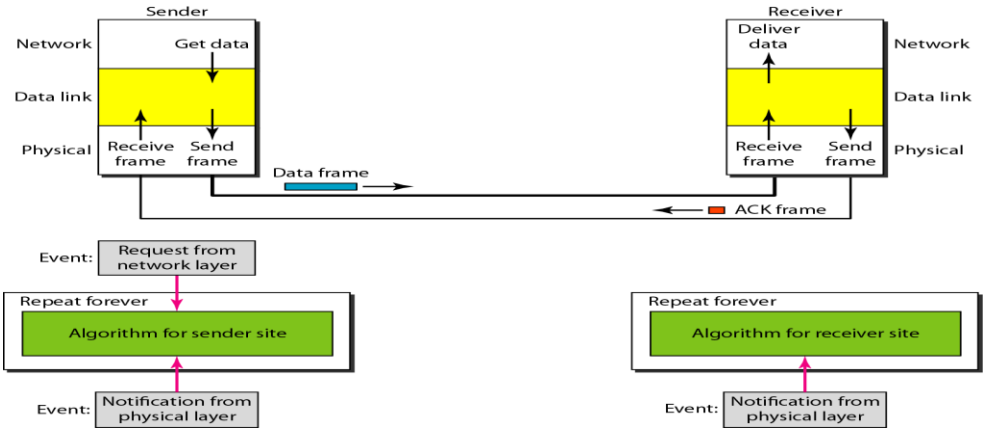


Figure illustrates the mechanism. Comparing this figure with Figure simple protocol, we can see the traffic on the forward channel (from sender to receiver) and the reverse channel. At any time, there is either one data frame on the forward channel or one ACK frame on the reverse channel. We therefore need a half-duplex link.

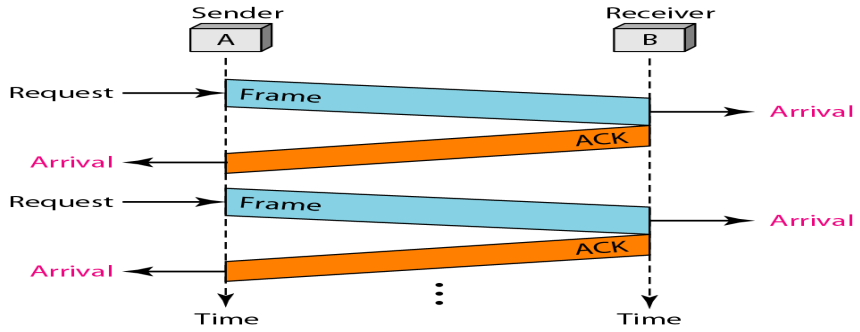
Sender-site algorithm for Stop-and-Wait Protocol

```
1  while(true)                          //Repeat forever
2  canSend = true                       //Allow the first frame to go
3  {
4    WaitForEvent();                    // Sleep until an event occurs
5    if(Event(RequestToSend) AND canSend)
6    {
7       GetData();
8       MakeFrame();
9       SendFrame();                    //Send the data frame
10      canSend = false;                //Cannot send until ACK arrives
11   }
12   WaitForEvent();                    // Sleep until an event occurs
13   if(Event(ArrivalNotification)      // An ACK has arrived
14    {
15      ReceiveFrame();                 //Receive the ACK frame
16      canSend = true;
17    }
18 }
```

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

```
1  while(true)                          //Repeat forever
2  {
3    WaitForEvent();                    // Sleep until an event occurs
4    if(Event(ArrivalNotification))     //Data frame arrives
5    {
6       ReceiveFrame();
7       ExtractData();
8       Deliver(data);                  //Deliver data to network layer
9       SendFrame();                    //Send an ACK frame
10   }
11 }
```

Figure shows an example of communication using this protocol. It is still very simple. The sender sends one frame and waits for feedback from the receiver. When the ACK arrives, the sender sends the next frame. Note that sending two frames in the protocol involves the sender in four events and the receiver in two events.



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 can ignore the error or we need to add error control to our protocols. We discuss the protocol that uses error control.

Stop-and-Wait ARQ (FOR FLOW CONTROL and ERROR CONTROL)

Our first protocol, called the Stop-and-Wait Automatic Repeat Request (Stop-andWait 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.

Lost frames are more difficult to handle than corrupted ones. In our previous protocols, there was no way to identify a frame. The received frame could be the correct one, or a duplicate, or a frame out of order. The solution is to number the frames. When the receiver receives a data frame that is out of order, this means that frames were either lost or duplicated.

The completed and lost frames need to be resent in this protocol. If the receiver does not respond when there is an error, how can the sender know which frame to resend? To remedy this problem, the sender keeps 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. The ACK frame for this protocol has a sequence number field. In this protocol, the sender simply discards a corrupted ACK frame or ignores an out-of-order one.

Sequence Numbers

The protocol specifies that frames need to be numbered. This is done by using sequence numbers. A field is added to the data frame to hold the sequence number of that frame.

One important consideration is the range of the sequence numbers. Since we want to minimize the frame size, we look for the smallest range that provides unambiguous communication. The sequence numbers of course can wrap around. For example, if we decide that the field is m bits long, the sequence numbers start from 0, go to 2m - 1, and then are repeated.

Let us reason out the range of sequence numbers we need. Assume we have used x as a sequence number; we only need to use x + 1 after that. There is no need for x + 2. To show this, assume that the sender has sent the frame numbered x. Three things can happen

1. The frame arrives safe and sound at the receiver site; the receiver sends an acknowledgment. The acknowledgment arrives at the sender site, causing the sender to send the next frame numbered as x + 1.

2. The frame arrives safe and sound at the receiver site; the receiver sends an acknowledgment, but the acknowledgment is corrupted or lost. The sender resends the frame (numbered x) after the time-out. Note that the frame here is a duplicate. The receiver can recognize this fact because it expects frame x + I but frame x was received.

3. The frame is corrupted or never arrives at the receiver site; the sender resends the frame (numbered x) after the time-out.

Acknowledgment Numbers

Since the sequence numbers must be suitable for both data frames and ACK frames, we use this convention: The acknowledgment numbers always announce the sequence number of the next frame expected by the receiver. For example, if frame 0 has arrived safe and sound, the receiver sends an ACK frame with acknowledgment 1 (meaning frame 1 is expected next). If frame 1 has arrived safe and sound, the receiver sends an ACK frame with acknowledgment 0 (meaning frame 0 is expected).

Design

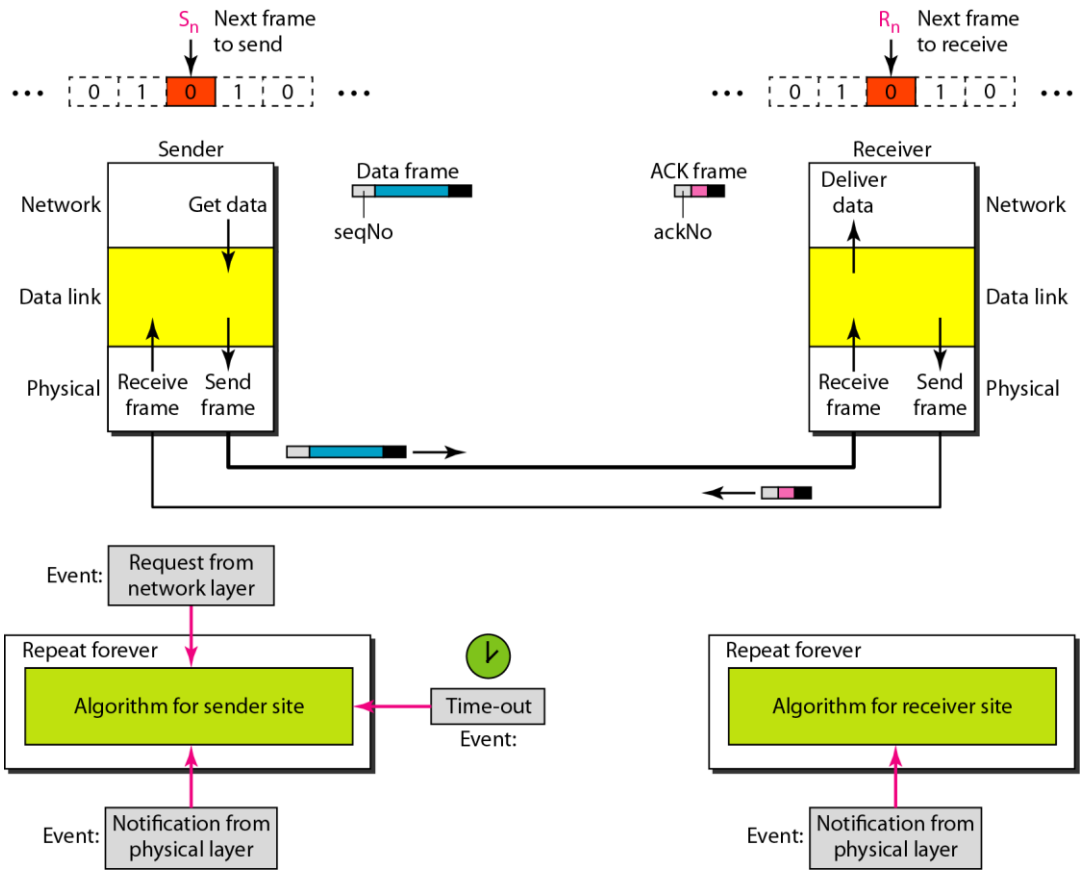


Figure above shows the design of the Stop-and-Wait ARQ Protocol. The sending device keeps a copy of the last frame transmitted until it receives an acknowledgment for that frame. A data frames uses a seq No (sequence number); an ACK frame uses an ackNo (acknowledgment number). The sender has a control variable, which we call Sn (sender, next frame to send), that holds the sequence number for the next frame to be sent (0 or 1).

The receiver has a control variable, which we call Rn (receiver, next frame expected), that holds the number of the next frame expected. When a frame is sent, the value of Sn is incremented (modulo-2), which means if it is 0, it becomes 1 and vice versa. When a frame is received, the value of Rn is incremented (modulo-2), which means if it is 0, it becomes 1 and vice versa. Three events can happen at the sender site; one event can happen at the receiver site. Variable Sn points to the slot that matches the sequence number of the frame that has been sent, but not acknowledged; Rn points to the slot that matches the sequence number of the expected frame

Algorithms

Algorithm below is for the sender site.

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

```
1   n = 0;                              // Frame 0 should be sent first
2   anSend = true;                      // Allow the first request to go
3   hile(true)                          // Repeat forever
4   {
5     WaitForEvent();                   // Sleep until an event occurs
6     if(Event(RequestToSend) AND canSend)
7     {
8       GetData();
9       MakeFrame(Sn);                  //The seqNo is Sn
10      StoreFrame(Sn);                 //Keep copy
11      SendFrame(Sn);
12      StartTimerO;
13      Sn = Sn + 1;
14      canSend = false;
15    }
16    WaitForEvent();                   // Sleep
17      if(Event(ArrivalNotification)   // An ACK has arrived
18      {
19        ReceiveFrame(ackNo);          //Receive the ACE fram
20        if(not corrupted AND ackNo = Sn) //Valid ACK
21          {
22            Stoptimer{};
23            PurgeFrame(Sn_1);         //Copy is not needed
24            canSend = true;
25          }
26       }
27
28      if(Event(TimeOUt)               // The timer expired
29      {
30       StartTimer();
31       ResendFrame(Sn_1);             //Resend a copy check
32      }
33  }
```

Analysis

We first notice the presence of Sn' the sequence number of the next frame to be sent. This variable is initialized once (line 1), but it is incremented every time a frame is sent (line 13) in preparation for the next frame. However, since this is modulo-2 arithmetic, the sequence numbers are 0, 1,0, 1, and so on. Note that the processes in the first event (SendFrame, StoreFrame, and PurgeFrame) use an Sn defining the frame sent out. We need at least one buffer to hold this frame until we are sure that it is received safe and sound. Line 10 shows that before the frame is sent, it is stored. The copy is used for resending a corrupt or lost frame. We are still using the canSend variable to prevent the network layer from making a request before the previous frame is received safe and sound. If the frame is not corrupted and the ackNo of the ACK frame matches the sequence number of the next frame to send, we stop the timer and purge the copy of the data frame we saved. Otherwise, we just ignore this event and wait for the next event to happen. After each frame is sent, a timer is started. When the timer expires (line 28), the frame is resent and the timer is restarted.

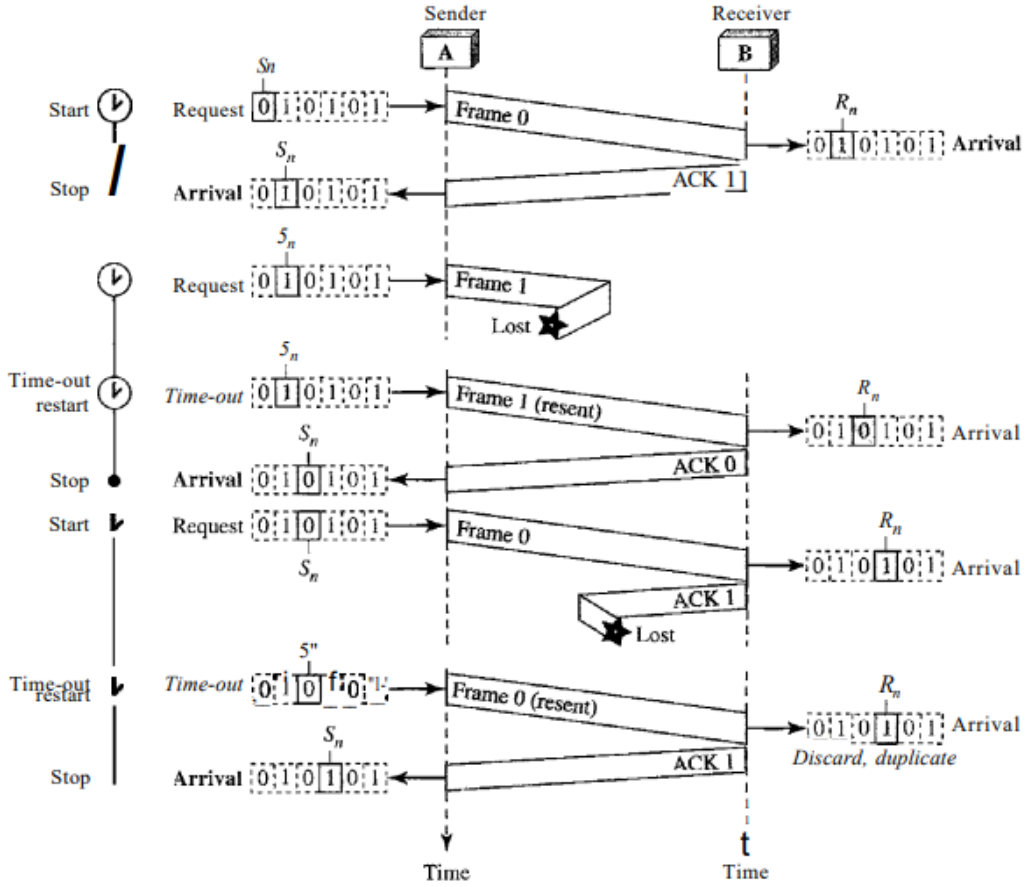
Algorithm below shows the procedure at the receiver site

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

```
1     = 0;                          II Frame 0 expected to arrive firs
2   hile(true)
3   {
4     WaitForEvent();               II Sleep until an event occurs
5     if(Event(ArrivalNotification»     //Data frame arrives
6     {
7         ReceiveFrame()i
8         if(corrupted(frame»i
9             sleep() i
10        if(seqNo == Rn)               //Valid data frame
11        {
12          ExtractData();
13           DeliverData()i             //Deliver data
14          Rn = Rn + 1;
15        }
16          SendFrame(Rn);              //Send an ACK
17    }
18  }
```

Analysis

Figure below shows an example of Stop-and-Wait ARQ



First, all arrived data frames that are corrupted are ignored. If the seqNo of the frame is the one that is expected (Rn), the frame is accepted, the data are delivered to the network layer, and the value of Rn is incremented. However, there is one subtle point here. Even if the sequence number of the data frame does not match the next frame expected, an ACK is sent to the sender. This ACK, however, just reconfirms the previous ACK instead of confirming the frame received. This is done because the receiver assumes that the previous ACK might have been lost; the receiver is sending a duplicate frame. The resent ACK may solve the problem before the time-out does it.

Sliding window protocol

Sliding window protocols are data link layer protocols for reliable and sequential delivery of data frames. The sliding window is also used in Transmission Control Protocol.

In this protocol, multiple frames can be sent by a sender at a time before receiving an acknowledgment from the receiver. The term sliding window refers to the imaginary boxes to hold frames. Sliding window method is also known as windowing. The Sliding Window ARQ (Automatic Repeat request) protocols are of two categories –

Go-Back-N Automatic Repeat Request (ARQ)

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 this, we discuss one protocol that can achieve this goal; in the next section, we discuss a second.

The first is called Go-Back-N Automatic Repeat Request (the rationale for the name will become clear later). In this protocol we can send several frames before receiving acknowledgments; we keep a copy of these frames until the acknowledgments arrive.

Sequence Numbers

Frames from a sending station are numbered sequentially. However, because we need to include the sequence number of each frame in the header, we need to set a limit. If the header of the frame allows m bits for the sequence number, the sequence numbers range from 0 to 2m - 1. For example, if m is 4, the only sequence numbers are 0 through 15 inclusive. However, we can repeat the sequence. So the sequence numbers are

0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 0, 1,2, 3, 4, 5, 6, 7, 8, 9, 10, 11, ...

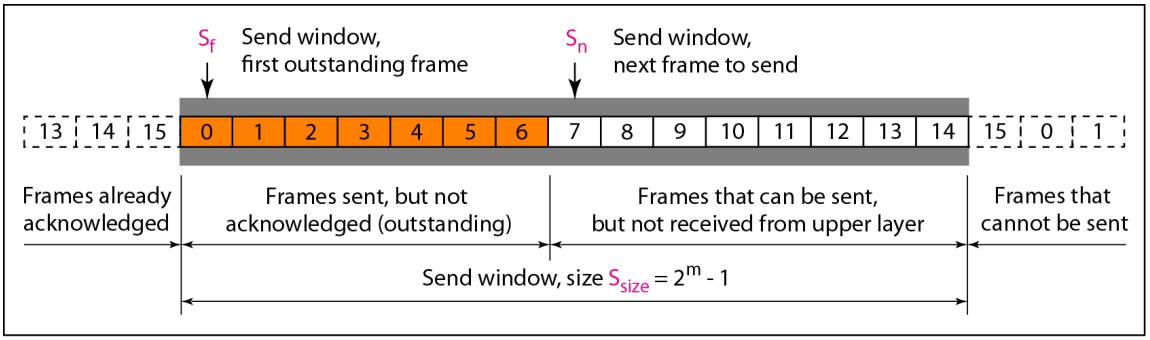
In other words, the sequence numbers are modulo-2^m

Sliding Window

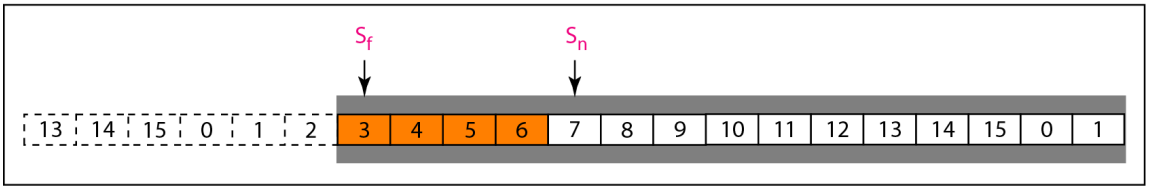
In this protocol, the sliding window is an abstract concept that defines the range of sequence numbers that is the concern of the sender and receiver. In other words, the sender and receiver need to deal with only part of the possible sequence numbers. The range which is the concern of the sender is called the send sliding window; the range that is the concern of the receiver is called the receive sliding window.

The send window is an imaginary box covering the sequence numbers of the data frames which can be in transit. In each window position, some of these sequence numbers define the frames that have been sent; others define those that can be sent. The maximum size of the window is 2m - 1 for some reasons. We let the size be fixed and set to the maximum value, but we will see in future chapters that some protocols may have a variable window size. Figure below shows a sliding window of size 15 (m =4).

The window at any time divides the possible sequence numbers into four regions. The first region, from the far left to the left wall of the window, defines the sequence numbers belonging to frames that are already acknowledged. The sender does not worry about these frames and keeps no copies of them. The second region, colored in Figure 11.12a, defines the range of sequence numbers belonging to the frames that are sent and have an unknown status. The sender needs to wait to find out if these frames have been received or were lost. We call these outstanding frames. The third range, white in the figure, defines the range of sequence numbers for frames that can be sent; however, the corresponding data packets have not yet been received from the network layer. Finally, the fourth region defines sequence numbers that cannot be used until the window slides.



a. Send window before sliding

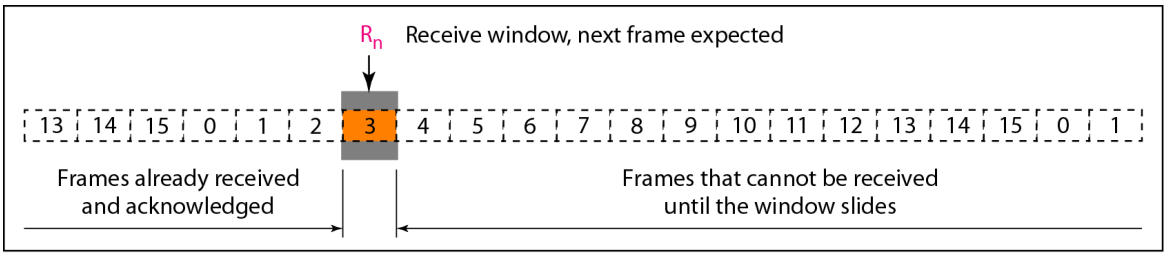


b. Send window after sliding

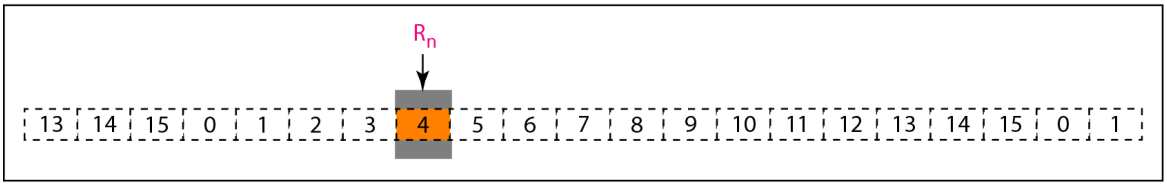
The window itself is an abstraction; three variables define its size and location at any time. We call these variables **Sf** (send window, the first outstanding frame), **Sn** (send window, the next frame to be sent), and **Ssize** (send window, size). The variable **Sf** defines the sequence number of the first (oldest) outstanding frame. The variable **Sn** holds the sequence number that will be assigned to the next frame to be sent. Finally, the variable **Ssize** defines the size of the window, which is fixed in our protocol.

Figure b above shows how a send window can slide one or more slots to the right when an acknowledgment arrives from the other end. As we will see shortly, the acknowledgments in this protocol are cumulative, meaning that more than one frame can be acknowledged by an ACK frame. In Figure b, frames 0, I, and 2 are acknowledged, so the window has slid to the right three slots. Note that the value of **Sf** is 3 because frame 3 is now the first outstanding frame.

The receive window makes sure that the correct data frames are received and that the correct acknowledgments are sent. The size of the receive window is always I. The receiver is always looking for the arrival of a specific frame. Any frame arriving out of order is discarded and needs to be resent. Figure below shows the receive window



a. Receive window



b. Window after sliding

Note that we need only one variable **Rn** (receive window, next frame expected) to define this abstraction. The sequence numbers to the left of the window belong to the frames already received and acknowledged; the sequence numbers to the right of this window define the frames that cannot be received. Any received frame with a sequence number in these two regions is discarded. Only a frame with a sequence number matching the value of **Rn** is accepted and acknowledged.

The receive window also slides, but only one slot at a time. When a correct frame is received (and also a frame is received only one at a time), the window slides.

Timers

Although there can be a timer for each frame that is sent, in our protocol we use only one. The reason is that the timer for the first outstanding frame always expires first; we send all outstanding frames when this timer expires.

Acknowledgment

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

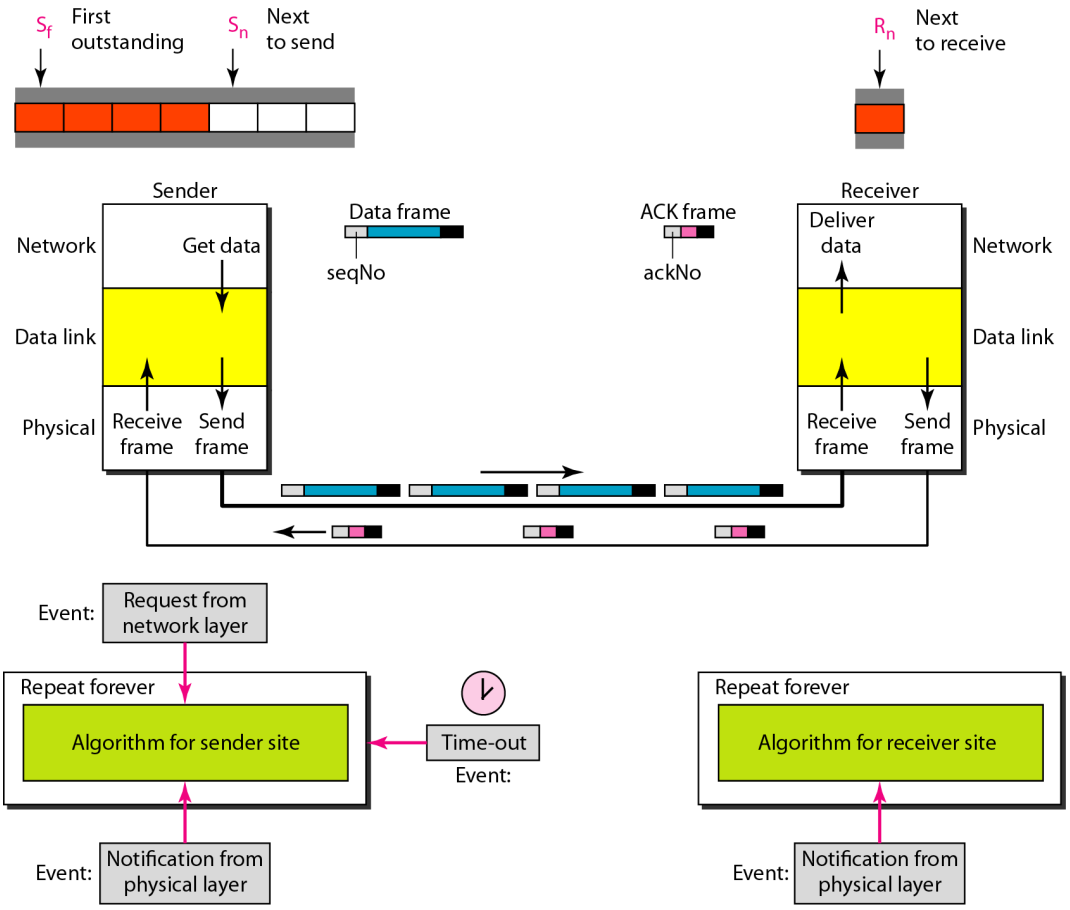
receiver does not have to acknowledge each frame received. It can send one cumulative acknowledgment for several frames.

Resending a Frame

When the timer expires, the sender resends all outstanding frames. For example, suppose the sender has already sent frame 6, but the timer for frame 3 expires. This means that frame 3 has not been acknowledged; the sender goes back and sends frames 3, 4,5, and 6 again. That is why the protocol is called Go-Back-N ARQ**.**

Design

Figure below shows the design for this protocol. As we can see, multiple frames can be in transit in the forward direction, and multiple acknowledgments in the reverse direction. The idea is similar to Stop-and-Wait ARQ; the difference is that the send window allows us to have as many frames in transition as there are slots in the send window



Send Window Size We can now show why the size of the send window must be less than 2m. As an example, we choose m =2, which means the size of the window can be 2m - 1, or 3. Figure above compares a window size of 3 against a window size of 4. If the size of the window is 3 (less than 22) and all three acknowledgments are lost, the frame ° timer expires and all three frames are resent. The receiver is now expecting frame 3, not frame 0, so the duplicate frame is correctly discarded. On the other hand, if the size of the window is 4 (equal to 22) and all acknowledgments are lost, the sender will send a duplicate of frame 0. However, this time the window of the receiver expects to receive frame 0, so it accepts frame 0, not as a duplicate, but as the first frame in the next cycle. This is an error.

Algorithms

Algorithm below shows the procedure for the sender in this protocol

```
1  Sw = 2^m - 1;
2  Sf = 0;
3  Sn = 0
4
5  while (true)                          //Repeat forever
6  {
7   WaitForEvent();
8    if(Event(RequestToSend»             //A packet to send
9    {
10       if(Sn-Sf >= Sw)                 IIIf window is full
11             Sleep();
12       GetData();
13       MakeFrame(Sn);
14       StoreFrame(Sn);
15       SendFrame(Sn);
16       Sn = Sn + 1;
17       if(timer not running)
18             StartTimer{);
19   }
20
21   if{Event{ArrivalNotification»       IIACK arrives
22   {
23       Receive(ACK);
24       if{corrupted{ACK»
25             Sleep();
26       if{{ackNo>sf)&&{ackNO<=Sn»      IIIf a valid ACK
27       While(Sf <= ackNo)
28        {
29         PurgeFrame{Sf);
30         Sf = Sf + 1;
31        }
32        StopTimer();
33   }
34
35   if{Event{TimeOut»                   liThe timer expires
36   {
37    StartTimer();
38    Temp = Sf;
39    while{Temp < Sn);
40      {
41       SendFrame(Sf);
42       Sf = Sf + 1;
43      }
44   }
45 }
```

Analysis

This algorithm first initializes three variables. Unlike Stop-and-Wait ARQ, this protocol allows several requests from the network layer without the need for other events to occur; we just need to be sure that the window is not full (line 12). In our approach, if the window is full the request is just ignored and the network layer needs to try again. Some implementations use other methods such as enabling or disabling the network layer. The handling of the arrival event is more complex than in the previous protocol. If we receive a corrupted ACK, we ignore it. If the

adeNa belongs to one of the outstanding frames, we use a loop to purge the buffers and move the left wall to the right. The time-out event is also more complex. We first start a new timer. We then resend all outstanding frames.

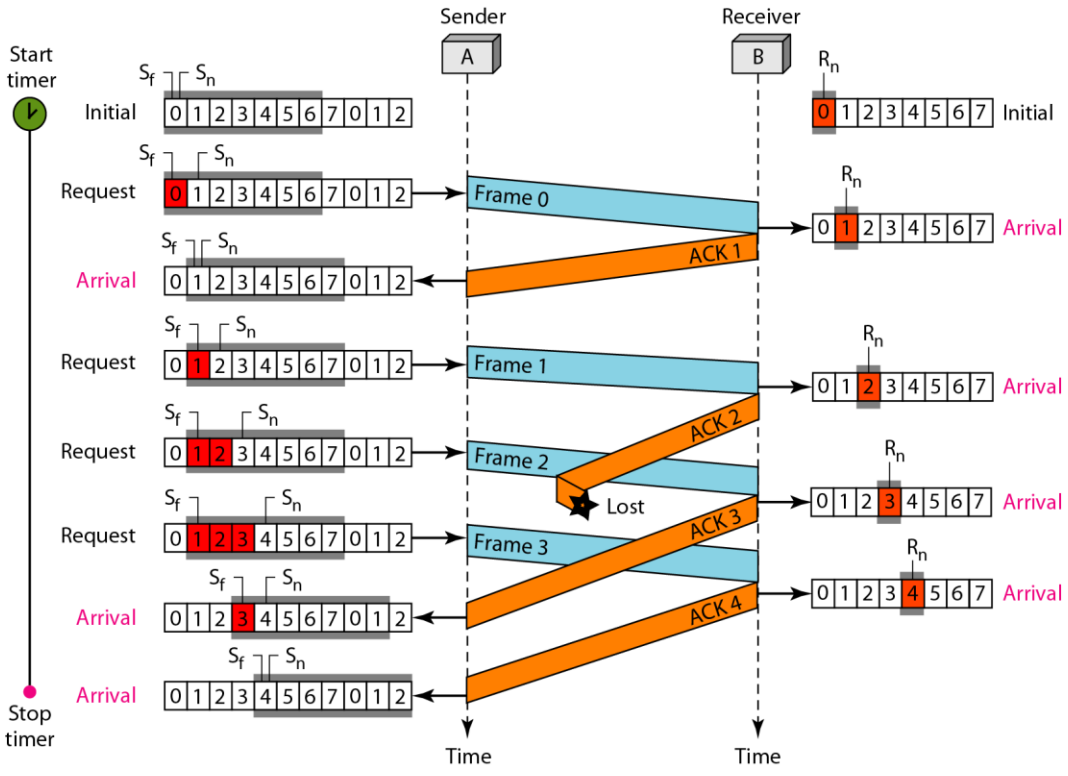
Algorithm below is the procedure at the receiver site

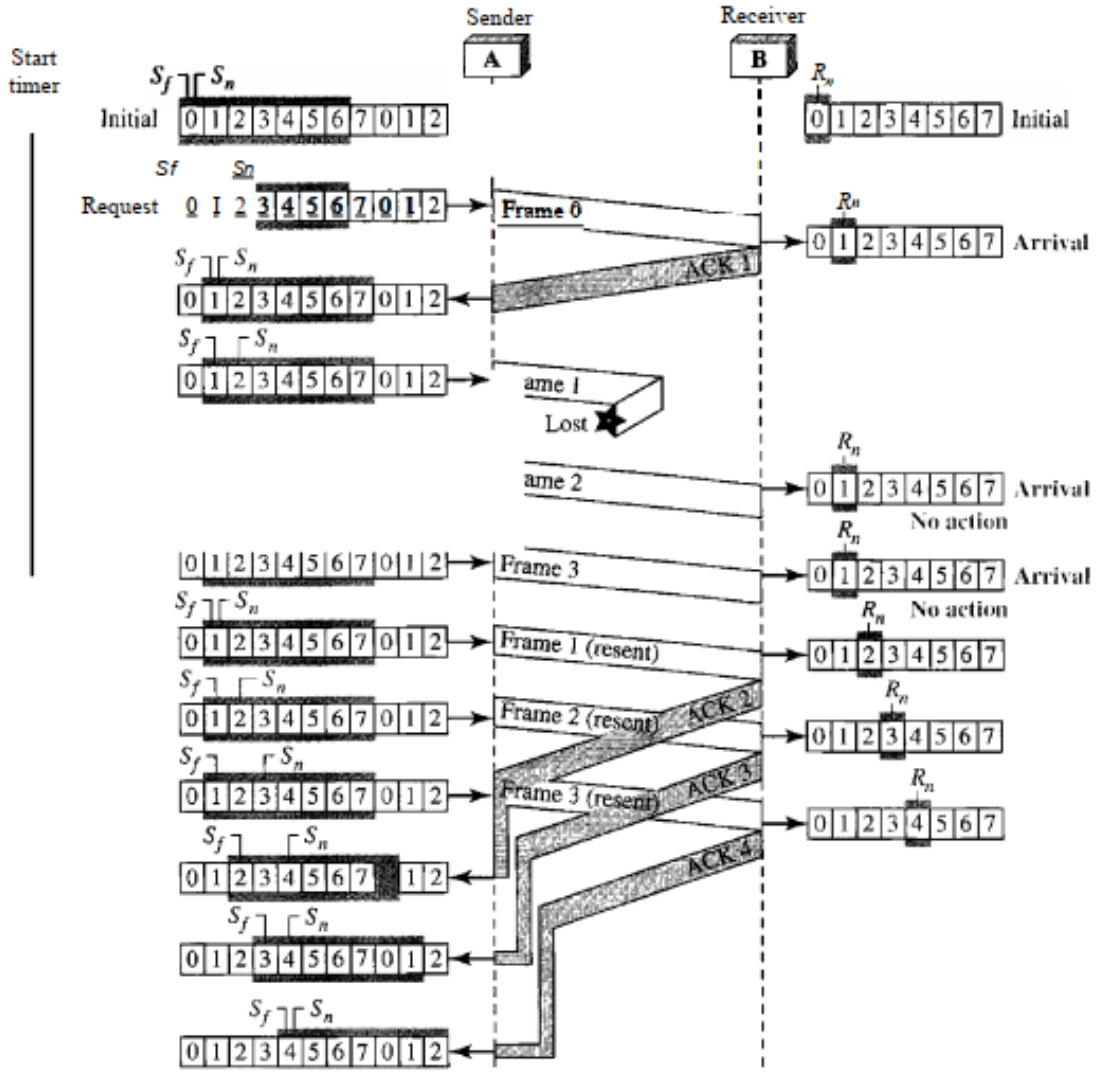
```
1  Rn = 0;
2
3  while (true)                          //Repeat forever
4  {
5    WaitForEvent();
6
7    if(Event{ArrivalNotification»   /Data frame arrives
8    (
9       Receive(Frame);
10      if(corrupted(Frame»
11           Sleep(};
12      if(seqNo == Rn)                  //If expected frame
13      {
14        DeliverData()i                 //Deliver data
15        Rn = Rn + 1;                   //Slide window
16        SendACK(Rn);
17      }
18   }
19 }
```

Analysis

This algorithm is simple. We ignore a corrupt or out-of-order frame. If a frame arrives with an expected sequence number, we deliver the data, update the value of **Rn**, and send an ACK with the ackNa showing the next frame expected

Flow diagram





Selective Repeat ARQ Protocol

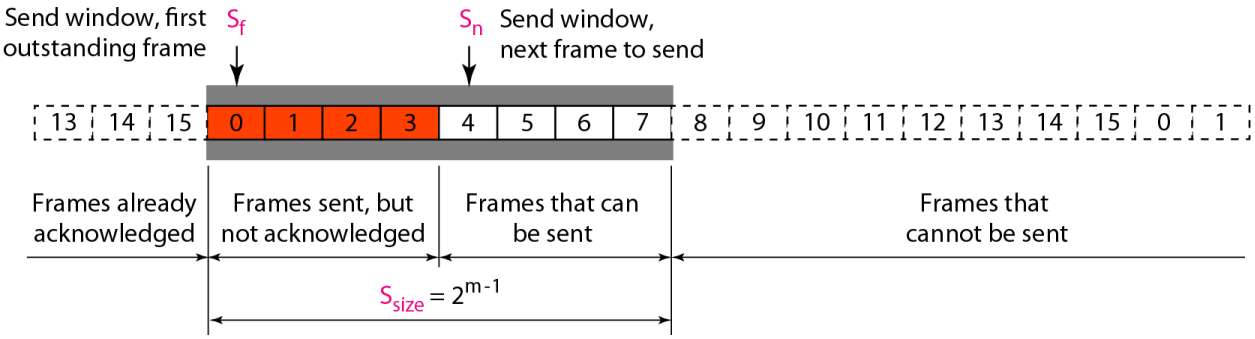
Selective Repeat Automatic Repeat Request Go-Back-N ARQ simplifies the process at the receiver site. The receiver keeps track of only one variable, and there is no need to buffer out-of-order frames; they are simply discarded. However, this protocol is very inefficient for a noisy link. In a noisy link a frame has a higher probability of damage, which means the resending of multiple frames. This resending uses up the bandwidth and slows down the transmission. For noisy links, there is another mechanism that does not resend N frames when just one frame is damaged; only the damaged frame is resent. This mechanism is called Selective Repeat ARQ. It is more efficient for noisy links, but the processing at the receiver is more complex

Windows

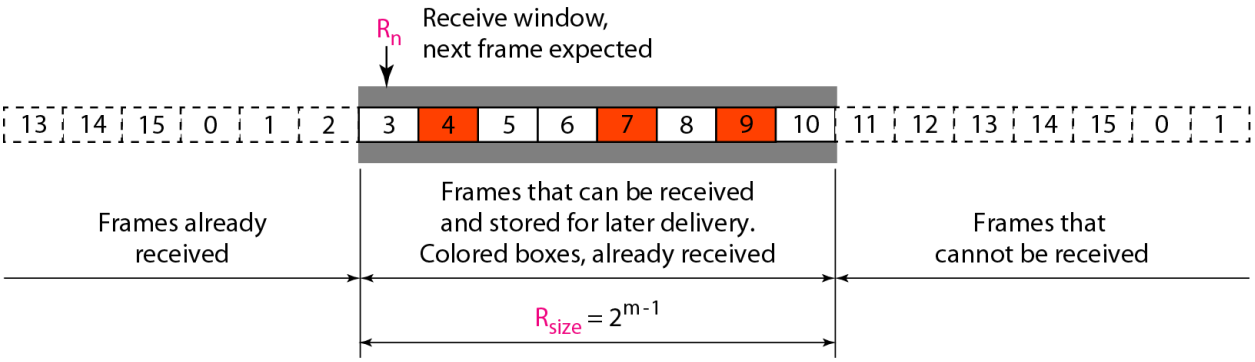
The Selective Repeat Protocol also uses two windows: a send window and a receive window. However, there are differences between the windows in this protocol and the ones in Go-Back-N.

First, the size of the send window is much smaller; it is 2m- I . The reason for this will be discussed later. Second, the receive window is the same size as the send window.

The send window maximum size can be 2m- I . For example, if m = 4, the sequence numbers go from 0 to 15, but the size of the window is just 8 (it is 15 in the Go-Back-N Protocol). The smaller window size means less efficiency in filling the pipe, but the fact that there are fewer duplicate frames can compensate for this. The protocol uses the same variables as we discussed for Go-Back-N.



The receive window in Selective Repeat is totally different from the one in GoBack-N. First, the size of the receive window is the same as the size of the send window (2m- I). The Selective Repeat Protocol allows as many frames as the size of the receive window to arrive out of order and be kept until there is a set of in-order frames to be delivered to the network layer. Because the sizes of the send window and receive window are the same, all the frames in the send frame can arrive out of order and be stored until they can be delivered. We need, however, to mention that the receiver never delivers packets out of order to the network layer. Figure below shows the receive window in this protocol. Those slots inside the window that are colored define frames that have arrived out of order and are waiting for their neighbors to arrive before delivery to the network layer.



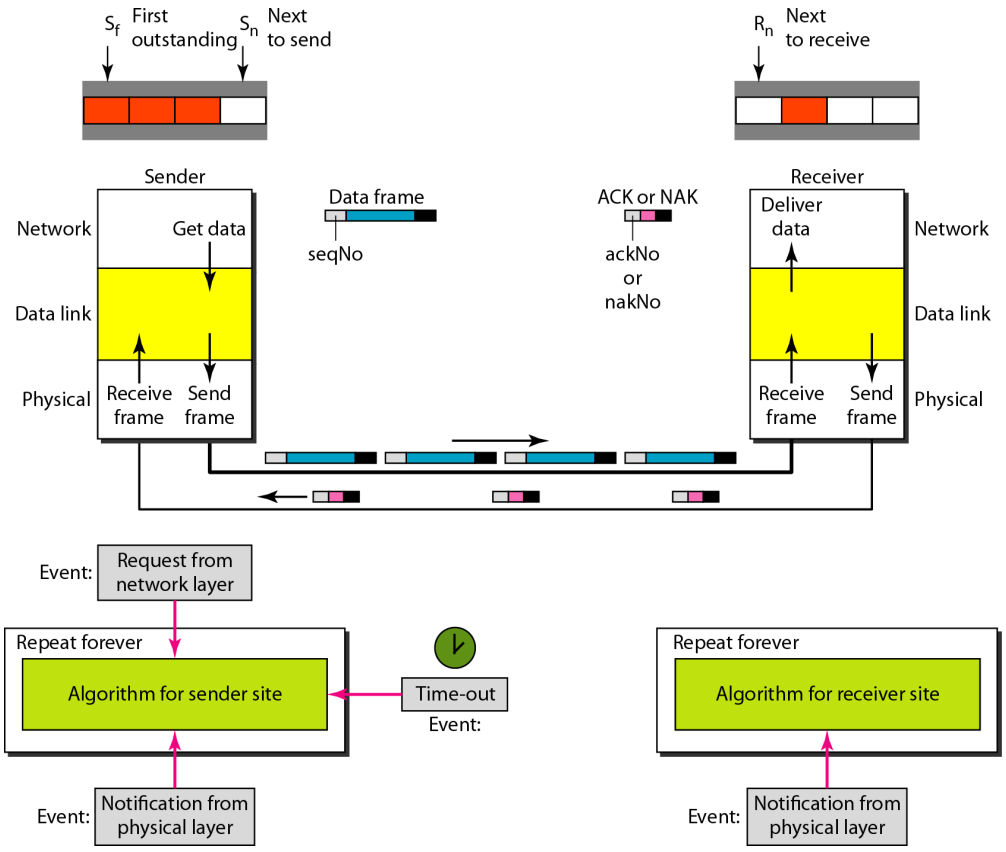
Design

The design in this case is to some extent similar to the one we described for the 00- Back-N, but more complicated, as shown in Figure below.

Window Sizes

We can now show why the size of the sender and receiver windows must be at most onehalf of 2m. For an example, we choose m = 2, which means the size of the window is 2m/2, or 2. Figure 11.21 compares a window size of 2 with a window size of 3.

If the size of the window is 2 and all acknowledgments are lost, the timer for frame 0 expires and frame 0 is resent. However, the window of the receiver is now expecting frame 2, not frame 0, so this duplicate frame is correctly discarded. When the size of the window is 3 and all acknowledgments are lost, the sender sends a duplicate of frame O. However, this time, the window of the receiver expects to receive frame 0 (0 is part of the window), so it accepts frame 0, not as a duplicate, but as the first frame in the next cycle. This is clearly an error



Algorithms

Algorithm below shows the procedure for the sender

```
1  Sw = 2^(m-1) ;
2  Sf = 0;
3  Sn = 0;
4
5  while (true)                          //Repeat forever
6  {
7    WaitForEvent();
8    if(Event(RequestToSend))            //There is a packet to send
9    {
10     if(Sn-Sf >= Sw)                   //If window is full
11           Sleep();
12     GetData();
13     MakeFrame(Sn);
14     StoreFrame(Sn);
15     SendFrame(Sn);
16     Sn = Sn + 1;
17     StartTimer(Sn);
18   }
19
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(sf < ackNo)
35          {
36           Purge(sf);
37           StopTimer(sf);
38           Sf = Sf + 1;
39          }
40        }
41   }
42
43   if(Event(TimeOut(t)))               //The timer expires
44   {
45    StartTimer(t);
46    SendFrame(t);
47   }
48 }
```

Analysis

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, we just resend the corresponding frame. If a valid ACK arrives, we use a loop to purge the buffers, stop the corresponding timer. and move the left wall of the window. The time-out event is simpler here; only the frame which times out is resent.

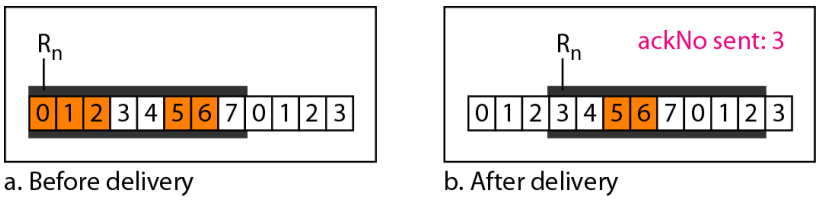
Algorithm below shows the procedure for the receiver.

```
1  Rn = 0;
2  NakSent = false;
3  AckNeeded = false;
4  Repeat(for all slots)
5      Marked(slot) = false;
6
7  !while (true)                                  IIRepeat forever
8  {
9    WaitForEvent()i
10
11   if{Event{ArrivalNotification»                jData frame arrives
12   {
13       Receive(Frame);
14       if(corrupted(Frame»&& (NOT NakSent)
15       {
16        SendNAK(Rn);
17        NakSent = true;
18        Sleep{);
19       }
20       if(seqNo <> Rn)&& (NOT NakSent)
21       {
22        SendNAK(Rn);
23        NakSent = true;
24        if {(seqNo in window)&&(IMarked(seqNo»
25        {
26         StoreFrame{seqNo)
27         Marked(seqNo)= true;
28         while(Marked(Rn)
29         {
30          DeliverData(Rn);
31          Purge(Rn);
32          Rn = Rn + 1;
33          AckNeeded = true;
34         }
35          if(AckNeeded);
36          {
37          SendAck(Rn);
38          AckNeeded = false;
39          NakSent = false;
40          }
41        }
42       }
43   }
44 }
```

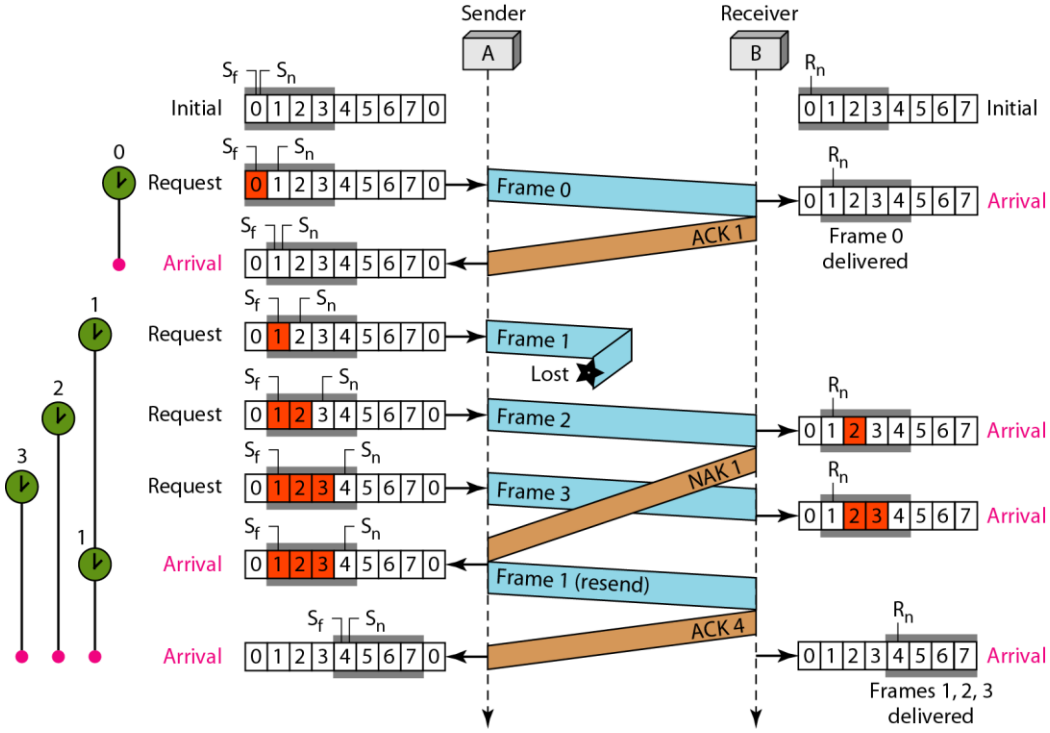
Analysis

Here we need more initialization. In order not to overwhelm the other side with NAKs, we use a variable called NakSent. To know when we need to send an ACK, we use a variable called AckNeeded. Both of these are initialized to false. We also use a set of variables to mark the slots in the receive window once the corresponding frame has arrived and is stored. If we

receive a corrupted frame and a NAK has not yet been sent, we send a NAK to tell the other site that we have not received the frame we expected. If the frame is not corrupted and the sequence number is in the window, we store the frame and mark the slot. If contiguous frames, starting from Rn have been marked, we deliver their data to the network layer and slide the window. Figure below shows this situation



Flow diagram



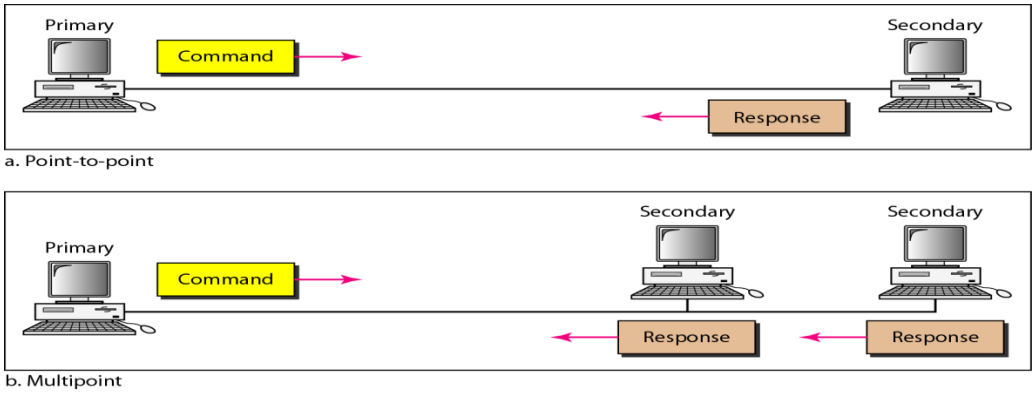
High-level Data Link Control (HDLC) Protocol

High-level Data Link Control (HDLC) is a bit-oriented protocol for communication over point-to-point and multipoint links. It implements the ARQ mechanisms we discussed in this chapter.

Configurations and Transfer Modes

HDLC provides two common transfer modes that can be used in different configurations**: normal response mode (NRM) and asynchronous balanced mode (ABM).**

In normal response mode (NRM), the station configuration is unbalanced. We have one primary station and multiple secondary stations. A primary station can send commands; a secondary station can only respond. The NRM is used for both point-to-point and multiple-point links, as shown in Figure.



Asynchronous Balanced Mode

In asynchronous balanced mode (ABM), the configuration is balanced. The link is point-to-point, and each station can function as a primary and a secondary (acting as peers), as shown in Figure This is the common mode today.

Frames

HDLC defines three types of frames: information frames (I-frames), supervisory frames (S-frames), and unnumbered frames (U-frames). Each type of frame serves as an envelope for the transmission of a different type of message.

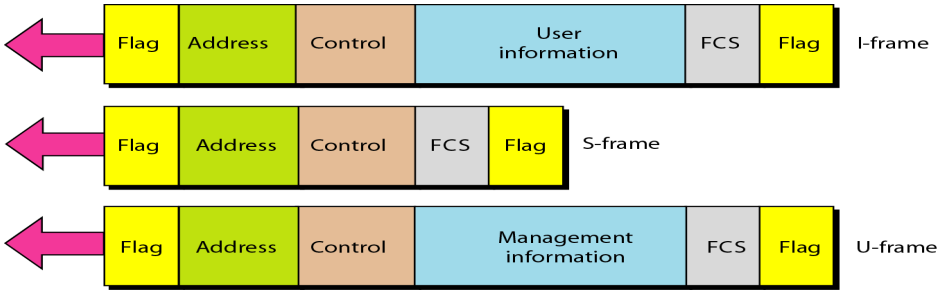
- I-frames are used to transport user data and control information relating to user data (piggybacking).
- S-frames are used only to transport control information.
- U-frames are reserved for system management. Information carried by U-frames is intended for managing the link itself.

Frame Format

Each frame in HDLC may contain up to six fields, as shown in Figure : a beginning flag field, an address field, a control field, an information field, a frame check sequence (FCS) field, and an ending flag field. In multiple-frame transmissions, the ending flag of one frame can serve as the beginning flag of the next frame.

Flag field: The flag field of an HDLC frame is an 8-bit sequence with the bit pattern 01111110 that identifies both the beginning and the end of a frame and serves as a synchronization pattern for the receiver.

Address field: The second field of an HDLC frame contains the address of the secondary station. An address field can be 1 byte or several bytes long, depending on the needs of the network.

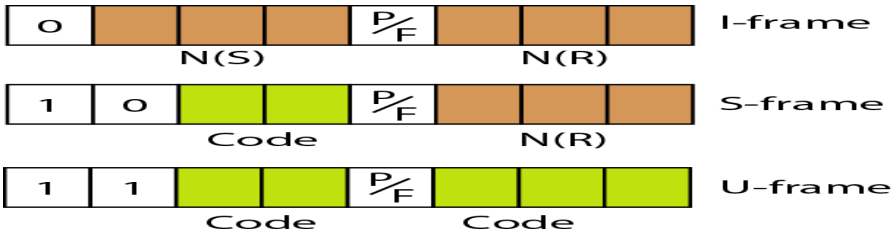


Control field. The control field is a 1- or 2-byte segment of the frame used for flow and error control

Information field. The information field contains the user's data from the network layer or management information. Its length can vary from one network to another.

FCS field. The frame check sequence (FCS) is the HDLC error detection field. It can contain either a 2- or 4-byte ITU-T CRC.

Control Field The control field determines the type of frame and defines its functionality.



Control Field for I-Frames

I-frames are designed to carry user data from the network layer. In addition, they can include flow and error control information (piggybacking). The subfields in the control field are used to define these functions. The first bit defines the type. If the first bit of the control field is 0, this means the frame is an I-frame. The next 3 bits, called N(S), define the sequence number of the frame. Note that with 3 bits, we can define a sequence number between 0 and 7 .The last 3 bits, called N(R), correspond to the acknowledgment number when piggybacking is used. The single bit between N(S) and N(R) is called the P/F bit. The P/F field is a single bit with a dual purpose. It has meaning only when it is set (bit = 1) and can mean poll or final. It means poll when the frame is sent by a primary station to a secondary (when the address field contains the address of the receiver). It means final when the frame is sent by a secondary to a primary (when the address field contains the address of the sender).

Control Field for S-Frames

Code	Description	N(R)
00	Receive Ready (RR)	ACK number
10	Receive Not Ready (RNR)	ACK number
01	Reject (REJ)	NAK
11	Selective Reject (SREJ)	NAK

S-frames do not have information fields. If the first 2 bits of the control field is 10, this means the frame is an S-frame. The last 3 bits, called N(R), corresponds to the acknowledgment number (ACK) or negative acknowledgment number (NAK) depending on the type of S-frame. The 2 bits called code is used to define the type of S-frame itself. With 2 bits, we can have four types of S-frames.

Receive ready (RR). If the value of the code subfield is 00, it is an RR S-frame. This kind of frame acknowledges the receipt of a safe and sound frame or group of frames. In this case, the value N(R) field defines the acknowledgment number.

Receive not ready (RNR). If the value of the code subfield is 10, it is an RNR S-frame. It acknowledges the receipt of a frame or group of frames, and it announces that the receiver is busy and cannot receive more frames. It acts as a kind of congestion control mechanism by asking the sender to slow down.

Reject (REJ). If the value of the code subfield is 01, it is a REJ S-frame. This is a NAK frame, but not like the one used for Selective Repeat ARQ. It is a NAK that can be used in Go-Back-N ARQ to improve the efficiency of the process by informing the sender, before the sender time expires, that the last frame is lost or damaged. The value of N(R) is the negative acknowledgment number.

Selective reject (SREJ). If the value of the code subfield is 11, it is an SREJ S-frame. This is a NAK frame used in Selective Repeat ARQ. Note that the HDLC Protocol uses the term selective reject instead of selective repeat. The value of N(R) is the negative acknowledgment number.

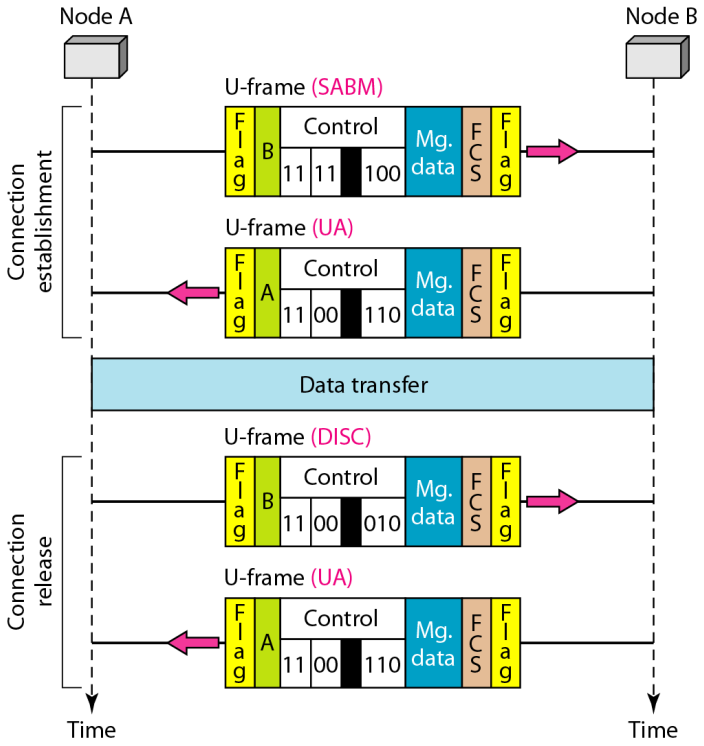
Control Field for U-Frames

Unnumbered frames are used to exchange session management and control information between connected devices. Unlike S-frames, U-frames contain an information field, but one used for system management information, not user data. As with S-frames, however, much of the information carried by U-frames is contained in codes included in the control field. U-frame codes are divided into two sections: a 2-bit prefix before the P/F bit and a 3-bit suffix after the P/F bit. Together, these two segments (5 bits) can be used to create up to 32 different types of U-frames. Some of the more common types are shown in Table

Code	*Command*	*Response*	*Meaning*
00 001	SNRM		Set normal response mode
11 011	SNRME		Set normal response mode, extended
11 100	SABM	**DM**	Set asynchronous balanced mode or **disconnect mode**
11 110	SABME		Set asynchronous balanced mode, extended
00 000	UI	**UI**	Unnumbered information
00 110		**UA**	**Unnumbered acknowledgment**
00 010	DISC	**RD**	Disconnect or **request disconnect**
10 000	SIM	**RIM**	Set initialization mode or **request information mode**
00 100	UP		Unnumbered poll
11 001	RSET		Reset
11 101	XID	**XID**	Exchange ID
10 001	FRMR	**FRMR**	Frame reject

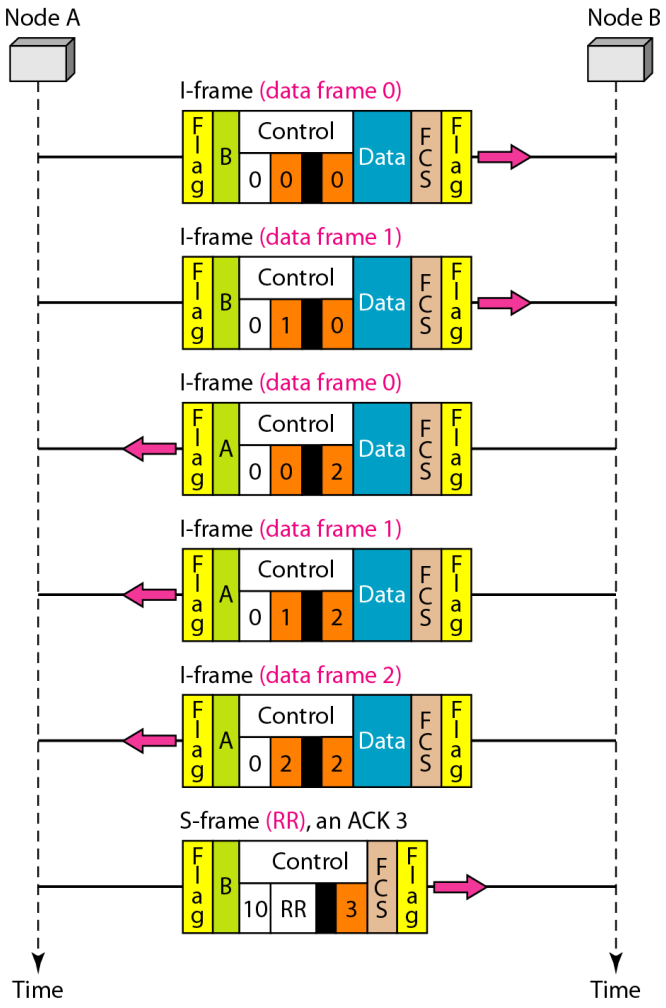
Figure below shows how U-frames can be used for connection establishment and connection release. Node A asks for a connection with a set asynchronous balanced mode (SABM) frame; node B gives a positive response with an unnumbered acknowledgment (UA) frame. After these two exchanges, data can be transferred between the two nodes. After data transfer, node A sends a DISC (disconnect) frame to release the connection; it is confirmed by node B responding with a UA (unnumbered acknowledgment)

Example of connection and disconnection



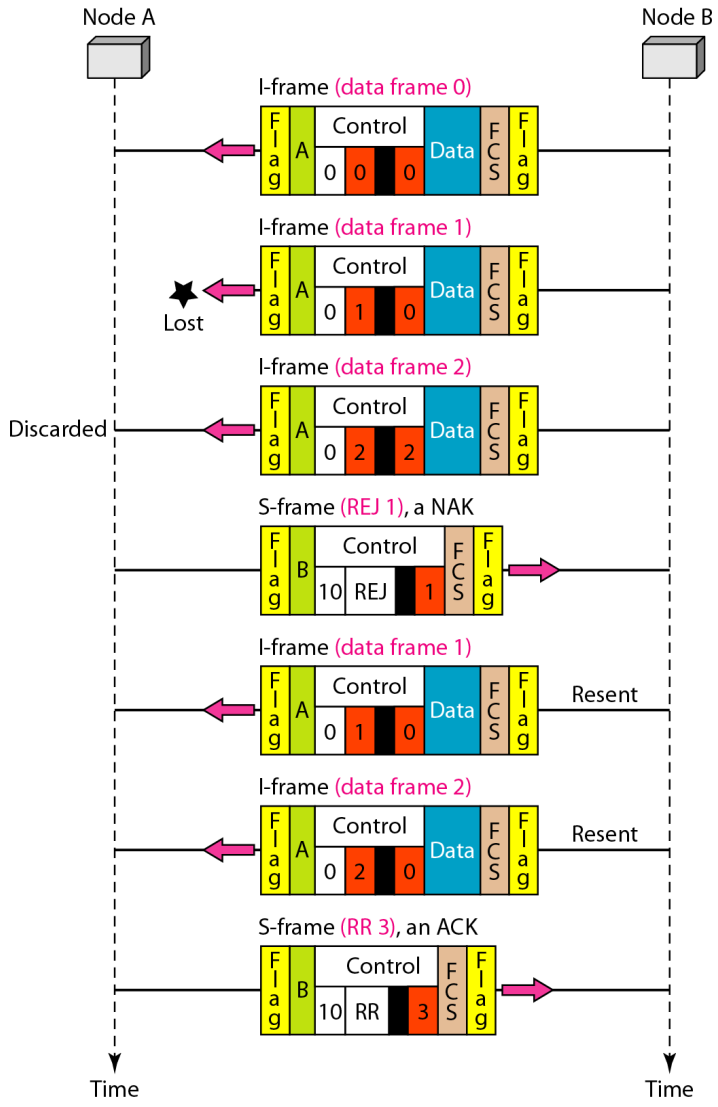
Piggybacking without Error

Figure below shows an exchange using piggybacking. Node A begins the exchange of information with an I-frame numbered 0 followed by another I-frame numbered 1. Node B piggybacks its acknowledgment of both frames onto an I-frame of its own. Node B's first I-frame is also numbered 0 [N(S) field] and contains a 2 in its N(R) field, acknowledging the receipt of Ns frames 1 and 0 and indicating that it expects frame 2 to arrive next. Node B transmits its second and third I-frames (numbered 1 and 2) before accepting further frames from node A. Its N(R) information, therefore, has not changed: B frames 1 and 2 indicate that node B is still expecting Ns frame 2 to arrive next. Node A has sent all its data. Therefore, it cannot piggyback an acknowledgment onto an I-frame and sends an S-frame instead. The RR code indicates that A is still ready to receive. The number 3 in the N(R) field tells B that frames 0, 1, and 2 have all been accepted and that A is now expecting frame number 3



Piggybacking with Error

Figure below shows an exchange in which a frame is lost. Node B sends three data frames (0, 1, and 2), but frame 1 is lost. When node A receives frame 2, it discards it and sends a REI frame for frame 1. Note that the protocol being used is Go-Back-N with the special use of an REI frame as a NAK frame. The NAK frame does two things here: It confirms the receipt of frame ° and declares that frame 1 and any following frames must be resent. Node B, after receiving the REI frame, resends frames 1 and 2. Node A acknowledges the receipt by sending an RR frame (ACK) with acknowledgment number 3.



POINT-TO-POINT PROTOCOL (PPP)

HDLC is a general protocol that can be used for both point-to-point and multipoint configurations; one of the most common protocols for point-to-point access is the Point-to-Point Protocol (PPP).

PPP provides several services:

1. PPP defines the format of the frame to be exchanged between devices.

2. PPP defines how two devices can negotiate the establishment of the link and the exchange of data.

3. PPP defines how network layer data are encapsulated in the data link frame.

4. PPP defines how two devices can authenticate each other.

5. PPP provides multiple network layer services supporting a variety of network layer protocols.

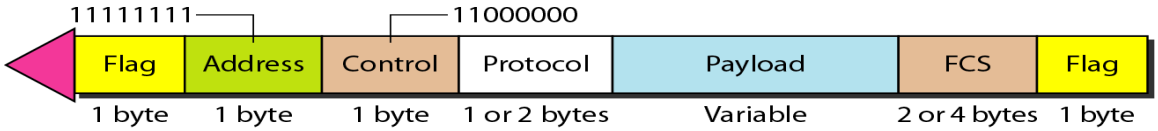
6. PPP provides connections over multiple links.

7. PPP provides network address configuration. This is particularly useful when a home user needs a temporary network address to connect to the Internet.

Framing

PPP is a byte-oriented protocol

Frame Format



Flag. A PPP frame starts and ends with a 1-byte flag with the bit pattern 01111110. Although this pattern is the same as that used in HDLC, there is a big difference. PPP is a byte-oriented protocol; HDLC is a bit-oriented protocol. The flag is treated as a byte.

Address. The address field in this protocol is a constant value and set to 11111111 (broadcast address). During negotiation (discussed later), the two parties may agree to omit this byte.

Control. This field is set to the constant value 11000000. PPP does not provide any flow control. Error control is also limited to error detection.

Protocol. The protocol field defines what is being carried in the data field: either user data or other information. This field is by default 2 bytes long, but the two parties can agree to use only 1 byte.

Payload field. This field carries either the user data or other information.

FCS. The frame check sequence (FCS) is simply a 2-byte or 4-byte standard CRC

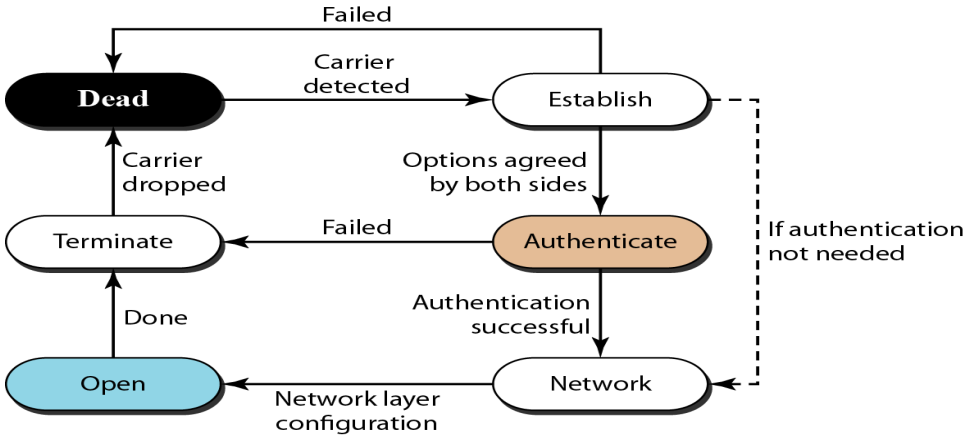
Byte Stuffing

PPP is a byte-oriented protocol using byte

stuffing with the escape byte 01111101.

Transition Phases

A PPP connection goes through phases which can be shown in a transition phase diagram



Dead. In the dead phase the link is not being used. There is no active carrier (at the physical layer) and the line is quiet.

Establish. When one of the nodes starts the communication, the connection goes into this phase. In this phase, options are negotiated between the two parties. If the negotiation is successful, the system goes to the authentication phase (if authentication is required) or directly to the networking phase.

Authenticate. The authentication phase is optional; the two nodes may decide, during the establishment phase, not to skip this phase. However, if they decide to proceed with authentication, they send several authentication packets. If the result is successful, the connection goes to the networking phase; otherwise, it goes to the termination phase.

Network. In the network phase, negotiation for the network layer protocols takes place. PPP specifies that two nodes establish a network layer agreement before data at the network layer can be exchanged. The reason is that PPP supports multiple protocols at the network layer. If a node is running multiple protocols simultaneously at the network layer, the receiving node needs to know which protocol will receive the data.

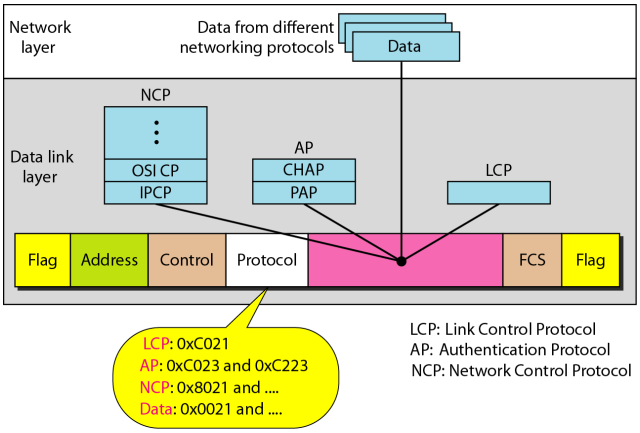
Open. In the open phase, data transfer takes place. When a connection reaches this phase, the exchange of data packets can be started. The connection remains in this phase until one of the endpoints wants to terminate the connection.

Terminate. In the termination phase the connection is terminated. Several packets are exchanged between the two ends for house cleaning and closing the link.

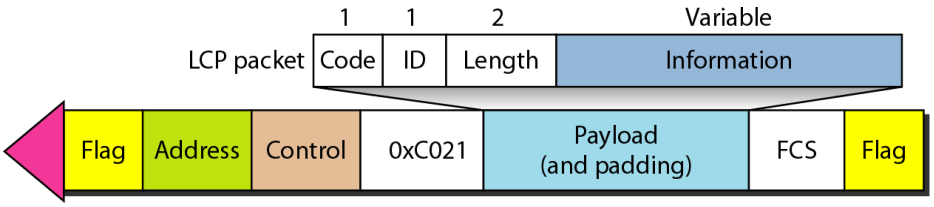
Multiplexing in PPP

Although PPP is a data link layer protocol, PPP uses another set of other protocols to establish the link, authenticate the parties involved, and carry the network layer data. Three sets of protocols are defined to make PPP powerful: the Link Control Protocol (LCP), two Authentication Protocols (APs), and several Network Control Protocols (NCPs). At any moment, a PPP packet can carry data from one of these protocols in its data field, as shown in Figure below. Note that there is one LCP, two APs, and several NCPs. Data may also come from several different network layers.

Link Control Protocol



The Link Control Protocol (LCP) is responsible for establishing, maintaining, configuring, and terminating links. It also provides negotiation mechanisms to set options between the two endpoints. Both endpoints of the link must reach an agreement about the options before the link can be established. See Figure below All LCP packets are carried in the payload field of the PPP frame with the protocol field set to C021 in hexadecimal.



The code field defines the type of LCP packet. There are 11 types of packets as shown in Table

Table 11.2 ***LCP packets***

Code	*Packet Type*	*Description*
0x01	Configure-request	Contains the list of proposed options and their values
0x02	Configure-ack	Accepts all options proposed
0x03	Configure-nak	Announces that some options are not acceptable
0x04	Configure-reject	Announces that some options are not recognized
0x05	Terminate-request	Request to shut down the line
0x06	Terminate-ack	Accept the shutdown request
0x07	Code-reject	Announces an unknown code
0x08	Protocol-reject	Announces an unknown protocol
0x09	Echo-request	A type of hello message to check if the other end is alive
0x0A	Echo-reply	The response to the echo-request message
0x0B	Discard-request	A request to discard the packet

Authentication Protocols

Authentication plays a very important role in PPP because PPP is designed for use over dial-up links where verification of user identity is necessary. Authentication means validating the identity of a user who needs to access a set of resources. PPP has created two protocols for authentication: Password Authentication Protocol and Challenge Handshake Authentication Protocol. Note that these protocols are used during the authentication phase

PAP

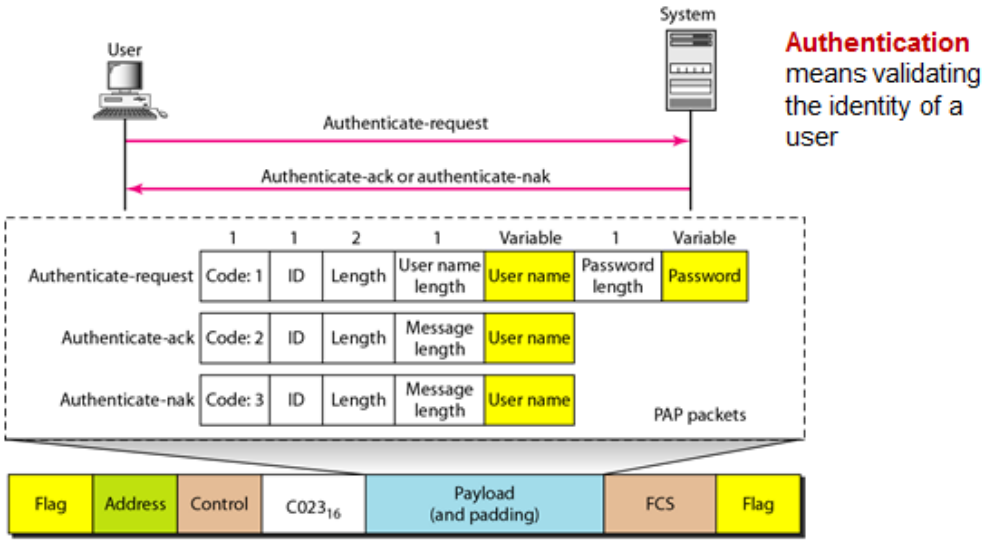
The Password Authentication Protocol (PAP) is a simple authentication procedure with a two-step process:

1. The user who wants to access a system sends an authentication identification (usually the user name) and a password.
2. The system checks the validity of the identification and password and either accepts or denies connection.

Figure below shows the three types of packets used by PAP and how they are actually exchanged. When a PPP frame is carrying any PAP packets, the value of the protocol field is OxC023. The three PAP packets are authenticate-request, authenticate-ack, and authenticate-nak.

The first packet is used by the user to send the user name and password. The second is used by the system to allow access. The third is used by the system to deny access.

Figure 11.36 *PAP packets encapsulated in a PPP frame*



PAP - Password Authentication Protocol
CHAP -Challenge Handshake Authentication Protocol

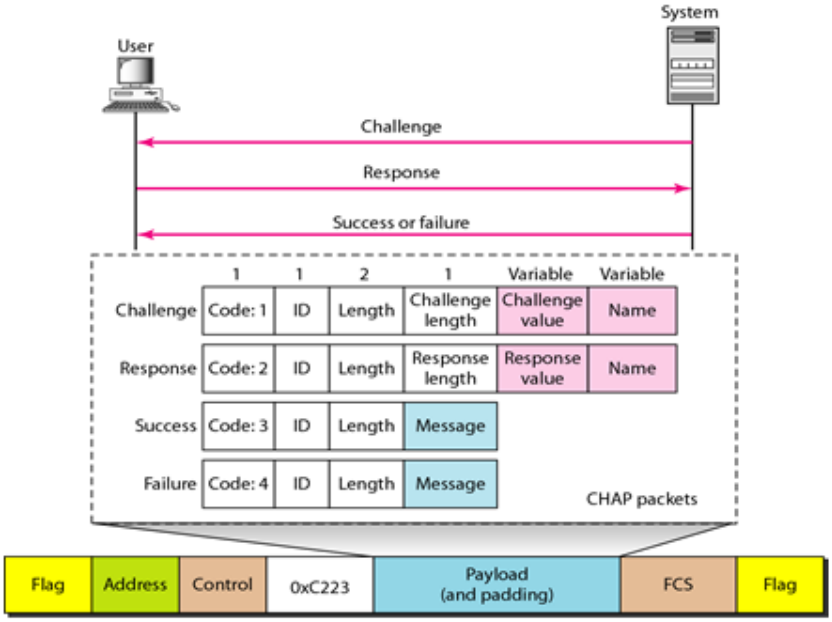
CHAP packets encapsulated in a PPP frame

The Challenge Handshake Authentication Protocol (CHAP) is a three-way hand-shaking authentication protocol that provides greater security than PAP. In this method, the password is kept secret; it is never sent online.

1. The system sends the user a challenge packet containing a challenge value, usually a few bytes.

2. The user applies a predefined function that takes the challenge value and the user's own password and creates a result. The user sends the result in the response packet to the system.

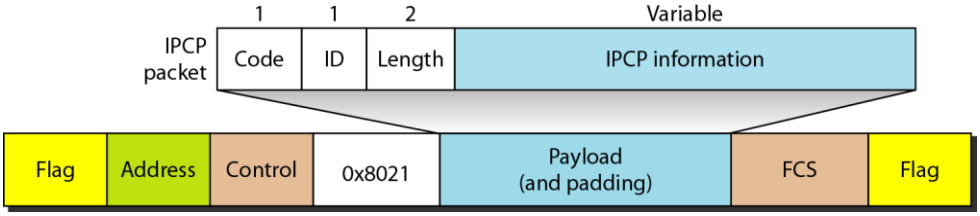
Figure 11.37 *CHAP packets encapsulated in a PPP frame*



3. The system does the same. It applies the same function to the password of the user (known to the system) and the challenge value to create a result. If the result created is the same as the result sent in the response packet, access is granted; otherwise, it is denied. CHAP is more secure than PAP, especially if the system continuously changes the challenge value. Even if the intruder learns the challenge value and the result, the password is still secret. Figure above shows the packets and how they are used

IPCP packet encapsulated in PPP frame

One NCP protocol is the Internet Protocol Control Protocol (IPCP). This protocol configures the link used to carry IP packets in the Internet. IPCP is especially of interest to us. The format of an IPCP packet is shown in Figure below. Note that the value of the protocol field in hexadecimal is 8021



Other Protocols

There are other NCP protocols for other network layer protocols. The OSI Network Layer Control Protocol has a protocol field value of 8023; the Xerox NS IDP Control Protocol has a protocol field value of 8025; and so on. The value of the code and the format of the packets for these other protocols are the same as shown in Table

Table 11.4 *Code value for IPCP packets*

Code	*IPCP Packet*
0x0!	Configure-request
0x02	Configure-ack
0x03	Configure-nak
0x04	Configure-reject
0x05	Terminate-request
0x06	Terminate-ack
0x07	Code-reject