Appendix

Pioneering Protocols

STUDENTS STUDYING ancient civilizations and languages often wonder what is the point in studying cultures that no longer exist. How will the way people lived hundreds and thousands of years ago benefit you in the here and now? Often, the answer given is that current civilizations are based in many ways upon the ancient civilizations that existed long before the present moment. The same argument, to a lesser degree, can be made for some of the early protocols that shaped and guided the world of computer networks and data communications to where it is today. Unlike ancient civilizations that no longer exist, however, some of the early network protocols still exist today and actively support computer networks. One of the earliest protocols for providing a data link connection between terminals and a mainframe computer is the BISYNC protocol. Although you may be hard pressed to find a current system still using BISYNC, the concepts and principles introduced with BISYNC can be found in more modern protocols.

Synchronous data link control (SDLC) protocol was designed to replace BISYNC and is a more modern data link control protocol. Although many descendants of SDLC have appeared over the years, SDLC is still used in IBM-type systems today. SDLC's close cousin, high-level data link control (HDLC) protocol, is also still found in some non-IBM systems. Many of the concepts introduced in SDLC (and then HDLC) have carried over into many different types of protocols.

The final topic of this appendix is the pioneering protocol for packet-switched networks—X.25. When packet-switched networks were introduced in the 1960s, the modern-day Internet (also a packet-switched network) was barely a twinkle in someone's eye. Although X.25 is still offered by some telecommunications carriers across the country, its descendants—frame relay and asynchronous transfer mode—are quickly replacing the original protocol. Once again, however, many concepts were introduced with X.25 that are used in other similar protocols today. Let's examine each of these protocols, beginning with the oldest—BISYNC.

BISYNC Transmission

Introduced in the mid-1960s, IBM's **BISYNC** protocol was designed to provide a general-purpose data link protocol for point-to-point and multipoint connections. Although the protocol itself is very old and rarely used today, the concepts introduced with BISYNC have carried over into many protocols currently in use. Because of this carryover, it is worthwhile to examine this classic protocol.

BISYNC (or BSC) is a half duplex protocol—data may be transmitted in both directions but not at the same time. Stated another way, it is a stop-and-wait protocol, in that one side sends a message, then stops and waits for a reply. Typically, one end of the connection is termed the primary, and the opposite end is termed the secondary (one secondary if point-to-point, multiple secondaries if multipoint). In most situations, the primary controls the dialog and may perform polls and selects (in which the primary sends to the secondary) of the secondary counterparts.

BISYNC is also called a character-oriented protocol. The messages or packets transmitted between primary and secondary are collections of individual characters, many of which are special control characters that drive the dialog. For example, if the primary wishes to poll the first of multiple secondaries, the following packet is transmitted:

SYN SYN address1 ENQ

The SYN character establishes synchronization of the incoming message with the receiver and precedes all message packets. It can also be inserted into the middle of longer messages to maintain synchronization. The *address1* field is the address of the intended secondary. The ENQ character, or inquiry, is used to initiate a poll. Note that, as stated earlier, each control character is an individual character. If you examine the ASCII character set, you will see that the SYN character is a valid ASCII character with the decimal value of 22; the ENQ character is a decimal 5.

If the addressed secondary has nothing to transmit, it responds with:

SYN SYN EOT

The EOT character signifies the End Of Transmission, or that the secondary has nothing to send. If the secondary had something to send to the primary, it would respond with a message such as:

SYN SYN STX text EOT BCC

The STX character signals the Start of TeXt, and data (text) follows. The BCC character is the Block Check Count, an error checksum appended to the end of the data.

If an optional header with control information is included in the packet, the message would look something like the following:

SYN SYN SOH header STX text EOT BCC

SOH indicates the Start Of Header.

Upon receipt of the data at the primary, if there is no checksum error, the primary responds with:

SYN SYN ACK0

The ACK0 character is a positive acknowledgement for even-sequenced packets.

If the packet was corrupted during transmission and the primary receives a garbled message, it replies with:

SYN SYN NAK

The NAK character is a negative acknowledgement. Retransmitting the message then becomes the responsibility of the secondary.

Table A-1 lists the more commonly found BISYNC control codes and their meanings.

Table A-1
Commonly found BISYNC characters

SYN SYN STX DLE ETX DLE ETX ETX BCC
SYN SYN DLE STX DLE DLE ETX DLE DLE ETX DLE ETX BCC

Transparency

When a sender transmits a message (or packet) to a receiver, how does the receiver know when the incoming message has ended? As Table A-1 shows, if it is the end of a block of data, the sender inserts an ETB character at the end of the data. The receiver inputs this ETB character and realizes that the end of the block has been reached and what should follow is the checksum. If it is the end of the text, the sender inserts an ETX character, and, once again, the receiver responds accordingly.

Since the receiver is carefully watching for an ETB or ETX, what would happen if an ETB or ETX suddenly appeared in the middle of the data? The receiver would erroneously assume that the end of the text had been reached and the following characters compose the checksum—which they don't—resulting in an incorrect checksum calculation.

Why would a sender insert an ETB or ETX into the middle of the text, causing all this confusion? Suppose a sender is transmitting a memory listing of a program. A memory listing consists of multiple bytes, each byte having values of hexadecimal 00 to hexadecimal FF. If one of those bytes just happens to have the exact same value as the ordinal value of the ETX character, the receiver will believe it has received the ETX character and act accordingly. To avoid this problem, a technique is needed that allows the binary equivalent of the ETX character to occur in the text but not be recognized as the control character ETX. Such a technique is termed transparency. More generally, transparency is a scheme in which any bit sequence can be included within the text, even those that may appear as control characters.

To enable transparency, the Data Link Escape character (DLE) precedes the STX character. Then, to end the stream of text characters, a DLE ETX character would be transmitted, rather than just an ETX character. The DLE would also be inserted in front of STX, ETB, ITB, ETX, ENQ, DLE, and SYN control characters.

This system seems fairly straightforward until you ask: Couldn't the binary equivalent of DLE ETX also appear as data within the text? Yes, it could. To solve this problem, an extra DLE is inserted before each DLE *in the text and only within the text*. Thus, if the sender encounters a DLE ETX sequence in the text during transmission, it inserts an extra DLE creating DLE DLE ETX. The receiver will see the DLE DLE ETX and discard the first DLE. Since it encountered two DLEs before an ETX, the receiver knows it is not receiving the control sequence DLE ETX but simply text. The only single DLE followed by an ETX should occur after the end of the text. Figure A-1 demonstrates a before and after example.

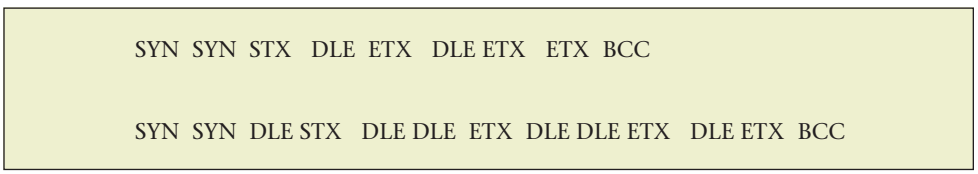

Figure A-1
Example of transparency before and after DLE insertion

Figure A-2 is a more complete example of two stations engaged in data transfer using the BISYNC protocol. Station A (the primary) is polling Stations B, C, and D (the secondaries), but only Station D has data to return to Station A. After

Station D sends its data to Station A, Station A informs Station C that it is going to receive data and to get ready. The data is transmitted but arrives garbled, so Station C asks Station A to retransmit.

Figure A-2
Two stations engaged in data transfer using the BISYNC protocol

Station A	Direction of Transmission	Station B, C, or D	Description
SYN SYN addrB ENQ	→		Poll to station B
	←	SYN SYN EOT	B has no data to send
SYN SYN addrC ENQ	→		Poll to station C
	←	SYN SYN EOT	C has no data to send
SYN SYN addrD ENQ	→		Poll to station D
	←	SYN SYN STX text ETX BCC	D sends first part of data
SYN SYN ACK0	→		Acknowledge data
	←	SYN SYN STX text EOT BCC	D sends remaining data
SYN SYN ACK1	→		Acknowledge data
SYN SYN addrB ENQ	→		Poll to station B
	←	SYN SYN EOT	B has no data to send
SYN SYN EOT SYN SYN addrC ENQ	→		Primary selects station C
	←	SYN SYN ACK0	C says OK, I'm ready, send data
SYN SYN STX text EOT BCC	→		Data is sent
	←	SYN SYN NAK	Data arrives garbled
SYN SYN STX text EOT BCC	→		Data sent again
	←	SYN SYN ACK1	Acknowledge data

Synchronous Data Link Control (SDLC)

IBM created Synchronous Data Link Control (SDLC) in the mid-1970s to replace BISYNC. It is a bit synchronous protocol in which the receiver examines individual bits looking for control information. Unlike BISYNC, SDLC is capable of supporting both half duplex and full duplex connections. SDLC is a full duplex data link protocol, because both sender and receiver may transmit at the same time. You will learn shortly about the mechanism that allows SDLC to support a full duplex connection.

Another difference between BISYNC and SDLC is that SDLC is code independent, whereas BISYNC is code dependent. BISYNC requires a data code, such as ASCII or EBCDIC, that includes control codes such as SOH, STX, or ETX in the code set. SDLC does not rely on character codes to control execution but relies instead on particular bit patterns to indicate a command.

Figure A-3 shows the basic packet format for SDLC. The flag field is an eight-bit field with the unique value 01111110. The receiver scans the incoming bit stream looking for the pattern 01111110. When it encounters the pattern, the receiver knows the beginning of the packet has arrived and continues to scan for the remainder of the packet.

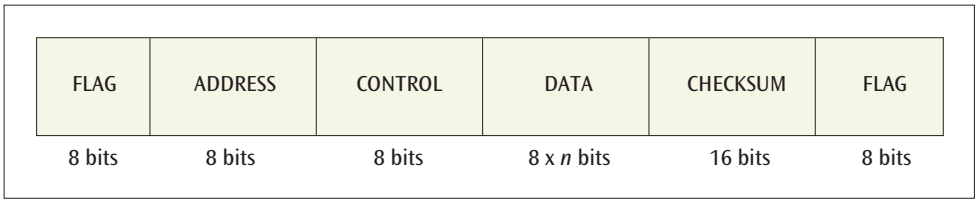

Figure A-3
Basic packet format for SDLC

Note that the same flag is used to signal the end of the packet. Just as it looked for the beginning of the packet, the receiver scans the incoming bit stream looking for this unique pattern for the ending. When the ending flag has been encountered, the receiver knows to mark the end of the packet.

As with the control codes in BISYNC, if the bit pattern 01111110 occurs in the data, the receiver will interpret it as signaling the end of the packet erroneously. SDLC needs a way of preventing the bit pattern 01111110 from occurring in the data. As the data and CRC portions of the message are transmitted, the transmitter watches for five 1s in a row. If it encounters five 1s, the transmitter automatically inserts a 0. With this procedure, the data and CRC portions of the packet can never contain 01111110. The receiver also scans the incoming bit stream, and if it finds five 1s immediately followed by a 0 within the data field of the packet, it discards the extra 0. This technique of inserting an extra bit is known as *bit stuffing*.

The address field contains an eight-bit address used to identify sender or receiver. In an unbalanced configuration (the host computer is the primary or master, the terminal is the secondary or slave), the address field always contains the address of the secondary station. (In a balanced configuration, the host computer and terminal have equal power, much like peers.)

The control field describes the type of packet for this particular message. In SDLC, there are three different types of packets:

- I (Information) packets are used to send data, flow control information, and error control information.
- S (Supervisory) packets are used to send flow and error control but no data.
- U (Unnumbered) packets are used to send supplemental link control information.

Figure A-4 shows a further breakdown of the control field. The N(S) and N(R) fields are the send and receive counts for the packets that have been transmitted from and received at a particular station. A station is capable of transmitting packets to another station and receiving packets from another station at the same time. Unlike BISYNC, which sends one packet then awaits a reply, SDLC may send multiple packets to another station before waiting for a reply.

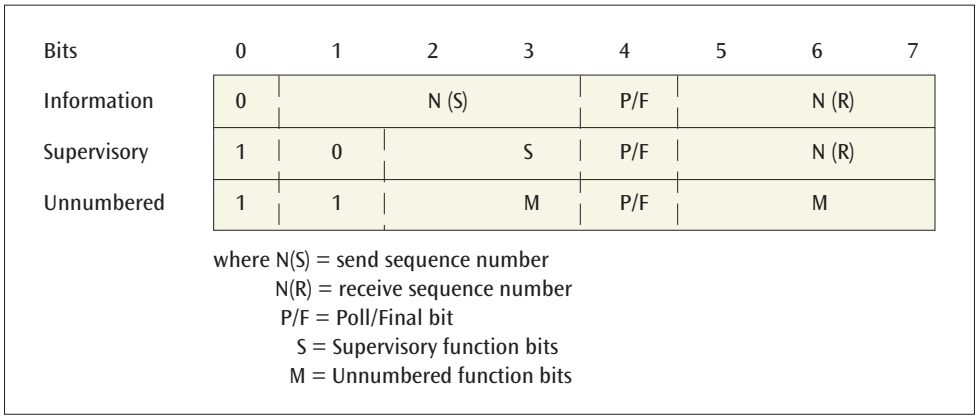

Figure A-4
Bit contents of the control field of an SDLC packet

Likewise, a receiving station does not have to acknowledge every single packet. Instead, a receiving station may wait until several packets have arrived before acknowledging any or all of the packets. For example, Station A may send seven packets (numbered 0, 1, 2, 3, 4, 5, and 6) to Station B. As the packets arrive at Station B, Station B may wait until the fourth packet (numbered 3) arrives before sending an acknowledgement. Station B would return an acknowledgment to Station A with the N(R) count set at 4, implying that packets 0 through 3 were accepted correctly, and packet 4 is the next packet expected. The N(R) and N(S) counts always reflect the next packet number expected or sent. At a later time, Station B will acknowledge the remaining three packets by transmitting a packet with the N(R) count set to 7.

So that a transmitting station does not overwhelm a receiving station with a flood of packets, SDLC has a technique that limits the number of packets a station may transmit at one time. The window size states how many packets may be unacknowledged at any given time. Assume that the window size is 7, and Station A sends six packets to another station. Station A can still send one more packet, since the window size is 7 and only six packets have been sent. If the receiving station acknowledges four of those packets, Station A can then send up to five more packets, since two of the original six packets transmitted have not yet been acknowledged. Because the number of packets that can be transmitted grows and shrinks with transmissions and acknowledgments, the technique has more accurately been called a sliding window.

The P/F bit is the Poll/Final bit. If a primary is polling a secondary, the P/F bit is a poll bit and is set to 1. If a secondary is sending multiple messages to a primary, the last message will have the P/F bit set to 1, and it will act as a final bit.

Following the control field is the data, which is of variable length but always a multiple of eight bits. After the data is the cyclic redundancy checksum, followed by the ending flag (01111110).

The S and M subfields of the control field are used by SDLC to define further the type of Supervisory or Unnumbered packets. The S bits, which exist only within the Supervisory format, are used to specify flow and error control information. Three types of Supervisory messages are available:

- Receive Ready (RR) 00—positive acknowledgment; ready to receive an Information packet

- Receive Not Ready (RNR) 01—positive acknowledgment but not ready to receive Information packets
- Reject (REJ) 10—negative acknowledgment; go back to the *n*th packet, and resend all packets from the *n*th packet on

The M bits, used only within Unnumbered packets, represent a command when the packet comes from a primary station, and a response when the packet comes from a secondary station. The available Unnumbered packet Commands follow:

- Nonsequenced information (NSI): C/R_1:00 - C/R_2:000
- Set Normal Response Mode (SNRM): 00-001
- Disconnect (DISC): 00-010
- Optional Response Poll (ORP): 00-100
- Set Initialization Mode (SIM): 10-000
- Request Station ID (XID): 11-101
- Request Task Response (TEST): 00-111
- Configure for Test (CFGR): 10-011

The available Unnumbered packet Responses follow:

- Nonsequenced Information (UI): 00-000
- Nonsequenced Acknowledgement (UA): 00-110
- Request for Initialization (RIM): 10-000
- Command Reject (FRMR): 10-100 reject packet, cannot make sense of it
- Request Online (DM): 11-000
- Test Response/Beacon (BCN): 11-111
- Disconnect Request (RD): 00-010

To better understand SDLC and its commands, you need to examine several examples. The first example, shown in Figure A-5, demonstrates the primary polling Station A followed by Station A requesting initialization information.

Figure A-5
Example showing polling and request for initialization

→ FLAG, Address-A, 10-00-1-000, CRC, FLAG Primary polls A

Decoding the Control Field *(10-00-1-000)*: *10*=Supervisory Format, *00*=Receive Ready, *1*=Poll Bit is On, *000*=N(R)=0

← FLAG, Address-A, 11-11-1-000, CRC, FLAG A requests online

Nonsequenced Format, Request Online, Final Bit is On

→ FLAG, Address-A, 11-00-1-001, CRC, FLAG Primary sets A to Normal Mode

Nonsequenced Format, Set Normal Response Mode, Poll Bit is On

← FLAG, Address-A, 11-00-1-110, CRC FLAG A acknowledges

Nonsequenced Format, Unnumbered Acknowledge, Final Bit is On

In the second example, shown in Figure A-6, the primary polls Station A to see if it has data to send, and A replies with three packets of data.

Figure A-6
Primary polls A and A responds with three packets of data

→ FLAG, Address-A, 10-00-1-000, CRC, FLAG	Primary polls A
← FLAG, Address-A, 0-000-0-000, data, CRC, FLAG	A sends first packet (data packet #0)
←FLAG, Address-A, 0-001-0-000, data, CRC, FLAG	A sends second packet (data packet #1)
← FLAG, Address-A, 0-010-1-000, data, CRC, FLAG	A sends final packet (data packet #2, final bit)
→ FLAG, Address-A, 10-00-1-011, CRC, FLAG	Primary acks packets 0-2 Primary says packet #3 next expected packet

In a third example, shown in Figure A-7, the primary polls Station B, and Station B replies with several data packets. Due to a transmission error, one packet arrives garbled and the primary requests retransmission.

Figure A-7
Primary polls Station B and B replies with several data packets

→ FLAG, Address-B, 10-00-1-000, CRC, FLAG	Primary polls B
← FLAG, Address-B, 0-000-0-000, data, CRC, FLAG	B sends first packet
← FLAG, Address-B, 0-001-0-000, data, CRC, FLAG	B sends second packet
← FLAG, Address-B, 0-010-0-000, data, CRC, FLAG	B sends third packet
← FLAG, Address-B, 0-011-0-000, data, CRC, FLAG	B sends fourth packet
← FLAG, Address-B, 0-100-0-000, data, CRC, FLAG	B sends fifth packet
← FLAG, Address-B, 0-101-0-000, data, CRC, FLAG	B sends sixth packet
← FLAG, Address-B, 0-110-0-000, data, CRC, FLAG	B sends seventh packet, stops
packet 110 arrives garbled which results in a CRC error	
→ FLAG, Address-B, 10-00-1-110, CRC, FLAG	Primary acks packets 000-101 but says Reject, go back and resend packet 110 and all subsequent packets again.
← FLAG, Address-B, 0-110-0-000, data, CRC, FLAG	B resends seventh packet
← FLAG, Address-B, 0-111-0-000, data, CRC, FLAG	B sends eighth packet
← FLAG, Address-B, 0-000-1-000, data, CRC, FLAG	B sends ninth and final packet
→ FLAG, Address-B, 10-00-1-001, CRC, FLAG	Primary acks packets 110-000

These greatly simplified examples show a primary communicating with only one secondary. However, you should note that it is quite possible for the primary to carry on concurrent conversations with multiple secondaries.

High-Level Data Link Control (HDLC)

High-level Data Link Control (HDLC) is a data link standard created by ISO that closely resembles SDLC. Typically, anyone dealing with IBM products and software would use SDLC. HDLC is used by anyone dealing with non-IBM products. Note that the two protocols are very similar but not exactly the same. It is possible to make HDLC behave like SDLC, but it is not necessarily possible to make SDLC behave like HDLC. Do not assume that the two protocols are interchangeable.

There are a number of major differences between HDLC and SDLC. For example, SDLC allows for only an eight-bit address, whereas HDLC can be extended to an address size that is a multiple of eight bits. To extend the address in HDLC, a 0 is inserted in the high-order bit position of each octet, except the last octet of the address, which has a 1 in the high-order bit position.

A further difference is that a balanced configuration is available in HDLC, but not in SDLC. In HDLC's balanced configuration, a command packet contains the destination address, and a response packet contains the sending address.

Like the extended address, the control field in HDLC may be 8 bits (as in SDLC) or 16 bits in length. The 16-bit control field allows for 7-bit N(R) and N(S) counts, thus allowing a larger window size for packet transmission. HDLC also allows for an extended checksum. The cyclic checksum field may be either 16 bits (as in SDLC) or an extended 32-bit checksum.

Unique to HDLC is the selective reject command. Supervisory packets in SDLC have Receive Ready, Receive Not Ready, and Reject (go-back-N). HDLC adds Selective Reject (SREJ), which informs the sender that a message was in error and to resend that one message but NOT all the messages that followed it. HDLC also has additional Unnumbered Commands.

Of primary interest here is the fact that multiple modes of dialog between sender and receiver may be established. The following modes exist in HDLC:

- Set Normal Response Mode (SNRM) is a primary (master)–secondary (slave) type arrangement.
- Set Normal Response Mode Extended (SNRME) is the same as SNRM except control field is 16 bits in length as opposed to standard eight-bit length.
- Set Asynchronous Response Mode (SARM) allows the secondary to initiate transmission without explicit permission of the primary, but the primary still retains responsibility of the line (initialization, error recovery, and logical disconnection).
- Set Asynchronous Response Mode Extended (SARME) is the same as SARM but in extended mode (16-bit control field).
- Set Asynchronous Balanced Mode (SABM) allows either station to initiate transmission without explicit permission from the other station. No station implicitly retains responsibility. This arrangement is a peer-to-peer connection, as opposed to the primary-secondary configuration of SDLC.
- Set Asynchronous Balanced Mode Extended (SABME) is the same as SABM but in extended mode.

Public Data Networks and X.25

Data transfer over short distances is often performed by local area networks, metropolitan area networks, and dial-up modem transmission. However, when the distance covered encompasses a state or a country, LANs and MANs can no longer do the job. Dial-up long-distance transmission using voice-grade telephone lines and modems has improved in quality and reliability over the years but is still plagued by occasional transmission noise and high telephone costs. One possible solution, presented by a number of companies, is to provide a user with a local connection to a long-haul network for a fee. The long-haul company deals with the details of transmitting the data across the network. As shown in Figure A-8, the user at location A (DTE) transmits its data to the network host 1 (DCE). The network has the responsibility of transmitting the data across the subnet to the destination DCE, where the user at site B may receive the information. Networks such as these are termed public data networks, or PDNs.

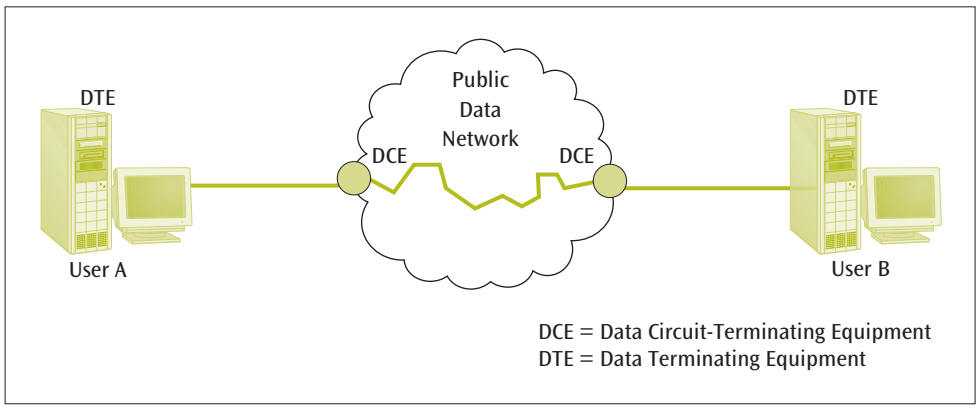

Figure A-8
Two users accessing a public data network

As an analogy, a PDN is similar to a package delivery system such as the United Parcel Service (UPS). You take the package to the UPS depot, and UPS ships it across the country in the best manner possible. The person sending the package does not necessarily care how the package gets there, just as long as it arrives undamaged in a reasonable length of time.

One advantage of using a PDN is that User A pays only for the number of characters of data transferred. This situation is like talking on the telephone to a friend and being charged only for the number of words spoken, not the overall connect time.

So that a user may connect a DTE terminal to a PDN DCE, the International Telecommunications Union (ITU) created the X.25 standard in 1974. The X.25 standard allows a uniform approach to making a connection to the network station. Since the use of X.25 involves a high-speed synchronous interface and requires a fair of amount of software and computing power, the device that is the DTE should be more powerful than a simple dumb terminal. If a user does not have a powerful enough workstation or is using a low-speed asynchronous DTE, X.25 cannot be used. Luckily, ITU provided ways to connect low-speed, asynchronous terminals to a PDN. These low-speed, asynchronous methods will be discussed later.

The three levels of X.25

X.25, much like the OSI model, is divided into levels. Since X.25 originated before the OSI model, however, the three levels do not precisely fit into the three layers of the OSI model. The lowest level of X.25, the **physical level**, follows the X.21 standard introduced in Chapter Four. Almost any physical layer protocol can be used since, like any network model, the levels are disjointed from one another. Since X.21 is not widely used, X.25 also allows the use of EIA-232 or V.24/V.28 protocols. When you use EIA-232 or V.24/V.28, the physical level interface is termed X.21 bis (secondary standard).

The second level, the **data link level**, is responsible for creating a cohesive, error-free connection between the user's DTE and the network DCE. Since these responsibilities are virtually identical to the data link layer of the OSI model, X.25 allows two variations on the HDLC data link protocol: Link Access Protocol (LAP) and LAP-B. As a data packet is passed from the network level of the DTE to the data link level, the network-level data packet is encapsulated with beginning and ending 8-bit flags, control field, address field, and a frame check sequence (Figure A-9).

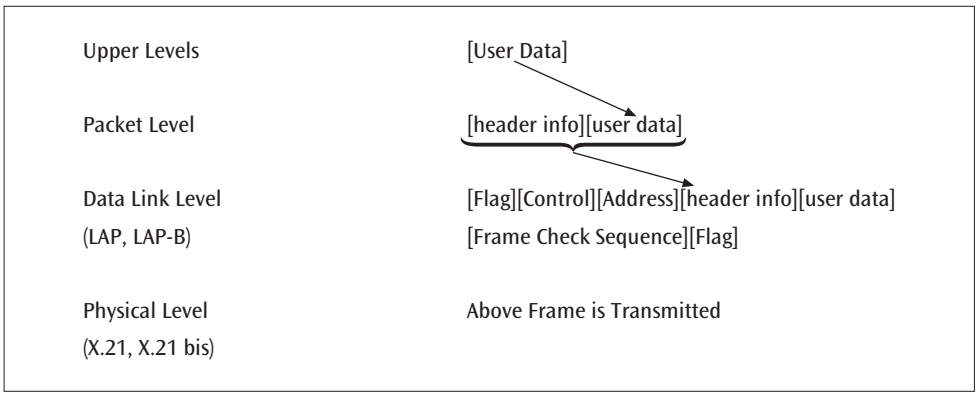

Figure A-9
X.25 levels and the flow of data between levels

It is important to note that the LAP fields (flag, control, address, frame check sequence, and flag) remain with the data packet only for the transmission from DTE to DCE. Once the packet arrives at the DCE, and before it is placed onto the network, all LAP fields are removed.

The third level of the X.25 protocol is termed the **packet level**. This level deviates the most from the OSI's third layer, the network layer. The packet level's main responsibility is to establish a connection between the user (DTE) and the network, and finally to the receiver (DTE) at the other end of the circuit (Figure A-10).

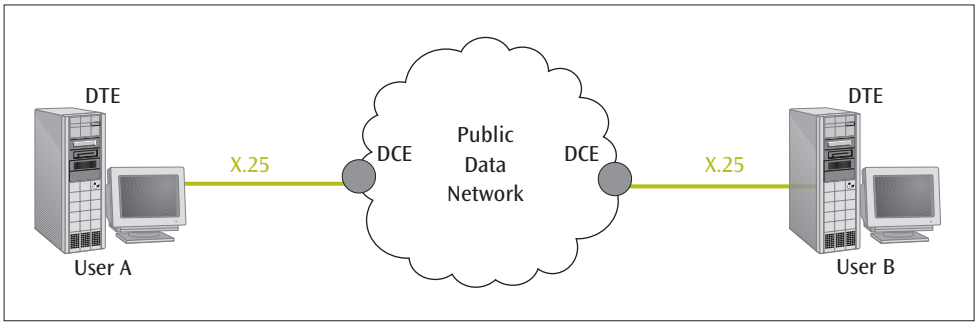

Figure A-10
X.25 connection of user to packet network

Since it is possible for a user to establish multiple connections simultaneously, each network connection is identified by a 4-bit logical group number plus an 8-bit logical channel number. Although all possible connections are not used, there is sufficient room for many concurrent logical connections over each physical channel.

A user of an X.25 PDN can choose from among four interface options. A user may establish a **permanent virtual circuit**, which is quite similar to a leased line in a public telephone network. Thus, before a session begins, the two users (or sender and receiver) and the network administration reach an agreement for a permanent virtual connection, and a logical connection number is assigned. This assigned logical connection number is then used for all future transmissions between the two users.

A second type of interface, the **virtual call**, is similar to the standard telephone call. The sending or originating DTE issues a Call Request packet, which is sent to the network. The network routes the Call Request packet to the destination DTE, which can accept or reject the request for a connection. If the destination DTE accepts, a Call Accepted packet is sent to the network, which transposes the packet into a Call Connected packet. The Call Connected packet arrives at the originating DTE and a virtual circuit is established. The two DTEs are now in a data transfer state until either DTE issues a Clear Request.

The permanent virtual circuit and virtual call interface designs are connection-oriented, in that a virtual circuit must be created with the consent of both DTEs and the network. After going through all the work to establish a connection, you would imagine that a relatively large number of data packets will be transferred between the two DTEs. But what if one DTE wishes to send only one packet of information to another DTE? Spending so much time making a connection for only one packet of data seems wasteful.

In situations in which only one packet of information is sent, a third connection strategy, the fast select, is available. **Fast select** allows one DTE to transfer data to another DTE without using call establishment and call termination procedures. The originating DTE sends a fast select packet to the network. This packet may also contain up to 128 bytes of user data. When the network delivers this packet to the destination DTE, the destination DTE has two response choices: Clear Request or Call Accepted. The first choice is to return a Clear Request packet, which may also contain 128 bytes of user data. When the originating DTE receives the Clear Request packet, that DTE realizes "I accepted your packet, thank you very much." The destination DTE could also respond with a Call Accepted packet, which then invokes the standard X.25 data transfer and clearing procedures of a virtual call.

The final interface option is the **fast select with immediate clear**. This technique is similar to the fast select, but the destination DTE has only one possible response: to return a Clear Request. Upon receipt of the Clear Request, the originating DTE returns a Clear Confirmation to the destination DTE. Thus, if a DTE has only one packet of data to send, this fourth and final interface choice is the simplest.

Packet types

So that X.25 may establish connections, transfer error-free data packets, and clear connections, several types of packets exist (Table A-2).

Table A-2
X.25 packet types

From DCE to DTE	Packet Types	From DTE to DCE
	Call Setup and Clearing	
Incoming Call		Call Request
Call Connected		Call Accepted
Clear Indication		Clear Request
DCE Clear Confirmation		DTE Clear Confirmation
	Data and Interrupt	
DCE Data		DTE Data
DCE Interrupt		DTE Interrupt
DCE Interrupt Confirmation		DTE Interrupt Confirmation
	Flow Control and Reset	
DCE Receive Ready		DTE Receive Ready
DCE Receive Not Ready		DTE Receive Not Ready
		DTE Reject
Reset Indication		Reset Request
DCE Reset Confirmation		DTE Reset Confirmation
	Restart	
Restart Indication		Restart Request
DCE Restart Confirmation		DTE Restart Confirmation
	Diagnostic	
Diagnostic		
	Registration	
Registration Confirmation		Registration Request

The call setup and clearing packets are used to create a virtual call. The originating DTE issues a Call Request packet, the network delivers an Incoming Call packet to the destination DTE, the destination DTE responds with a Call Accepted packet, and the network delivers a Call Connected packet to the originating DTE. A similar dialog follows for clearing a call.

The data and interrupt packets are used for transferring data and sending interrupts. Similar to HDLC commands, the flow control packets are used to acknowledge or suspend the actions of the DTEs. The Reset and Restart packets are used by the network and DTEs to recover from errors (such as loss of a packet), congestion, loss of the network's internal virtual circuit, or sequence number error. The Reset packet can be used to reinitialize a virtual circuit, and the Restart packet is reserved for more serious calamities.