**1.2       Communication Protocols**

Some of the communication protocols and their overview are presented here so that we understand how to use the protocols, and to provide references to more detailed descriptions of the actual design and imple­mentation of the protocols.

Following protocol suites are presented in this document.

•     the TCP/IP protocol suite (the Internet protocols),

•     Xerox Networking Systems (Xerox NS or XNS),

•     IBM's Systems Network Architecture (SNA),

•     IBM's NetBIOS,

•     the OSI protocols,

•     Unix-to-Unix Copy (UUCP).

For each protocol suite an overview is presented, and the relevant portions for the protocol's layers are discussed. The description starts from the bottom layer and work our way up.  Some of the protocols like XNS and OSI are just presented in brief.

XNS, SNA, are actually rarely used these days because of the popularity of the TCP/IP. NetBIOS is still in use in PC world and UUCP is used with the Unix systems only.

**1.2.1 TCP/IP—the Internet Protocols**

During the late 1960s and the 1970s the Advanced Research Projects Agency (ARPA) of the Department of Defense (DoD) sponsored the development of the ARPANET. The ARPANET included military, university, and research sites, and was used to support computer science and military research projects. (ARPA is now called DARPA, with the first letter of the acronym standing for "Defense.") In 1984 the DoD split the ARPANET into two networks—the ARPANET for experimental research, and the MILNET for military use. In the early 1980s a new family of protocols was specified as the standard for the ARPANET and associated DoD networks. Although the accurate name for this family of protocols is the "DARPA Internet protocol suite," it is com­monly referred to as the TCP/IP protocol suite, or just TCP/IP.

In 1987 the National Science Foundation (NSF) funded a network that connects the six national supercomputer centers together. This network is called the *NSFNET.* Physi­cally this network connects 13 sites using high-speed leased phone lines and this is called the NSFNET backbone. About eight more backbone nodes are currently planned. Addi­tionally the NSF has funded about a dozen regional networks that span almost every state. These regional networks are connected to the NSFNET backbone, and the NSFNET backbone is also connected to the DARPA Internet. The NSFNET backbone and the regional networks all use the TCP/IP protocol suite.

There are several interesting points about TCP/IP.

•      It is not vendor-specific.

•      It has been implemented on everything from personal computers to the largest supercomputers.

•      It is used for both LANs and WANs.

•      It is used by many different government agencies and commercial sites, not just DARPA-funded research projects.

The DARPA-funded research has led to the interconnection of many different individual networks into what appears as a single large network—an *internet,* as described in the previous chapter. We refer to this internet as just the Internet (capitalized).

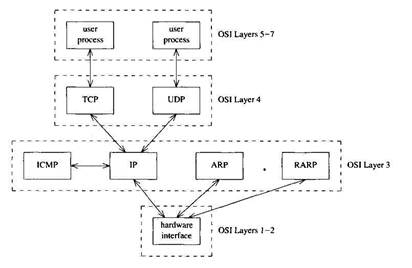
It is important to realize that while sites on the Internet use the TCP/IP protocols, many other organizations (with no government affiliation whatsoever) have established their own internets using the same TCP/IP protocols. At one extreme we have the Inter­net using TCP/IP to connect more than 150,000 computers throughout the United States, Europe and Asia, and at the other extreme we could have a network consisting of only two personal computers in the same room connected by an Ethernet using the same TCP/IP protocols.

To avoid having to qualify everything with "the TCP/IP protocol suite" or "the DARPA Internet protocol suite," we use the capitalized word *Internet,* as in "an Internet address," or "the Internet protocols," to refer to the TCP/IP protocol suite. This is to avoid confusion with internets that use protocols other than TCP/IP.

One reason for the increased use of the TCP/IP protocols during the 1980s was their inclusion in the BSD Unix system around 1982. This, along with the use of BSD Unix in technical workstations, allowed many organizations and university departments to estab­lish their own LANs.

**1.2.1.1       Overview**

Although the protocol family is referred to as TCP/IP, there are more members of this family than TCP and IP. Figure 4 shows the relationship of the protocols in the proto­col suite along with their approximate mapping into the OSI model.

**[](http://www.dsbaral.com.np/subject/network-programming/communication-protocols/clip_image002.jpg?attredirects=0)**

**Figure 4.** Layering in the Internet protocol suite

TCP              *Transmission Control Protocol.* A connection-oriented protocol that provides a reliable, full-duplex, byte stream for a user process. Most Internet application programs use TCP. Since TCP uses IP (as shown in Figure 4) the entire Internet protocol suite is often called the TCP/IP protocol family.

UDP              *User Datagram Protocol.* A connectionless protocol for user processes. Unlike TCP, which is a reliable protocol, there is no guarantee that UDP datagrams ever reach their intended destination.

ICMP           *Internet Control Message Protocol.* The protocol to handle error and control information between gateways and hosts. While ICMP messages are transmitted using IP datagrams, these messages are normally generated by and processed by the TCP/IP networking software itself, not user processes.

IP                  *Internet Protocol.* IP is the protocol that provides the packet delivery service for TCP, UDP, and ICMP. Note from the Figure 4 that user processes nor­mally do not need to be involved with the IP layer.

ARP             *Address Resolution Protocol.* The protocol that maps an Internet address into a hardware address. This protocol and the next, RARP, are not used on all networks. Only some networks need it.

RARP          *Reverse Address Resolution Protocol.* The protocol that maps a hardware address into an Internet address.

There are other protocols in the Internet protocol suite that we do not consider here—GGP (Gateway-to-Gateway Protocol) and VMTP (Versatile Message Transaction Protocol), for example.

**1.2.1.2       Data-Link Layer**

The ARPANET consists of about 50 special purpose computers that are connected together using leased telephone lines at 57.6 Kbps. The host computers and gateways on the ARPANET are then connected to these special purpose computers. The 13 backbone sites of the NSFNET are connected with Tl leased phone lines, which operate at 1.544 Mbps. Most 4.3BSD systems using the TCP/IP suite for a LAN use Ethernet tech­nology. Products also exist that use a token ring for a LAN using TCP/IP. There also exist implementations using an RS-232 serial line protocol (at speeds from 1200 bps to 19.2 Kbps) called the Serial Line Internet Protocol (SLIP). There are a variety of other data-link connections in use by TCP/IP networks: satellite links and packet radio, for example.

**1.2.1.3       Network Layer—IP**

**IP Datagrams**

The IP layer provides a connectionless and unreliable delivery system. It is connection­less because it considers each IP datagram independent of all others. Any association between datagrams must be provided by the upper layers. Every IP datagram contains the source address and the destination address (described below) so that each datagram can be delivered and routed independently. The IP layer is unreliable because it does not guarantee that IP datagrams ever get delivered or that they are delivered correctly. Reli­ability must also be provided by the upper layers. The IP layer computes and verifies a checksum that covers its own 20-byte header (that contains, for example, the source and destination addresses). This allows it to verify the fields that it needs to examine and pro­cess. But if an IP header is found in error, it is discarded, with the assumption that a higher layer protocol will retransmit the packet.

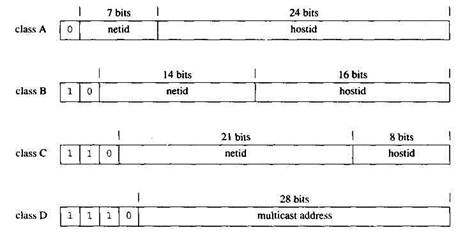
As we saw in the gateway examples from the previous chapter, it is the IP layer that handles routing through an Internet. The IP layer is also responsible for fragmentation, as described in the previous chapter. For example, if a gateway receives an IP datagram that is too large to transmit across the next network, the IP module breaks up the data­gram into fragments and sends each fragment as an IP packet. (Technically, the protocol unit exchanged by the end-to-end IP layers is an IP datagram. An IP datagram can be fragmented into smaller IP packets. When fragmentation does occur, the IP layer dupli­cates the source address and destination address into each IP packet, so the resulting IP packets can be delivered independently of each other.) The fragments are reassembled into an IP datagram only when they reach their final destination. If any of the fragments are lost or discarded, the entire datagram is discarded by the destination host.

The IP layer provides an elementary form of flow control. When IP packets arrive at a host or gateway so fast that they are discarded, the IP module sends an ICMP source quench message to the original source informing that system that the data is arriving too fast. With 4.3BSD, for example, the ICMP source quench is passed to the TCP module (assuming it was a TCP message that caused the source quench), which then decreases the amount of data being sent on that connection.

**Internet Addresses**

Every protocol suite defines some type of addressing that identifies networks and com­puters. An Internet address occupies 32 bits and encodes both a network ID and a host ID. The host ID is relative to the network ID. Every host on a TCP/IP internet must have a unique 32-bit address. TCP/IP addresses on the Internet are assigned by a central authority—Internet Assigned Number Authority (IANA).

        A 32-bit Internet address has one of the four formats shown in Figure 5

**[](http://www.dsbaral.com.np/subject/network-programming/communication-protocols/clip_image004.jpg?attredirects=0)**

**Figure 5** Internet address formats**.**

Class A addresses are used for those networks that have a lot of hosts on a single net­work, while Class C addresses allow for more networks but fewer hosts per network. For network addresses assigned by the NIC, only the type of address (Class A, B, or C) and the network ID is assigned. The requesting organization then has responsibility for assigning individual host addresses on that network. We won't consider multicast addresses (Class D) any further.

Internet addresses are usually written as four decimal numbers, separated by decimal points. Each decimal digit encodes one byte of the 32-bit Internet address. For example, the 32-bit hexadecimal value 0x0102FF04 is written as 1.2.255.4. This example is a Class A address with a network ID of 1 and a host ID of 0x02FF04. A sample Class B address is 128.3.0.5, and a sample class C address is 192.43.235.6.

Every IP datagram contains the 32-bit Internet address of the source host and the 32-bit Internet address of the destination host, in every 20-byte IP header. Since an Inter­net address is comprised of a network ID and a host ID, gateways can easily extract the network ID field from a 32-bit address and route IP datagrams based solely on the net­work ID. This is an important concept for routing, as it means that a gateway needs only to know the location of other networks, and does not to need to know the location of every host on an Internet.

Recall that a multihomed host is connected to two or more networks. This implies that it must have two or more Internet addresses, one for each network that it is con­nected to. This means that every Internet address specifies a unique host, but each host does not have a unique address.

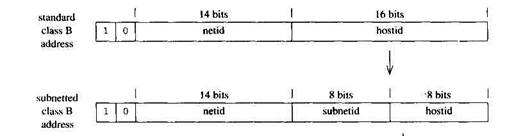
**Subnet Addresses**

Any organization with an Internet address of any class can subdivide the available host address space in any way it desires, to provide subnetworks. For example, if you have a Class B address, there are 16 bits allocated for the host ID. If your organization wants to assign host IDs to its 150 hosts, that are in turn organized into 10 physical networks, there are two different ways to do this.

•      You can allocate host IDs of 1 through 150, ignoring the physical network struc­  
ture. This requires that all the gateway systems among the 150 hosts know where  
each individual host is located, for routing purposes. Adding a new host requires  
that each gateway's routing table be updated.

•      You can allocate some of the high-order bits from the host ID, say the high-order  
8 bits, for the network ID within your subnetwork. These 8 bits are independent  
of the Class B network ID. The remaining 8 bits of the Class B host ID you then  
use to identify the individual hosts on each internal network.   Using this tech­  
nique, your gateway systems can extract the 8-bit internal network ID and use it  
for routing, instead of having to know where each of the 150 hosts are located.  
Adding a new host on an existing internal network doesn't require any changes to  
the internal gateways.

A picture of what the second option is doing is given in Figure 6.

**[](http://www.dsbaral.com.np/subject/network-programming/communication-protocols/clip_image006.jpg?attredirects=0)**

**Figure 6.** Class B Internet address with subnetting

This feature adds another level to the Internet address hierarchy.

•      network ID

•      subnet ID within network

•      host ID within subnet

**Address Resolution**

If we have an Ethernet LAN consisting of hosts using the TCP/IP protocols, we have two types of addresses: 32-bit Internet addresses and 48-bit Ethernet addresses. Recall from the previous chapter that the 48-bit Ethernet addresses are typically assigned by the manufacturer of the interface board and are all unique. We have the following address resolution problems:

•     If we know the Internet address of the other host that we want to communicate  
with, how does the IP layer determine which Ethernet address corresponds to that  
host? This is the *address resolution problem.*

•     When a diskless workstation is initialized (bootstrapped), the operating system  
can usually determine its own 48-bit Ethernet address from its interface hardware.  
But we do not want to embed the workstation's 32-bit Internet address into the  
operating system image, as this prevents us from using the same image for multi­  
ple workstations. How can the diskless workstation determine its Internet address  
at bootstrap time? This is the *reverse address resolution problem.*

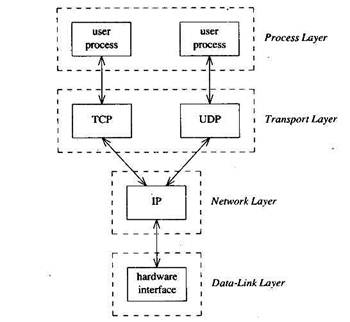
The first problem is solved using the Internet *Address Resolution Protocol (ARP),* and the second uses the Internet *Reverse Address Resolution Protocol (RARP).*

The ARP allows a host to broadcast a special packet on the Ethernet that asks the host with a specified Internet address to respond with its Ethernet address. Every host on the Ethernet receives this broadcast packet, but only the specified host should respond. Once the requesting host receives the response it can maintain the mapping between the Internet address and the Ethernet address for all future packets destined for the same Internet address.

The RARP is intended for a LAN with diskless workstations. One or more systems on the LAN are the RARP servers and contain the 32-bit Internet address and its corresponding 48-bit Ethernet address for each workstation. This allows the operating system for each workstation to be generated without having to have its Internet address as part of its configuration. When the workstation is initialized, it obtains its 48-bit Ethernet address from the interface hardware and broadcasts an Ethernet RARP packet containing its Ethernet address and asking for its Internet address. Every host on the Ethernet LAN receives this broadcast, but only the RARP servers should respond.

**1.2.1.4       Transport Layer—UDP and TCP**

User processes interact with the TCP/IP protocol suite by sending and receiving either TCP data or UDP data. The relationship of TCP and UDP to our simplified 4-layer model shown in Figure 7.

**[](http://www.dsbaral.com.np/subject/network-programming/communication-protocols/clip_image008.jpg?attredirects=0)**

**Figure 7.**  4-layer model showing UDP, TCP, and IP.

These two protocols are sometimes referred to as TCP/IP or UDP/IP, to indicate that both use IP also.

TCP provides a connection-oriented, reliable, full-duplex, byte-stream service to an application program. UDP, on the other hand, provides a connectionless, unreliable data­gram service. Figure 8 compares IP, UDP, and TCP against the modes of service.

|  |  |  |  |
| --- | --- | --- | --- |
|  | **IP** | **UDP** | **TCP** |
| connection-oriented ? | no | no | yes |
| message boundaries ? | yes | yes | no |
| data checksum ? | no | opt. | yes |
| positive ack. ? | no | no | yes |
| timeout and rexmit ? | no | no | yes |
| duplicate detection ? | no | no | yes |
| sequencing ? | no | no | yes |
| flow control ? | no | no | yes |

**Figure 8.** Comparison of protocol features for IP, UDP, and TCP.

Notice that we list IP as not having flow control, since the source quench feature described earlier is not an end-to-end flow control technique.

Since the IP layer provides an unreliable, connectionless delivery service for TCP, it is the TCP module that contains the logic necessary to provide a reliable, virtual circuit for a user process. TCP handles the establishment and termination of connections between processes, the sequencing of data that might be received out of order, the end-to-end reliability (checksums, positive acknowledgments, timeouts), and the end-to-end flow control.

UDP provides only two features that are not provided by IP: port numbers (described below) and an optional checksum to verify the contents of the UDP datagram. But these two features are enough reason for a user process to use UDP instead of trying to use IP directly, when a connectionless datagram protocol is required.

**Port Numbers**

It is possible for more than one user process at a time to be using either TCP or UDP. This requires some method for identifying the data associated with each user process. Both TCP and UDP use 16-bit integer *port numbers* for this identifi­cation.

When a client process wants to contact a server, the client must have a way of identi­fying the server that it wants. If the client knows the 32-bit Internet address of the host on which the server resides it can contact that host, but how does the client identify the particular server process? To solve this problem, both TCP and UDP have defined a group of *well-known ports.* (These are the Internet-specific *well-known addresses*) For example, every TCP/IP implementation that supports FTP, the File Transfer Protocol, assigns the well-known port of 21 (decimal) to it. TFTP, the Trivial File Transfer Protocol, is assigned the UDP port of 69.

Let's take this client-server interaction to the next step and assume that a client sends a message to the FTP server on some host by sending a message to port 21 on that host. How does the FTP server know where to send its response? First, the server can obtain the 32-bit Internet address of the client from the IP datagram, since these data­grams contain the source and destination Internet addresses in the 20-byte IP header. The client process also requests an unused port number from the TCP module on its local host. The server can obtain the 16-bit port number from the TCP header. As long as the client's TCP module does not reassign this port number to some other process before the first client is finished, there won't be any conflict. When TCP or UDP assign unique port numbers for user processes, they are called *ephemeral port numbers* (short lived). The process that receives the ephemeral port number (the client process in this example) doesn't care what value it is.   It is the other end of the communication link (the server) that needs it. TCP and UDP port numbers in the range 1 through 255 are reserved.  All the well-known ports are in this range.  Some operating systems reserve additional ports for privileged programs (4.3BSD reserves the ports 1-1023 for superuser processes), and the ephemeral ports are above these reserved ports. A hierarchical addressing scheme that involves multiple layers is described.

•      The IP datagram contains the source and destination Internet addresses in its IP  
header.  These two 32-bit values uniquely identify the two host systems that are  
communicating.

•      Also contained in the IP header is a protocol identifier, so that the IP module can  
determine if an IP datagram is for TCP, UDP, or some other protocol module that  
uses IP, which isn't discussed in this text.

•      The UDP header and the TCP header both contain the source port number and the  
destination port number. These two 16-bit integer values are used by the protocol  
modules to identify a particular user process.   Note that the TCP ports are  
independent of the UDP ports, since the IP header specifies the protocol.   TCP  
port 1035, for example, is independent of UDP port 1035.

The addition of this control information by the dif­ferent protocol modules is *encapsulation.* The combination of information from different sources using identifiers such as port numbers, protocol types, and Internet addresses is called *multiplexing.*

The 5-tuple that defines an *association* in the Internet suite consists of

•      the protocol (TCP or UDP),

•      the local host's Internet address (a 32-bit value),

•      the local port number (a 16-bit value),

•      the foreign host's Internet address (a 32-bit value),

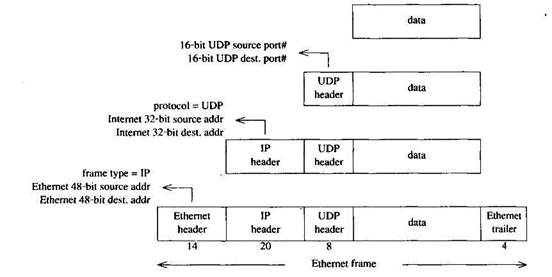
•      the foreign port number (a 16-bit value).

An example could be

(tcp,    128.10.0.3,    1500,    128.10.0.7,   21}

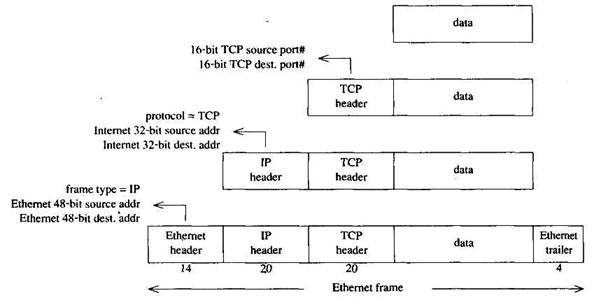
Figure 9 diagrams the encapsulation that takes place with UDP data on an Ethernet.

If the length of the IP datagram (the data, plus the UDP header, plus the IP header) is greater than the MTU of the network access layer, then the IP layer has to fragment the datagram before it is passed to the network access layer. If this happens, the receiving IP layer has to reassemble the fragments into a single datagram before it is passed to the upper layers (e.g., UDP). Whether fragmentation takes place or not, the size of the data message (the datagram) exchanged by the two UDP layers is the same.

[](http://www.dsbaral.com.np/subject/network-programming/communication-protocols/clip_image010.jpg?attredirects=0)

**Figure 9.** Encapsulation of UDP data on an Ethernet

Similarly, the encapsulation that takes place with TCP data is shown in Figure 10.

**[](http://www.dsbaral.com.np/subject/network-programming/communication-protocols/clip_image012.jpg?attredirects=0)**

**Figure 10.** Encapsulation of TCP data on an Ethernet

As with UDP, it is also possible for the TCP layer to pass messages to the IP layer that exceed the underlying network's MTU. If this happens, the sending IP layer does frag­mentation and the receiving IP layer does reassembly. For performance reasons, how­ever, most TCP implementations try to prevent IP fragmentation.

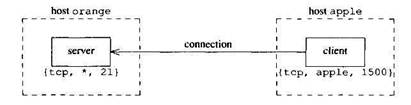
**Concurrent Servers**

With a concurrent server, what happens if the child process that is spawned by the main server continues to use its well-known port number while servicing a long request? Let's examine a typical sequence. First, the server is started on the host orange and it does a passive open using its well-known port number (21, for this example). It is now waiting for a client request.

[http://www.dsbaral.com.np/_/rsrc/1313656178861/subject/network-programming/communication-protocols/clip_image014.jpg](http://www.dsbaral.com.np/subject/network-programming/communication-protocols/clip_image014.jpg?attredirects=0)

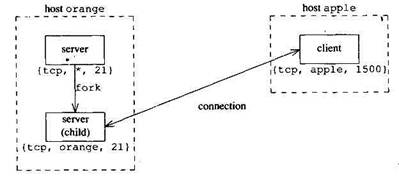
The notation {tcp, \*, 21} is used to indicate that the server is waiting for a TCP connection on any connected network (i.e., interface), on port 21. If the host on which the server is running is connected to more than one TCP/IP network, the server can specify that it only wants to accept a connection from a client on one specific network. Our asterisk notation indicates that a connection will be accepted on any network.

At some later time a client starts up and executes an active open to the server. The client has an ephemeral port number assigned to it by the protocol module (TCP, in this example). Assume the ephemeral port number is 1500 for this example, and assume that the name of the client's host is apple. This is shown in Figure 11.

**[](http://www.dsbaral.com.np/subject/network-programming/communication-protocols/clip_image016.jpg?attredirects=0)**

**Figure 11.** Connection from client to server

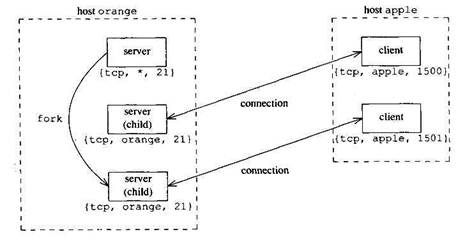
The notation {tcp, apple, 1500} is the client's *half association* or *socket.* Here we are designating the Internet address as apple, using the name of the host instead of its 4-digit notation. When the server receives the client's connection request, it forks a copy of itself, passing the connection to the child process, as shown in Figure 12.

**[](http://www.dsbaral.com.np/subject/network-programming/communication-protocols/clip_image018.jpg?attredirects=0)**

**Figure 12.** Concurrent server passes connection to child

The association is {tcp, orange, 21, apple, 1500 }. The server process that was waiting for the client connection now returns to its wait loop, letting the child pro­cess handle the client's request.

Assume that another client process on the host apple requests a connection to the same server. The client's TCP module assigns it a new ephemeral port number, say 1501, so that the half association {tcp, apple, 1501} is unique on the host apple. This gives us the picture shown in Figure 13.

**[](http://www.dsbaral.com.np/subject/network-programming/communication-protocols/clip_image020.jpg?attredirects=0)**

**Figure 13.** Second client connection passed to another child

Note that even though there are two identical half associations on the orange host, {tcp,   orange,   21}, the two complete associations are unique.

{tcp, orange, 21, apple, 1500}

{tcp, orange, 21, apple, 1501}

The TCP module on the orange system is able to determine which server child process is to receive a given data message, based on the source Internet address and the source TCP port number.

It is assumed in the examples above that the TCP module assigns a unique ephemeral port number to a process (e.g., a client) if the process does not need to assign a well-known port to itself (e.g., a server). This is what normally happens. There are instances, however, when a process can request that a specific port be used. The Internet FTP has this requirement, and it is interesting to examine this as another example of Internet addressing and port assignment.

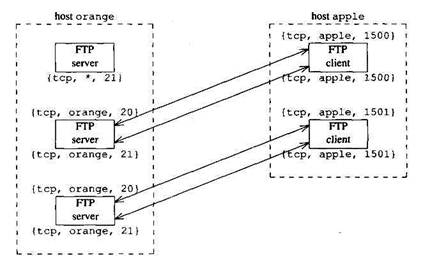
FTP normally runs using the standard client and concurrent-server relationship, as shown above. User commands are passed from the client to the server across the TCP connection, with the server's responses being returned on the connection. But when we request a file to be transferred between the client and server, another TCP connection is established between the two processes. This provides one connection for commands and one connection for data. The client initiates the transfer by sending a command across the existing connection to the server. The client also creates a new communication chan­nel, using the same port number that it is using for the control connection. The client does a passive open on this new channel, waiting for the server to complete the connec­tion. This is different from the previous examples, as the client now has to tell its TCP module that it is OK to use the same port number, even though the half association, {tcp, apple, 1500 }, for example, is not unique on the client's host. (This is done using the SO\_REUSEADDR socket option.) To guaran­tee a unique association when the server completes the connection with this new client port, the server must use some other port. FTP dictates that the server uses the well-known port 20 for this purpose. If we assume that both the client processes in our previ­ous examples are FTP clients, and both have established a data connection with their respective servers, we have the picture shown in Figure 14. All four associations are unique, as required by the Internet protocol suite.

{tcp, orange,  20, apple, 1500}

{tcp, orange,  21, apple, 1500}

{tcp, orange,  20, apple, 1501}

{tcp, orange,  21, apple, 1501}

**[](http://www.dsbaral.com.np/subject/network-programming/communication-protocols/clip_image022.jpg?attredirects=0)**

**Figure 14.** FTP example with two clients and two servers

**Buffering and Out-of-Band Data**

UDP is a datagram service. Every datagram written by a user process has a header encapsulated by the UDP layer, and then the datagram is passed to the IP layer for transmis­sion. There is no reason for the UDP layer to let the user's data hang around in the UDP layer. The data is transmitted as soon as the IP layer can get to it, so the concepts of buffering and out-of-band data do not apply to UDP.

TCP, however, presents a byte-stream service to the user process, and provides both of these features. In TCP terminology the flush output command is called the *push* flag, and out-of-band data is called *urgent data.* By definition, TCP supports any amount of out-of-band data, but not all implementations support more than one byte at a time of urgent data.

**Buffer Sizes and Limitations**

By definition the maximum size of an IP datagram is 65,536 bytes. Assuming a 20-byte IP header, this leaves up to 65,516 bytes for other data in the datagram. Realize, how­ever, that as soon as the size of an IP datagram exceeds the size of the underlying network's MTU, fragmentation occurs. Furthermore, not all TCP/IP implementations support IP fragmentation, let alone fragmentation of a 65,536-byte datagram. Every TCP/IP implementation must support a minimum IP datagram size of 576 bytes. Note that a network can have an MTU smaller than 576, but any host must be able to reassemble the fragmented IP packets into at least a 576-byte IP datagram. The actual maximum size of an IP datagram depends on the IP software on both ends, as well as the software on every gateway between the two ends.

Since UDP packets are transmitted using IP, if the result of adding the UDP header and the IP header causes the datagram to exceed the network's MTU, fragmentation occurs. This means, for example, that sending 2048-byte UDP packets on an Ethernet guarantees fragmentation.

TCP is different, since it breaks up the data into what it calls *segments.* The segment size used by TCP is agreed on between the two ends when a connection is established.

**1.2.1.5       Application Layer**

There are several application programs that are provided by almost every TCP/IP imple­mentation. One strength of the TCP/IP protocol suite is the availability of these standard applications for a variety of operating environments.

**FTP—File Transfer Protocol**

FTP is a program used to transfer files from one system to another. It provides a rich set of features and options, such as user authentication, data conversion, directory listings, and the like.

A typical sequence of events is for an interactive user to invoke an FTP client pro­cess on the local system. This client process establishes a connection with an FTP server process on the remote system using TCP. FTP establishes two connections between the client and server processes—one for control information (commands and responses) and the other for the data being transferred. The interactive user is prompted for access infor­mation on the remote system (login name and password, if required), and files can then be transferred in both directions. FTP handles both binary and text files.

**TFTP—Trivial File Transfer Protocol**

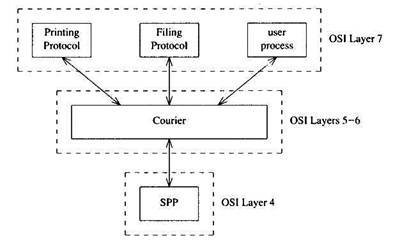
TFTP is a simpler protocol than FTP. While it provides for file transfer between a client process and a server process, it does not provide user authentication or some of the other features of FTP (listing directories, moving between directories, etc.). TFTP uses UDP, not TCP.

**TELNET—Remote Login**

TELNET provides a remote login facility. It allows an interactive user on a client system to start a login session on a remote system. Once a login session is established, the client process passes the user's keystrokes to the server process. As with the FTP program, TELNET uses TCP.

**SMTP—Simple Mail Transfer Protocol**

SMTP provides a protocol for two systems to exchange electronic mail using a TCP con­nection between the two systems.



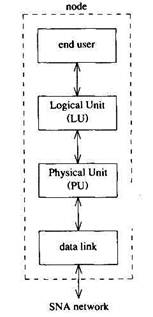
**1.2.3     SNA—Systems Network Architecture**

SNA, Systems Network Architecture, was originally released by IBM in 1974. It has since been enhanced many times and is now the predominant method used to form inter­nets of various IBM computers. Additionally, many vendors other than IBM provide various levels of SNA support, allowing many non-IBM systems to participate in SNA networks. SNA is an architecture, not a product. There exist software and hardware products from both IBM and other vendors that implement different portions of SNA.

SNA is covered here because most vendors of UNIX systems now support SNA, in one form or another. Frequently the support is for LU 6.2, which is the emphasis of this section. Unfortunately there is no common pro­gramming interface across all these products, so it is not possible to say much about the C interface to an SNA implementation.

SNA was originally designed to support the networking of nonprogrammable devices, such as terminals (termed "workstations" by IBM) and printers, to IBM main­frames. (An IBM "mainframe" is a system based on the System/370 architecture, as opposed to other IBM systems, both large and small.) Because of this non peer-to-peer evolution, where one system (the mainframe) is always the master in the communication relationship, SNA appears more complicated than the other communication architectures that are discussed (TCP/IP, for example). SNA has always had a terminology all its own, and has been confounded by the mainframe requirements just to implement and configure an SNA network—CICS, VTAM, NCP, cluster controllers, and the like. In the past years SNA has adopted the protocols and implementations that allow two user processes to communicate easily with each other, and without the requirement of a mainframe in the communication link.

The user of a network is defined to be an *end user,* which is either a user at a termi­nal or an application program (process). An end user interacts with a *logical unit* (referred to as an *LU)* to access the network. The type of services provided by the LU to the end user differs for each type of LU. LU types 2, 3, 4, and 7 support communication between processes and terminals, while LU types 1, 6.1, and 6.2 are for communication between two processes. The LU in turn interacts with a *physical unit* (referred to as a *PU).* The PU is, in essence, the operating system of the node, and it controls the data-link connections between the node and the network. It is shown in Figure 17.

**[](http://www.dsbaral.com.np/subject/network-programming/sna/clip_image002.jpg?attredirects=0)**

**Figure 17.** SNA node with logical unit and physical unit

If we consider the PU as the operating system of the node, then the data-link portion is the device driver of the node. The type of node is specified by the PU type. Five types of nodes are currently supported—1, 2.0, 2.1, 4, and 5. These are referred to, for exam­ple, as a "Type 2.1 node," or "T2.1 node," or as PU 2.1. The term "network host" or a "system on a network," is referred to by IBM as a *node*—an addressable unit on a network. IBM uses the term node, since an SNA node can be a nonprogrammable device such as a terminal or printer.

Our interest in this text is only with LU 6.2 and PU 2.1, as these are the strategic IBM products, and the only SNA implementations that adequately support communica­tion between processes.

The bibliography lists several SNA manuals available from IBM that provide addi­tional details on all the features mentioned here.

**1.2.3.1       Overview**

SNA was developed in a layered approach that is similar to the OSI model. The seven SNA layers are shown in Figure 18.

Since SNA predates the OSI model, there is no exact mapping from the seven SNA layers into the seven OSI layers. An approximate mapping between the two, along with a consolidation of the seven SNA layers into the corresponding SNA functions, is also shown in Figure 18.

In the seven SNA layers, the LU performs the functions of layers 4, 5, and 6— transmission control, data-flow control, and presentation services. The top layer, transac­tion services, is called the user process. The bottom two layers are com­bined into what we have called the network access layer, as these are the two layers that change depending on the interface types supported by the node (SDLC, token ring, etc.).

Note that the four SNA functions in Figure 18 correspond to our simplified 4-layer model. What is called a user process is called a transaction program (TP) by IBM.

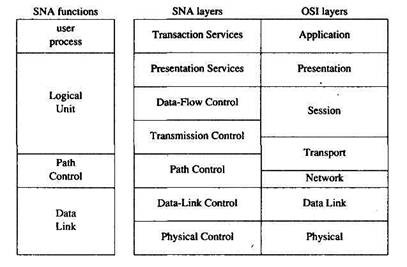
Two forms of SNA networks have evolved.

•   *Subarea SNA networks.*

These networks are built around IBM mainframes that maintain centralized con­trol over the network. Internets of independent SNA subarea networks can be formed using the SNA network interconnect (SNI) facility.

•   *APPN networks.*

APPN stands for "Advanced Peer-to-Peer Networking" and was introduced by IBM in 1986 for the System/36. It has also been called SNA/LEN (low entry net­working). In 1988 the capability was introduced for APPN networks to be con­nected together through an SNA subarea network. APPN is limited to PU Type 2.1 nodes and supports LU 6.2. APPN has features not found in subarea networks, such as dynamic routing. Baratz et al. [1985] describe the prototype implementation of APPN.

**[](http://www.dsbaral.com.np/subject/network-programming/sna/clip_image004.jpg?attredirects=0)**

**Figure 18. 7-layer SNA model and approximate mapping between SNA layers and OSI model**

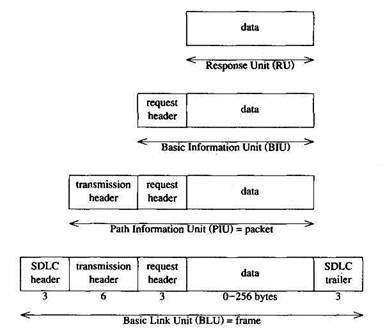
Today most of the more than 20,000 SNA networks in existence are subarea networks based on IBM mainframes. But the increased shift towards smaller systems and the increased capabilities of APPN should lead to the creation of many APPN-based net­works.

**1.2.3.2       Data-Link Layer**

The predominant form of network access used in SNA networks is SDLC—synchronous data-link control. Typical SDLC links operate between 4.8 Kbps and 57.6 Kbps over leased or switched telephone lines. The token ring is now starting to be used for LANs. One IBM product, the RT, supports LU 6.2 over an Ethernet. IBM mainframes can com­municate with each other using SNA over a System/370 data channel.

**1.2.3.3       Path-Control Layer**

In the TCP/IP and XNS protocol suites, the IP layer and the IDP layer both provided an unreliable datagram delivery service to the layers above. In SNA we encounter a dif­ferent scenario where the SNA path-control layer provides a virtual circuit service to its upper layer (the LU). This means that the path-control layer, and the data-link layer beneath it, provide error control, flow control, and sequencing.

**[](http://www.dsbaral.com.np/subject/network-programming/sna/clip_image006.jpg?attredirects=0)**

**Figure 19.** Encapsulation of SNA data on an SDLC link

Path control is the layer responsible for moving packets around in an SNA network. Figure 19 shows the encapsulation that each layer applies to a message of user data being transmitted by an SNA node, assuming an SDLC data link.

The request header is the LU header, and the transmission header is the PC (path control) header, if we follow the conventions from the TCP/IP and XNS protocol suites where the name of an encapsulated header is the name of the layer that adds the header. There is another header that is sometimes added by the LU—a function management header (FMH) that carries control information between LUs. This header is placed between the request header and the data, with a bit turned on in the request header specifying that a function management header is present.

In Figure 19 the data portion as being from 0 to 256 bytes in length is shown. The maximum response unit size is typically 256 or 512 bytes, but this size can be negotiated to other values by the two LUs.

Every LU in a given SNA network must have a unique name, from 1 to 8 characters. In an internet of SNA networks, every network must have a unique name, also from 1 to 8 characters. A particular LU in an internet is identified by its *network-qualified LU name.* This consists of

•     network name,

•     LU name within the network.

The network-qualified LU name is written as *netname.LUname* using a period between the two names. The actual mapping of this network-qualified LU name to a physical address is handled by a portion of the LU called *directory services.*

The size of the transmission header that path control uses for moving packets through a network ranges from 6 to 26 bytes, depending on the node types. A 6-byte transmission header is shown in Figure 19. These 6-byte headers are used for traffic between two adjacent Type 2.1 nodes. The 26-byte transmission header is used for data exchanges between two adjacent subarea nodes that support explicit routing and virtual routing. A complete treatment of SNA routing and SNA address formats is beyond the scope of this text. Fortunately, it is not a requisite to understanding the use of LU 6.2, which is the intent of our coverage of SNA.

Path control also does packet fragmentation and reassembly, termed *segmenting* in SNA, when a packet must be divided into pieces before being passed to the data-link layer. This is similar to the fragmentation and reassembly done by the IP layer in TCP/IP.

**1.2.3.4       LU 6.2—APPC**

LU 6.2 is also referred to as *APPC* for *Advanced Program-to-Program Communication* and was released by IBM in 1982. LU 6.2 can use either PU 2.0 or PU 2.1. The major difference is that PU 2.0 can only be used to communicate with an IBM mainframe, and has inherent in it the master-slave relationship, where the mainframe is the master. PU2.1, however, is newer, and systems implementing PU 2.1 can communicate with each other, without the need for a mainframe in the communication path. All newer IBM implementations of SNA (System/36, System/38, AS/400, RT, PC) support PU 2.1.

LU 6.2 provides a connection-oriented, reliable, half-duplex service to an application program. Note that this is the first half-duplex service that we have encountered. Both TCP and SPP are full-duplex protocols. LU 6.2, however, allows data flow between the user processes only in a single direction at a time. In Figure 20 LU 6.2 is added to the table of protocol features that is being built. This time a row for the connection-oriented protocols indicating whether the protocol provides a full-duplex data stream is added to the user process.

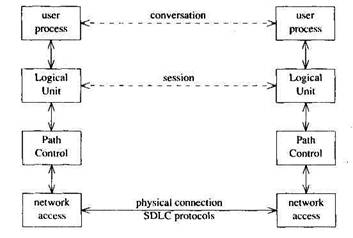
|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
|  | **IP** | **IDP** | **UDP** | **PEX** | **TCP** | **SPP** | **LU6.2** |
| connection-oriented ? | no | no | no | no | yes | yes | yes |
| message boundaries ? | yes | yes | yes | yes | no | yes | yes |
| data checksum ? | no | yes | opt. | yes | yes | yes | no |
| positive ack. ? | no | no | no | no | yes | yes | yes |
| timeout and rexmit ? | no | no | no | yes | yes | yes | yes |
| duplicate detection ? | no | no | no | no | yes | yes | yes |
| sequencing ? | no | no | no | no | yes | yes | yes |
| flow control ? | no | no | no | no | yes | yes | yes |
| full-duplex ? |  |  |  |  | yes | yes | no |

**Figure 20.** Comparison of protocol features: Internet, XNS, and SNA

Notice that LU 6.2 does not provide an end-to-end data checksum, which is different from the other connection-oriented protocols like—TCP and SPP. The developers of SNA consider the hop-by-hop reliability of the data-link layer (the cyclic redundancy check provided by the SDLC protocol, for example) to be an adequate test for corrupted data. Yet LU 6.2 provides positive acknowledgments, timeout and retransmission, duplicate detec­tion, and sequencing, so it is still considered a "reliable" protocol.

**Sessions and Conversations**

In SNA terminology, the peer-to-peer connection between two user processes is called a *conversation.* The peer-to-peer connection between two LUs is called a *session.* A ses­sion is usually a long-term connection between two LUs, while a conversation is often of a shorter duration. This is shown in Figure 21.

**[](http://www.dsbaral.com.np/subject/network-programming/sna/clip_image008.jpg?attredirects=0)**

**Figure 21.** SNA sessions and conversations

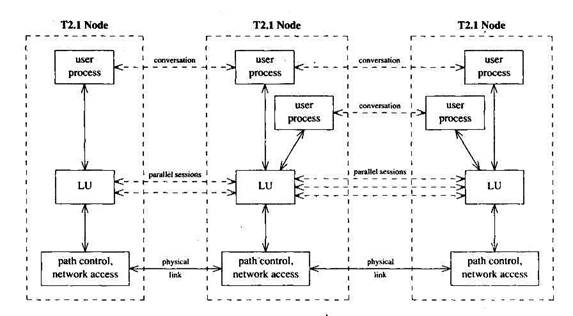
Using Figure 21 some of the features provided by a Type 2.1 node that are not available with a Type 2.0 node can be listed.

•      A T2.1 node supports multiple hardware links.

•      A T2.1 node supports *parallel sessions* for its LUs.  This means that two LUs, each in a different node, can have multiple sessions with each other.

•      A T2.1 node supports *multiple sessions* for its LUs. This means that a given LU can have sessions with more than one partner LU at the same time.

An example is shown in following Figure 22.

**[](http://www.dsbaral.com.np/subject/network-programming/sna/clip_image010.jpg?attredirects=0)**

**Figure 22.** Example of sessions and conversations between Type 2.1 nodes

Sessions are expensive to establish, so a typical LU establishes a certain number of sessions with its partner LUs. This forms a pool of active sessions for the LU to manage. When a user process wants to allocate a conversation, it requests the use of a session" from the LU for the conversation. The LU picks an available session from its pool and dedicates this session to the conversation. The user process can use this conversation for a single transaction with its partner process, or it can hold on to the conversation for a long time. When the process is finished with the conversation, the process deallocates it. This allows the LU to put the conversation back into an available pool.

Sessions are usually established by a network operator or when the system is initialized, so the typical user has no control over the session allocations. Note that LU 6.2 differs from both TCP and SPP in its explicit allocation and use of sessions. While a TCP con­nection between two user processes implies that the two TCP modules have a "session" of some form between them, we do not have to preallocate a certain number of sessions between two TCP modules. There is no inherent limit on the number of "sessions" that a TCP module can have with other TCP modules, other than any resource limits in its implementation (table sizes, number of processes allowed by the operating system, etc.). LU 6.2 requires more user control in specifying the number of sessions to allocate between LUs.

In our client-server model, a server process is identified as a process that had exe­cuted a passive open, awaiting a connection from a client process.  This implied that the server process was already executing when it did the passive open. The server was identified by a well-known address that was protocol-dependent. LU 6.2 is different in that the client specifies .the *name* of the partner program to execute—called the TP (transaction program) name. Furthermore, the target LU usually creates a new program instance (i.e., a new process) for every conversation that is started with a given TP name. Each LU associates a conversation with a unique identifier, which is called a conver­sation ID, and this ID associates a packet with a particular process. (Since a conversation has exclusive use of a session, we can call this either a conversation ID or a session ID.) In an APPN network the conversation ID is a 16-bit integer. This 16-bit value is carried in the transmission header and is used by APPN to specify the user process. From our definition of an *association* from the previous chapter, for SNA it consists of

•     the protocol (LU 6.2),

•     the local network-qualified LU name,

•     the local conversation ID,

•     the foreign network-qualified LU name,

•     the foreign conversation ID.

An example of this 5-tuple could be

{Lu62,    NET1.LUJ0E,    5,    NET1.LUMIKE,    3}

Again, we are not concerned with the conversion of a network-qualified LU name, such as NET1.LUJOE, into its internal representation as an SNA address.

**LU 6.2 Protocol Boundary**

The interface between a user process and the LU is called "presentation services" in SNA. There are two interfaces possible for a user process to LU 6.2:

•     mapped conversations,

•     unmapped conversations.

An unmapped conversation is also called a basic conversation. The major difference between the two types of conversations is in the format of the data that is exchanged between the process and the LU.

The interface between a user process and LU 6.2 is defined as a collection of *verbs* that a transaction program can execute to request a service from the LU. This defines the protocol boundary between the program and LU 6.2. The actual mapping of these verbs into an application program interface (API) depends on the specific -LU 6.2 software being used. An API could be a set of functions that a user process can call. As men­tioned earlier, there is no standard API for LU 6.2, not even among different IBM prod­ucts. Therefore, instead of describing an actual API, the different LU 6.2 verbs that most LU 6.2 implementations provide are described to an application program.

Here a brief summary of the LU 6.2 verbs are provided. Only the names of the mapped conversation verbs are listed—unless otherwise noted, a basic verb that does a similar function is available, and its name is formed by removing the MC\_prefix. For example, ALLOCATE is the basic conversation equivalent of MC\_ALLOCATE.

MC\_ALLOCATE                               Allocates a conversation with another program. Required arguments are the name of the program to execute and the name of the LU where that program is located. The process issuing the ALLOCATE starts in the send state and the partner process starts in the receive state.

MC\_CONFIRM                                  Sends a confirmation request to the remote process and waits for a reply. This allows the two processes in a conversation to synchronize their processing with each other.

MC\_CONFIRMED                            Sends a confirmation reply to the remote process.

MC\_DEALLOCATE                          Deallocates a conversation.

MC\_FLUSH                                        Forces the transmission of the local send buffer to the other LU. In general, LU 6.2 waits until a full buffer is available before sending anything to the other pro­cess.

MC\_GET\_ATTRIBUTES                 Obtains information about a conversation, such as the name of the partner LU.

MC\_PREPARE\_TO\_RECEIVE      Changes the conversation from send to receive state. Recall that LU 6.2 is a half-duplex protocol, not full-duplex.

MC\_RECEIVE\_AND\_WAIT          Waits for information to be received from the partner process. The information returned can be user data or a confirmation request.

MC\_RECEIVE\_IMMEDIATE        Receives any information that is available in the local LU's buffer, but does not wait for information to arrive.

MC\_REQUEST\_TO\_SEND             Notifies the partner process that this process wants to send data. When the local process receives a "send" indication from the partner process, the state of the conversation changes.

MC\_SEND\_DATA                             Sends one data record to the partner process.  For a mapped conversation, one user data record is option­ally mapped into a mapped conversation record (MCR), as described below. For an unmapped conversation, some portion of a logical record is sent. Note that regardless of the conversation type, this verb might not cause any data to be sent to the other process, depending on the buffering in the local LU. See the MC\_FLUSH verb above.

MC\_END\_ERROR                            Informs the partner process that the local process has detected an application error of some form.

There are additional verbs defined by LU 6.2 that are considered "control operator verbs." These are used to start sessions, change the number of sessions, stop sessions, and the like. Their usage is product dependent and they are not considered here.

The LU 6.2 protocol boundary defines numerous return codes from each of the verbs. For example, the SEND\_DATA verb can return that all is OK and additionally it can indicate that the partner process has issued a REQUEST\_TO\_SEND.

**Logical Records and GDS Variables**

All user process data consists of a 2-byte length field (LL) followed by zero or more bytes of data. This is called a *logical record* and is shown in Figure 23.

**[http://www.dsbaral.com.np/_/rsrc/1313659730148/subject/network-programming/sna/clip_image012.jpg](http://www.dsbaral.com.np/subject/network-programming/sna/clip_image012.jpg?attredirects=0)**

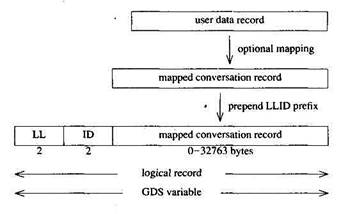
**Figure 23.** SNA logical record

The length includes the two bytes occupied by the length field, so its value is always greater than or equal to two.t

Two processes using the basic conversation verbs exchange logical records. A pro­cess that is writing logical records need not write a complete logical record with every SEND\_DATA verb. That is, a logical record consisting of 500 bytes (an LL field of 500, followed by 498 bytes of user data) can be written in three pieces—100 bytes, followed by a SEND\_DATA of 250 bytes, followed by a SEND\_DATA of 150 bytes. Similarly, the receiving process can read the logical record (RECEIVE\_AND\_WAIT) in whatever sized chunks it desires. A process must write a complete logical record, based on the LL value.

In the above example with an LL of 500, the process cannot write 400 bytes and then do a read. Not writing a complete logical record is called *truncation* and generates an error indication for the receiving process.

For mapped conversations each MC\_SEND\_DATA verb executed by the process gen­erates a single *user data record.* If a mapping is being done, the mapping is applied to the user data record, generating the *mapped conversation record (MCR).* A 2-byte LL field and a 2-byte ID field are prepended to the beginning of the MCR, generating a logi­cal record. This is shown in Figure 24.

**[](http://www.dsbaral.com.np/subject/network-programming/sna/clip_image014.jpg?attredirects=0)**

**Figure 24.** SNA mapped conversation record

This logical record is the same as the logical record that is written using the basic conver­sation verbs, except a 2-byte ID field follows the LL field. This guarantees that the length of a logical record from an MC\_SEND\_DATA verb is greater than or equal to 4 bytes (since the lengths of the LL field and the ID field are include in the length). For aft user process data, the ID field is set to 0xl2FF by LU 6.2. Other values of the ID field are used by IBM applications and by the LU for special purposes.

A logical record containing an ID field is also called a *GDS variable.* GDS is an acronym for Generalized Data Stream. A GDS variable can be comprised of multiple logical records, if the length of the MCR exceeds 32,763 bytes.

No matter how the data was written, using either the basic conversation write or the mapped conversation write, the LU is free to break up the logical records into *response units (RUs)* as it wishes. The maximum response unit size is negotiated by the two partner LUs when the session is started and its value is typically 256 bytes for an SDLC data link. The LU appends the *request header (RH)* to the response unit and passes the buffer to path control, as shown earlier.

**Confirmation**

One parameter for the ALLOCATE verbs is the synchronization level. This can be set to NONE or CONFI RM.t Specifying the confirmation option allows the user processes to use the CONFIRM and CONFIRMED verbs, which were described earlier. This provides an end-to-end confirmation that is built into the LU 6.2 protocol itself. Note that the same type of feature can be built into any protocol, if the two user processes agree on a con­vention for indicating a confirmation.

**Buffering and Out-of-Band Data**

LU 6.2 internally buffers almost everything that one process sends to another. To cir­cumvent this, the FLUSH verbs are provided to force the LU to send what is has accumu­lated in its buffer. For example, when a program ALLOCATES a conversation with another program and then executes a SEND\_DATA to transfer data to that process, the LU. by default, won't send the allocate until its buffer is full. This means that the initiat­ing process won't know if there was an error invoking the partner program until some number of SEND  DATA verbs cause the allocate to be sent to the other side.

LU 6.2 has no out-of-band data mechanism, comparable to TCP and SPP. A user process must issue a SEND\_ERROR verb to notify the other pro­cess that attention is required. It is then up to the user processes to determine how to handle things.

**Well-Known Addresses**

If the first character of the remote transaction program name (TP name) in an ALLOCATE is a special character (OxOO-OxOD or 0xl0-0x3F), then the remote TP is a *service transaction program.* These programs are typically supplied by IBM and only privileged processes can allocate conversations with them. Three service TPs are DIA, DDM, and SNADS, described in the next section. This feature corresponds to the privileged port feature of 4.3BSD for ports in the Internet domain and in the XNS domain.

**Option Sets**

The LU 6.2 specification provides a base set of features that any implementation must provide, along with 41 optional features that can also be provided by a given product. For example, one option allows conversations between programs at the same LU. Another option allows a process to flush the LU's buffer using the FLUSH verbs.

Unfortunately, this laxness in the "official" specification of the protocol makes por­tability of LU 6.2 applications less than desired. But most real-world implementations of LU 6.2 on computer systems of general interest (i.e., ignoring special purpose systems that weren't intended to be programmed by end users) tend to support options that should have been in the base set. Furthermore, the publication by IBM of their CPI-Communications API [IBM 1988] gives some indication about the features in LU 6.2 which will be supported in the future.

**1.2.3.5       Application Layer**

IBM is building distributed applications using LU 6.2. IBM differentiates between two types of user processes that are at the layer above the LU:

•     application transaction programs,

•     service transaction programs.

The only real difference is that service TPs are supplied by IBM and a service TP can provide a service for an application TP. Application TPs are user processes. Service TPs can only be invoked by privileged processes, as described above. From a network pro­gramming perspective the difference is negligible. Both types of TPs access the network in the same way through the LU.

**DIA—Document Interchange Architecture**

DIA defines the functions required for interchanging documents between different IBM office systems. DIA is implemented on top of LU 6.2. That is, DIA uses LU 6.2 for communicating between different computer systems. DIA provides the following func­tions: document library services (storing and retrieving documents), document distribu­tion (delivering documents to others in the network), and file transfer. DIA specifies the protocols and data structures used to exchange information between two DIA processes—a DIA client process and a DIA server process. An implementation of DIA usually provides a command language for an interactive user to execute DIA commands. These user commands (such as sign on, sign off, search, retrieve, deliver, delete, etc.) are translated into the appropriate DIA functions that are exchanged between the user's DIA client process and the DIA server process, using LU 6.2. *DISOSS* (Distributed Office Support System) is the IBM mainframe implementation of DIA that runs under CICS.

One type of document that is exchanged using DIA is one whose format is specified by DCA. DCA (Document Content Architecture) specifies the internal format of a docu­ment that can be exchanged between IBM office systems—the data stream and how it is to be interpreted. There are two forms of DCA, revisable form and final form. The revisable form specifies the structure of the document while it can be edited or formatted. The final form specifies the structure of the completed document in a device independent format. The intent of DCA is to allow a document to be moved between different IBM office systems, in either a revisable form or a final format.

**SNADS—SNA Distribution Services**

SNADS is an *asynchronous* distribution service for moving distribution objects (such as a document or an electronic mail message) from a source node to a destination node. Asynchronous means that it initiates the sending of the object, but the actual delivery might take place later. Furthermore, the delivery might require temporary storage in one or more intermediate nodes. The asynchronous nature of SNADS is similar to the Unix electronic mail delivery service provided by the Unix uucp program.

Contrast this with a *synchronous* delivery service where the source and destination are connected together through an LU 6.2 conversation. This synchronous capability is similar to the TCP/IP SMTP application, which uses a TCP connection between the source host and the destination host.

**DDM—Distributed Data Management**

DDM is another application available from IBM that uses LU 6.2. DDM is IBM's archi­tecture for transparent remote file access in an SNA network. DDM allows an applica­tion program to access the records in a file on another system. To do this, the application program calls the local DDM interface to request a record from a file. If the interface recognizes the request as one for a local file, the local file is accessed. Otherwise, the local interface (the DDM client process) establishes an LU 6.2 session with the DDM server process on the remote system, and the server accesses the record and returns it to the client.

**1.2.4     NetBIOS**

In 1984 IBM released its first LAN, the IBM PC Network. It was similar in concept to an Ethernet, but ran at 2 Mbps, whereas most Ethernets operate at 10 Mbps. The interface card for the IBM PC (called an "adapter card" by IBM) was developed by Sytek, Inc., and contained on it the first implementation of NetBIOS. The name *NetBIOS* is derived from the name BIOS for the "basic input output system" for the IBM PC. The BIOS was contained in read-only memory on the PC and provided an interface between a pro­gram on the PC and the actual hardware. Similarly, NetBIOS provides an interface between a program and the actual hardware on the interface card.

When IBM introduced its token ring LAN in 1985, it provided an implementation of NetBIOS for the token ring. The original PC Network implementation of NetBIOS was implemented in read-only memory on the interface card, while the token ring version was a software module. Despite the implementation differences, the token ring version pro­vided the same interface to an application program as was provided by the original PC network.

The third implementation of NetBIOS by IBM occurred when the IBM PS/2 systems were introduced and the IBM LAN Support Program was available. This software pack­age consists of device drivers and interface support for all of IBM's LAN interfaces.

NetBIOS is a software interface, not a network protocol. For example, the data packets that are exchanged across the IBM PC Network differ from those on a token ring network. The actual frame that is transmitted by two different data-link layers is expected to be different, since the data link header and trailer are different for each type of data link (token ring, Ethernet, SDLC, etc.). But the packet that is passed to the data-link layer should not change for a given protocol. For example, in the TCP/IP protocol suite, the IP datagram that starts with the IP header is the same, regardless of the data link being used. With NetBIOS this packet equivalency is not true. Nevertheless, the inter­faces provided by all IBM implementations of NetBIOS are equivalent, providing a con­sistent software interface that has become a de facto standard for personal computers. In addition, there exist implementations of NetBIOS that use TCP and UDP as the underly­ing transport protocols, and standards exist for this in the Internet (RFC 1001 and RFC 1002).

**1.2.4.1       Overview**

NetBIOS was designed for a group of personal computers, all sharing a common broad­cast medium (the IBM PC Network, like an Ethernet).  It pro­vides both a connection-oriented service (virtual circuit) and a connectionless (datagram) service. It supports both broadcast and multicast. Four types of service are provided by NetBIOS:

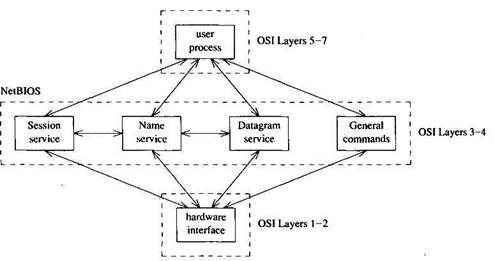
•     name service,

•     session service,

•     datagram service,

•     general commands.

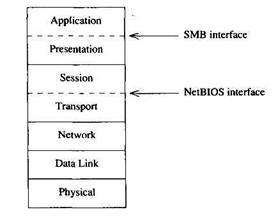
Figure 25 shows the relationship of these four services. In most implementations, a single box providing some form of datagram delivery (similar to the IP layer in the TCP/IP suite) is probably used.

**[](http://www.dsbaral.com.np/subject/network-programming/netbios/clip_image002.jpg?attredirects=0)**

**Figure 25.** Relationship of NetBIOS services

But unlike the IP layer, with NetBIOS the user process has no access to any services other than the ones described in the following sections, so the actual implementation need not concern us.

In many PC environments the application that NetBIOS is being used for is file shar­ing. In this case, another protocol interface exists above NetBIOS. This interface is called the Server Message Block protocol (SMB) and it is shown in Figure 26.

**[](http://www.dsbaral.com.np/subject/network-programming/netbios/clip_image004.jpg?attredirects=0)**

**Figure 26.** Relationship of NetBIOS and SMB to OSI model

Dunsmuir [1989] shows the format of the SMB header and gives a listing of all the SMB functions: create directory, open file, read from file, and so on. Line printer access is automatically handled by this software interface by providing three SMB operations that are for print files, not regular files. These open, write, and close a print spool file.

**1.2.4.2       Name Service**

Names are used to identify resources in NetBIOS. For example, for two processes to participate in a conversation, each must have a name. The client process identifies the specific server by the server's name, and the server can determine the name of the client. The name space is flat (that is, it is not hierarchical) and each name consists of from 1 to 16 alphanumeric characters. Upper case is different from lower case and names cannot start with an asterisk or with the three characters "IBM".

There are two types of names: *unique names* and *group names. A* unique name must be unique across the network. (As stated earlier, NetBIOS was designed for a LAN, so the name must typically be unique on the local network.) A group name does not have to be unique and all processes that have a given group name belong to the group.

There are four commands pertaining to name service.

ADD\_ NAME                           Add a unique name

ADD\_GROUP\_NAME           Add a group name

DELETE\_NAME                     Delete name

FIND\_NAME                           Determine if a name is registered

To obtain a unique name or a group name, a process must bid for the use of the name. This is done by broadcasting a notice that the process wants to use the name, as either a unique name or a group name—by issuing either the ADD\_NAME command or the ADD\_GROUP\_NAME command, respectively. If no objections are received from any other NetBIOS node, the name is considered registered by the requesting node. An objection occurs for an ADD\_NAME if some other node responds that the name is currently in use on that node as either a unique name or a group name. The only restric­tion on the ADD\_GROUP\_NAME command is that no other node can be using the name as a unique name.

Implied in the NetBIOS specification is that each NetBIOS node maintains a table of all names that processes on that node currently own. These names continue to be owned by the NetBIOS node, until the names are specifically deleted, or until the node is powered off or reset. Realize that names are handled by two different entities: name registration is done by NetBIOS for a process that issues a ADD\_NAME command, but it is NetBIOS that maintains the name table. Even though the process that registered the name might cease to exist, unless the name is specifically deleted with the DELETE\_NAME command, the node's NetBIOS name table continues to know the name.

Both the ADD\_NAME and ADD\_GROUP\_NAME commands return a *local name num­ber* that is a small integer identifying the name. This number is used for the datagram commands and for the RECEIVE\_ANY command (both described below).

By default, the IBM PC LAN Support Program transmits a name registration request six times, at half-second intervals, until it considers the name registered by itself. This implies a 3-second delay every time an application that requires a new name is started for the first time.

The FIND\_NAME command was added with the token ring implementation of Net­BIOS and determines if a particular name has been registered by any other NetBIOS node.

**1.2.4.3       Session Service**

The NetBIOS session service provides a connection-oriented, reliable, full-duplex mes­sage service to a user process. The data is organized into messages and each message can be between 0 and 131,071 bytes. NetBIOS does not provide any form of out-of-band data.

The following commands provide session service:

CALL                                 Call—active open

LISTEN                             Listen—passive open

SEND                                 Send session data

SEND\_NO\_ACK             Send session data, no acknowledgment

RECEIVE                         Receive session data

RECEIVE\_ANY              Receive session data

HANG\_UP                        Terminate session

SESSION\_STATUS      Retrieve session status

NetBIOS requires one process to be the client and another to be the server. The server first issues a passive open with the LISTEN command. The client then connects with the server when the client executes the CALL command.

The LISTEN command requires the caller (typically a server process) to specify both the local name (which must be in the local NetBIOS name table) and the remote name. The local name is the *well-known address* that the server is known by. The remote name is the name of the specific client with which a session can be established. The caller can specify the remote name as an asterisk, allowing any remote process that specifies the local name to establish a connection. Since most servers are willing to accept a connection from any process (perhaps doing some form of authentication once the session is established), specifying the remote name as an asterisk is the typical scenario. Both ends of the session can access the name of the other end.

Both the LISTEN command and the CALL command return a *local session number* to the calling process. This small integer is then used for SEND and RECEIVE com­mands to specify a particular session (since a process can have more than one session active at any time). The local session number is also used by the HANGUP command to specify which session is to be terminated. When a session is terminated, all pending data is first transferred.

In the normal SEND operation, the NetBIOS module waits for a positive acknowl­edgment frpm the other system before returning to the caller. Similarly, the RECEIVE command sends an acknowledgment to the other system before passing the received data to the caller. This provides an end-to-end verification that the data was received. When the token ring implementation of NetBIOS appeared, IBM introduced the SENDNOACK command, which does not perform the acknowledgments between the NetBIOS modules. The reason for this is that the data-link layer of the token ring net­work performs acknowledgments between the two data-link layers.

The IBM PC implementations provide a CHAINSEND command that combines two user buffers into a single message. The reason for this command is that the count associ­ated with a normal SEND command is a 16-bit integer, which allows values between 0 and 65535. By combining two sends into a single operation, messages up to 131,071 bytes can be exchanged. This command, however, is an interface issue between the Net­BIOS implementation and the user process.

The RECEIVE command receives data for a particular session that the process has open. The local session number that was returned by the CALL or LISTEN specifies the session. The RECEIVE\_ANY command allows a process to receive the next message from any of its current session partners. NetBIOS returns the actual session number corresponding to the received data.

**1.2.4.4       Datagram Service**

NetBIOS supports datagrams up to 512 bytes in length. Datagrams can be sent to a specific name (either a unique name or a group name) or can be broadcast to the entire local area network. As with other datagram services, such as UDP/1P, the NetBIOS data­grams are connectionless and unreliable. There are four datagram commands.

SEND\_DATAGRAM                                     Send datagram

SEND\_BROADCAST\_DATAGRAM         Send broadcast datagram

RECEIVE\_DATAGRAM                              Receive datagram

RECEIVE\_BROADCAST\_DATAGRAM  Receive broadcast datagram

The SEND\_DATAGRAM command requires the caller to specify the name of the destina­tion. The name can be either a unique name or a group name. If the destination is a group name, then every member of the group receives the datagram. The caller of the RECEIVE\_DATAGRAM command must specify the local name for which it wants to receive datagrams. Datagrams addressed to this name are received by the caller. The RECEIVE\_DATAGRAM command also returns the name of the sender, in addition to the ' actual datagram data. If NetBIOS receives a datagram (i.e., the NetBIOS module has previously registered the name to which the datagram was addressed), but there are no RECEIVE\_DATAGRAM commands pending, then the datagram is discarded.

The SEND\_BROADCAST\_DATAGRAM command sends the message to every NetBIOS system on the local network. When a broadcast datagram is received by a Net­BIOS node, every process that has issued a RECEIVE\_BROADCAST\_\_DATAGRAM com­mand receives the datagram. If none of these commands are outstanding when the broadcast datagram is received, the datagram is discarded. As with a normal datagram, the name of the broadcast datagram source is also returned to the receiver.

**1.2.4.5       General Commands**

There are four general commands.

RESET                              Reset NetBIOS

CANCEL                          Cancel an asynchronous command

ADAPTER\_STATUS      Fetch adapter status

UNLINK                           Unlink from bootstrap server

The RESET command clears the NetBIOS name and session tables, and also aborts any existing sessions.

The CANCEL command assumes that NetBIOS commands can be issued *asynchronously* by a user process. That is, the user process starts a command but does not wait for it to complete. Some method is required for the process to be notified when the command completes or for the process to check if a specific command is done or not. If asynchronous commands are supported by the NetBIOS implementation, then the CANCEL command cancels a specific outstanding command. If the command being can­celled is a SEND, then the associated session is aborted.

The STATUS command returns interface-specific status associated with either a local name or a remote name. Additionally, it returns the NetBIOS name table for that Net­BIOS node (either the local node or a remote node). Unfortunately, for the IBM PC implementations of NetBIOS, the actual contents of the adapter status information that is returned to the caller depends on the adapter type (PC Network or token ring).

The UNLINK command was used with the original PC Network interface when a diskless workstation was bootstrapped from a remote disk drive.

**1.2.4.6       NetBIOS Summary**

In Figure 27 the two NetBIOS services are added to the table that is being built. The terms **NbS** is used for the NetBIOS session services and **NbD** is used for the datagram ser­vices. Note that the NetBIOS session service is like LU 6.2—both assume the underly­ing data link provides reliability.

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | **IP** | **IDP** | **UDP** | **PEX** | **NbD** | **TCP** | **SPP** | **LU6.2** | **NbS** |
| connection-oriented ? | no | no | no | no | no | yes | yes | yes | yes |
| message boundaries ? | yes | yes | yes | yes | yes | no | yes | yes | yes |
| data checksum ? | no | yes | opt. | yes | no | yes | yes | no | no |
| positive ack. ? | no | no | no | no | no | yes | yes | yes | yes |
| timeout and rexmit ? | no | no | no | yes | no | yes | yes | yes | yes |
| duplicate detection ? | no | no | no | no | no | yes | yes | yes | yes |
| sequencing ? | no | no | no | no | no | yes | yes | yes | yes |
| flow control ? | no | no | no | no | no | yes | yes | yes | yes |
| full-duplex ? |  |  |  |  |  | yes | yes | no | yes |

**Figure 27.** Comparison of protocol features: Internet, XNS, SNA, and NetBIOS

Using our definition of an *association* from the previous chapter, for the NetBIOS session service it consists of

•     the protocol (NetBIOS session service),

•     the source name,

•     the source session number,

•     the destination name,

•     the destination session number.

An example could be

{NbS, JOESXT, 4, PRINTER, 7}

For the NetBIOS datagram service, only the protocol and the names are required, as there is no concept of a session for a datagram. Note also that the name of a partner process implies both the host name and the server's name.

**1.2.5     OSI Protocols**

As stated in the previous chapter, the OSI model provides a framework within which standards can be developed for protocols at each layer. The protocol stan­dards developed by ISO and other related organizations (CCITT, for example) are known as *OSI protocols.* This is in contrast to other networking protocols, most of which predate the OSI model, which have been developed by organizations other than ISO or CCITT. TCP/IP, XNS, and SNA, for example, are protocol suites that are not based on ISO stan­dards.

OSI protocols have become popular lately as many organizations (such as the U.S. government) have stated their intentions to move towards networks based on ISO standards. Unfortunately, networks based on OSI protocols are still in their infancy. Working examples of the lower layers exist, but most of the standards at these lower layers (layer 1-3) were developed before the OSI model. Standards exist and are currently being developed for the upper layers and for specific applications.

One confusing feature of the ISO standards is that their terminology differs from existing networking terminology. For example, what we call a client and server are termed as initiator and responder, respectively, in ISO-world. The concepts of an iterative server and a concurrent serer are called static responder and a dynamic responder. The packets or messages that are exchanged by peer layers are termed protocol data unit in OSI model.

**1.2.5.1       Data-Link Layer**

The data-link layer provides services to the network layer. LANs that use the OSI proto­cols typically use the IEEE 802 standards for the data-link layer and the physical layer. This provides for the IEEE 802.2 logical link control as the interface between the net­work layer and the data-link layer. The lower portion of the data-link layer, along with the physical layer, is then Ethernet (802.3), token bus (802.4), or token ring (802.5). These four IEEE standards have comparable ISO standards: 8802/2, 8802/3, 8802/4, and 8802/5. The 802.2 standard allows either a connection-oriented service or a connection­less service to be provided to the network layer.

Networks that use the OSI protocols with point-to-point connections typically use the link access procedure (LAP) that is part of the X.25 standard. This protocol is similar to the SDLC proto­col used by SNA for point-to-point links.

**1.2.5.2       Network Layer**

ISO standard 8348 defines the services provided by the network layer for the presenta­tion layer. The original version of the standard provided only for a connection-oriented network service (CONS). An addendum provides for a connectionless network service (CLNS) also.

X.25 is the name used to describe the widely used connection-oriented protocol net­work layer protocol. X.25 is a CCITT standard that first appeared in 1974. X.25 encom­passes layers 1, 2, and 3, not just the network layer. ISO standard 8878 describes how X.25 can be used to provide a connection-oriented network service.

ISO standard 8473 defines the protocol used to provide the connectionless network service. This protocol is similar to the Internet Protocol, IP. One difference is that the Internet IP uses fixed-length address fields in its IP header (the 32-bit network ID and host ID value) while the OSI IP uses variable-length address fields.

**1.2.5.3       Transport Layer**

The task of the transport layer is to provide reliable, end-to-end data transfer for users of the transport layer. ISO standard 8072 provides the definition of the services provided by the transport layer. As with the network layer, the original standard only defined the ser­vices for a connection-oriented transmission, with an addendum specifying the services for connectionless transmission.

One service that the connection-oriented transport layer must provide is expedited data, which is called out-of-band. Few specifics are given, however, other than the requirement that up to 16 bytes of expedited data be sent in a single opera­tion. Additionally, the service definition requires that normal data sent after expedited data must not be delivered to the peer before the expedited data.

The definition of the transport layer services also includes features such as establish­ing a connection between two endpoints, and the negotiation of parameters during con­nection establishment.

The specification of the actual connection-oriented transport layer protocols is given in ISO standard 8073. Included in this standard is the definition of three different types of network services that are provided to the transport layer, types A, B, and C.

Type A       A reliable network service. The network layer and the data-link layer handle all error conditions.

Type B       A reliable network service with error notification. Although most error conditions are handled by the network layer and the data-link layer for this type of service, there can be some notifications to the transport layer that something has gone wrong. A reset notification from the network layer requires that both transport ends resynchronize. A restart notifica­tion requires that both transport ends establish a new connection.

Type C       An unreliable network service. This is the type of service provided by datagram-oriented networks.

X.25 networks provide a type B network service, since both resets and restarts are possi­ble. But, it is often assumed that an X.25 network provides a reliable type A service.

Given these three types of network services, there are five different classes of connection-oriented transport protocols: classes 0 through 4. We can classify the five protocol classes by the type of network service they are intended to be used with (A, B, or C), whether they can detect errors on their own, whether they can recover from errors that are signaled by the network layer, and whether they do multiplexing. This is shown in Figure 28. Multiplexing here means the ability to have two or more transport con­nections over a single network connection.

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | Transport | | Network | | Error | | Error | | Multiplexing ? | |
| protocol class | | service type | | detection ? | | recovery ? | |  | |  |
|  | 0 | | A | | no | | no | | no | |
|  | 1 | | B | | no | | yes | | no | |
|  | 2 | | A | | no | | no | | yes | |
|  | 3 | | B | | no | | yes | | yes | |
|  | 4 | | C | | yes | | yes | | yes | |
|  |  |  |  |  |  |  |  |  |  |  |

**Figure 28.** ISO connection-oriented transport protocol classes

These five classes are sometimes called TP0 through TP4. TP0 is a simple protocol—everything is handled by the lower layers. TP1 can be used with an X.25 network service, although if a reliable X.25 service is assumed, TP0 can be used instead. TP4 is similar to the Internet TCP, since TP4 assumes an unreliable network layer. TP4 could be used with the ISO connec­tionless network layer.

**1.2.5.4       Session Layer**

The session layer provides services to a user process, in addition to the services provided by the transport layer. ISO standard 8326 defines the services to be provided by the ses­sion layer and ISO standard 8327 defines the session layer protocol.

Two of the services provided by the session layer to the layers above it are session establishment and session release. A session is similar in concept to a transport connec­tion. During the life of a session there are two possible ways for the session layer to han­dle the transport connection that it needs for the session: a single transport connection can be used for the entire session, or two or more transport connections can be used for the entire session. In the latter case, it must be transparent to the user of the session layer that the actual transport connection has changed. It is also possible for a session layer to have consecutive sessions use a single transport connection. One restriction, however, is that the session layer cannot multiplex several sessions on a single transport connection.

Another service that can be provided by the session layer is dialog management. This feature provides a half-duplex, flip-flop form of data exchange. To manage this feature, an imaginary token is maintained by the two session layers. Only the end that holds the token can transmit data. During the session establishment, it is determined which end gets the token to start. One end can also ask the other end for the token when it wants to transmit data. This half-duplex, flip-flop mode of operation is similar to the SNA LU 6.2 protocol.

There are other services that the session layer can provide: synchronization, activity management, and exception reporting. Furthermore, the ISO standard defines four sub­sets of the session services, realizing that few applications, if any, need all the features that the session layer can provide. These four subsets are called kernel, BCS (basic com­bined subset), BSS (basic synchronized subset), and BAS (basic activity subset). The simplest of these, the kernel, must be provided with any implementation. All the kernel subset provides is session establishment and data transfer.

There is nothing similar to the session layer in the TCP/IP protocol suite.

**1.2.5.5       Presentation Layer**

The presentation layer is concerned with the representation of the data that is being exchanged. This can include conversion of the data between different formats (ASCII, EBCDIC, binary), data compression, and encryption. Additionally, the presentation layer must make the services of the session layer available to the application. Much of the presentation layer, therefore, is just a pass-through of application requests (establish a session, terminate a session, etc.) to the session layer.

ISO standard 8822 defines the services for the presentation layer and ISO standard 8823 defines the protocols.

One task, of the presentation layer is to convert the application data into some stan­dard form. To explain this the ISO terminology of *abstract syntax* and *transfer syntax* are used*.* The application layer deals with an abstract syntax. This includes items such as "an integer whose value is 1." This is an abstract description that does not say how the data value is represented. A transfer syntax, however, specifies exactly how this data value is represented. For example, it could be represented as 16-bit integer in twos complement binary format with the most significant bit transferred first. To convert from an abstract syntax to a transfer syntax, *encoding rules* are applied by the presentation layer. Two presentation layers exchange data in the transfer format, while the two appli­cation layers exchange data in the abstract format.

ISO standard 8824 specifies an abstract syntax called ASN.l. This stands for "abstract syntax notation 1". The encoding rules for converting ASN.l data structures into a bit stream for transmission are contained in ISO standard 8825.

SNA supports some features that resemble the presentation layer.

**1.2.5.6       Application Layer**

**Common Application Service Elements**

CASE stands for "common application service elements." It is intended to provide capabilities that are useful to a variety of applications. Currently there are only two CASEs.

•     Association Control Service Elements (ACSE)

This element allows the user process to establish and release associations with a peer. There is a one-to-one relationship between associations and presentation layer connections.

•      Commitment, Concurrency, and Recovery (CCR)

CCR provides atomic actions between application entities. An atomic action is a set of operations, with either all operations being done or none of the operations being done—there is no in-between. Atomic operations and the techniques used by CCR have been used by distributed database systems and transaction process­ing systems for many years.

ISO standard 8649 defines the CASE services and ISO standard 8650 defines the CASE protocols.

**Electronic Mail**

In 1984 CC1TT defined a set of protocols for what it calls MHS (message handling sys­tem). The CCITT recommendations are defined in their X.400-series. These were incor­porated in the OSI model at the application layer where they are called MOTIS (message-oriented text interchange system). X.400 provides for more than simple text-oriented electronic mail. It provides for a variety of message types, including text, fac­simile, and digitized voice, for example.

Electronic mail under Unix is usually divided into two pieces. The user agent (UA) is the program the user interacts with the interactive user to send or receive mail. The user agent then communicates with a message transfer agent (MTA) that delivers the mail. Typical user agents are /usr/ucb/Mail on 4.3BSD, /bin/mail and mailx on System V, and a variety of other programs. The typical message transfer agent on 4.3BSD is sendmail. X.400 is concerned with all aspects of message handling—the user agent and the message transfer agent.

**Directory Services**

Directory services (DS) are similar to a telephone book. It maps names of people and services into their corresponding attributes (addresses, etc.). It is intended that the direc­tory services be usable by the message handling systems (MHS) and other OSI applica­tions. Directory services are sometimes classified as "white pages" or "yellow pages," similar to a telephone book, depending whether you are searching for a name or a ser­vice.

ISO standard 9594 and the CCITT X.500 recommendation specify all the details of the OSI directory.

**Virtual Terminal**

The OSI virtual terminal (VT) allows various terminals to be used. The intent is to iso­late applications from the differences in terminal characteristics. ISO standards 9040 and 9041 describe the virtual terminal services and protocols, respectively.

When a virtual terminal connection is started, the two peer entities negotiate the parameters of the terminal that can be supported. An example of the types of parameters that can be specified for the virtual terminal are: number of dimensions (two for a stan­dard CRT, three for a bit-mapped display), maximum coordinate in each dimension, allowable character sets, and so on. Some example operations that can be done are: move cursor to absolute position, enter characters starting at current position, and erase this line from cursor to end.

The ISO virtual terminal can be used to provide a remote login client and server, similar to the Internet TELNET application.

**File Transfer, Access, and Management**

The OSI file transfer, access, and management application (FTAM) is built around the concept of a virtual filestore. This virtual filestore presents a standard interface to its users. It is up to the software to map this virtual filestore into the actual filesystem being used. ISO standard'8571 specifies the services and protocols used by FTAM.

**1.2.6     UUCP— Unix-to-Unix Copy**

UUCP is the generic name used to describe a set of programs that can be used to copy files between different systems and to execute commands on other systems. An early version was made available with Version 7 Unix in 1978 and was intended for communi­cation between systems using dial-up telephone lines. Today there are two major flavors of UUCP in use, the version distributed with 4.3BSD, which is derived from the Ver­sion 7 software, and a version known as Honey DanBer UUCP. (This name is derived from the login names of its three authors, Peter Honeyman, David A. Nowitz, and Brian E. Redman.) The Honey DanBer version is distributed with System V Release 3 where it is officially called BNU—Basic Networking Utilities. All versions of UUCP communi­cate with each other, so for our purposes it doesn't matter which specific version is described.

UUCP is a collection of programs. The four that we're interested in are as follows:

uucp         This program can be invoked by users to copy a file from one system to another. The term uucp is used to refer to this specific program, and the term UUCP to refer to the collection of programs, uucp is patterned after the Unix cp command, which copies one or more files. A typical use of uucp is

uucp main.c apple\!~uucp

This command says to copy the file main. c in the current directory to the login directory of uucp on the system named apple.

uux            This program spools a command for execution on another system. Although a user can execute this command, frequently this command is generated auto­matically by the mail software or the news software. (For additional details on the Usenet news system, refer to Tanenbaum [19891.)

uucico    This program is usually run as a daemon process to perform the actions that have been requested by previous uucp or uux commands. Most Unix sys­tems invoke the uucico program automatically at various times of the day from the /usr/lib/crontab file. This is one fundamental feature of the UUCP system—it is a batch mode system. The uucp and uux commands just queue work, and the work is executed at some later time by the uucico process.

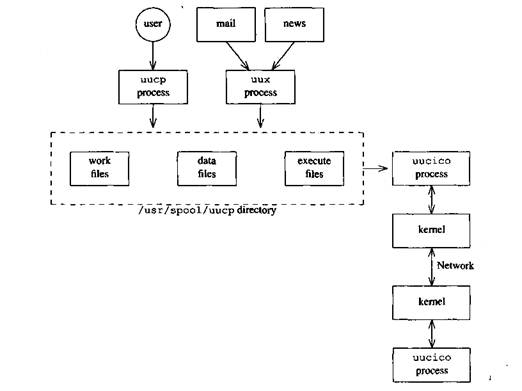
uuxqt      Executes files that were generated by uux. Normally uuxqt is invoked by uux or it is spawned by uucico to process execution files that have been received from another system.

Other programs exist in a typical UUCP software package—a program to display the jobs queued for transmission, one to remove a job that's been queued, and so on.

Figure 29 shows the typical operation of UUCP. Note that it is the uucico processes that communicate with each other across the network. There is no capability for a user process to use the UUCP protocols to communicate with some other user pro­cess.

The original uucico process supported a single data-link protocol known as the 'g' protocol. This was developed for dial-up or hardwired terminal lines and typically uses 64-byte packets. Its throughput is limited to around 9000 baud, even with higher line speeds. The 4.3BSD version supports two additional protocols. The 't' protocol assumes that the communication channel is error-free and no checksums are used. This protocol is typically used with TCP links. The 'f protocol is used for X.25 links and relies on the flow control provided by the data stream. The T protocol also applies a checksum only to the entire file (not to each packet) and uses a 7-bit data path, instead of the usual 8-bit data path.

Our overview of UUCP is to show where it belongs in relation to the other protocols described in this chapter. There is no ability for user processes to communicate across the network using UUCP. However, UUCP is an important piece of the networking tools used by most Unix sites.

**[](http://www.dsbaral.com.np/subject/network-programming/uucp/clip_image002.jpg?attredirects=0)**

**Figure 29.** Overview of UUCP processes

**1.2.7     Protocol Comparisons**

The comparison of the different protocols that are discussed are presented here in different aspects..

**Fragmentation and Reassembly**

TCP/UDP/IP          IP handles this. TCP and UDP need not worry about it, other than per­formance implications. Once a packet is fragmented, it is not reassem­bled until it reaches the destination host.

XNS                        Never done.  If a packet must be fragmented by some data-link layer, then it must be reassembled by the other end of the link, to prevent any other systems from knowing that it happened. The packet size is almost always 576 bytes, which is less than optimal for an Ethernet.

SNA                        Done by path control.

NetBIOS                An implementation detail that is not specified.

**Flow Control**

IP                       The IP layer sends ICMP source quench messages to a host when packets are arriving too quickly from that host. This is a form of hop-by-hop flow control.

TCP                   Provides end-to-end flow control whereby the receiver tells the sender how much data it can accept.

UDP                   No flow control is provided by UDP (other than the ICMP source quench feature of the underlying IP layer).

IDP                    The error protocol can be used to notify a host when a packet from that host has been discarded because of resource limitations. This is similar to the ICMP source quench.

SPP                    Provides end-to-end flow control, similar to TCP.

PEX                   No flow control is provided (other than the error control packets provided by the underlying IDP layer).

SNA                   Link-to-link flow control is provided by the data-link layer for most SNA data links (SDLC, token ring). Additionally, an end-to-end flow control is also provided by the LU.

NetBIOS           An implementation detail that is not specified.

**Reliability**

The issue of reliability is a touchy area when talking about whether some form of end-to-end reliability is needed. (This is one of the "religious topics" of networking.) Obvi­ously, if the underlying data-link and network layers are unreliable, the transport layer must provide an end-to-end check. This is how TCP and UDP operate. But with UDP the end-to-end reliability is optional and there are some implementations that disable it when running on a "reliable" LAN such as an Ethernet. The reason we say that an Ethernet is reliable is that every Ethernet frame has a 32-bit CRC appended to it, to verify that the data is exchanged without any errors between the two Ethernet interface cards. Other LAN technology, such as the token ring, also provides a 32-bit CRC to provide link-to-link reliability.

If the data-link layer is reliable, do we still need an end-to-end reliability check? Without answering this touchy question we should point out where the data could be cor­rupted, even if it is transferred between the two interface cards without any errors. First, the interface card contains memory in which the incoming and outgoing frames are stored. This memory can be a source of errors. The data is also transferred between the interface card and the computer's memory, typically along an input/output bus of some form. On some LANs there might also be bridges or repeaters, which can also be the source of errors.

IP                       Provides a 16-bit checksum that covers the IP header only.  The upper layers (TCP and UDP) are left to do their own checksum of the actual message, if desired.

TCP                   Assumes the IP layer is unreliable and provides a 16-bit checksum of the entire message.

UDP                   As with TCP, it provides a 16-bit checksum of the entire message.

XNS                   IDP provides a 16-bit checksum of the entire packet, therefore no additional end-to-end reliability checks are required by the upper layers (SPP and PEX).

SNA                   Assumes the data-link layer provides the reliability.  No form of end-to-end checksum is provided.

NetBIOS           Both the datagram service and the session service assume the data link layer is reliable, with no additional checksums provided.

**Sequencing**

The datagram protocols do not perform sequencing—IP, UDP, IDP, PEX, and NetBIOS datagram. The connection-oriented protocols all provide this feature—TCP, SPP, LU 6.2, and NetBIOS session. SNA assumes that the data-link layer does the sequenc­ing, with a verification of this taking place in the LU. NetBIOS also assumes the data link does the sequencing. TCP and SPP both do the sequencing themselves, assuming the underlying layers do not sequence the data.

**Timeout and Retransmission**

Most datagram protocols do not provide this service (IP, UDP, IDP, and NetBIOS). PEX, however, is a datagram service that provides a timeout and retransmission. As expected, the connection-oriented protocols do provide this service—TCP, SPP, LU 6.2, and NetBIOS. As with sequencing, both SNA and NetBIOS require the data link to han­dle this.