Unit-II

Data Link Layer: Need, Services Provided, Framing, Flow Control, Error control. Data Link Layer Protocol: Elementary & Sliding Window protocol: 1-bit, Go-Back-N, Selective Repeat, Hybrid ARQ. Protocol verification: Finite State Machine Models & Petri net models. ARP/RARP/GARP

SERVICES OF DATA ANALYTICS

The **data link layer** transforms the physical layer, a raw transmission facility, to a reliable link. It makes the physical layer appear error-free to the upper layer (network layer). Other responsibilities of the data link layer include the following:

Framing: The data link layer divides the stream of bits received from the network layer into manageable data units called **frames.**

Physical addressing: If frames are to be distributed to different systems on the network, the data link layer adds a header to the frame to define the sender and/or receiver of the frame. If the frame is intended for a system outside the sender's network, the receiver address is the address of the connecting device that connects the network to the next one.

Flow control: If the rate at which the data is absorbed by the receiver is less than the rate produced at the sender, the data link layer imposes a flow control mechanism to prevent overwhelming the receiver.

Error control: The data link layer adds reliability to the physical layer by adding mechanisms to detect and retransmit damaged or lost frames. It also uses a mechanism to recognize duplicate frames. Error control is normally achieved through a trailer added to the end of the frame.

Access control: When two or more devices are connected to the same link, data link layer protocols are necessary to determine which device has control over the link at any given time.

DATA LINK LAYER DESIGN ISSUES

The data link layer uses the services of the physical layer to send and receive bits over communication channels. It has a number of functions, including:

1. Providing a well-defined service interface to the network layer.
2. Dealing with transmission errors.
3. Regulating the flow of data so that slow receivers are not swamped by fast senders.

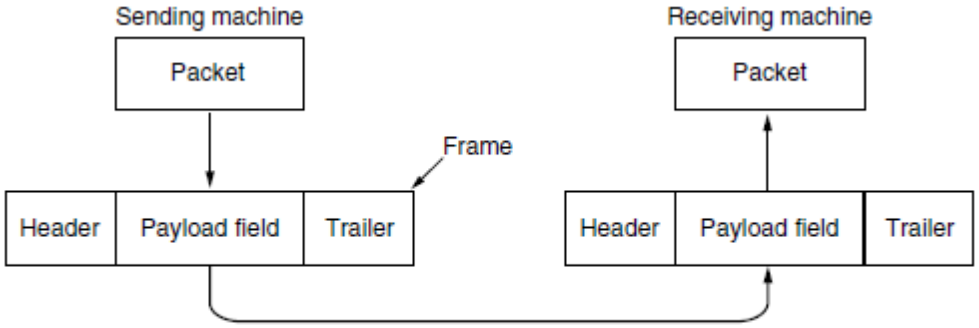


Figure 3-1. Relationship between packets and frames.

To accomplish these goals, the data link layer takes the packets it gets from the network layer and encapsulates them into **frames** for transmission. Each frame contains a frame header, a payload field for holding the packet, and a frame trailer, as illustrated in Fig. 3-1. Frame management forms the heart of what the data link layer does. In the following sections we will examine all the above mentioned issues in detail.

Services Provided to the Network Layer

The function of the data link layer is to provide services to the network layer. The principal service is transferring data from the network layer on the source machine to the network layer on the destination machine. On the source machine is an entity, call it a process, in the network layer that hands some bits to the data link layer for transmission to the destination. The job of the data link layer is to transmit the bits to the destination machine so they can be handed over to the network layer there, as shown in Fig. 3-2(a). The actual transmission follows the path of Fig. 3-2(b), but it is easier to think in terms of two data link layer processes communicating using a data link protocol.

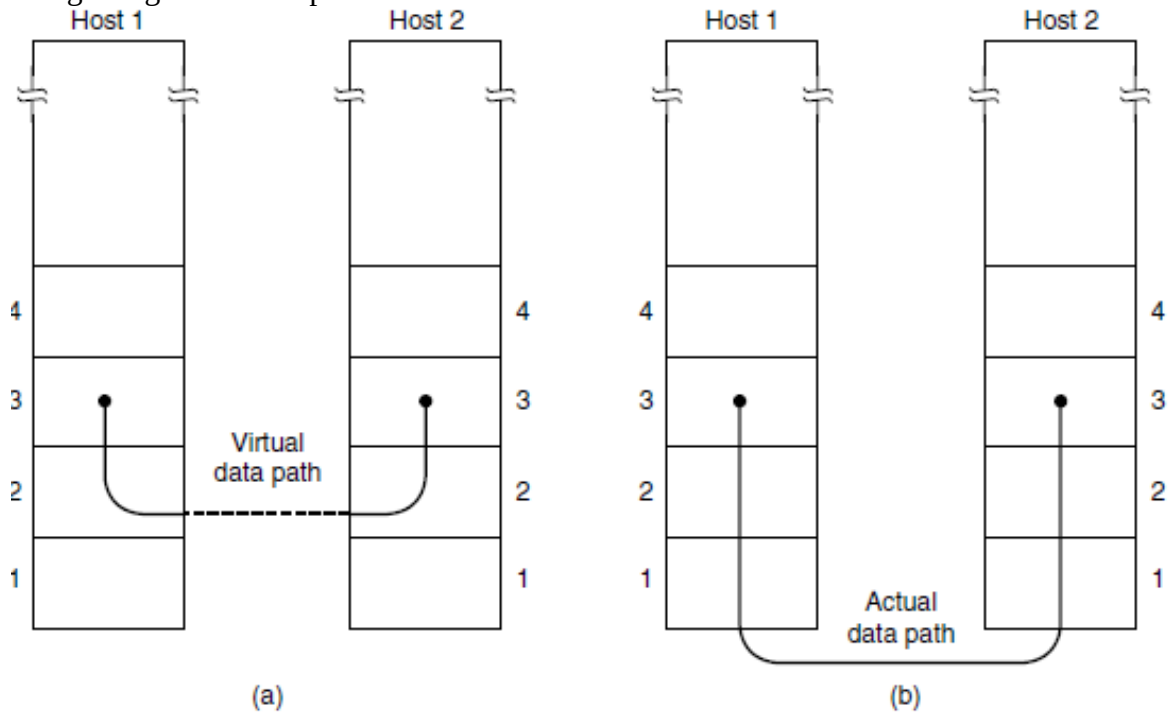


Figure 3-2. (a) Virtual communication. (b) Actual communication.

The data link layer can be designed to offer various services. The actual services that are offered vary from protocol to protocol. Three reasonable possibilities that we will consider in turn are:

1. Unacknowledged connectionless service: Unacknowledged connectionless service consists of having the source machine send independent frames to the destination machine without having the destination machine acknowledge them. Ethernet is a good example of a data link layer that provides this class of service. No logical connection is established beforehand or released afterward.

2. Acknowledged connectionless service: The next step up in terms of reliability is acknowledged connectionless service. When this service is offered, there are still no logical connections used, but each frame sent is individually acknowledged. In this way, the sender knows whether a frame has arrived correctly or been lost. If it has not arrived within a specified time interval, it can be sent again.

3. Acknowledged connection-oriented service: When connection-oriented service is used, transfers go through three distinct phases. In the first phase, the connection is established by having both sides initialize variables and counters needed to keep track of which frames have been received and which ones have not. In the second phase, one or more frames are actually transmitted. In the third and final phase, the connection is released, freeing up the variables, buffers, and other resources used to maintain the connection.

Framing

To provide service to the network layer, the data link layer must use the service provided to it by the physical layer. The usual approach is for the data link layer to break up the bit stream into discrete frames, compute a short token called a checksum for each frame, and include the checksum in the frame when it is transmitted. When a frame arrives at the destination, the checksum is recomputed. If the newly computed checksum is different from the one contained in the frame, the data link layer knows that an error has occurred and takes steps to deal with it (e.g., discarding the bad frame and possibly also sending back an error report).

Breaking up the bit stream into frames is more difficult than it at first appears. A good design must make it easy for a receiver to find the start of new frames while using little of the channel bandwidth. We will look at four methods:

1. Byte count.
2. Flag bytes with byte stuffing.
3. Flag bits with bit stuffing.
4. Physical layer coding violations.

Byte Count

The first framing method uses a field in the header to specify the number of bytes in the frame. When the data link layer at the destination sees the byte count, it knows how many bytes follow and hence where the end of the frame is. This technique is shown in Fig. 3-3(a) for four small example frames of sizes 5, 5, 8, and 8 bytes, respectively.

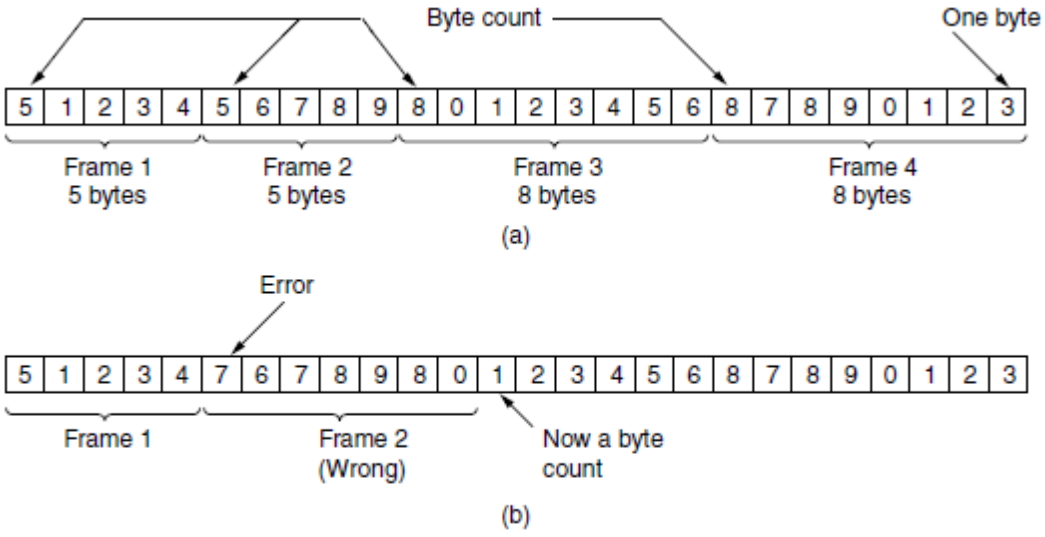


Figure 3-3. A byte stream. (a) Without errors. (b) With one error.

The trouble with this algorithm is that the count can be garbled by a transmission error. For example, if the byte count of 5 in the second frame of Fig. 3-3(b) becomes a 7 due to a single bit flip, the destination will get out of synchronization. It will then be unable to locate the correct start of the next frame. Even if the checksum is incorrect so the destination knows that the frame is bad, it still has no way of telling where the next frame starts.

Flag bytes with byte stuffing

The second framing method gets around the problem of resynchronization after an error by having each frame start and end with special bytes. Often the same byte, called a flag byte, is used as both the starting and ending delimiter. This byte is shown in Fig. 3-4(a) as FLAG. Two consecutive flag bytes indicate the end of one frame and the start of the next. Thus, if the receiver ever loses synchronization it can just search for two flag bytes to find the end of the current frame and the start of the next frame.

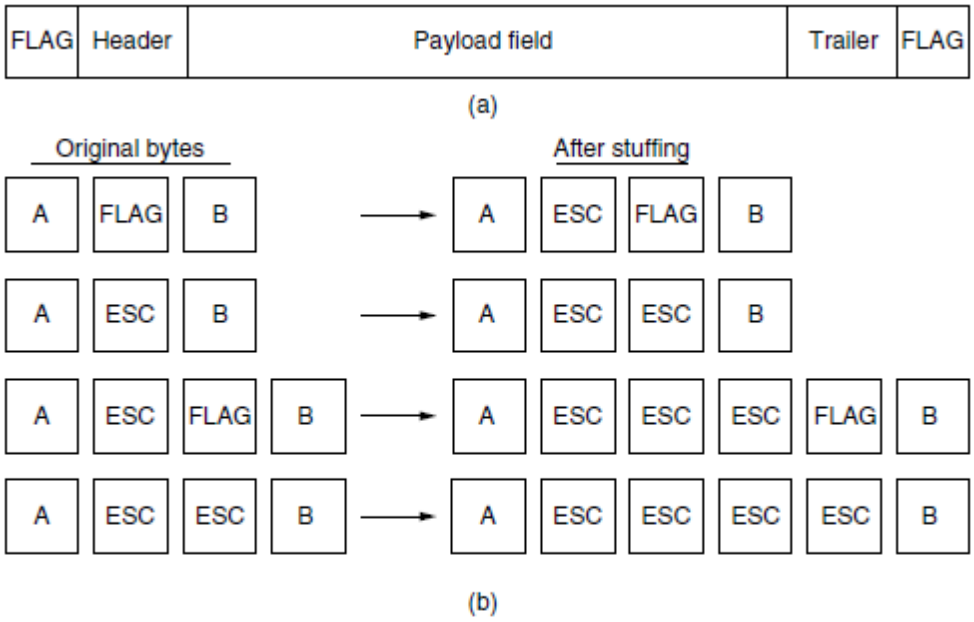


Figure 3-4. (a) A frame delimited by flag bytes. (b) Four examples of byte sequences before and after byte stuffing.

It may happen that the flag byte occurs in the data, especially when binary data such as photographs or songs are being transmitted. This situation would interfere with the framing. One way to solve this problem is to have the sender's data link layer insert a special escape byte (ESC) just before each ''accidental'' flag byte in the data. Thus, a framing flag byte can be distinguished from one in the data by the absence or presence of an escape byte before it. The data link layer on the receiving end removes the escape bytes before giving the data to the network layer. This technique is called **byte stuffing.**

Flag bits with bit stuffing

The third method of delimiting the bit stream gets around a disadvantage of byte stuffing, which is that it is tied to the use of 8-bit bytes. Framing can be also be done at the bit level, so frames can contain an arbitrary number of bits made up of units of any size. It was developed for the once very popular HDLC (Highlevel Data Link Control) protocol. Each frame begins and ends with a special bit pattern, 01111110 or 0x7E in hexadecimal. This pattern is a flag byte. Whenever the sender's data link layer encounters five consecutive 1s

in the data, it automatically stuffs a 0 bit into the outgoing bit stream. This bit stuffing is analogous to byte stuffing, in which an escape byte is stuffed into the outgoing character stream before a flag byte in the data. It also ensures a minimum density of transitions that help the physical layer maintain synchronization. USB (Universal Serial Bus) uses bit stuffing for this reason.

When the receiver sees five consecutive incoming 1 bits, followed by a 0 bit, it automatically destuffs (i.e., deletes) the 0 bit. Just as byte stuffing is completely transparent to the network layer in both computers, so is bit stuffing. If the user data contain the flag pattern, 01111110, this flag is transmitted as 011111010 but stored in the receiver's memory as 01111110. Figure 3-5 gives an example of bit stuffing.

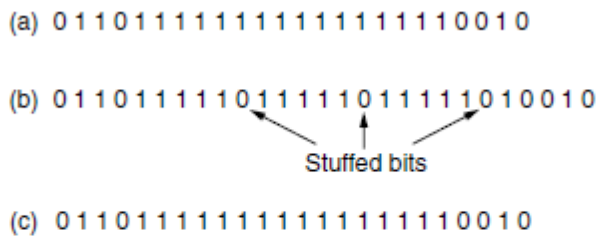


Figure 3-5. Bit stuffing. (a) The original data. (b) The data as they appear on the line. (c) The data as they are stored in the receiver's memory after destuffing.

Physical layer coding violations

The last method of framing is to use a shortcut from the physical layer. the encoding of bits as signals often includes redundancy to help the receiver. This redundancy means that some signals will not occur in regular data. For example, in the 4B/5B line code 4 data bits are mapped to 5 signal bits to ensure sufficient bit transitions. This means that 16 out of the 32 signal possibilities are not used. We can use some reserved signals to indicate the start and end of frames. In effect, we are using ‘‘coding violations’’ to delimit frames. The beauty of this scheme is that, because they are reserved signals, it is easy to find the start and end of frames and there is no need to stuff the data.

Error Control:

Having solved the problem of marking the start and end of each frame, we come to the next problem: how to make sure all frames are eventually delivered to the network layer at the destination and in the proper order. For unacknowledged connectionless service it might be fine if the sender just kept outputting frames without regard to whether they were arriving properly. But for reliable, connection-oriented service it would not be fine at all.

The usual way to ensure reliable delivery is to provide the sender with some feedback about what is happening at the other end of the line. Typically, the protocol calls for the receiver to send back special control frames bearing positive or negative acknowledgements about the incoming frames. If the sender receives a positive acknowledgement about a frame, it knows the frame has arrived safely. On the other hand, a negative acknowledgement means that something has gone wrong and the frame must be transmitted again.

An additional complication comes from the possibility that hardware troubles may cause a frame to vanish completely (e.g., in a noise burst). In this case, the receiver will not react at all, since it has no reason to react. Similarly, if the acknowledgement frame is lost, the sender will not know how to proceed. It should be clear that a protocol in which the sender transmits a frame and then waits for an acknowledgement, positive or

negative, will hang forever if a frame is ever lost due to, for example, malfunctioning hardware or a faulty communication channel.

This possibility is dealt with by introducing timers into the data link layer. When the sender transmits a frame, it generally also starts a timer. The timer is set to expire after an interval long enough for the frame to reach the destination, be processed there, and have the acknowledgement propagate back to the sender. Normally, the frame will be correctly received and the acknowledgement will get back before the timer runs out, in which case the timer will be canceled.

Flow Control

Another important design issue that occurs in the data link layer (and higher layers as well) is what to do with a sender that systematically wants to transmit frames faster than the receiver can accept them. This situation can occur when the sender is running on a fast, powerful computer and the receiver is running on a slow, low-end machine.

Two approaches are commonly used. In the first one, **feedback-based flow control**, the receiver sends back information to the sender giving it permission to send more data, or at least telling the sender how the receiver is doing. In the second one, **rate-based flow control**, the protocol has a built-in mechanism that limits the rate at which senders may transmit data, without using feedback from the receiver.

Various feedback-based flow control schemes are known, but most of them use the same basic principle. The protocol contains well-defined rules about when a sender may transmit the next frame. These rules often prohibit frames from being sent until the receiver has granted permission, either implicitly or explicitly. For example, when a connection is set up the receiver might say: "You may send me *n* frames now, but after they have been sent, do not send any more until I have told you to continue."

ELEMENTARY DATA LINK PROTOCOLS

To start with, we assume that the physical layer, data link layer, and network layer are independent processes that communicate by passing messages back and forth. A common implementation is shown in Fig. 3-10.

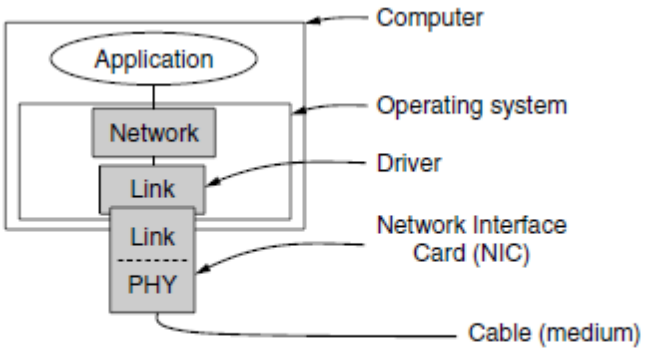


Figure 3-10. Implementation of the physical, data link, and network layers.

The physical layer process and some of the data link layer process run on dedicate hardware called a **NIC (Network Interface Card)**. The rest of the link layer process and the network layer process run on the main CPU as part of the operating system, with the software for the link layer process often taking the form of a device driver.

An unrestricted simplex protocol

In order to appreciate the step by step development of efficient and complex protocols such as SDLC, HDLC etc., we will begin with a simple but unrealistic protocol. In this protocol:

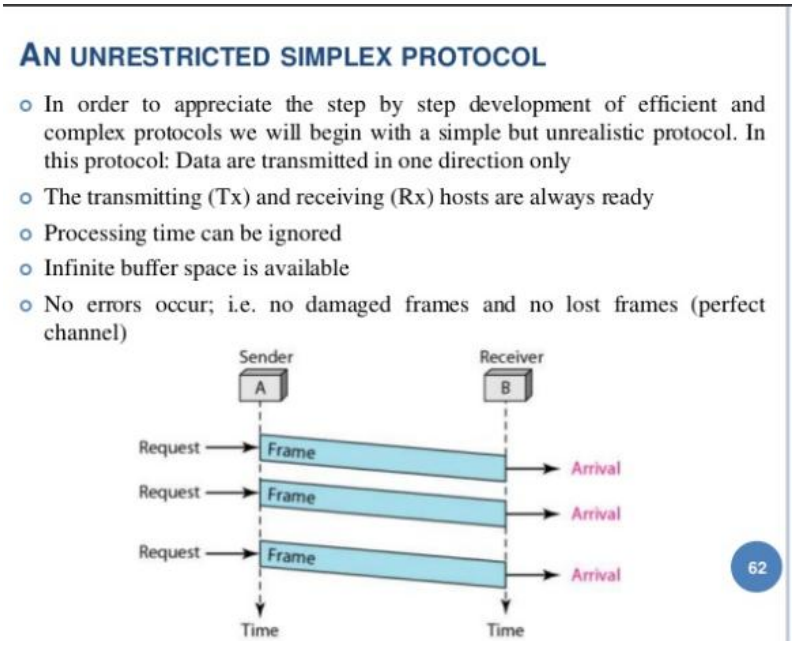
- Data are transmitted in one direction only
- The transmitting (Tx) and receiving (Rx) hosts are always ready
- Processing time can be ignored
- Infinite buffer space is available
- No errors occur; i.e. no damaged frames and no lost frames (perfect channel)

The protocol consists of two procedures, a sender and receiver as depicted below:

```
/* protocol 1 */

Sender()
{
      forever
      {
            from_host(buffer);
            S.info = buffer;
            sendf(S);
      }
}

Receiver()
{
      forever
      {
            wait(event);
            getf(R);
            to_host(R.info);
      }
}
```



A simplex stop-and-wait protocol

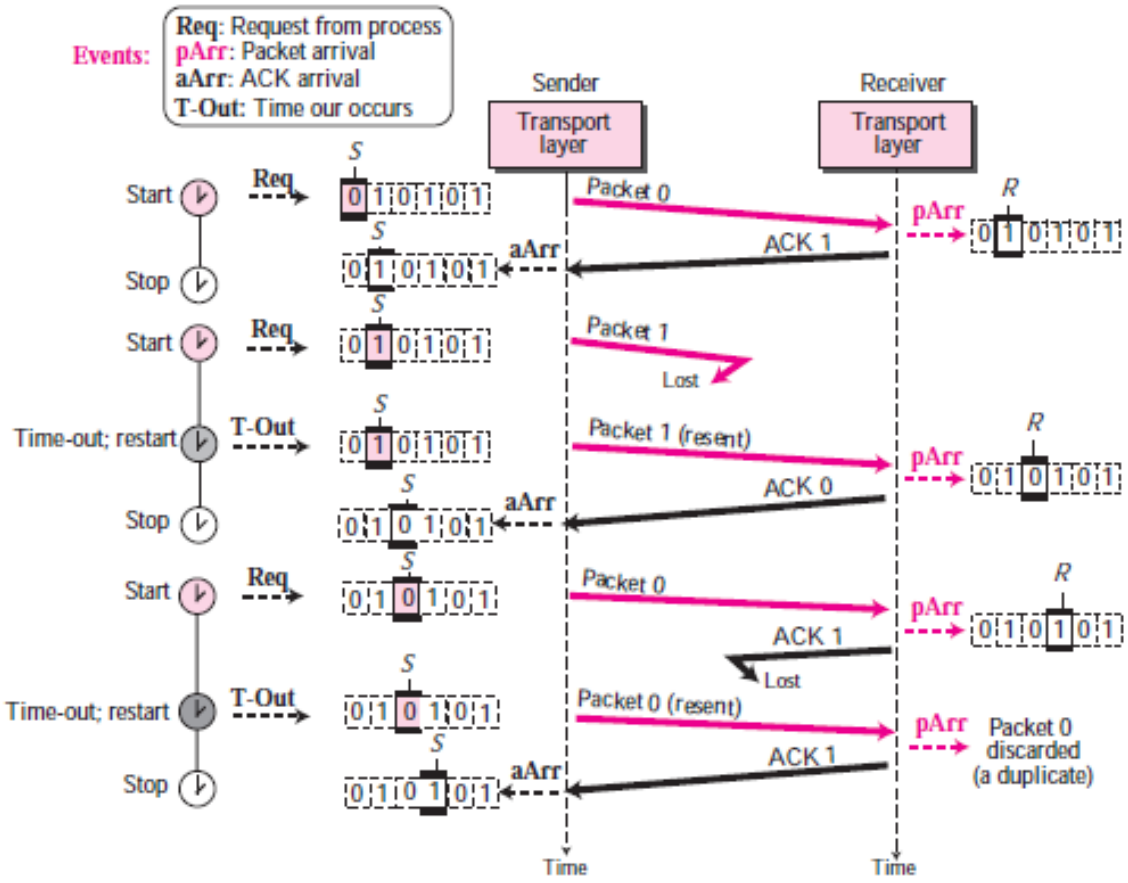
In this protocol we assume that

- Data are transmitted in one direction only
- No errors occur (perfect channel)
- The receiver can only process the received information at a finite rate

These assumptions imply that the transmitter cannot send frames at a rate faster than the receiver can process them.

The problem here is how to prevent the sender from flooding the receiver.

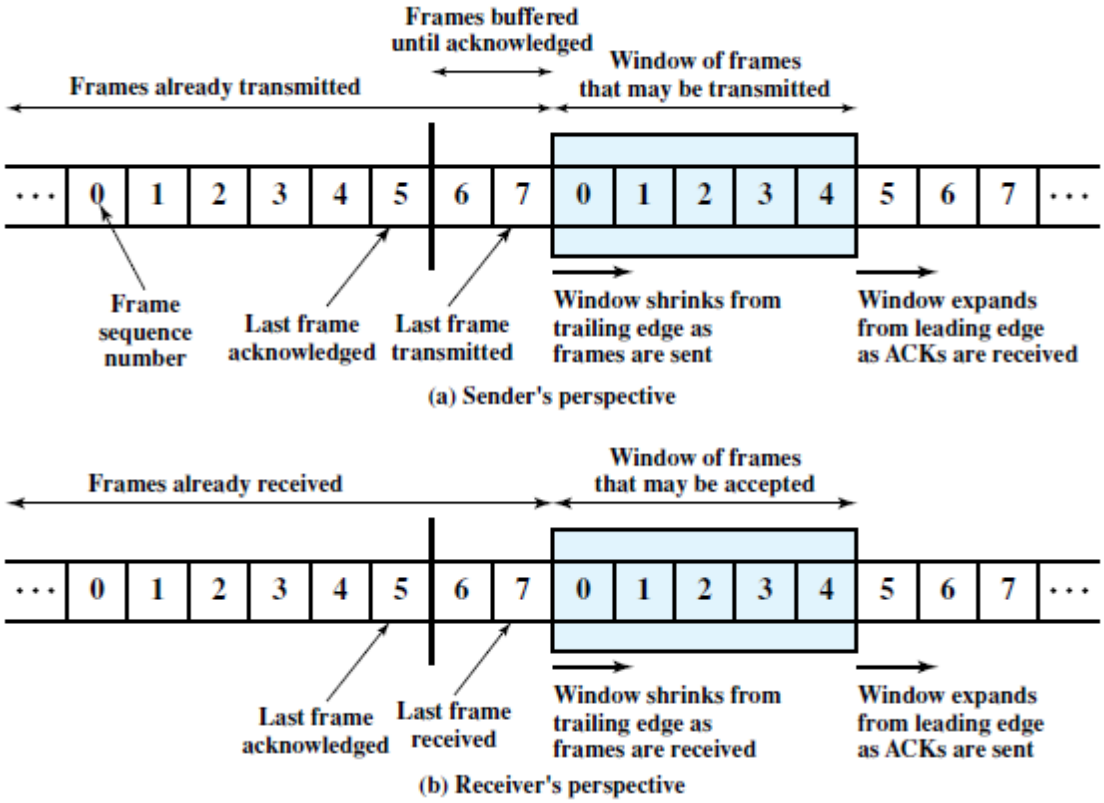
A general solution to this problem is to have the receiver provide some sort of feedback to the sender. The process could be as follows: The receiver send an acknowledge frame back to the sender telling the sender that the last received frame has been processed and passed to the host; permission to send the next frame is granted. The sender, after having sent a frame, must wait for the acknowledge frame from the receiver before sending another frame. This protocol is known as *stop-and-wait*.



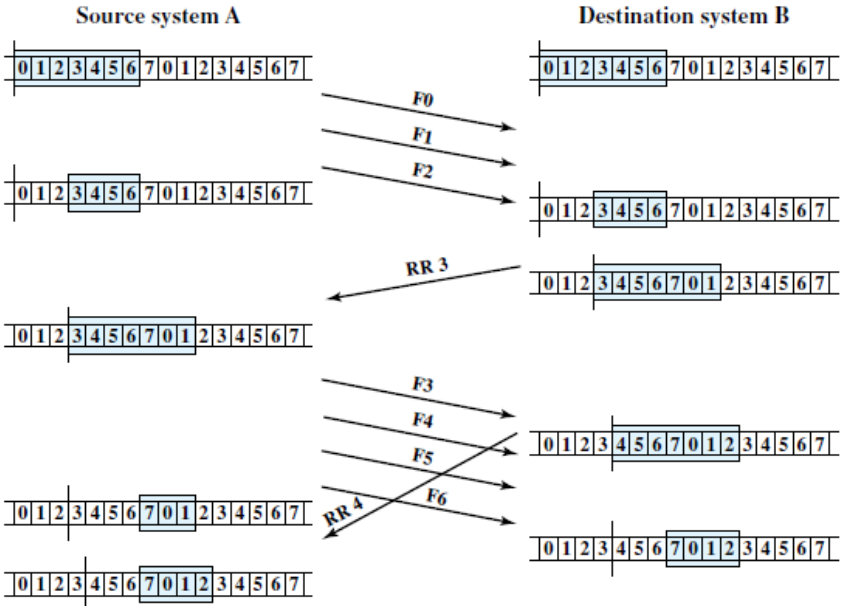
SLIDING WINDOW PROTOCOL:

In sliding window method, multiple frames are sent by sender at a time before needing an acknowledgment. Multiple frames sent by source are acknowledged by receiver using a single ACK frame. Figure below is a

useful way of depicting the sliding-window process. It assumes the use of a 3-bit sequence number, so that frames are numbered sequentially from 0 through 7, and then the same numbers are reused for subsequent frames.



If two stations exchange data, each needs to maintain two windows, one for transmit and one for receive, and each side needs to send the data and acknowledgments to the other. To provide efficient support for this requirement, a feature known as **piggybacking** is typically provided. Each **data frame** includes a field that holds the sequence number of that frame plus a field that holds the sequence number used for acknowledgment.



A One-Bit Sliding Window Protocol

A sliding window protocol with a maximum window size of 1 is called a one-bit sliding window protocol. Such a protocol uses stop-and-wait since the sender transmits a frame and waits for its acknowledgement before sending the next one.

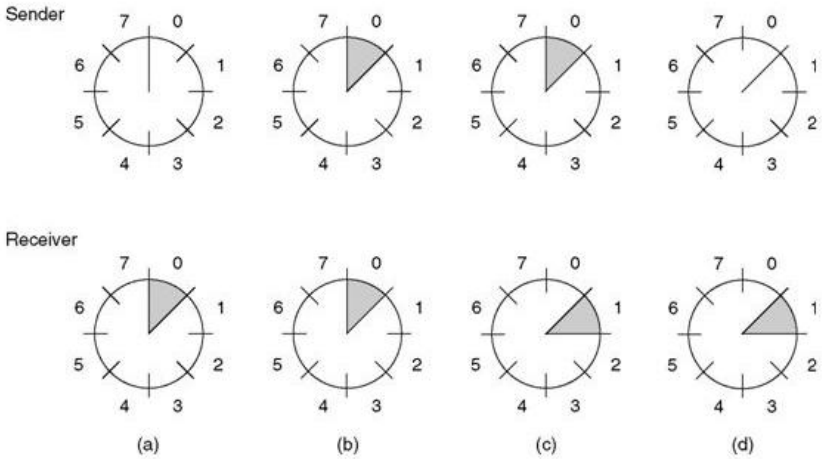
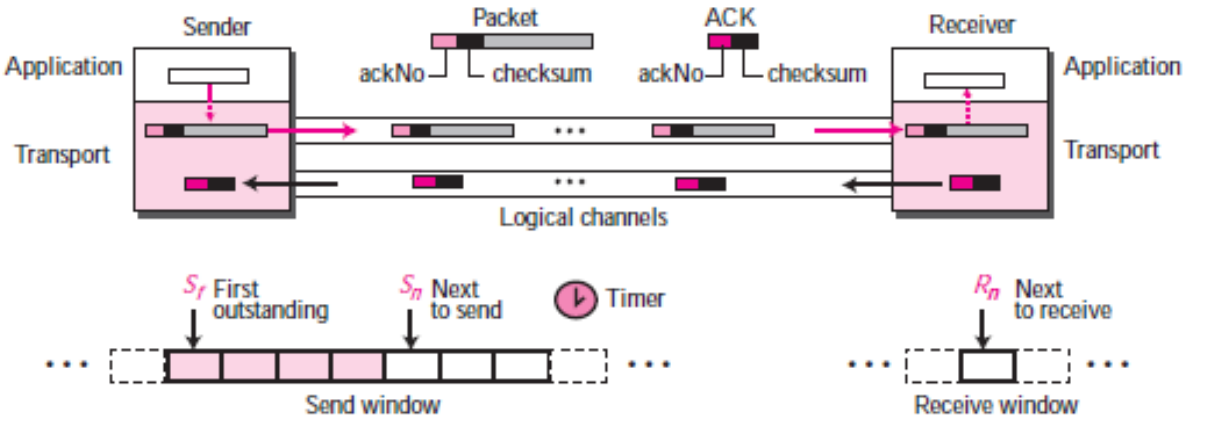


Fig.3-13 A sliding window of size 1, with a 3-bit sequence number.
(a) Initially.
(b) After the first frame has been sent.
(c) After the first frame has been received.
(d) After the first acknowledgement has been received.

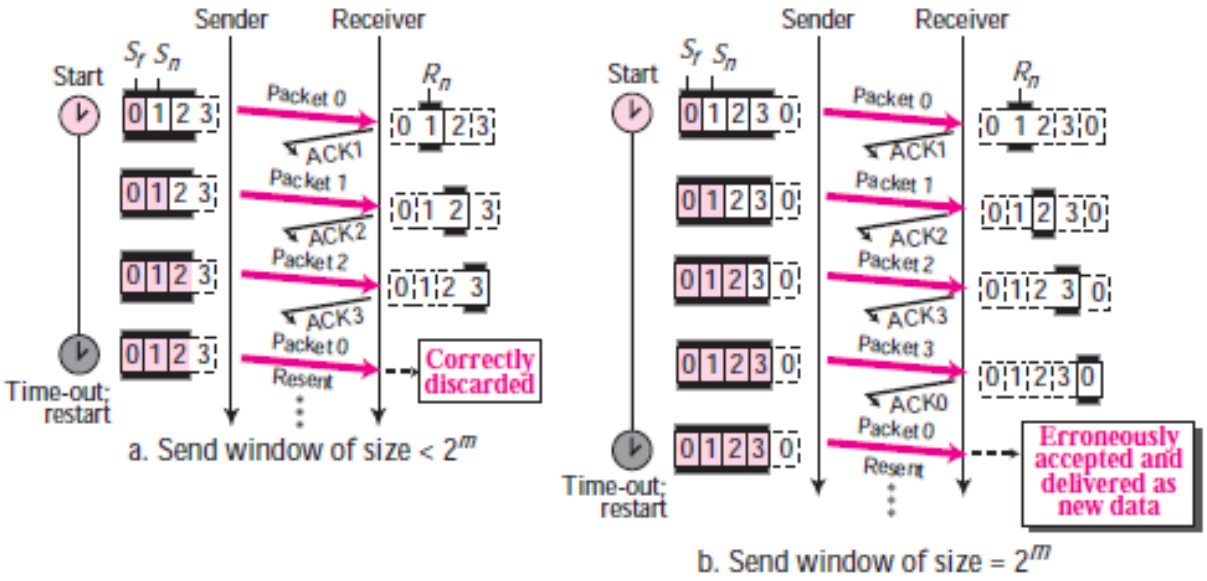
Go-Back-N

To improve the efficiency of transmission (fill the pipe), multiple packets must be in transition while the sender is waiting for acknowledgment. In other words, we need to let more than one packet be outstanding to keep the channel busy while the sender is waiting for acknowledgment. The key to Go-back-*N* is that we can send several packets before receiving acknowledgments, but the receiver can only buffer one packet. We keep a copy of the sent packets until the acknowledgments arrive.

Figure 13.22 *Go-Back-N protocol*



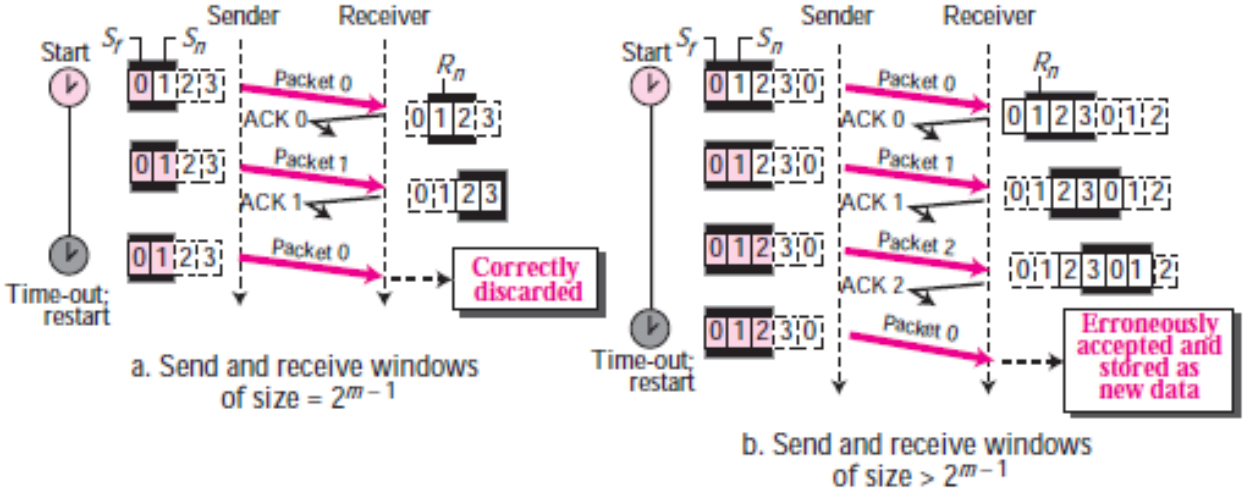
In the Go-Back-*N* protocol, the size of the send window must be less than 2*m*; the size of the receive window is always 1.



Selective Repeat

The Go-Back-*N* protocol simplifies the process at the receiver. The receiver keeps track of only one variable, and there is no need to buffer out-of-order packets; they are simply discarded. However, this protocol is inefficient if the underlying network protocol loses a lot of packets. Each time a single packet is lost or corrupted, the sender resends all outstanding packets although some of these packets may have been received safe and sound, but out of order. If the network layer is losing many packets because of congestion in the network, the resending of all of these outstanding packets makes the congestion worse, and eventually more packets are lost. This has an avalanche effect that may result in the total collapse of the network.

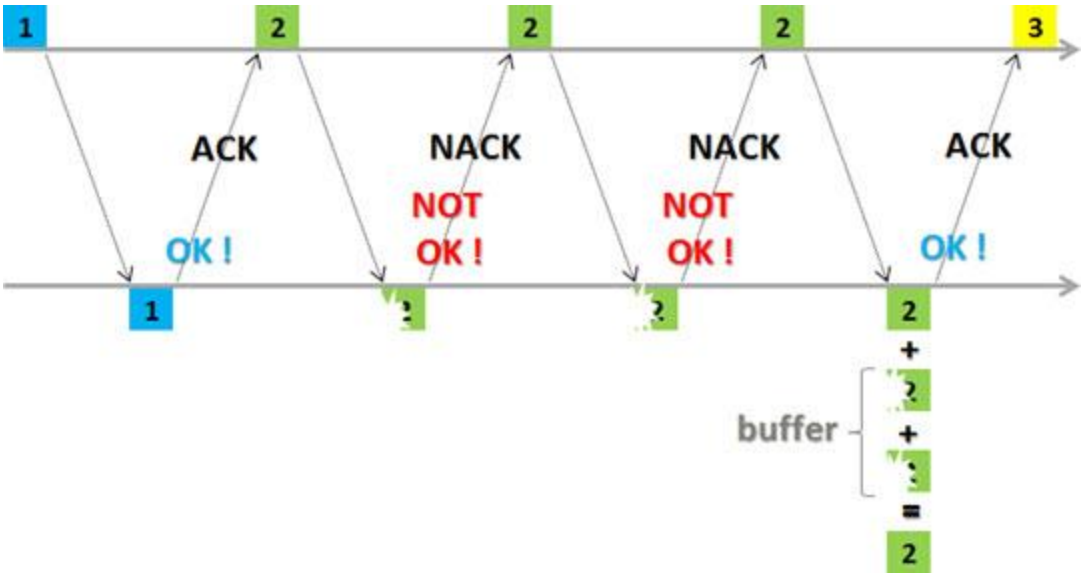
In the Selective-Repeat protocol, an acknowledgment number defines the sequence number of the error-free packet received.



Hybrid ARQ

Hybrid ARQ transmission schemes combine the conventional ARQ with forward error correction. In this scheme packet with error is stored in a buffer and NACK is generated, after retransmission if packet again received with error receiver can try to merge both packet and try to generate actual packet. If actual packet is

generated ACK will be sent to sender else NACK is send again till getting proper data. This scheme is useful in noisy channel.



Bit oriented protocols:

Bit-oriented protocols transmit information without regard to character boundaries and thus handle all types of information images. Bit-oriented protocols are much less overhead-intensive, as compared to byte-oriented protocols, also known as character-oriented protocols. Bit-oriented protocols are usually full-duplex (FDX) and operate over dedicated, four-wire circuits. Examples include Synchronous Data Link Control (SDLC) and the High-Level Data Link Control (HDLC).

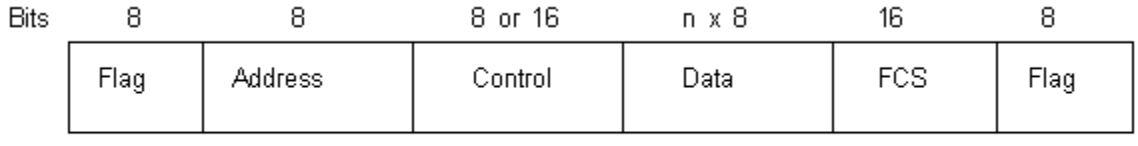
SDLC:

Synchronous Data Link Control (SDLC) is the oldest layer 2 protocol designed by IBM in 1975 to carry **Systems Network Architecture (SNA)** traffic.

In SDLC, a link station is a logical connection between adjacent nodes. Only one **Primary Link Station** is allowed on an SDLC line. A device can be set up as a Primary or a Secondary link station. A device configured as a Primary link station can communicate with both PU 2.0 nodes and PU 2.1 nodes (APPN) and controls the secondary devices. If the device is set up as a secondary link station then it acts as a PU 2.0 device and can communicate with Front End Processors (FEP), but only communicates with the primary device when the primary allows it, i.e. the primary sets up and tears down the connections and controls the secondaries.

A primary station issues commands, controls the link and initiates error-recovery. A device set up as a secondary station can communicate to a FEP, exist with other secondary devices on an SDLC link and exist as a secondary PU 2.0 device.

SDLC supports line speeds up to 64Kb/s e.g. V.24 (RS-232) at 19.2Kb/s, V.35 (up to 64Kb/s) and X.21.



SDLC Frame

- **Flag** - Begins and ends the error checking procedure with **0x7E** which is **01111110** in binary.
- **Address** - This is only the secondary address since all communication occurs via the single primary device. The address can be an individual, group or broadcast address.
- **Control** - this identifies the frame's function and can be one of the following:
    - **Information (I)** - contains the **Send Sequence Number** which is the number of the next frame to be sent, and the **Receive Sequence Number** which is the number of the next frame expected to be received. The is also a **Poll Final Bit (P/F)** which performs error checking.
    - **Supervisory (S)** - this can report on status, ask for and stop transmission and acknowledge **I** frames.
    - **Unnumbered (U)** - this does not have sequence numbers (hence 'unnumbered'), it can be used to start up secondaries and can sometimes have an Information field.
- **Data** - can contain **Path Information Unit (PIU)** or **Exchange Identification (XID)**.
- **Frame Check Sequence (FCS)** - this check is carried out on the sending AND receiving of the frame.

HDLC (High-level Data Link Control)

The most important data link control protocol is HDLC. Not only is HDLC widely used, but it is the basis for many other important data link control protocols, which use the same or similar formats and the same mechanisms as employed in HDLC.

Basic Characteristics

To satisfy a variety of applications, HDLC defines three types of stations, two \link configurations, and three data transfer modes of operation. The three station types are

• **Primary station:** Responsible for controlling the operation of the link. Frames issued by the primary are called commands.
• **Secondary station:** Operates under the control of the primary station. Frames issued by a secondary are called responses. The primary maintains a separate logical link with each secondary station on the line.
• **Combined station:** Combines the features of primary and secondary. A combined station may issue both commands and responses.

The two link configurations are
• **Unbalanced configuration:** Consists of one primary and one or more secondary stations and supports both full-duplex and half-duplex transmission.
• **Balanced configuration:** Consists of two combined stations and supports both full-duplex and half-duplex transmission.

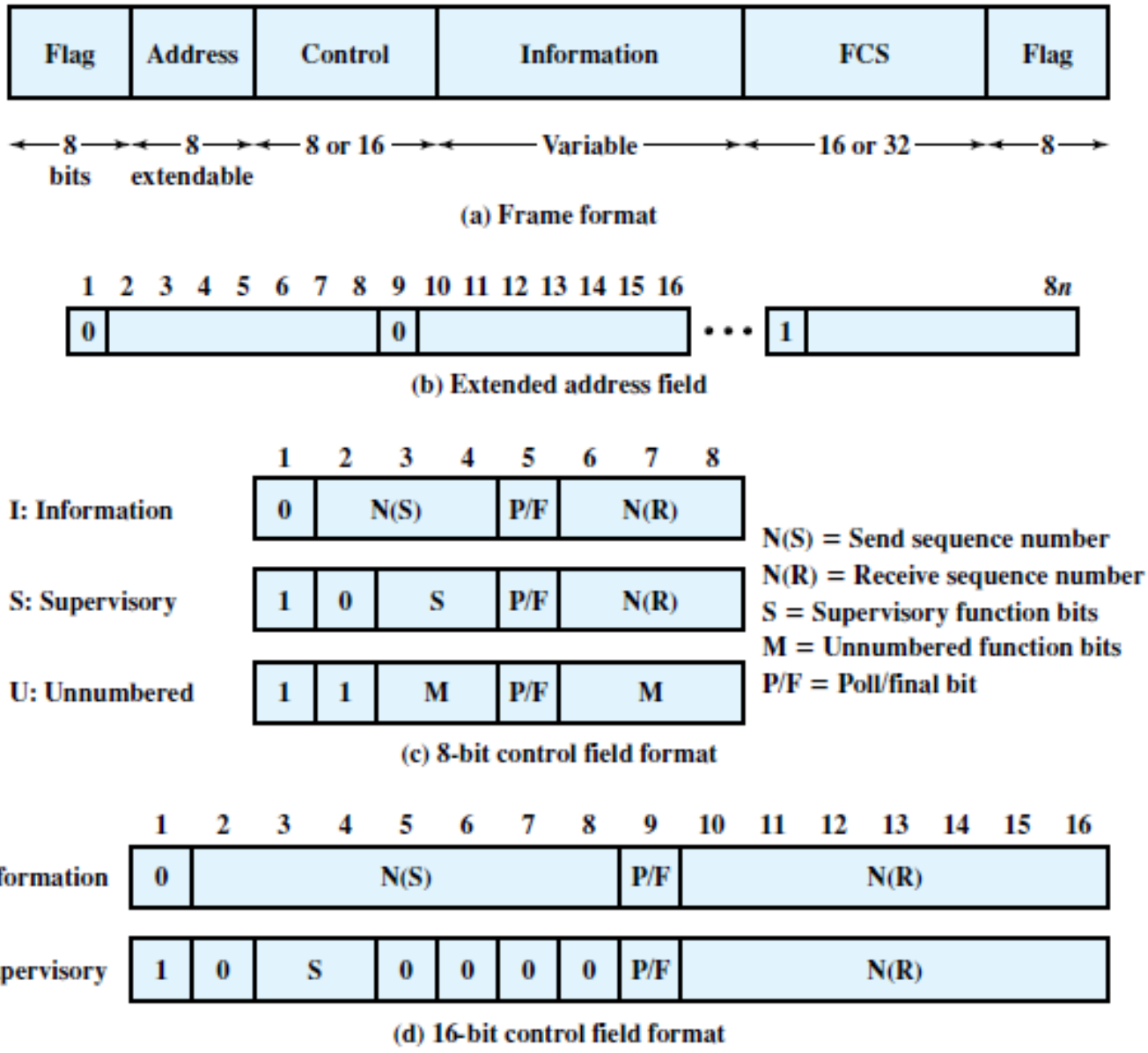
The three data transfer modes are
• **Normal response mode (NRM):** Used with an unbalanced configuration. The primary may initiate data transfer to a secondary, but a secondary may only transmit data in response to a command from the primary.
• **Asynchronous balanced mode (ABM):** Used with a balanced configuration. Either combined station may initiate transmission without receiving permission from the other combined station.
• **Asynchronous response mode (ARM):** Used with an unbalanced configuration. The secondary may initiate transmission without explicit permission of the primary. The primary still retains responsibility for the line, including initialization, error recovery, and logical disconnection.

NRM is used on multidrop lines, in which a number of terminals are connected to a host computer. The computer polls each terminal for input. NRM is also sometimes used on point-to-point links, particularly if the link connects a terminal or other peripheral to a computer. ABM is the most widely used of the three modes; it

makes more efficient use of a full-duplex point-to-point link because there is no polling overhead. ARM is rarely used; it is applicable to some special situations in which a secondary may need to initiate transmission.

Frame Structure

HDLC uses synchronous transmission. All transmissions are in the form of frames, and a single frame format suffices for all types of data and control exchanges. Figure depicts the structure of the HDLC frame. The flag, address, and controlfields that precede the information field are known as a **header**. The FCS and flag fields following the data field are referred to as a **trailer**.



Flag Fields

Flag fields delimit the frame at both ends with the unique pattern 01111110.A single flag may be used as the closing flag for one frame and the opening flag for the next. On both sides of the user-network interface, receivers are continuously hunting for the flag sequence to synchronize on the start of a frame. While receiving a frame, a station continues to hunt for that sequence to determine the end of the frame (Flag Bit with Bit stuffing).

Address Field
The address field identifies the secondary station that transmitted or is to receive the frame. This field is not needed for point-to-point links but is always included for the sake of uniformity. The address field is usually 8 bits long but, by prior agreement, an extended format may be used in which the actual address length is a multiple of 7 bits. The leftmost bit of each octet is 1 or 0 according as it is or is not the last octet of the address field. The remaining 7 bits of each octet form part of the address. The single-octet address of 11111111 is interpreted as the all-stations address in both basic and extended formats.

Control Field
HDLC defines three types of frames, each with a different control field format. **Information frames** (I-frames) carry the data to be transmitted for the user (the logic above HDLC that is using HDLC). Additionally, flow and error control data, using the ARQ mechanism, are piggybacked on an information frame. **Supervisory frames** (S-frames) provide the ARQ mechanism when piggybacking is not used. **Unnumbered frames** (U-frames) provide supplemental link control functions. The first one or two bits of the control field serves to identify the frame type.

Information Field
The information field is present only in I-frames and some U-frames. The field can contain any sequence of bits but must consist of an integral number of octets. The length of the information field is variable up to some system defined maximum.

Frame Check Sequence Field The frame check sequence (FCS) is an error detecting code calculated from the remaining bits of the frame, exclusive of flags.

Operation
HDLC operation consists of the exchange of I-frames, S-frames, and U-frames between two stations. The various commands and responses defined for these frame types are listed in Table 7.1. In describing HDLC operation, we will discuss these three types of frames.

The operation of HDLC involves three phases. First, one side or another initializes the data link so that frames may be exchanged in an orderly fashion. During this phase, the options that are to be used are agreed upon. After initialization, the two sides exchange user data and the control information to exercise flow and error control. Finally, one of the two sides signals the termination of the operation.

Initialization Either side may request initialization by issuing one of the six set mode commands. This command serves three purposes:
1. It signals the other side that initialization is requested.
2. It specifies which of the three modes (NRM, ABM, ARM) is requested.
3. It specifies whether 3- or 7-bit sequence numbers are to be used.

If the other side accepts this request, then the HDLC module on that end transmits an unnumbered acknowledged (UA) frame back to the initiating side. If the request is rejected, then a disconnected mode (DM) frame is sent.

Data Transfer When the initialization has been requested and accepted, then a logical connection is established. Both sides may begin to send user data in I-frames, starting with sequence number 0.The N(S)

and N(R) fields of the I-frame are sequence numbers that support flow control and error control. An HDLC module sending a sequence of I-frames will number them sequentially, modulo 8 or 128, depending on whether 3- or 7-bit sequence numbers are used, and place the sequence number in N(S). N(R) is the acknowledgment for I-frames received; it enables the HDLC module to indicate which number I-frame it expects to receive next.

Table 7.1 HDLC Commands and Responses

Name	Command/ Response	Description
Information (I)	C/R	Exchange user data
Supervisory (S)		
Receive ready (RR)	C/R	Positive acknowledgment; ready to receive I-frame
Receive not ready (RNR)	C/R	Positive acknowledgment; not ready to receive
Reject (REJ)	C/R	Negative acknowledgment; go back N
Selective reject (SREJ)	C/R	Negative acknowledgment; selective reject
Unnumbered (U)		
Set normal response/extended mode (SNRM/SNRME)	C	Set mode; extended = 7-bit sequence numbers
Set asynchronous response/extended mode (SARM/SARME)	C	Set mode; extended = 7-bit sequence numbers
Set asynchronous balanced/extended mode (SABM, SABME)	C	Set mode; extended = 7-bit sequence numbers
Set initialization mode (SIM)	C	Initialize link control functions in addressed station
Disconnect (DISC)	C	Terminate logical link connection
Unnumbered Acknowledgment (UA)	R	Acknowledge acceptance of one of the set-mode commands
Disconnected mode (DM)	R	Responder is in disconnected mode
Request disconnect (RD)	R	Request for DISC command
Request initialization mode (RIM)	R	Initialization needed; request for SIM command
Unnumbered information (UI)	C/R	Used to exchange control information
Unnumbered poll (UP)	C	Used to solicit control information
Reset (RSET)	C	Used for recovery; resets N(R), N(S)
Exchange identification (XID)	C/R	Used to request/report status
Test (TEST)	C/R	Exchange identical information fields for testing
Frame reject (FRMR)	R	Report receipt of unacceptable frame

S-frames are also used for flow control and error control. The receive ready (RR) frame acknowledges the last I-frame received by indicating the next I-frame expected. The RR is used when there is no reverse user data traffic (I-frames) to carry an acknowledgment. Receive not ready (RNR) acknowledges an I-frame, as with RR, but also asks the peer entity to suspend transmission of I-frames. When the entity that issued RNR is again ready, it sends an RR. REJ initiates the go-back-N ARQ. It indicates that the last I-frame received has been rejected and that retransmission of all I-frames beginning with number N(R) is required. Selective reject (SREJ) is used to request retransmission of just a single frame.

Disconnect Either HDLC module can initiate a disconnect, either on its own initiative if there is some sort of fault, or at the request of its higher-layer user. HDLC issues a disconnect by sending a disconnect (DISC)

frame. The remote entity must accept the disconnect by replying with a UA and informing its layer 3 user that the connection has been terminated. Any outstanding unacknowledged I-frames may be lost, and their recovery is the responsibility of higher layers.

BISYNC

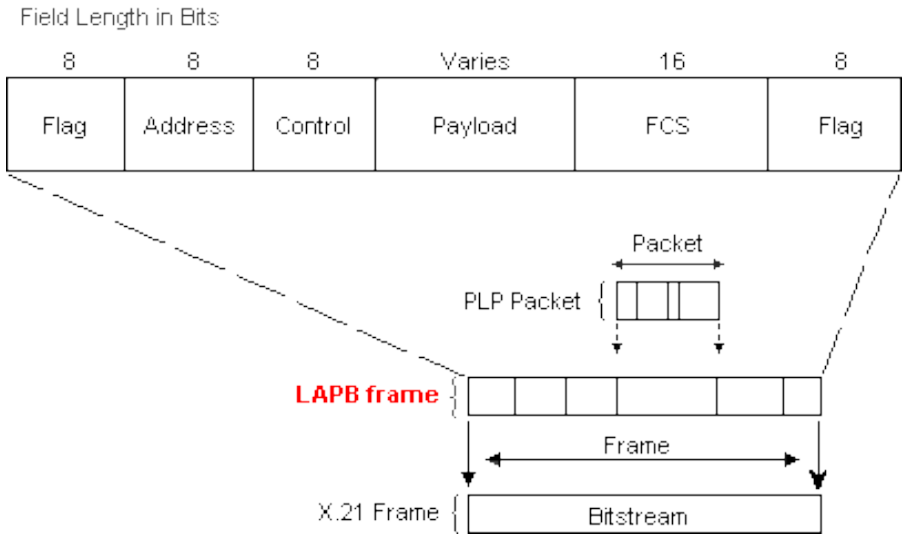
Binary synchronous communications, or BISYNC, is a character (byte)-oriented form of communication developed by IBM in the 1960s. It was originally designed for batch transmissions between the IBM S/360 mainframe family and IBM 2780 and 3780 terminals.

BISYNC is character oriented, meaning that groups of bits (bytes) are the main elements of transmission, rather than a stream of bits. It starts with two sync characters that the receiver and transmitter use for synchronizing. This is followed by a start of header (SOH) command, and then the header. Following this are the start of text (STX) command and the text. Finally, an end of text (EOT) command and a cyclic redundancy check (CRC) end the frame. The CRC provides error detection and correction.

Link Access Procedure (LAP) & LAP-B (balance)

The LAP protocols are part of a group of data link layer protocols for framing and transmitting data across point-to-point links. It is a full-duplex protocol, meaning that each station can send and receive commands and responses over separate channels to improve throughput. The protocol is bit oriented, meaning that the data is monitored bit by bit.

Bit-oriented information in the LAPB frame defines the structure for delivering data and command/response messages between communicating systems. The frame format for LAPB is similar to the frame type for HDLC.



Flag bit:

The binary pattern 01111110 is used to mark the beginning of the LAPB frame. This pattern has to be be unique so the transmitter in a LAPB connection uses a bit stuffing technique to make sure that the flag pattern appears only in the flag fields.

Address:

This field can have two values:

Binary Value	Transmission	
	Command	Response
0000001	DTE->DCE	DCE->DTE
0000011	DTE<-DCE	DCE<-DTE

Neither of the two values is actually an address. Because communication is in balanced mode, the communication link is full duplex and either DTE or DCE may initiate or terminate communication. The point of including an address is to indicate whether DTE or DCE is in control of the communication.

Control:
The control field is used to indicate command and response frames and indicates whether the frame is an I-frame, an S-frame, or a U-frame. In addition, this field contains the frame's sequence number and its function (for example, whether receiver-ready or disconnect). Control frames vary in length depending on the frame type.

Data:
Contains upper-layer data in the form of an encapsulated PLP packet.

FCS:
Handles error checking and ensures the integrity of the transmitted data.

LAPB FRAME TYPES	
Information (I-Frame)	Information's carry upper layer information and some control information. Information frames pass sequencing, flow control, error detection, and recovery information to the receiver to assist in reconstructing the upper layer data stream.
Supervisory (S-Frame)	Carries control information. Used to request and suspend transmission, report status, and Acknowledge receipt of I-Frames. The following messages are sent using s-frames. RECEIVE READY Acknowledges the current frames received and indicates which frame is expected next. REJECT Signals rejection of frame by the receiver. RECEIVE NOT READY Used for flow control. Indicates the receiver is not ready to receive additional data and the transmitter should wait until a RECEIVED READY message is sent.
Unnumbered (U-Frame)	Used for link setup, link disconnection and error reporting. No sequence numbers used.

Protocol verification:
Realistic protocols and their implementations are very complicated. How to verify that an implementation of a protocol is correct?
For this two concepts for the verification of protocols are used
● Finite State Machine (FSM) Models
● Petri Net Models

Finite State Machine Models
Construct a finite state machine for a protocol
● The machine is always in a specific state specific state
● The states consist of all the values of its variables
● Often, a large number of states can be grouped
● Number of state is 2^n, where n is the number of bits needed to represent all variables
● Well-known techniques from graph theory allow the determination of which states are reachable and which are not

Reachability analysis
● Incompleteness: protocol machine is in a state and the protocol does not specify what Incompleteness: protocol machine is in a state and the protocol does not specify what to do
● Deadlock: no exit or progress from a state
● Extraneous transition: event occurs in a state in which it should not occur

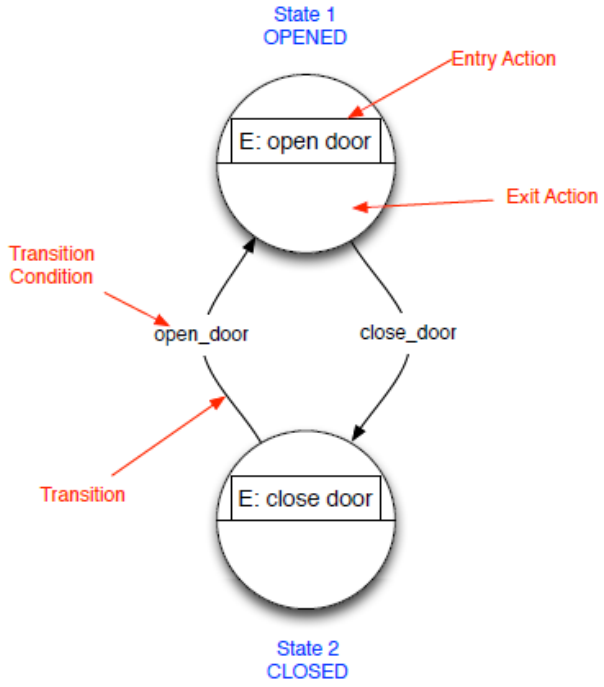
Finite State Machine Models
● Formal definition of a protocol machine as a quadruple (S, M, I, T)
● **S** set of states the processes and channel can be in set of states the processes and channel can be in
● **M** set of frames that can be exchanged over the channel
● **I** set of initial states of the processes
● **T** set of transitions between states

Finite state machine

A finite state machine (FSM) is a model of behavior composed of a finite number of states, transitions between those states, and actions. A finite state machine is an abstract model of a machine with a primitive (sometimes read-only) internal memory.
To summarize it, a state machine can be described as:

- An initial state or record of something stored someplace
- A set of possible input events
- A set of new states that may result from the input
- A set of possible actions or output events that result from a new state

A finite state machine is one that has a limited or finite number of possible states. Finite state machine can be used both as a development tool for approaching and solving problems and as a formal way of describing the solution for later developers and system maintainers. There are a number of ways to show state machines, from simple tables through graphically animated illustrations.



A current state is determined by past states of the system (recording information about the past), therefore, reflects the input changes from the system start to the present moment.
A transition indicates a state change and is described by a condition that would need to be fulfilled to enable the transition.

Figure 13.15 shows the representation of a transport layer using an FSM. Using this tool, each transport layer (sender or receiver) is taught as a machine with a finite number of states. The machine is always in one of the states until an event occurs. Each event is associated with two reactions: defining the list (possibly empty) of actions to be performed and determining the next state (which can be the same as the current state). One of the states must be defined as the initial state, the state in which the machine starts when it turns on.

Figure 13.15 *Connectionless and connection-oriented service represented as FSMs*

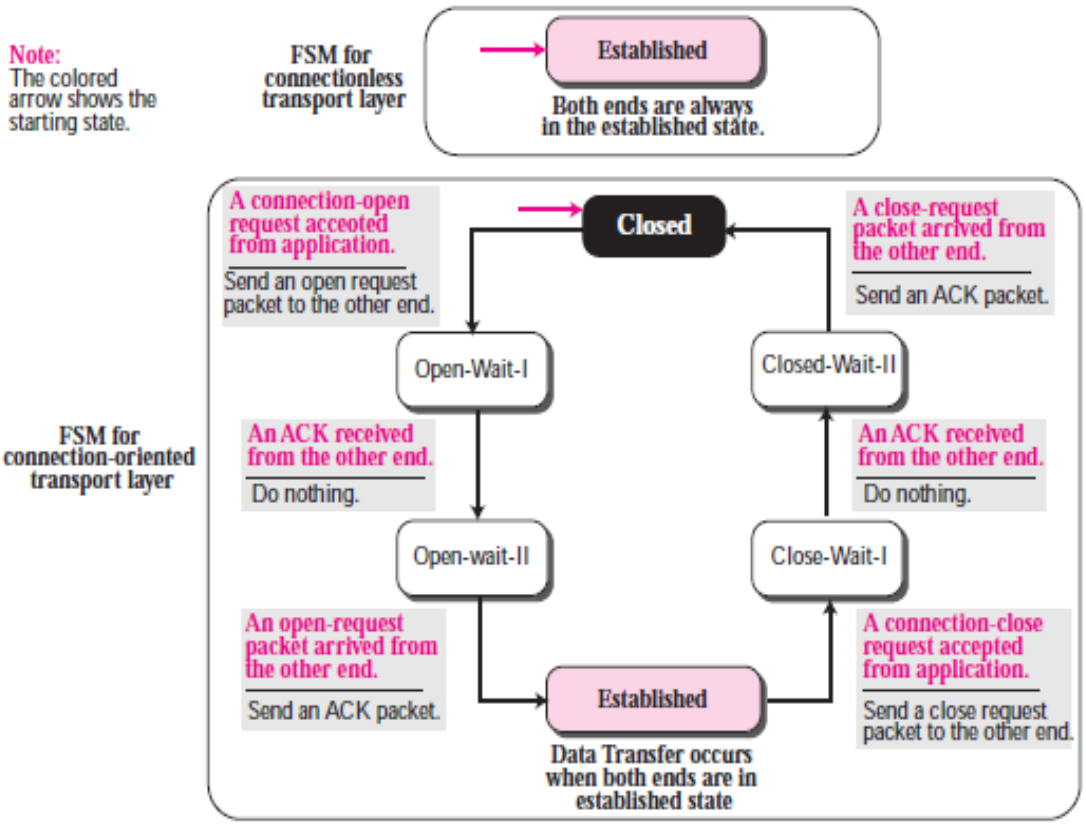


Figure 13.20 shows the FSMs for the Stop-and-Wait protocol. Since the protocol is a connection-oriented protocol, both ends should be in the *established* state before exchanging data packets.

Figure 13.20 *FSM for the Stop-and-Wait protocol*

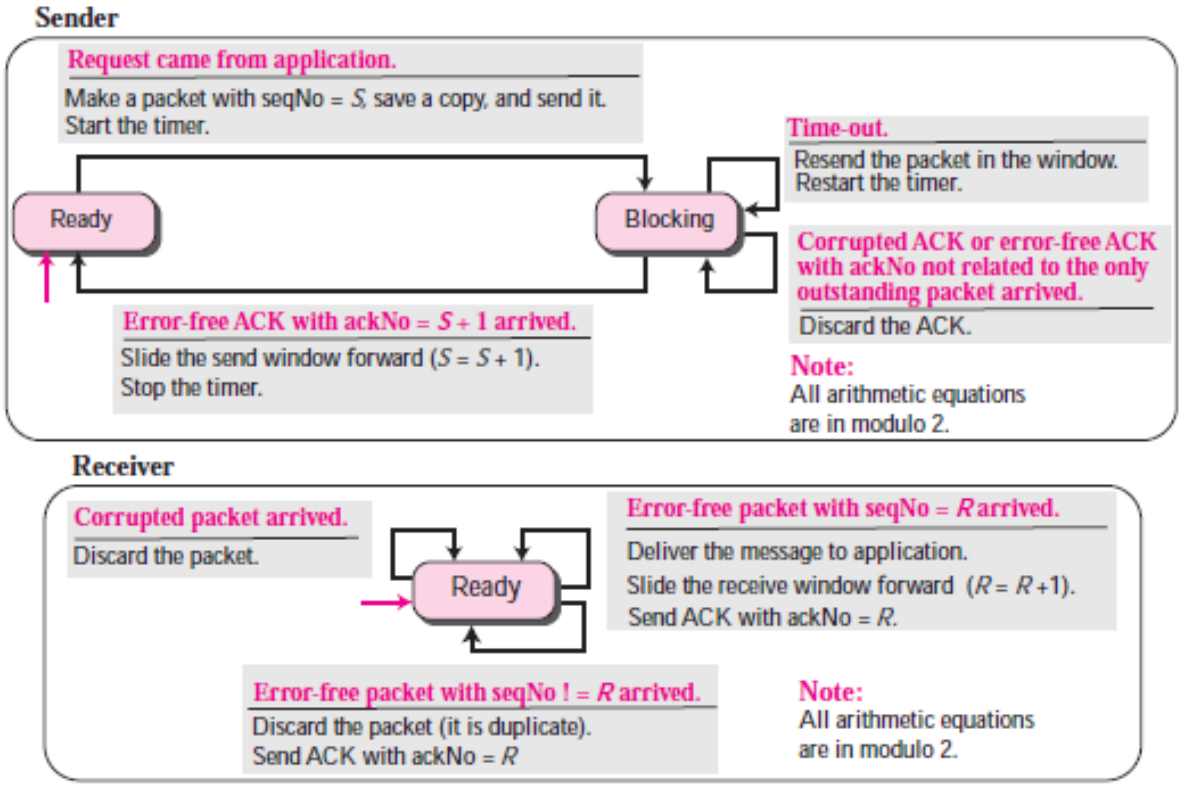
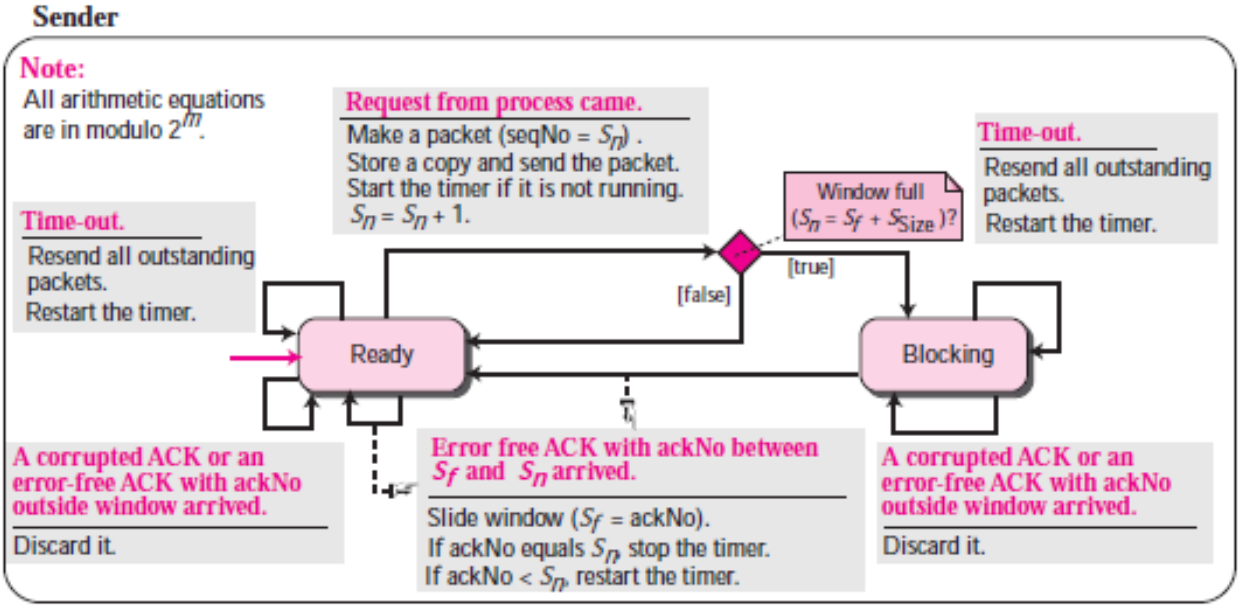
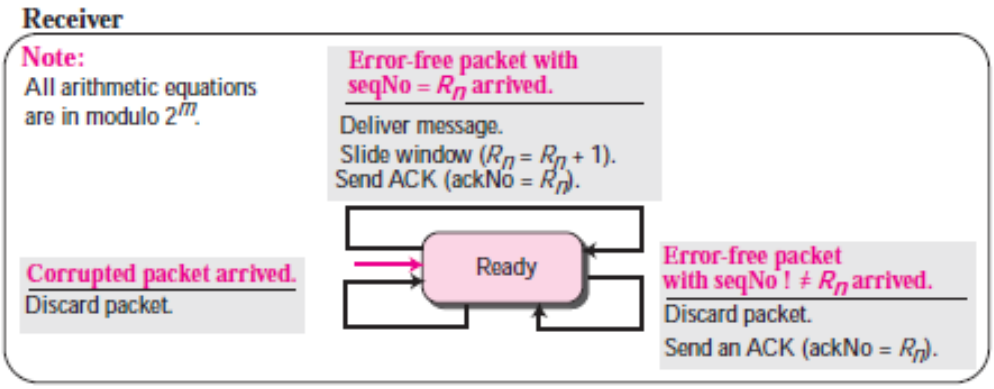


Figure 13.26 shows the FSMs for the GBN protocol.

Figure 13.26 *FSM for the Go-Back-N protocol*

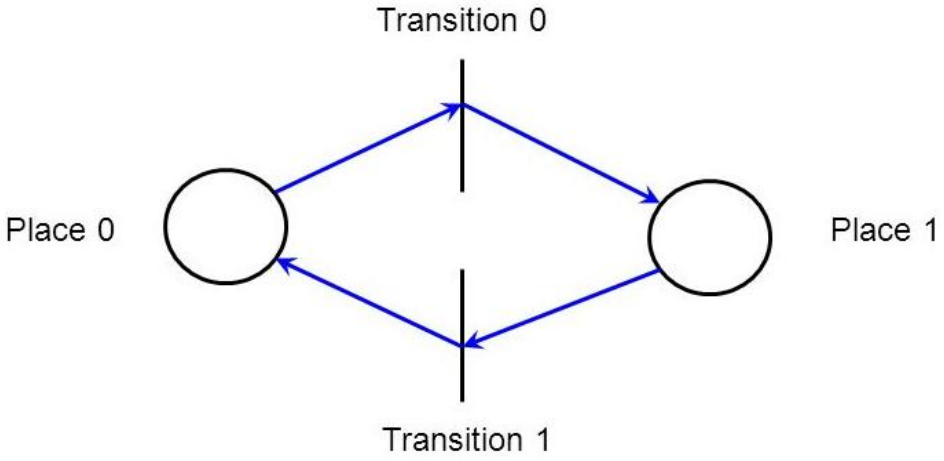




Petrinet Model

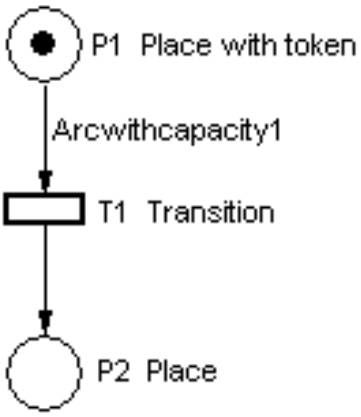
A Petri net, also known as a **place/transition (PT) net**, is one of several mathematical modeling languages applied in a variety of areas: Office automation, work-flows, flexible manufacturing, programming languages, protocols and networks, hardware structures, real-time systems, performance evaluation, operations research, embedded systems, defense systems, telecommunications, Internet, e-commerce and trading, railway networks, biological systems.

It is a class of discrete event dynamic system. A Petri net is a directed bipartite graph, in which the nodes represent transitions (i.e. events that may occur, represented by bars) and places (i.e. conditions, represented by circles). The directed arcs describe which places are pre- and/or post conditions for which transitions (signified by arrows).



The Basics:

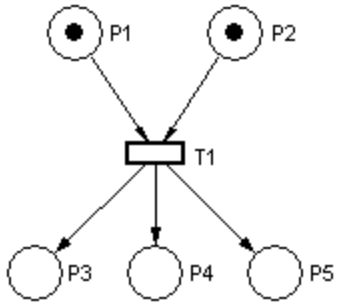
A Petri Net is a collection of directed arcs connecting places and transitions. Places may hold tokens. The state or marking of a net is its assignment of tokens to places. Here is a simple net containing all components of a Petri Net:



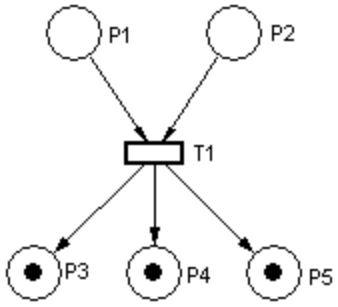
Arcs have capacity 1 by default; if other than 1, the capacity is marked on the arc. Places have infinite capacity by default, and transitions have no capacity, and cannot store tokens at all. With the rule that arcs can only connect places to transitions and vice versa, we have all we need to begin using Petri Nets. A few other features and considerations will be added as we need them.

A transition is enabled when the number of tokens in each of its input places is at least equal to the arc weight going from the place to the

transition. An enabled transition may fire at any time. When fired, the tokens in the input places are moved to output places, according to arc weights and place capacities. This results in a new marking of the net, a state description of all places.

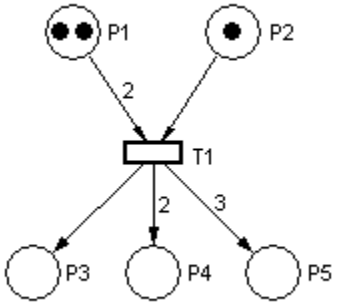


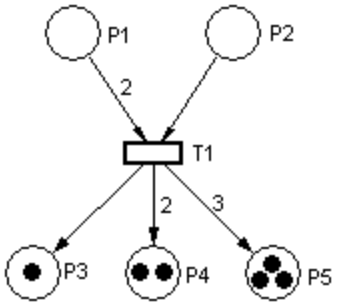
Enabled.ready to file



Firing complete

When arcs have different weights, we have what might at first seem confusing behavior. Here is a similar net, ready to fire:

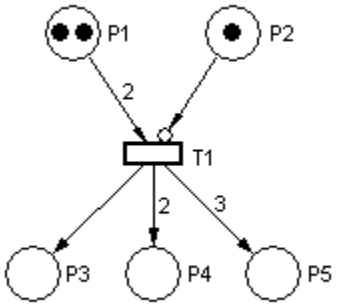




and here it is after firing:

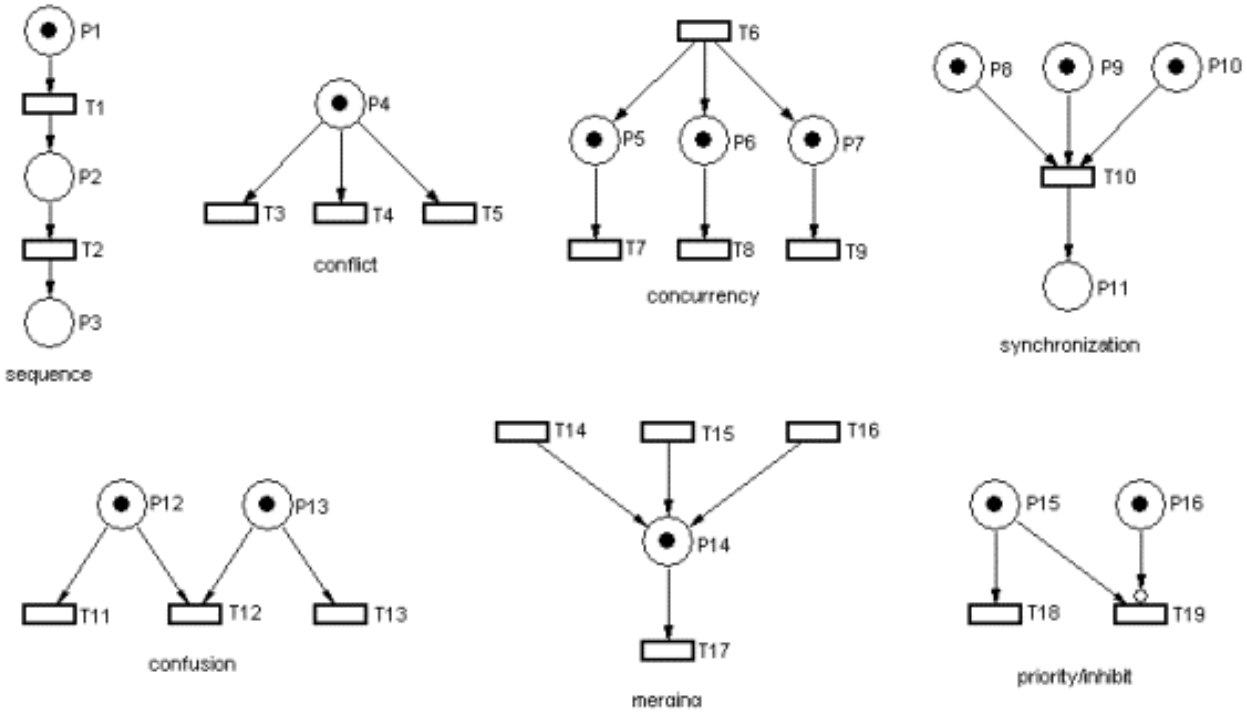
When a transition fires, it takes the tokens that enabled it from the input places; it then distributes tokens to output places according to arc weights. If the arc weights are all the same, it appears that tokens are moved across the transition. If they differ, however, it appears that tokens may disappear or be created. That, in fact, is what happens; think of the transition as removing its enabling tokens and producing output tokens according to arc weight.

A special kind of arc, the inhibitor arc, is used to reverse the logic of an input place. With an inhibitor arc, the absence of a token in the input place enables, not the presence:



This transition cannot fire, because the token in P2 inhibits it.

Here is a collection of primitive structures that occur in real systems, and thus we find in Petri Nets.



Sequence is obvious - several things happen in order.
Conflict is not so obvious. The token in P4 enables three transitions; but when one of them fires, the token is removed, leaving the remaining two disabled. Unless we can control the timing of firing, we don't know how this net is resolved.

Concurrency, again, is obvious; many systems operate with concurrent activities and this model it well. Synchronization is also modeled well using Petri Nets; when the processes leading into P8, P9 and P10 are finished, all three are synchronized by starting P11.

Confusion is another not so obvious construct. It is a combination of conflict and concurrency. P12 enables both T11 and T12, but if T11 fires, T12 is no longer enabled.

Merging is not quite the same as synchronization, since there is nothing requiring that the three transitions fire at the same time, or that all three fire before T17; this simply merges three parallel processes.

The **priority/inhibit** construct uses the inhibit arc to control T19; as long as P16 has a token, T19 cannot fire.

Very sophisticated logic and control structures can be developed using these primitives.

THE ARP PROTOCOL

Anytime a host or a router has an IP datagram to send to another host or router, it has the logical (IP) address of the receiver. But the IP datagram must be encapsulated in a frame to be able to pass through the physical network. This means that the sender needs the physical address of the receiver. A mapping corresponds a logical address to a physical address. Figure 8.1 shows the position of the ARP in the TCP/IP protocol suite. ARP accepts a logical address from the IP protocol maps the address to the corresponding physical address and passes it to the data link layer.

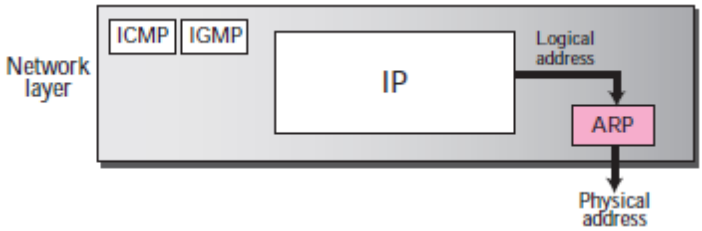


Figure 8.1

ARP associates an IP address with its physical address. On a typical physical network, such as a LAN, each device on a link is identified by a physical or station address that is usually imprinted on the NIC.

Anytime a host, or a router, needs to find the physical address of another host or router on its network, it sends an ARP query packet. The packet includes the physical and IP addresses of the sender and the IP address of the receiver. Because the sender does not know the physical address of the receiver, the query is broadcast over the network (see Figure 8.2).

Every host or router on the network receives and processes the ARP query packet, but only the intended recipient recognizes its IP address and sends back an ARP response packet. The response packet contains the recipient's IP and physical addresses. The packet is unicast directly to the inquirer using the physical address received in the query packet.

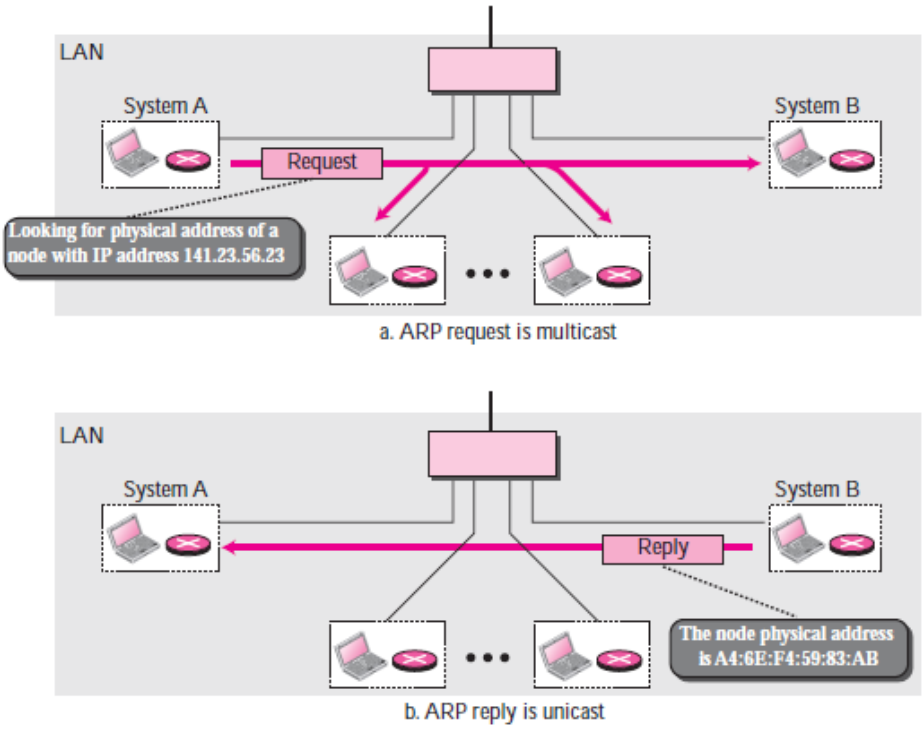


Figure 8.2

In Figure 8.2a, the system on the left (A) has a packet that needs to be delivered to another system (B) with IP address 141.23.56.23. System A needs to pass the packet to its data link layer for the actual delivery, but it does not know the physical address of the recipient. It uses the services of ARP by asking the ARP protocol to send a broadcast ARP request packet to ask for the physical address of a system with an IP address of 141.23.56.23.

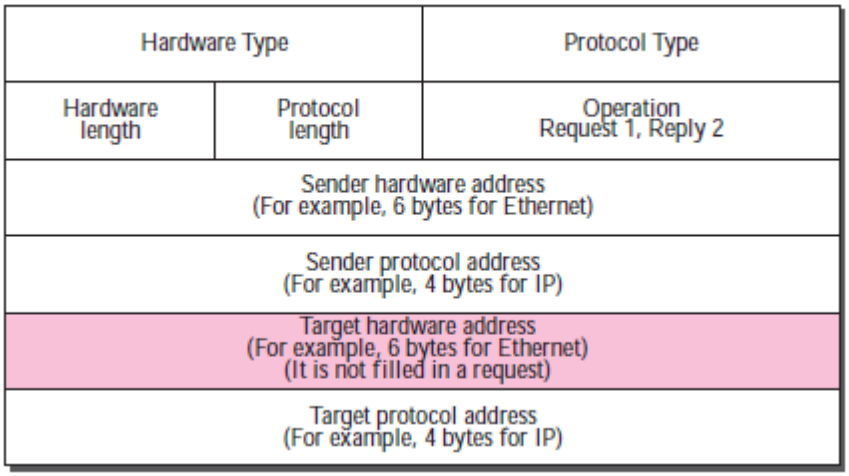
This packet is received by every system on the physical network, but only system B will answer it, as shown in Figure 8.2b. System B sends an ARP reply packet that includes its physical address. Now system A can send all the packets it has for this destination using the physical address it received.

Packet Format

Figure 8.3 shows the format of an ARP packet. The fields are as follows:

- **Hardware type:** This is a 16-bit field defining the type of the network on which ARP is running. Each LAN has been assigned an integer based on its type. For example, Ethernet is given the type 1. ARP can be used on any physical network.
- **Protocol type:** This is a 16-bit field defining the protocol. For example, the value of this field for the IPv4 protocol is 080016. ARP can be used with any higher-level protocol.
- **Hardware length:** This is an 8-bit field defining the length of the physical address in bytes. For example, for Ethernet the value is 6.
- **Protocol length:** This is an 8-bit field defining the length of the logical address in bytes. For example, for the IPv4 protocol the value is 4.

8.3 *ARP packet*

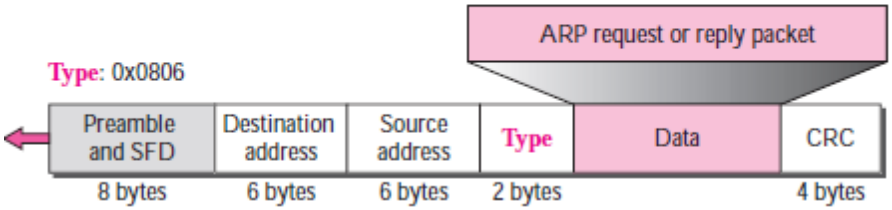


- **Operation:** This is a 16-bit field defining the type of packet. Two packet types are defined: ARP request (1), ARP reply (2).
- **Sender hardware address:** This is a variable-length field defining the physical address of the sender. For example, for Ethernet this field is 6 bytes long.
- **Sender protocol address:** This is a variable-length field defining the logical (for example, IP) address of the sender. For the IP protocol, this field is 4 bytes long.
- **Target hardware address:** This is a variable-length field defining the physical address of the target. For example, for Ethernet this field is 6 bytes long. For an ARP request message, this field is all 0s because the sender does not know the physical address of the target.
- **Target protocol address:** This is a variable-length field defining the logical (for example, IP) address of the target. For the IPv4 protocol, this field is 4 bytes long.

Encapsulation

An ARP packet is encapsulated directly into a data link frame. For example, in Figure 8.4 an ARP packet is encapsulated in an Ethernet frame. Note that the type field indicates that the data carried by the frame is an ARP packet.

8.4 *Encapsulation of ARP packet*



Operation

Let us see how ARP functions on a typical internet. First we describe the steps involved. Then we discuss the four cases in which a host or router needs to use ARP.

Steps Involved

These are seven steps involved in an ARP process:

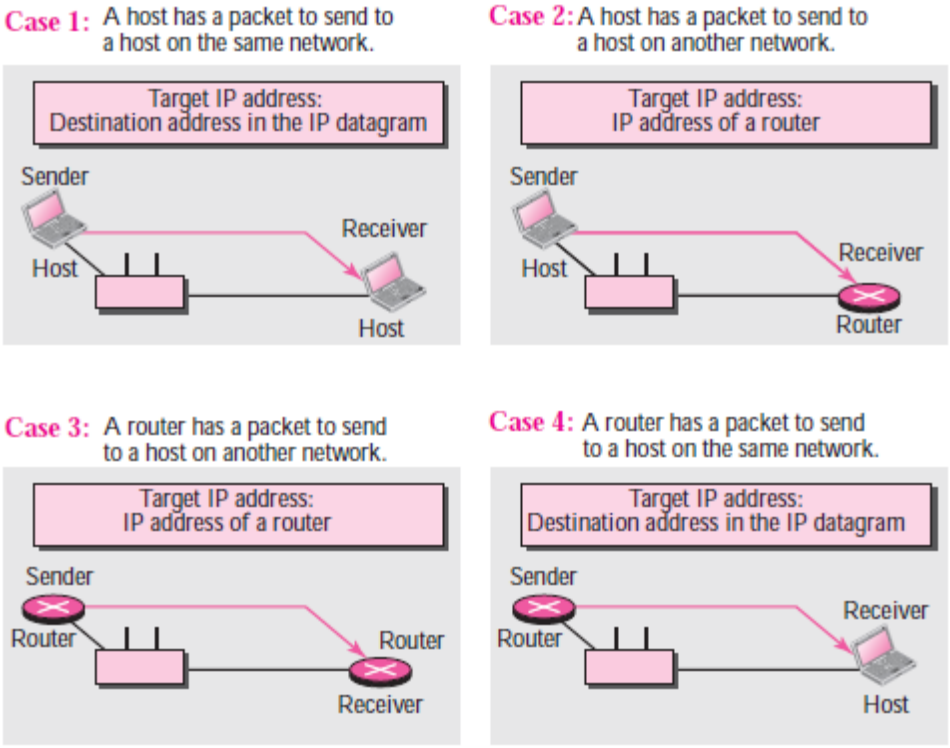
1. The sender knows the IP address of the target.
2. IP asks ARP to create an ARP request message, filling in the sender physical address, the sender IP address, and the target IP address. The target physical address field is filled with 0s.
3. The message is passed to the data link layer where it is encapsulated in a frame using the physical address of the sender as the source address and the physical broadcast address as the destination address.
4. Every host or router receives the frame. Because the frame contains a broadcast destination address, all stations remove the message and pass it to ARP. All machines except the one targeted drop the packet. The target machine recognizes the IP address.
5. The target machine replies with an ARP reply message that contains its physical address. The message is unicast.
6. The sender receives the reply message. It now knows the physical address of the target machine.
7. The IP datagram, which carries data for the target machine, is now encapsulated in a frame and is unicast to the destination.

Four Different Cases

The following are four different cases in which the services of ARP can be used (see figure).

- **Case 1:** The sender is a host and wants to send a packet to another host on the same network. In this case, the logical address that must be mapped to a physical address is the destination IP address in the datagram header.
- **Case 2:** The sender is a host and wants to send a packet to another host on another network. In this case, the host looks at its routing table and finds the IP address of the next hop (router) for this destination. If it does not have a routing table, it looks for the IP address of the default router. The IP address of the router becomes the logical address that must be mapped to a physical address.
- **Case 3:** The sender is a router that has received a datagram destined for a host on another network. It checks its routing table and finds the IP address of the next router. The IP address of the next router becomes the logical address that must be mapped to a physical address.
- **Case 4:** The sender is a router that has received a datagram destined for a host in the same network. The destination IP address of the datagram becomes the logical address that must be mapped to a physical address.

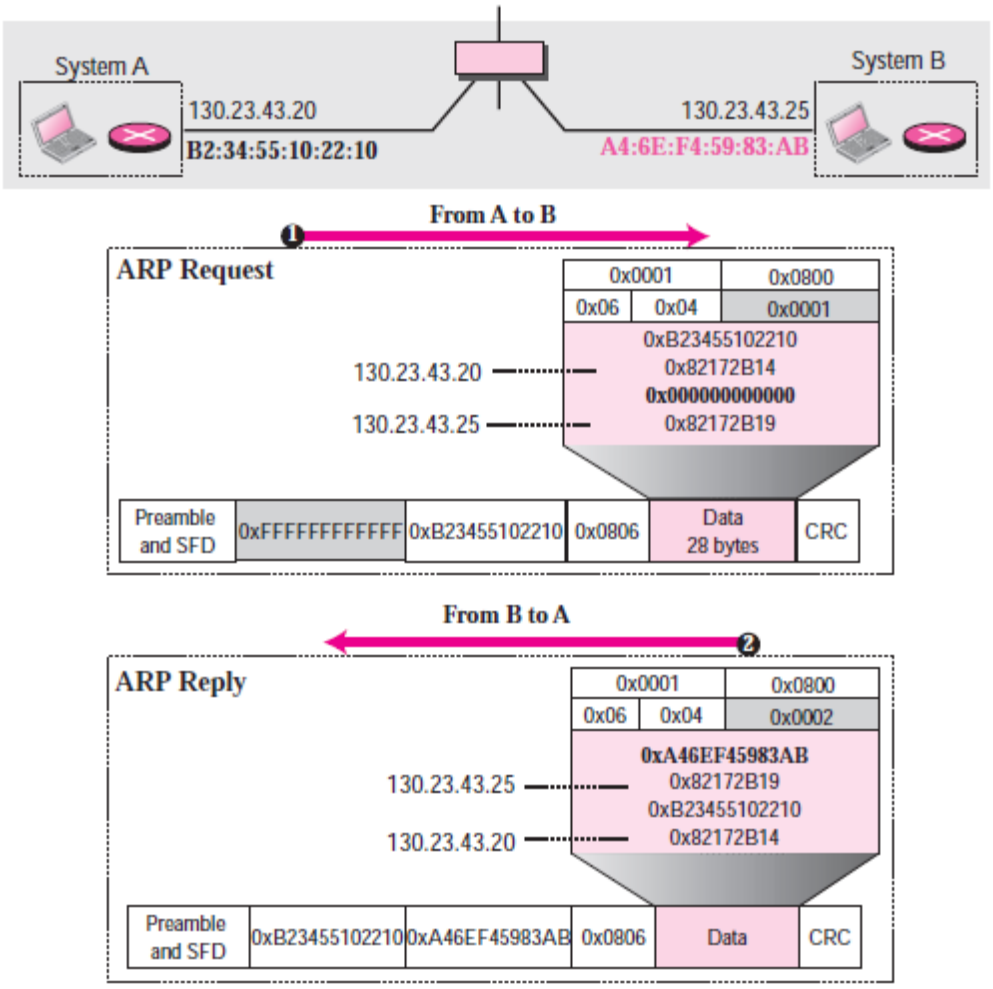
ure 8.5 *Four cases using ARP*



Example:
A host with IP address 130.23.43.20 and physical address B2:34:55:10:22:10 has a packet to send to another host with IP address 130.23.43.25 and physical address A4:6E:F4:59:83:AB (which is unknown to the first host). The two hosts are on the same Ethernet network. Show the ARP request and reply packets encapsulated in Ethernet frames.

Solution
Figure 8.6 show the ARP request and reply packets. Note that the ARP data field in this case is 28 bytes, and that the individual addresses do not fit in the 4-byte boundary. That is why we do not show the regular 4-byte boundaries for these addresses. Also note that the IP addresses are shown in hexadecimal. For information on binary or hexadecimal notation.

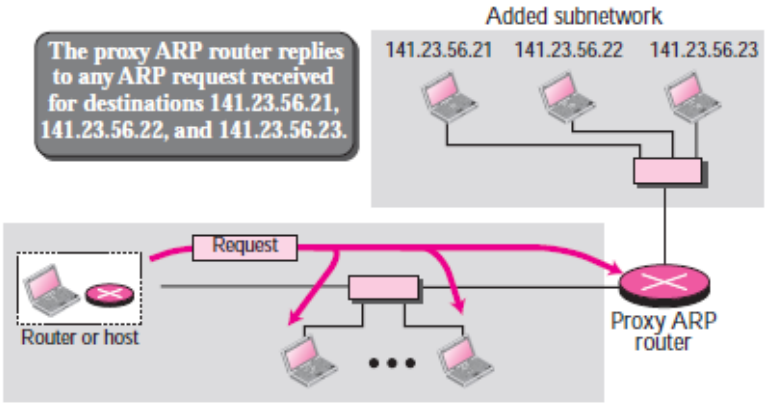
'igure 8.6 *Example 8.1*



Proxy ARP

A technique called *proxy* ARP is used to create a subnetting effect. A **proxy ARP** is an ARP that acts on behalf of a set of hosts. Whenever a router running a proxy ARP receives an ARP request looking for the IP address of one of these hosts, the router sends an ARP reply announcing its own hardware (physical) address. After the router receives the actual IP packet, it sends the packet to the appropriate host or router.

8.7 *Proxy ARP*



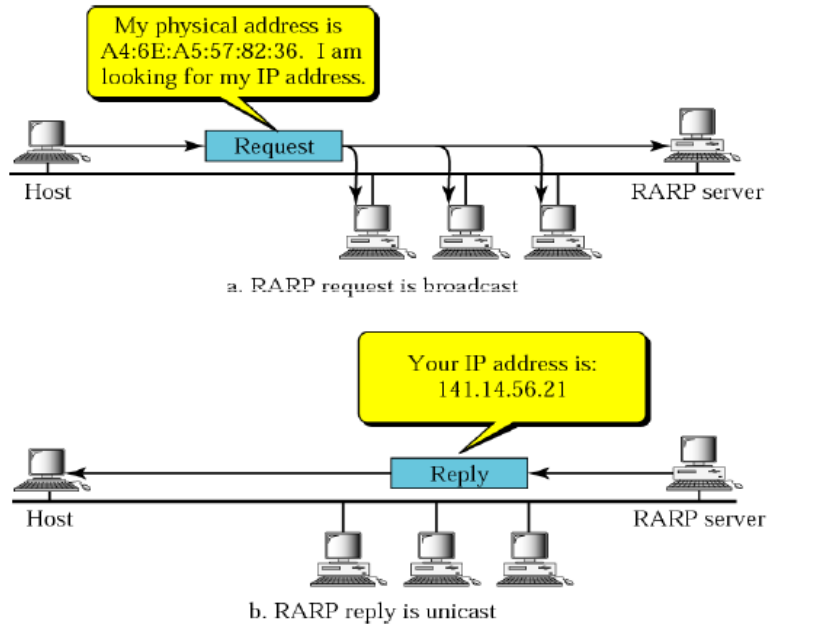
Let us give an example. In Figure 8.7 the ARP installed on the right-hand host will answer only to an ARP request with a target IP address of 141.23.56.23. However, the administrator may need to create a subnet without changing the whole system to recognize subnetted addresses. One solution is to add a router running a proxy ARP. In this case, the router acts on behalf of all of the hosts installed on the subnet. When it receives an ARP request with a target IP address that matches the address of one of its protégés (141.23.56.21, 141.23.56.22, and 141.23.56.23), it sends an ARP reply and announces its hardware address as the target hardware address. When the router receives the IP packet, it sends the packet to the appropriate host.

RARP

At the beginning of the Internet era, a protocol called Reverse Address Resolution Protocol (RARP) was designed to provide the IP address for a booted computer. RARP was actually a version of ARP. ARP maps an IP address to a physical address: RARP maps a physical address to an IP address. However, RARP is deprecated today for two reasons. First, RARP used the broadcast service of the data link layer, which means that a RARP server must be present in each network. Second, RARP can provide only the IP address of the computer, but a computer today needs all four pieces of information mentioned above.

RARP also uses same frame format as ARP and its encapsulation. RARP request packet is broadcast while reply packet is unicast.

gure 7.10 ***RARP operation***



GARP

Gratuitous ARP (GARP) is **used to update an ARP table of the hosts in a Broadcast Domain when the sender's IP address or MAC address has changed**. Other usages of GARP include detecting IP conflicts and during HA fail-overs.