NEAR EAST UNIVERSITY

Faculty of Engineering

Department of Computer Engineering

TCP/IP AND REAL TIME SYSTEM

Graduation Project
COM-400

Student: Nuray İşlek

Supervisor: Assoc. Prof. Dr. Doğan İbrahim

Nicosia 2001

TABLE OF CONTENTS

ACKNOWLEDGMENT

Education has given success to many students throughout life. And being one of these students must be the most honorable certificate, I could get. I believe that this honor does not only affect us but also to the teachers who have given us the strength of education.

I would like to thank all my teachers and especially Assoc. Prof. Dr. Doğan Ibrahim for helping me throughout my University Education. Being in the Department of Computer Engineering has not only given me an education about computer technology but also has given me a future that I have wanted all my life.

Thank You....

Nuray İşlek

950138

ABSTARCT

TCP and IP were developed by the U.S. Department of Defense (DOD) research project to connect a number different networks designed by different vendors into a network of networks (the "Internet"). It was initially successful because it delivered a few basic services that everyone needs (file transfer, electronic mail, remote logon) across a very large number of client and server systems. Several computers in a small department can use TCP/IP (along with other protocols) on a single LAN. The IP component provides routing from the department to the enterprise network, then to regional networks, and finally to the global Internet. On the battlefield a communications network will sustain damage, so the DOD designed TCP/IP to be robust and automatically recover from any node or phone line failure. This design allows the construction of very large networks with less central management. However, because of the automatic recovery, network problems can go undiagnosed and uncorrected for long periods of time.

As with all other communications protocol, TCP/IP is composed of layers: IP which is responsible for moving packet of data from node to node. IP forwards each packet based on a four byte destination address (the IP number). The Internet authorities assign ranges of numbers to different organizations. The organizations assign groups of their numbers to departments. IP operates on gateway machines that move data from department to organization to region and then around the world. TCP is responsible for verifying the correct delivery of data from client to server. Data can be lost in the intermediate network. TCP adds support to detect errors or lost data and to trigger retransmission until the data is correctly and completely received and Sockets - is a name given to the package of subroutines that provide access to TCP/IP on most systems.

In this project, the TCP/IP protocols have been studied and their applications to real-time programming are discussed. It is shown that TCP/IP can be used in real-time client-server based applications such as factory automation and so on.

3. Introduction

The Internet Protocol was developed to create a Network of Networks (the "Internet"). Individual machines are first connected to a LAN (Ethernet or Token Ring). TCP/IP shares the LAN with other uses (a Novell file server, Windows for Workgroups peer systems). One device provides the TCP/IP connection between the LAN and the rest of the world.

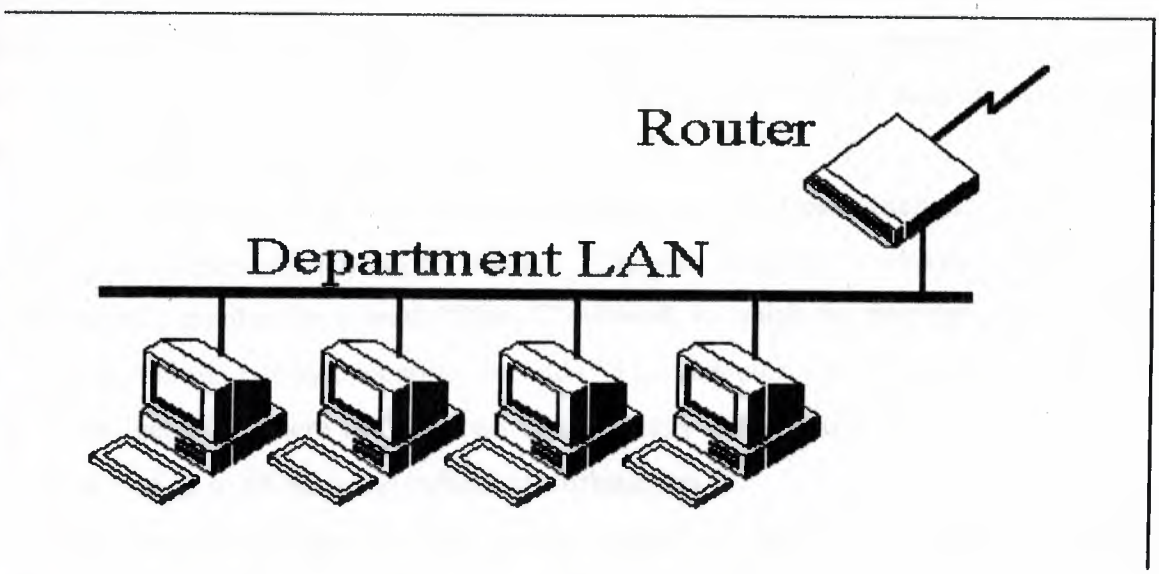


Figure 1. The connection between the LAN and the rest of the world.

To insure that all types of systems from all vendors can communicate, TCP/IP is absolutely standardized on the LAN. However, larger networks based on long distances and phone lines are more volatile. In the US, many large corporations would wish to reuse large internal networks based on IBM's SNA. In Europe, the national phone companies traditionally standardize on X.25. However, the sudden explosion of high speed microprocessors, fiber optics, and digital phone systems has created a burst of new options: ISDN, frame relay, FDDI, Asynchronous Transfer Mode (ATM). New technologies arise and become obsolete within a few years. With cable TV and phone companies competing to build the National Information Superhighway, no single standard can govern citywide, nationwide, or worldwide communications. The original design of TCP/IP as a Network of Networks fits nicely within the current technological uncertainty. TCP/IP data can be sent across a LAN, or it can be carried within an internal corporate SNA network, or it can piggyback on the cable TV

service. Furthermore, machines connected to any of these networks can communicate to any other network through gateways supplied by the network vendor.

3 .1 Addresses

Each technology has its own convention for transmitting messages between two machines within the same network. On a LAN, messages are sent between machines by supplying the six byte unique identifier (the "MAC" address). In an SNA network, every machine has Logical Units with their own network address. DECNET, Appletalk, and Novell IPX all have a scheme for assigning numbers to each local network and to each workstation attached to the network. On top of these local or vendor specific network addresses, TCP/IP assigns a unique number to every workstation in the world. This "IP number" is a four byte value that, by convention, is expressed by converting each byte into a decimal number (0 to 255) and separating the bytes with a period.

An organization begins by sending electronic mail to Hostmaster@INTERNIC.NET requesting assignment of a network number. It is still possible for almost anyone to get assignment of a number for a small "Class C" network in which the first three bytes identify the network and the last byte identifies the individual computer. Larger organizations can get a "Class B" network where the first two bytes identify the network and the last two bytes identify each of up to 64 thousand individual workstations.

The organization then connects to the Internet through one of a dozen regional or specialized network suppliers. The network vendor is given the subscriber network number and adds it to the routing configuration in its own machines and those of the other major network suppliers. The machines that manage large regional networks or the central Internet routers managed by the National Science Foundation can only locate these networks by looking each network number up in a table. There are potentially thousands of Class B networks, and millions of Class C networks, but computer memory costs are low, so the tables are reasonable. Customers that connect to the Internet, even customers as large as IBM, do not need to maintain any information on other networks. They send all external data to the regional carrier to which they subscribe, and the regional carrier maintains the tables and does the appropriate routing.

3.2 Subnets

Although the individual subscribers do not need to tabulate network numbers or provide explicit routing, it is convenient for most Class B networks to be internally managed as a much smaller and simpler version of the larger network organizations.

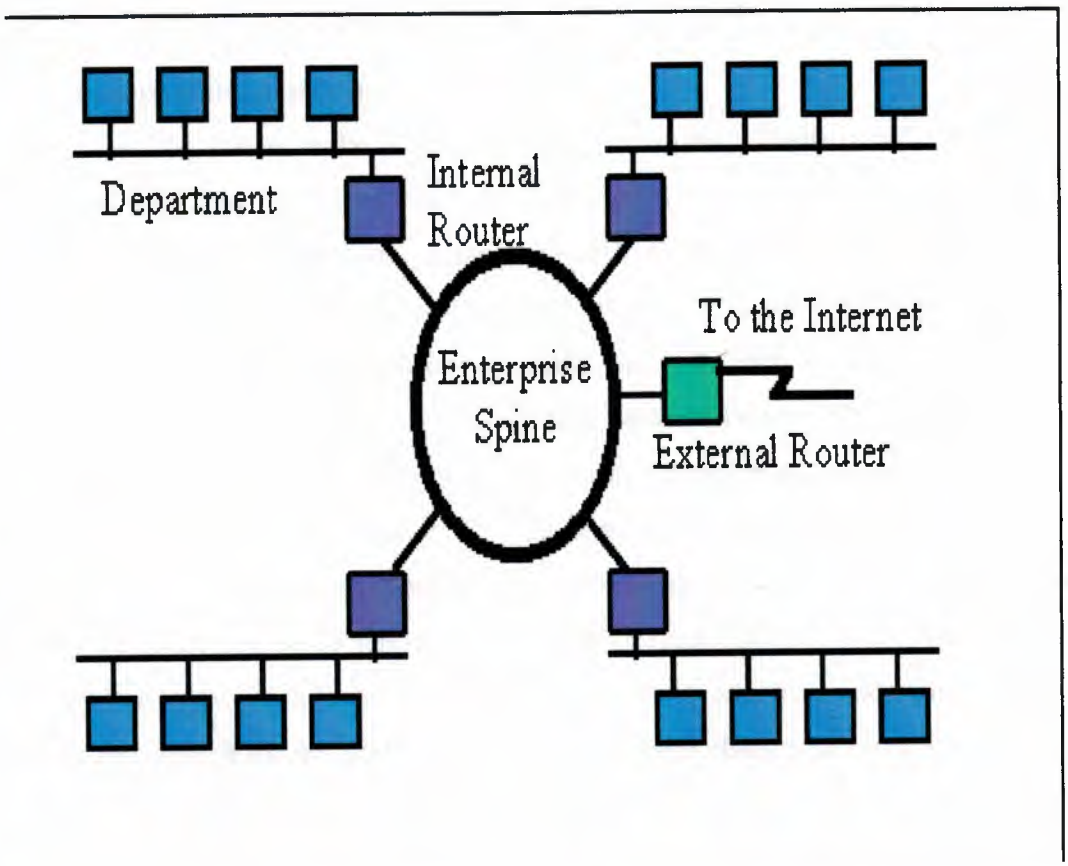


Figure 2. . It is common to subdivide the two bytes available for internal assignment into a one byte department number and a one byte workstation ID.

The enterprise network is built using commercially available TCP/IP router boxes. Each router has small tables with 255 entries to translate the one byte department number into selection of a destination Ethernet connected to one of the routers. Messages to the PC Lube and Tune server (130.132.59.234) are sent through the national and New England regional networks based on the 130.132 part of the number. Arriving at Yale, the 59 department ID selects an Ethernet connector in the C& IS building. The 234 selects a particular workstation on that LAN. The Yale network must be updated as new Ethernets and departments are added, but it is not effected by changes outside the university or the movement of machines within the department.

3.3 A Uncertain Path

Every time a message arrives at an IP router, it makes an individual decision about where to send it next. There is concept of a session with a preselected path for all traffic. Consider a company with facilities in New York, Los Angeles, Chicago and Atlanta. It could build a network from four phone lines forming a loop (NY to Chicago to LA to Atlanta to NY). A message arriving at the NY router could go to LA via either Chicago or Atlanta. The reply could come back the other way.

How does the router make a decision between routes? There is no correct answer. Traffic could be routed by the "clockwise" algorithm (go NY to Atlanta, LA to Chicago). The routers could alternate, sending one message to Atlanta and the next to Chicago. More sophisticated routing measures traffic patterns and sends data through the least busy link. If one phone line in this network breaks down, traffic can still reach its destination through a roundabout path. After losing the NY to Chicago line, data can be sent NY to Atlanta to LA to Chicago. This provides continued service though with degraded performance. This kind of recovery is the primary design feature of IP. The loss of the line is immediately detected by the routers in NY and Chicago, but somehow this information must be sent to the other nodes. Otherwise, LA could continue to send NY messages through Chicago, where they arrive at a "dead end." Each network adopts some Router Protocol which periodically updates the routing tables throughout the network with information about changes in route status.

If the size of the network grows, then the complexity of the routing updates will increase as will the cost of transmitting them. Building a single network that covers the entire US would be unreasonably complicated. Fortunately, the Internet is designed as a Network of Networks. This means that loops and redundancy are built into each regional carrier. The regional network handles its own problems and reroutes messages internally. Its Router Protocol updates the tables in its own routers, but no routing updates need to propagate from a regional carrier to the NSF spine or to the other regions (unless, of course, a subscriber switches permanently from one region to another).

3.4 Undiagnosed Problems

IBM designs its SNA networks to be centrally managed. If any error occurs, it is reported to the network authorities. By design, any error is a problem that should be corrected or repaired. IP networks, however, were designed to be robust. In battlefield conditions, the loss of a node or line is a normal circumstance. Casualties can be sorted out later on, but the network must stay up. So IP networks are robust. They automatically (and silently) reconfigure themselves

when something goes wrong. If there is enough redundancy built into the system, then communication is maintained. In 1975 when SNA was designed, such redundancy would be prohibitively expensive, or it might have been argued that only the Defense Department could afford it. Today, however, simple routers cost no more than a PC. However, the TCP/IP design that, "Errors are normal and can be largely ignored," produces problems of its own.

Data traffic is frequently organized around "hubs," much like airline traffic. One could imagine an IP router in Atlanta routing messages for smaller cities throughout the Southeast. The problem is that data arrives without a reservation. Airline companies experience the problem around major events, like the Super Bowl. Just before the game, everyone wants to fly into the city. After the game, everyone wants to fly out. Imbalance occurs on the network when something new gets advertised. Adam Curry announced the server at "mtv.com" and his regional carrier was swamped with traffic the next day. The problem is that messages come in from the entire world over high speed lines, but they go out to mvt.com over what was then a slow speed phone line.

Occasionally a snow storm cancels flights and airports fill up with stranded passengers. Many go off to hotels in town. When data arrives at a congested router, there is no place to send the overflow. Excess packets are simply discarded. It becomes the responsibility of the sender to retry the data a few seconds later and to persist until it finally gets through. This recovery is provided by the TCP component of the Internet protocol.

TCP was designed to recover from node or line failures where the network propagates routing table changes to all router nodes. Since the update takes some time, TCP is slow to initiate recovery. The TCP algorithms are not tuned to optimally handle packet loss due to traffic congestion. Instead, the traditional Internet response to traffic problems has been to increase the speed of lines and equipment in order to say ahead of growth in demand.

TCP treats the data as a stream of bytes. It logically assigns a sequence number to each byte. The TCP packet has a header that says, in effect, "This packet starts with byte 379642 and contains 200 bytes of data." The receiver can detect missing or incorrectly sequenced packets. TCP acknowledges data that has been received and retransmits data that has been lost. The TCP design means that error recovery is done end-to-end between the Client and Server machine. There is no formal standard for tracking problems in the middle of the network, though each network has adopted some ad hoc tools.

3.5 Need to Know

There are three levels of TCP/IP knowledge. Those who administer a regional or national network must design a system of long distance phone lines, dedicated routing devices, and very large configuration files. They must know the IP numbers and physical locations of thousands of subscriber networks. They must also have a formal network monitor strategy to detect problems and respond quickly.

Each large company or university that subscribes to the Internet must have an intermediate level of network organization and expertise. A half dozen routers might be configured to connect several dozen departmental LANs in several buildings. All traffic outside the organization would typically be routed to a single connection to a regional network provider.

However, the end user can install TCP/IP on a personal computer without any knowledge of either the corporate or regional network. Three pieces of information are required:

1. The IP address assigned to this personal computer
2. The part of the IP address (the subnet mask) that distinguishes other machines on the same LAN (messages can be sent to them directly) from machines in other departments or elsewhere in the world (which are sent to a router machine)
3. The IP address of the router machine that connects this LAN to the rest of the world.

In the case of the PCLT server, the IP address is 130.132.59.234. Since the first three bytes designate this department, a "subnet mask" is defined as 255.255.255.0 (255 is the largest byte value and represents the number with all bits turned on). It is a Yale convention (which we recommend to everyone) that the router for each department have station number 1 within the department network. Thus the PCLT router is 130.132.59.1. Thus the PCLT server is configured with the values:

- My IP address: 130.132.59.234
- Subnet mask: 255.255.255.0
- Default router: 130.132.59.1

The subnet mask tells the server that any other machine with an IP address beginning 130.132.59.* is on the same department LAN, so messages are sent to it directly. Any IP address beginning with a different value is accessed indirectly by sending the message through the router at 130.132.59.1 (which is on the departmental LAN).

4. What is TCP/IP?

TCP/IP is a set of protocols developed to allow cooperating computers to share resources across a network. It was developed by a community of researchers centered around the ARPAnet. Certainly the ARPAnet is the best-known TCP/IP network. However as of June, 87, at least 130 different vendors had products that support TCP/IP, and thousands of networks of all kinds use it. First some basic definitions. The most accurate name for the set of protocols we are describing is the "Internet protocol suite". TCP and IP are two of the protocols in this suite. (They will be described below.) Because TCP and IP are the best known of the protocols, it has become common to use the term TCP/IP or IP/TCP to refer to the whole family. It is probably not worth fighting this habit. However this can lead to some oddities. For example, I find myself talking about NFS as being based on TCP/IP, even though it doesn't use TCP at all. (It does use IP. But it uses an alternative protocol, UDP, instead of TCP. All of this alphabet soup will be unscrambled in the following pages.)

The Internet is a collection of networks, including the Arpanet, NSFnet, regional networks such as NYsernet, local networks at a number of University and research institutions, and a number of military networks. The term "Internet" applies to this entire set of networks. The subset of them that is managed by the Department of Defense is referred to as the "DDN" (Defense Data Network). This includes some research-oriented networks, such as the Arpanet, as well as more strictly military ones. (Because much of the funding for Internet protocol developments is done via the DDN organization, the terms Internet and DDN can sometimes seem equivalent.) All of these networks are connected to each other. Users can send messages from any of them to any other, except where there are security or other policy restrictions on access. Officially speaking, the Internet protocol documents are simply standards adopted by the Internet community for its own use. More recently, the Department of Defense issued a MILSPEC definition of TCP/IP. This was intended to be a more formal definition, appropriate for use in purchasing specifications. However most of the TCP/IP community continues to use the Internet standards. The MILSPEC version is intended to be consistent with it.

Whatever it is called, TCP/IP is a family of protocols. A few provide "low-level" functions needed for many applications. These include IP, TCP, and UDP. (These will be described in a bit more detail later.) Others are protocols for doing specific tasks, e.g.

transferring files between computers, sending mail, or finding out who is logged in on another computer. Initially TCP/IP was used mostly between minicomputers or mainframes. These machines had their own disks, and generally were self-contained. Thus the most important "traditional" TCP/IP services are:

- file transfer. The file transfer protocol (FTP) allows a user on any computer to get files from another computer, or to send files to another computer. Security is handled by requiring the user to specify a user name and password for the other computer. Provisions are made for handling file transfer between machines with different character set, end of line conventions, etc. This is not quite the same thing as more recent "network file system" or "netbios" protocols, which will be described below. Rather, FTP is a utility that you run any time you want to access a file on another system. You use it to copy the file to your own system. You then work with the local copy. (See RFC 959 for specifications for FTP.)
- remote login. The network terminal protocol (TELNET) allows a user to log in on any other computer on the network. You start a remote session by specifying a computer to connect to. From that time until you finish the session, anything you type is sent to the other computer. Note that you are really still talking to your own computer. But the telnet program effectively makes your computer invisible while it is running. Every character you type is sent directly to the other system. Generally, the connection to the remote computer behaves much like a dialup connection. That is, the remote system will ask you to log in and give a password, in whatever manner it would normally ask a user who had just dialed it up. When you log off of the other computer, the telnet program exits, and you will find yourself talking to your own computer. Microcomputer implementations of telnet generally include a terminal emulator for some common type of terminal. (See RFC's 854 and 855 for specifications for telnet. By the way, the telnet protocol should not be confused with Telenet, a vendor of commercial network services.)
- computer mail. This allows you to send messages to users on other computers. Originally, people tended to use only one or two specific computers. They would maintain "mail files" on those machines. The computer mail system is simply a way for you to add a message to another user's mail file. There are some problems with this in an environment where microcomputers are used. The most serious is that a micro is not well suited to receive computer mail. When you send mail, the mail

software expects to be able to open a connection to the addressee's computer, in order to send the mail. If this is a microcomputer, it may be turned off, or it may be running an application other than the mail system. For this reason, mail is normally handled by a larger system, where it is practical to have a mail server running all the time. Microcomputer mail software then becomes a user interface that retrieves mail from the mail server. (See RFC 821 and 822 for specifications for computer mail. See RFC 937 for a protocol designed for microcomputers to use in reading mail from a mail server.)

These services should be present in any implementation of TCP/IP, except that micro-oriented implementations may not support computer mail. These traditional applications still play a very important role in TCP/IP-based networks. However more recently, the way in which networks are used has been changing. The older model of a number of large, self-sufficient computers is beginning to change. Now many installations have several kinds of computers, including microcomputers, workstations, minicomputers, and mainframes. These computers are likely to be configured to perform specialized tasks. Although people are still likely to work with one specific computer, that computer will call on other systems on the net for specialized services. This has led to the "server/client" model of network services. A server is a system that provides a specific service for the rest of the network. A client is another system that uses that service. (Note that the server and client need not be on different computers. They could be different programs running on the same computer.) Here are the kinds of servers typically present in a modern computer setup. Note that these computer services can all be provided within the framework of TCP/IP.

- network file systems. This allows a system to access files on another computer in a somewhat more closely integrated fashion than FTP. A network file system provides the illusion that disks or other devices from one system are directly connected to other systems. There is no need to use a special network utility to access a file on another system. Your computer simply thinks it has some extra disk drives. These extra "virtual" drives refer to the other system's disks. This capability is useful for several different purposes. It lets you put large disks on a few computers, but still give others access to the disk space. Aside from the obvious economic benefits, this allows people working on several computers to share common files. It makes system maintenance and backup easier, because you don't have to worry about updating and backing up

copies on lots of different machines. A number of vendors now offer high-performance diskless computers. These computers have no disk drives at all. They are entirely dependent upon disks attached to common "file servers". (See RFC's 1001 and 1002 for a description of PC-oriented NetBIOS over TCP. In the workstation and minicomputer area, Sun's Network File System is more likely to be used. Protocol specifications for it are available from Sun Microsystems.)

- remote printing. This allows you to access printers on other computers as if they were directly attached to yours. (The most commonly used protocol is the remote lineprinter protocol from Berkeley Unix. Unfortunately, there is no protocol document for this. However the C code is easily obtained from Berkeley, so implementations are common.)
- remote execution. This allows you to request that a particular program be run on a different computer. This is useful when you can do most of your work on a small computer, but a few tasks require the resources of a larger system. There are a number of different kinds of remote execution. Some operate on a command by command basis. That is, you request that a specific command or set of commands should run on some specific computer. (More sophisticated versions will choose a system that happens to be free.) However there are also "remote procedure call" systems that allow a program to call a subroutine that will run on another computer. (There are many protocols of this sort. Berkeley Unix contains two servers to execute commands remotely: rsh and rexec. The man pages describe the protocols that they use. The user-contributed software with Berkeley 4.3 contains a "distributed shell" that will distribute tasks among a set of systems, depending upon load. Remote procedure call mechanisms have been a topic for research for a number of years, so many organizations have implementations of such facilities. The most widespread commercially-supported remote procedure call protocols seem to be Xerox's Courier and Sun's RPC. Protocol documents are available from Xerox and Sun. There is a public implementation of Courier over TCP as part of the user-contributed software with Berkeley 4.3. An implementation of RPC was posted to Usenet by Sun, and also appears as part of the user-contributed software with Berkeley 4.3.)
- name servers. In large installations, there are a number of different collections of names that have to be managed. This includes users and their passwords, names and network addresses for computers, and accounts. It becomes very tedious to keep this data up to date on all of the computers. Thus the databases are kept on a small number

of systems. Other systems access the data over the network. (RFC 822 and 823 describe the name server protocol used to keep track of host names and Internet addresses on the Internet. This is now a required part of any TCP/IP implementation. IEN 116 describes an older name server protocol that is used by a few terminal servers and other products to look up host names. Sun's Yellow Pages system is designed as a general mechanism to handle user names, file sharing groups, and other databases commonly used by Unix systems. It is widely available commercially. Its protocol definition is available from Sun.)

- terminal servers. Many installations no longer connect terminals directly to computers. Instead they connect them to terminal servers. A terminal server is simply a small computer that only knows how to run telnet (or some other protocol to do remote login). If your terminal is connected to one of these, you simply type the name of a computer, and you are connected to it. Generally it is possible to have active connections to more than one computer at the same time. The terminal server will have provisions to switch between connections rapidly, and to notify you when output is waiting for another connection. (Terminal servers use the telnet protocol, already mentioned. However any real terminal server will also have to support name service and a number of other protocols.)
- network-oriented window systems. Until recently, high- performance graphics programs had to execute on a computer that had a bit-mapped graphics screen directly attached to it. Network window systems allow a program to use a display on a different computer. Full-scale network window systems provide an interface that lets you distribute jobs to the systems that are best suited to handle them, but still give you a single graphically-based user interface. (The most widely-implemented window system is X. A protocol description is available from MIT's Project Athena. A reference implementation is publically available from MIT. A number of vendors are also supporting NeWS, a window system defined by Sun. Both of these systems are designed to use TCP/IP.)

Note that some of the protocols are not officially part of the Internet protocol suite. However they are implemented using TCP/IP, just as normal TCP/IP application protocols are. Since the protocol definitions are not considered proprietary, and since commercially-support implementations are widely available, it is reasonable to think of these protocols as being effectively part of the Internet suite. Note that the list above is simply a sample of the

sort of services available through TCP/IP. However it does contain the majority of the "major" applications. The other commonly-used protocols tend to be specialized facilities for getting information of various kinds, such as who is logged in, the time of day, etc.

Transmission Control Protocol/Internet Protocol (TCP/IP) is the main transport protocol used on the Internet for connectivity and transmission of data across heterogenous systems. It is an open standard which is available on most Unix systems, VMS and other minicomputer systems, many mainframe & supercomputing systems and some microcomputer & PC systems.

TCP/IP is a software solution for network connectivity. There is little assumption on the hardware system used for actual physical connections. The most common hardware solution is Ethernet, but TCP/IP will also run on Token-Ring, AT&T StarLAN, microwave & spread spectrum systems , LocalTalk (needs a gateway), Serial lines (modems, serial connections) and other systems as well. To run TCP/IP on a system you first need a hardware driver. On Macintosh systems, the hardware drivers are built into the system or is provided by the board manufacturer. On a PC system, there are different types of hardware drivers available both commercially and via public domain/shareware including the Packet driver specification by FTP Software, Inc., Microsoft's Network Device Interface Specification (NDIS), & Novell's Open Datalink Interface (ODI). Drivers for OS/2 systems are available from IBM and/or the board manufacturer (if they support OS/2). If a driver is not available for your hardware, look for a shim. This is a software device which translates between two driver specifications. There are shims for ODI-on-NDIS, NDIS-on-Packet driver. ODI-on-Packet driver, etc. usually publically available.

You then need a TCP/IP stack. This is package specific usually comes with every product. Each such stack has its own requirements for hardware drivers. you must find a combination of driver & TCP/IP stack which is compatible with your hardware & system. Macintosh's do not have a problem since most Macintosh systems use the MacTCP stack which is available from Apple and is provided with most if not all Macintosh TCP/IP packages. PC systems have something close to a standard in TCP applications called the Windows Sockets API (Winsock). [Note: This is not specific only to TCP/IP it is a general standard for networking on PC irrelevant of the transport protocol. Hence, there may be versions for NetBEUI, IPX, etc.]. The Winsock API is avaialble in 16 bit and 32 bit versions. The 32 bit versions are for Windows NT systems. Winsock is implemented in Dynamically Loaded Libraries or DLLs.

Currently work is under way to develop a freeware Winsock DLL but many commercial versions are available. With the TCP/IP stack in hand, you then need all the TCP/IP application programs such as Telnet, FTP, mail, etc. Just about every TCP/IP package has a corresponding set of applications although some do not provide all the different applications available.

5. General description of the TCP/IP protocols

TCP/IP is a layered set of protocols. In order to understand what this means, it is useful to look at an example. A typical situation is sending mail. First, there is a protocol for mail. This defines a set of commands which one machine sends to another, e.g. commands to specify who the sender of the message is, who it is being sent to, and then the text of the message. However this protocol assumes that there is a way to communicate reliably between the two computers. Mail, like other application protocols, simply defines a set of commands and messages to be sent. It is designed to be used together with TCP and IP. TCP is responsible for making sure that the commands get through to the other end. It keeps track of what is sent, and retransmitts anything that did not get through. If any message is too large for one datagram, e.g. the text of the mail, TCP will split it up into several datagrams, and make sure that they all arrive correctly. Since these functions are needed for many applications, they are put together into a separate protocol, rather than being part of the specifications for sending mail. You can think of TCP as forming a library of routines that applications can use when they need reliable network communications with another computer. Similarly, TCP calls on the services of IP. Although the services that TCP supplies are needed by many applications, there are still some kinds of applications that don't need them. However there are some services that every application needs. So these services are put together into IP. As with TCP, you can think of IP as a library of routines that TCP calls on, but which is also available to applications that don't use TCP. This strategy of building several levels of protocol is called "layering". We think of the applications programs such as mail, TCP, and IP, as being separate "layers", each of which calls on the services of the layer below it. Generally, TCP/IP applications use 4 layers:

- an application protocol such as mail
- a protocol such as TCP that provides services need by many applications
- IP, which provides the basic service of getting datagrams to their destination
- the protocols needed to manage a specific physical medium, such as Ethernet or a point to point line.

TCP/IP is based on the "catenet model". (This is described in more detail in IEN 48.) This model assumes that there are a large number of independent networks connected together by gateways. The user should be able to access computers or other resources on any of these networks. Datagrams will often pass through a dozen different networks before getting to their final destination. The routing needed to accomplish this should be completely invisible to the user. As far as the user is concerned, all he needs to know in order to access another system is an "Internet address". This is an address that looks like 128.6.4.194. It is actually a 32-bit number. However it is normally written as 4 decimal numbers, each representing 8 bits of the address. (The term "octet" is used by Internet documentation for such 8-bit chunks. The term "byte" is not used, because TCP/IP is supported by some computers that have byte sizes other than 8 bits.) Generally the structure of the address gives you some information about how to get to the system. For example, 128.6 is a network number assigned by a central authority to Rutgers University. Rutgers uses the next octet to indicate which of the campus Ethernets is involved. 128.6.4 happens to be an Ethernet used by the Computer Science Department. The last octet allows for up to 254 systems on each Ethernet. (It is 254 because 0 and 255 are not allowed, for reasons that will be discussed later.) Note that 128.6.4.194 and 128.6.5.194 would be different systems. The structure of an Internet address is described in a bit more detail later.

Of course we normally refer to systems by name, rather than by Internet address. When we specify a name, the network software looks it up in a database, and comes up with the corresponding Internet address. Most of the network software deals strictly in terms of the address. (RFC 882 describes the name server technology used to handle this lookup.)

TCP/IP is built on "connectionless" technology. Information is transfered as a sequence of "datagrams". A datagram is a collection of data that is sent as a single message. Each of these datagrams is sent through the network individually. There are provisions to open connections (i.e. to start a conversation that will continue for some time). However at some level, information from those connections is broken up into datagrams, and those datagrams are treated by the network as completely separate. For example, suppose you want to transfer a 15000 octet file. Most networks can't handle a 15000 octet datagram. So the protocols will break this up into something like 30 500-octet datagrams. Each of these datagrams will be sent to the other end. At that point, they will be put back together into the 15000-octet file. However while those datagrams are in transit, the network doesn't know that there is any

connection between them. It is perfectly possible that datagram 14 will actually arrive before datagram 13. It is also possible that somewhere in the network, an error will occur, and some datagram won't get through at all. In that case, that datagram has to be sent again.

Note by the way that the terms "datagram" and "packet" often seem to be nearly interchangable. Technically, datagram is the right word to use when describing TCP/IP. A datagram is a unit of data, which is what the protocols deal with. A packet is a physical thing, appearing on an Ethernet or some wire. In most cases a packet simply contains a datagram, so there is very little difference. However they can differ. When TCP/IP is used on top of X.25, the X.25 interface breaks the datagrams up into 128-byte packets. This is invisible to IP, because the packets are put back together into a single datagram at the other end before being processed by TCP/IP. So in this case, one IP datagram would be carried by several packets. However with most media, there are efficiency advantages to sending one datagram per packet, and so the distinction tends to vanish.

5.1 The TCP level

Two separate protocols are involved in handling TCP/IP datagrams. TCP (the "transmission control protocol") is responsible for breaking up the message into datagrams, reassembling them at the other end, resending anything that gets lost, and putting things back in the right order. IP (the "internet protocol") is responsible for routing individual datagrams. It may seem like TCP is doing all the work. And in small networks that is true. However in the Internet, simply getting a datagram to its destination can be a complex job. A connection may require the datagram to go through several networks at Rutgers, a serial line to the John von Neuman Supercomputer Center, a couple of Ethernets there, a series of 56Kbaud phone lines to another NSFnet site, and more Ethernets on another campus. Keeping track of the routes to all of the destinations and handling incompatibilities among different transport media turns out to be a complex job. Note that the interface between TCP and IP is fairly simple. TCP simply hands IP a datagram with a destination. IP doesn't know how this datagram relates to any datagram before it or after it.

It may have occurred to you that something is missing here. We have talked about Internet addresses, but not about how you keep track of multiple connections to a given system. Clearly it isn't enough to get a datagram to the right destination. TCP has to know which connection this datagram is part of. This task is referred to as "demultiplexing." In fact, there

are several levels of demultiplexing going on in TCP/IP. The information needed to do this demultiplexing is contained in a series of "headers". A header is simply a few extra octets tacked onto the beginning of a datagram by some protocol in order to keep track of it. It's a lot like putting a letter into an envelope and putting an address on the outside of the envelope. Except with modern networks it happens several times. It's like you put the letter into a little envelope, your secretary puts that into a somewhat bigger envelope, the campus mail center puts that envelope into a still bigger one, etc. Here is an overview of the headers that get stuck on a message that passes through a typical TCP/IP network:

We start with a single data stream, say a file you are trying to send to some other computer:

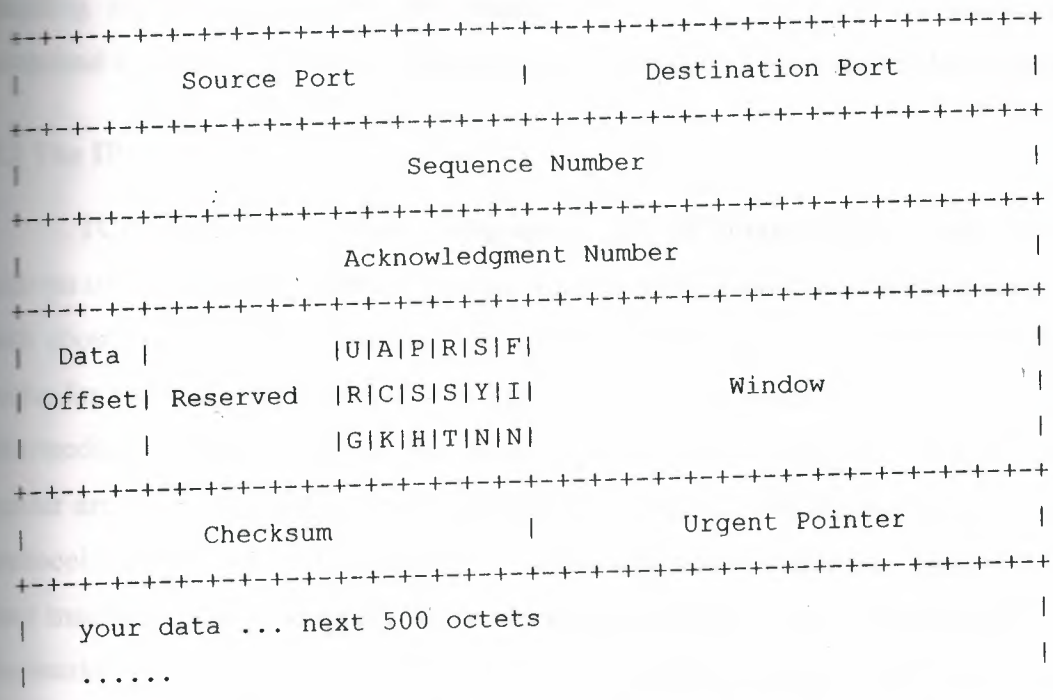
.....

TCP breaks it up into manageable chunks. (In order to do this, TCP has to know how large a datagram your network can handle. Actually, the TCP's at each end say how big a datagram they can handle, and then they pick the smallest size.)

....

TCP puts a header at the front of each datagram. This header actually contains at least 20 octets, but the most important ones are a source and destination "port number" and a "sequence number". The port numbers are used to keep track of different conversations. Suppose 3 different people are transferring files. Your TCP might allocate port numbers 1000, 1001, and 1002 to these transfers. When you are sending a datagram, this becomes the "source" port number, since you are the source of the datagram. Of course the TCP at the other end has assigned a port number of its own for the conversation. Your TCP has to know the port number used by the other end as well. (It finds out when the connection starts, as we will explain below.) It puts this in the "destination" port field. Of course if the other end sends a datagram back to you, the source and destination port numbers will be reversed, since then it will be the source and you will be the destination. Each datagram has a sequence number. This is used so that the other end can make sure that it gets the datagrams in the right order, and that it hasn't missed any. (See the TCP specification for details.) TCP doesn't number the datagrams, but the octets. So if there are 500 octets of data in each datagram, the first datagram might be numbered 0, the second 500, the next 1000, the next 1500, etc. Finally, I will mention the Checksum. This is a number that is computed by adding up all the octets in the datagram (more or less - see the TCP spec). The result is put in the header. TCP at the

other end computes the checksum again. If they disagree, then something bad happened to the datagram in transmission, and it is thrown away. So here's what the datagram looks like now.



If we abbreviate the TCP header as "T", the whole file now looks like this:

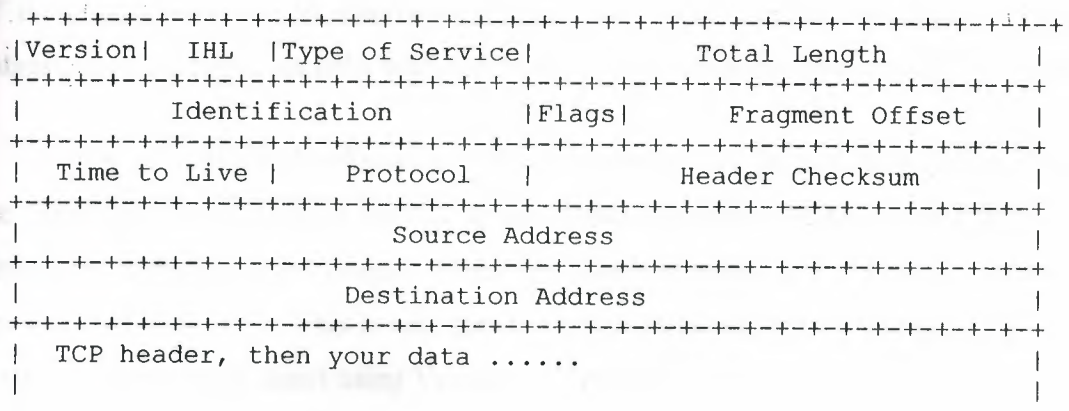
T.... T.... T.... T.... T.... T.... T....

You will note that there are items in the header that I have not described above. They are generally involved with managing the connection. In order to make sure the datagram has arrived at its destination, the recipient has to send back an "acknowledgement". This is a datagram whose "Acknowledgement number" field is filled in. For example, sending a packet with an acknowledgement of 1500 indicates that you have received all the data up to octet number 1500. If the sender doesn't get an acknowledgement within a reasonable amount of time, it sends the data again. The window is used to control how much data can be in transit at any one time. It is not practical to wait for each datagram to be acknowledged before sending the next one. That would slow things down too much. On the other hand, you can't just keep sending, or a fast computer might overrun the capacity of a slow one to absorb data. Thus each end indicates how much new data it is currently prepared to absorb by putting the number of octets in its "Window" field. As the computer receives data, the amount of space left in its window decreases. When it goes to zero, the sender has to stop. As the receiver processes the data, it increases its window, indicating that it is ready to accept more data. Often the same datagram can be used to acknowledge receipt of a set of data and to give

permission for additional new data (by an updated window). The "Urgent" field allows one end to tell the other to skip ahead in its processing to a particular octet. This is often useful for handling asynchronous events, for example when you type a control character or other command that interrupts output. The other fields are beyond the scope of this document.

5.2 The IP level

TCP sends each of these datagrams to IP. Of course it has to tell IP the Internet address of the computer at the other end. Note that this is all IP is concerned about. It doesn't care about what is in the datagram, or even in the TCP header. IP's job is simply to find a route for the datagram and get it to the other end. In order to allow gateways or other intermediate systems to forward the datagram, it adds its own header. The main things in this header are the source and destination Internet address (32-bit addresses, like 128.6.4.194), the protocol number, and another checksum. The source Internet address is simply the address of your machine. (This is necessary so the other end knows where the datagram came from.) The destination Internet address is the address of the other machine. (This is necessary so any gateways in the middle know where you want the datagram to go.) The protocol number tells IP at the other end to send the datagram to TCP. Although most IP traffic uses TCP, there are other protocols that can use IP, so you have to tell IP which protocol to send the datagram to. Finally, the checksum allows IP at the other end to verify that the header wasn't damaged in transit. Note that TCP and IP have separate checksums. IP needs to be able to verify that the header didn't get damaged in transit, or it could send a message to the wrong place. For reasons not worth discussing here, it is both more efficient and safer to have TCP compute a separate checksum for the TCP header and data. Once IP has tacked on its header, here's what the message looks like:



If we represent the IP header by an "I", your file now looks like this:

IT.... IT.... IT.... IT.... IT.... IT.... IT....

Again, the header contains some additional fields that have not been discussed. Most of them are beyond the scope of this document. The flags and fragment offset are used to keep track of the pieces when a datagram has to be split up. This can happen when datagrams are forwarded through a network for which they are too big. (This will be discussed a bit more below.) The time to live is a number that is decremented whenever the datagram passes through a system. When it goes to zero, the datagram is discarded. This is done in case a loop develops in the system somehow. Of course this should be impossible, but well-designed networks are built to cope with "impossible" conditions.

At this point, it's possible that no more headers are needed. If your computer happens to have a direct phone line connecting it to the destination computer, or to a gateway, it may simply send the datagrams out on the line (though likely a synchronous protocol such as HDLC would be used, and it would add at least a few octets at the beginning and end).

5.2.1 The Internet Protocol

Internet Protocol (IP) was originally designed to operate on top of Version 2 Ethernet. The Compendium has a separate section to discuss ETHERNET. Various components of the IP protocol family were differentiated by Ethertype number. IP is assigned Ethertype 0800 hex. The Internet Protocol (IP) is a network-layer (Layer 3) protocol that contains addressing information and some control information that enables packets to be routed. IP is documented in RFC 791 and is the primary network-layer protocol in the Internet protocol suite. Along with the Transmission Control Protocol (TCP), IP represents the heart of the Internet protocols. IP has two primary responsibilities: providing connectionless, best-effort delivery of datagrams through an internetwork; and providing fragmentation and reassembly of datagrams to support data links with different maximum-transmission unit (MTU) sizes.

When the IEEE developed the 802.3 standards for Ethernet they, essentially, replaced the Ethertype number with a Service Access Point identifier. It was necessary to include an option for embedding the original Ethertype inside a newer 802.3 frame in order to allow access to an IP Subnet. This is why there is a Sub-Network Access Protocol (SNAP) header in most IP frames that aren't using Version 2 Ethernet.

IP operates at OSI Layer 3 and provides the routing function in an IP network. Each communicating device is assigned an IP address. The address identifies the network (which may be divided into sub-networks) and the host. The term "host" refers to any communicating device in an IP network. Originally the term referred to a central host computer. Today it includes any PC, printer, gateway, file server, or other device that has an IP address and talks on an IP network.

The discussion of IP begins with a description of the addressing scheme, progresses through the routing function, and then expands on the addressing concepts used to create sub-networks. Troubleshooting IP is the process of troubleshooting routing on the network.

5.2.2 Background

The Internet protocols are the world's most popular open-system (nonproprietary) protocol suite because they can be used to communicate across any set of interconnected networks and are equally well suited for LAN and WAN communications. The Internet protocols consist of a suite of communication protocols, of which the two best known are the Transmission Control Protocol (TCP) and the Internet Protocol (IP). The Internet protocol suite not only includes lower-layer protocols (such as TCP and IP), but it also specifies common applications such as electronic mail, terminal emulation, and file transfer. This chapter provides a broad introduction to specifications that comprise the Internet protocols. Discussions include IP addressing and key upper-layer protocols used in the Internet.

Internet protocols were first developed in the mid-1970s, when the Defense Advanced Research Projects Agency (DARPA) became interested in establishing a packet-switched network that would facilitate communication between dissimilar computer systems at research institutions. With the goal of heterogeneous connectivity in mind, DARPA funded research by Stanford University and Bolt, Beranek, and Newman (BBN). The result of this development effort was the Internet protocol suite, completed in the late 1970s.

TCP/IP later was included with Berkeley Software Distribution (BSD) UNIX and has since become the foundation on which the Internet and the World Wide Web (WWW) are based. Documentation of the Internet protocols (including new or revised protocols) and policies are specified in technical reports called Request For Comments (RFCs), which are published and then reviewed and analyzed by the Internet community. Protocol refinements are published in

the new RFCs. To illustrate the scope of the Internet protocols (figure 3) maps many of the protocols of the Internet protocol suite and their corresponding OSI layers. This chapter addresses the basic elements and operations of these and other key Internet protocols. Internet protocols span the complete range of OSI model layers.

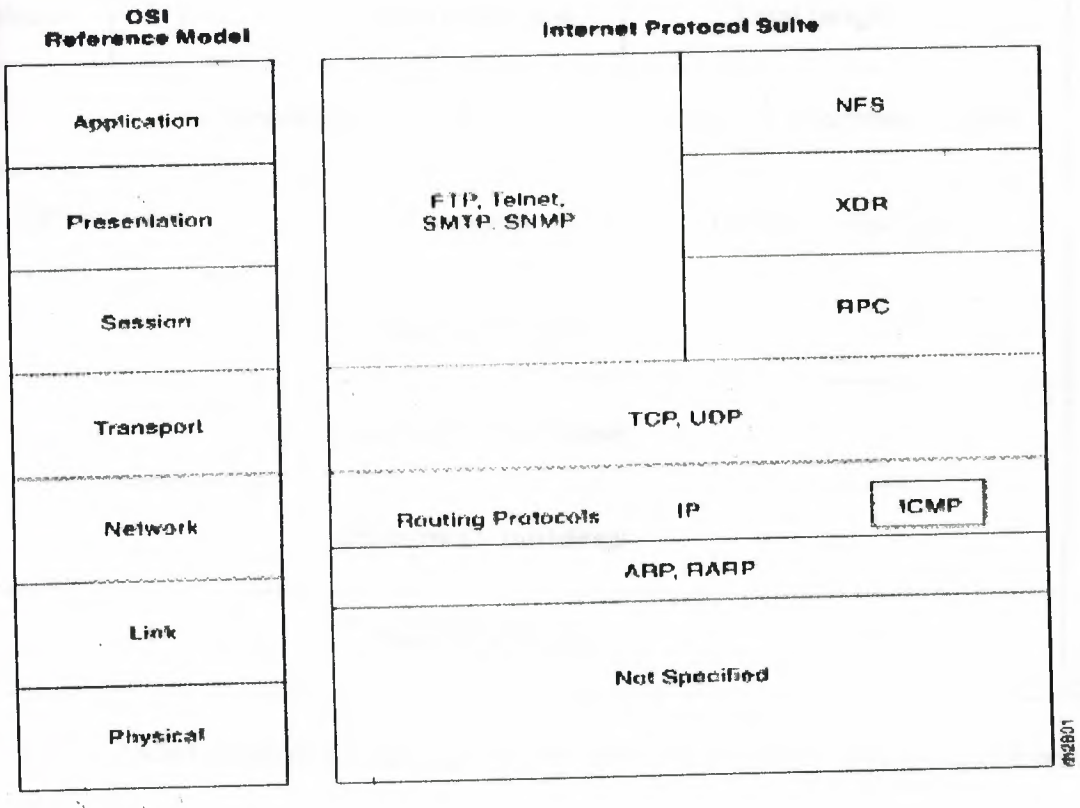


Figure 3. OSI model layers.

5.2.3 Thoughts about fixing problems...

The actual troubleshooting maxim is quite simple: Follow the frame from source to destination. Each station should be forwarding the frame to a correct destination; router to router; until the final destination is reached. If someone doesn't forward the frame correctly, and if the destination address is valid, then that station is misconfigured. To know what the expected forwarding will be from router to router it is necessary to understand the underlying subnet masking being used by the routers and by the nodes. The meaning of the dotted-

decimal IP address can only be ascertained by applying the mask using binary arithmetic to determine which bits are used to represent the network, the subnet (or subnets), and the host.

5.2.4 IP Packet Format

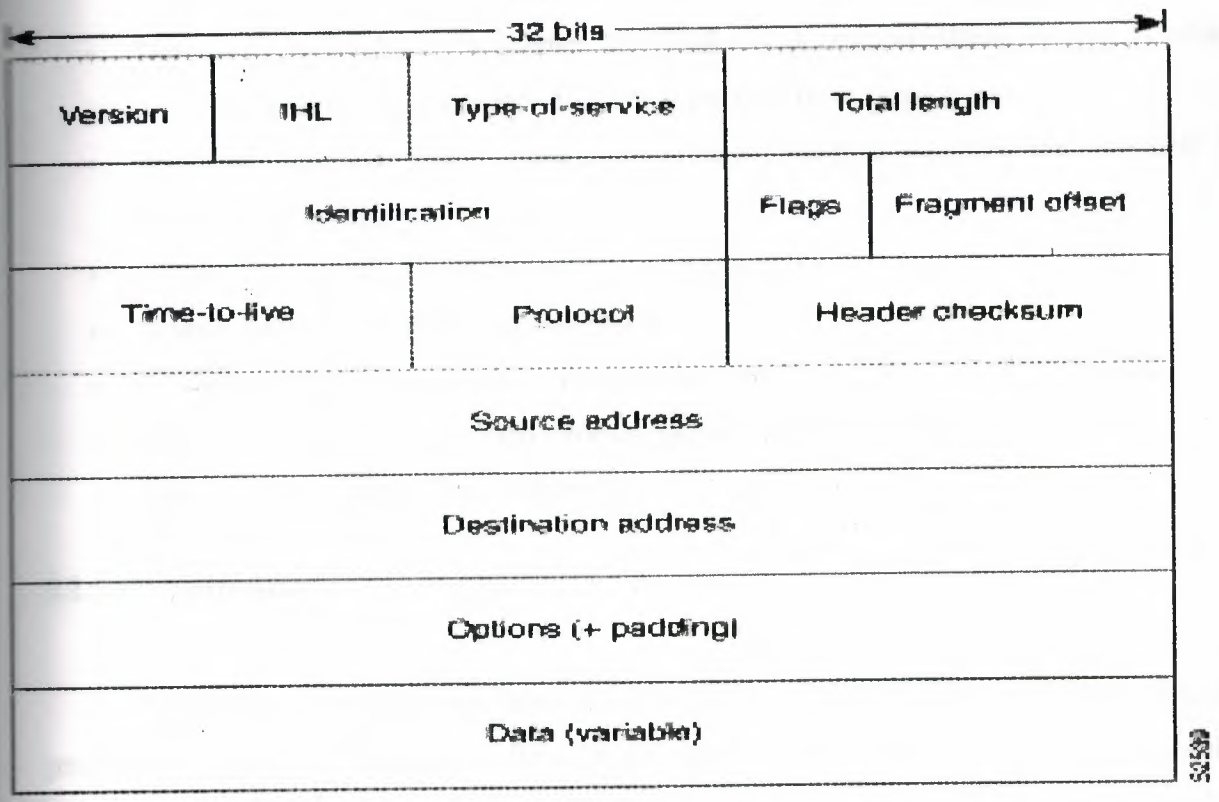


Figure 4. An IP packet contains several types of information. Fourteen fields comprise an IP packet.

- *Version*---Indicates the version of IP currently used.
- *IP Header Length* (IHL)---Indicates the datagram header length in 32-bit words.
- *Type-of-Service*---Specifies how an upper-layer protocol would like a current datagram to be handled, and assigns datagrams various levels of importance.
- *Total Length*---Specifies the length, in bytes, of the entire IP packet, including the data and header.
- *Identification*---Contains an integer that identifies the current datagram. This field is used to help piece together datagram fragments.
- *Flags*---Consists of a 3-bit field of which the two low-order (least-significant) bits control fragmentation. The low-order bit specifies whether the packet can be

fragmented. The middle bit specifies whether the packet is the last fragment in a series of fragmented packets. The third or high-order bit is not used.

- *Fragment Offset*---Indicates the position of the fragment's data relative to the beginning of the data in the original datagram, which allows the destination IP process to properly reconstruct the original datagram.
- *Time-to-Live*---Maintains a counter that gradually decrements down to zero, at which point the datagram is discarded. This keeps packets from looping endlessly.
- *Protocol*---Indicates which upper-layer protocol receives incoming packets after IP processing is complete.
- *Header Checksum*---Helps ensure IP header integrity.
- S*ource Address*---Specifies the sending node.
- *Destination Address*---Specifies the receiving node.
- *Options*---Allows IP to support various options, such as security.
- *Data*---Contains upper-layer information.

5.2.5 IP Addressing

As with any other network-layer protocol, the IP addressing scheme is integral to the process of routing IP datagrams through an internetwork. Each IP address has specific components and follows a basic format. These IP addresses can be subdivided and used to create addresses for subnetworks, as discussed in more detail later in this chapter.

Each host on a TCP/IP network is assigned a unique 32-bit logical address that is divided into two main parts: the network number and the host number. The network number identifies a network and must be assigned by the Internet Network Information Center (InterNIC) if the network is to be part of the Internet. An Internet Service Provider (ISP) can obtain blocks of network addresses from the InterNIC and can itself assign address space as necessary. The host number identifies a host on a network and is assigned by the local network administrator.

5.2.6 IP Address Format

The 32-bit IP address is grouped eight bits at a time, separated by dots, and represented in decimal format (known as *dotted decimal notation*). Each bit in the octet has a binary weight

(128, 64, 32, 16, 8, 4, 2, 1). The minimum value for an octet is 0, and the maximum value for an octet is 255. An IP address consists of 32 bits, grouped into four octets.

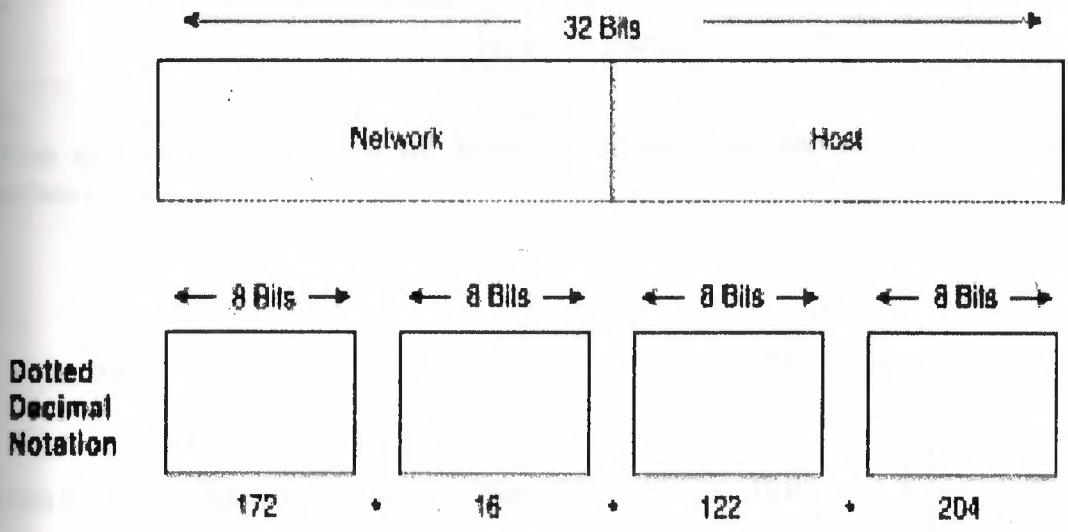


Figure 5. The basic format of an IP address.

5.2.7 IP Address Classes

IP addressing supports five different address classes: A, B,C, D, and E. Only classes A, B, and C are available for commercial use. The left-most (high-order) bits indicate the network class. Table 1 provides reference information about the five IP address classes.

Table 1: Reference Information About the Five IP Address Classes

IP Address Class	Format	Purpose	High-Order Bit(s)	Address Range	No. Bits Network/Host	Max. Hosts
A	N.H.H.H[1]	Few large organizations	0	1.0.0.0 to 126.0.0.0	7/24	16,777,214[2] (2^{24} - 2)
B	N.N.H.H	Medium-size organizations	1, 0	128.1.0.0 to 191.254.0.0	14/16	65, 543 (2^{16} - 2)
C	N.N.N.H	Relatively small organizations	1, 1, 0	192.0.1.0 to 223.255.254.0	22/8	245 (2^{8} - 2)

D	N/A	Multicast groups (RFC 1112)	1, 1, 1, 0	224.0.0.0 to 239.255.255.255	N/A (not for commercial use)	N/A
E	N/A	Experimental	1, 1, 1, 1	240.0.0.0 to 254.255.255.255	N/A	N/A

[1]N = Network number, H = Host number.
[2]One address is reserved for the broadcast address, and one address is reserved for the network.

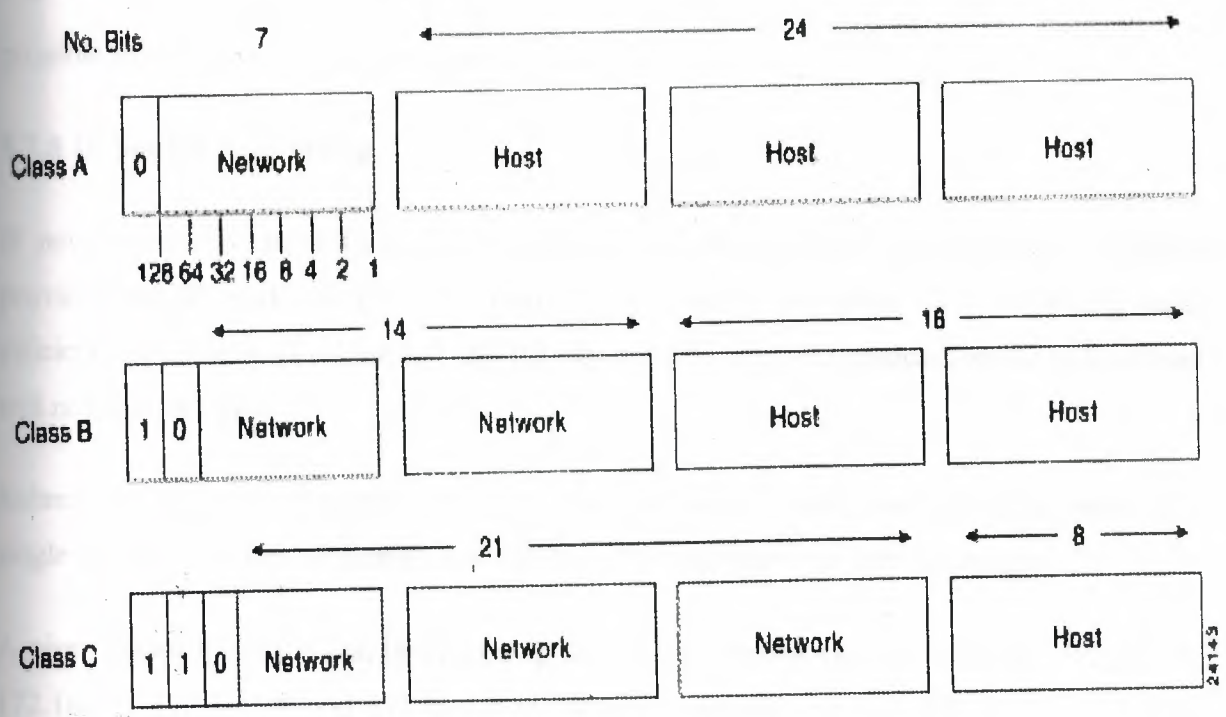


Figure 6. The format of the commercial IP address classes. (Note the high-order bits in each class.) IP address formats A, B, and C are available for commercial use.

The class of address can be determined easily by examining the first octet of the address and mapping that value to a class range in the following table. In an IP address of 172.31.1.2, for example, the first octet is 172. Because 172 falls between 128 and 191, 172.31.1.2 is a Class B address.

Address Class	First Octet in Decimal	High-Order Bits
Class A	1 D 126	0
Class B	128 D 191	10
Class C	192 D 223	110
Class D	224 D 239	1110
Class E	240 D 254	1111

Figure 7. The range of possible values for the first octet of each address class. A range of possible values exists for the first octet of each address class.

5.2.8 IP Subnet Addressing

IP networks can be divided into smaller networks called subnetworks (or subnets). Subnetting provides the network administrator with several benefits, including extra flexibility, more efficient use of network addresses, and the capability to contain broadcast traffic (a broadcast will not cross a router).

Subnets are under local administration. As such, the outside world sees an organization as a single network and has no detailed knowledge of the organization's internal structure.

A given network address can be broken up into many subnetworks. For example, 172.16.1.0, 172.16.2.0, 172.16.3.0, and 172.16.4.0 are all subnets within network 171.16.0.0. (All 0s in the host portion of an address specifies the entire network.)

5.2.9 IP Subnet Mask

A subnet address is created by "borrowing" bits from the host field and designating them as the subnet field. The number of borrowed bits varies and is specified by the subnet mask.

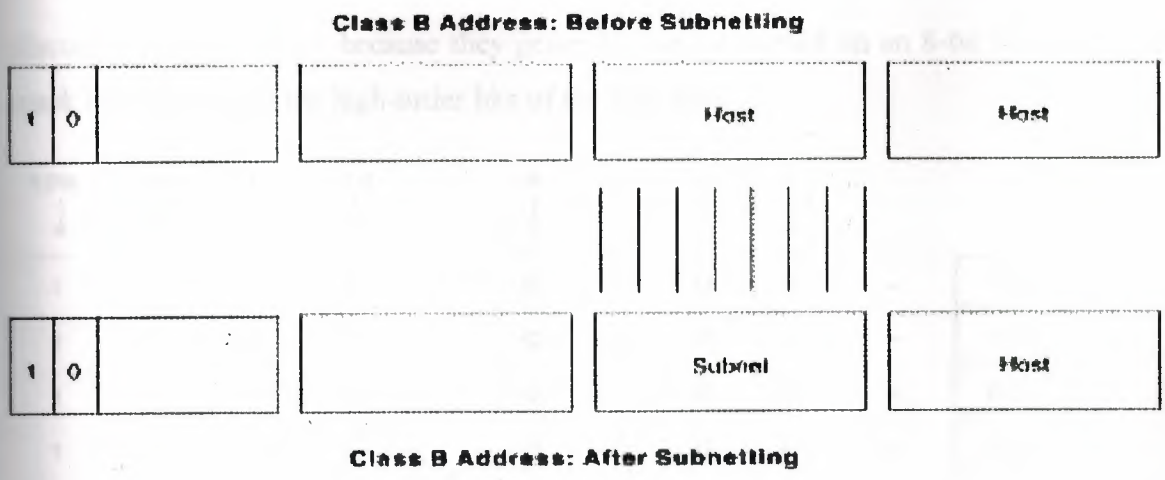


Figure 8. Shows how bits are borrowed from the host address field to create the subnet address field. Bits are borrowed from the host address field to create the subnet address field.

Subnet masks use the same format and representation technique as IP addresses. The subnet mask, however, has binary 1s in all bits specifying the network and subnetwork fields, and binary 0s in all bits specifying the host field.

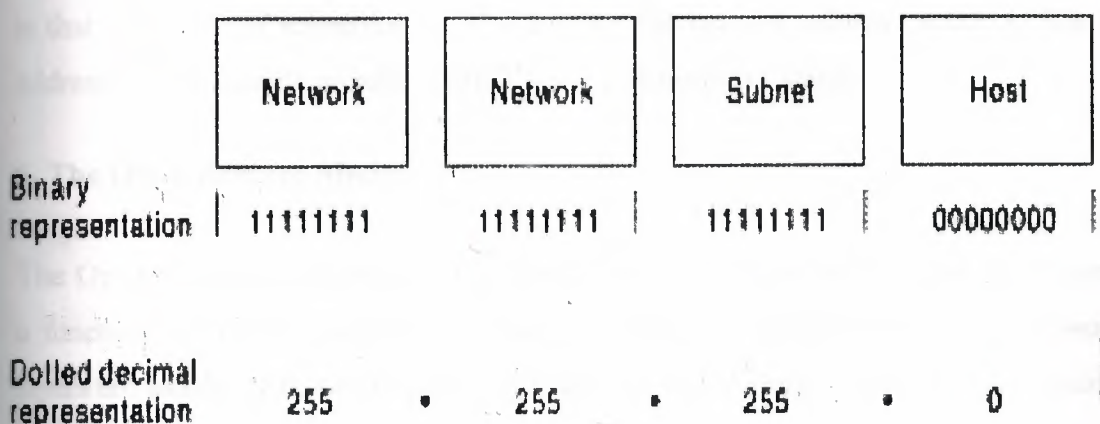


Figure 9. A sample subnet mask. A sample subnet mask consists of all binary 1s and 0s.

Subnet mask bits should come from the high-order (left-most) bits of the host field, as Figure 10 illustrates. Details of Class B and C subnet mask types follow. Class A addresses are not

discussed in this chapter because they generally are sub netted on an 8-bit boundary. Subnet mask bits come from the high-order bits of the host field.

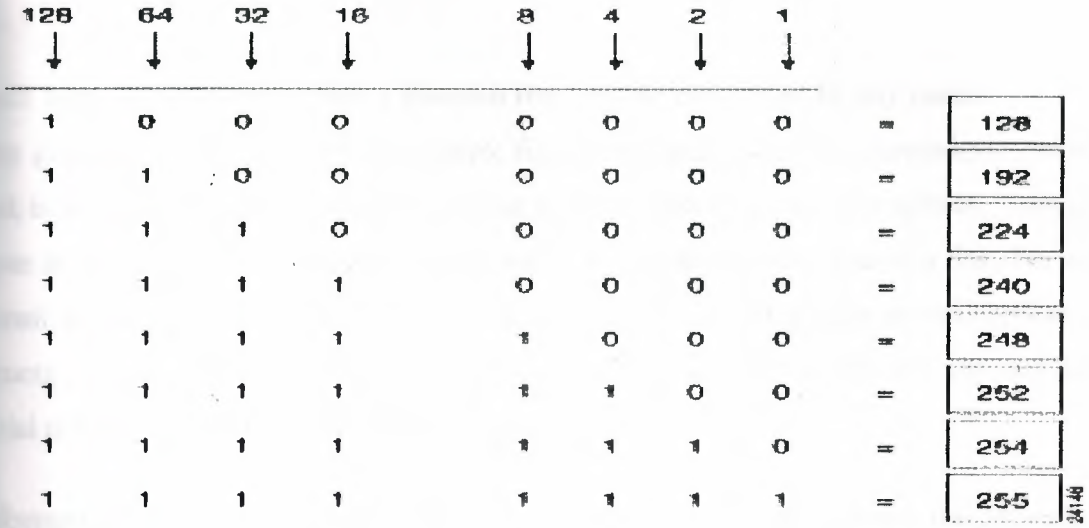


Figure 10

Various types of subnet masks exist for Class B and C subnets. The default subnet mask for a Class B address that has no subnetting is 255.255.0.0, while the subnet mask for a Class B address 171.16.0.0 that specifies eight bits of sub netting is 255.255.255.0. The reason for this is that eight bits of subnetting or 2^8 - 2 (1 for the network address and 1 for the broadcast address) = 254 subnets possible, with 2^8 - 2 = 254 hosts per subnet.

6. The OSI Reference Model

The Open Systems Interconnect (OSI) Reference Model has seven layers. Each layer defines a function performed when data is transferred between applications across a network. These layers are usually pictured as a stack of blocks, leading to the common term "protocol stack."

Application Layer	application programs that use the network
Presentation Layer	standardizes data presented to the applications
Session Layer	manages sessions between applications
Transport Layer	provides error detection and correction
Network Layer	manages network connections
Data Link Layer	provides data delivery across the physical connection

Physical Layer	defines the physical network media

Each layer of the stack defines a function that may be performed by any number of protocols. Any given protocol may perform multiple functions. Each protocol communicates with a **peer** that is an equivalent implementation of the same protocol on a remote system. Each protocol layer is only concerned with communication to a peer at the other end of a link. For example, e-mail is an application level protocol that communicates with a peer e-mail application on a remote system. The e-mail application does not care whether or not the physical layer is a serial modem line or a twisted pair ethernet connection.

Information is passed down through the layers until it is transmitted across the network, where it is passed back up the stack to the application at the remote end. Each layer relies on the other layers to perform their functions. The individual layers do not care how the other layers operate. They only need to know how to pass information up or down from one layer to another.

6.1 Open System Interconnection (OSI) Protocols

6.1.1 Background

The Open System Interconnection (OSI) protocol suite is comprised of numerous standard protocols that are based on the OSI reference model. These protocols are part of an international program to develop data-networking protocols and other standards that facilitate multivendor equipment interoperability. The OSI program grew out of a need for international networking standards and is designed to facilitate communication between hardware and software systems despite differences in underlying architectures.

The OSI specifications were conceived and implemented by two international standards organizations: the International Organization for Standardization (ISO) and the International Telecommunication Union-Telecommunication Standardization Sector (ITU-T). This chapter

provides a summary of the OSI protocol suite and illustrates its mapping to the general OSI reference model.

6.2 Open Systems Interconnection (OSI) Routing Protocol

6.2.1 Background

The International Organization for Standardization (ISO) developed a complete suite of routing protocols for use in the Open Systems Interconnection (OSI) protocol suite. These include *Intermediate System-to-Intermediate Systems (IS-IS)*, *End System-to-Intermediate System (ES-IS)*, and *Interdomain Routing Protocol (IDRP)*. This chapters addresses the basic operations of each of these protocols.

IS-IS is based on work originally done at Digital Equipment Corporation for DECnet/OSI (DECnet Phase V). IS-IS originally was developed to route in ISO *Connectionless Network Protocol* (CLNP) networks. A version has since been created that supports both CLNP and *Internet Protocol* (IP) networks; this version usually is referred to as *Integrated IS-IS* (it also has been called *Dual IS-IS)*.

OSI routing protocols are summarized in several ISO documents, including ISO 10589, which defines IS-IS. The American National Standards Institute (ANSI) X3S3.3 (network and transport layers) committee was the motivating force behind ISO standardization of IS-IS. Other ISO documents include ISO 9542 (which defines ES-IS) and ISO 10747 (which defines IDRP).

6.3 OSI Networking Terminology

The world of OSI networking uses some specific terminology, such as *end system* (ES), which refers to any nonrouting network nodes, and *intermediate system* (IS), which refers to a router. These terms form the basis for the ES-IS and IS-IS OSI protocols. The ES-IS protocol enables ESs and ISs to discover each other. The IS-IS protocol provides routing between ISs. Other important OSI networking terms include *area*, *domain*, *Level 1 routing*, and *Level 2 routing*. An area is a group of contiguous networks and attached hosts that is specified to be an area by a network administrator or manager. A domain is a collection of connected areas. Routing domains provide full connectivity to all end systems within them. Level 1 routing is routing within a Level 1 area, while Level 2 routing is routing between Level 1 areas. Figure 11

illustrates the relationship between areas and domains and depicts the levels of routing between the two. Areas exist within a larger domain and use Level 2 routing to communicate.

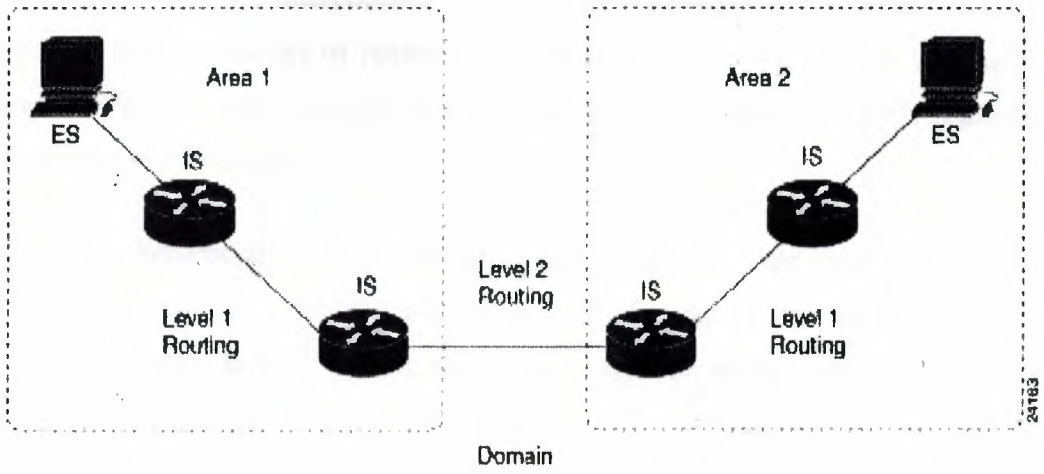


Figure 11

6.4 OSI Routing Operation

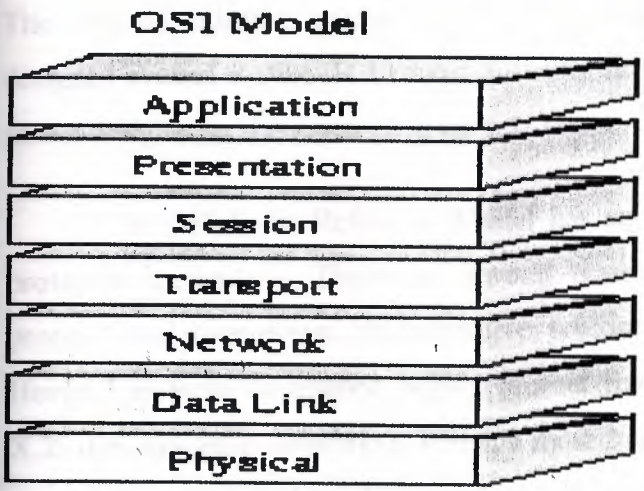
Each ES lives in a particular area. OSI routing begins when the ESs discovers the nearest IS by listening to ISH packets. When an ES wants to send a packet to another ES, it sends the packet to one of the ISs on its directly attached network. The router then looks up the destination address and forwards the packet along the best route. If the destination ES is on the same subnetwork, the local IS will know this from listening to ESHs and will forward the packet appropriately. The IS also might provide a *redirect* (RD) message back to the source to tell it that a more direct route is available. If the destination address is an ES on another subnetwork in the same area, the IS will know the correct route and will forward the packet appropriately. If the destination address is an ES in another area, the Level 1 IS sends the packet to the nearest Level 2 IS. Forwarding through Level 2 ISs continues until the packet reaches a Level 2 IS in the destination area. Within the destination area, ISs forward the packet along the best path until the destination ES is reached.

Link-state update messages help ISs learn about the network topology. First, each IS generates an update specifying the ESs and ISs to which it is connected, as well as the associated metrics. The update then is sent to all neighboring ISs, which forward (flood) it to their neighbors, and so on. (Sequence numbers terminate the flood and distinguish old updates from new ones.) Using these updates topology of the network. When the topology changes, new updates are sent.

6.5 OSI Seven-Layer Model

In the 1980s, the European-dominated International Standards Organization (ISO), began to develop its Open Systems Interconnection (OSI) networking suite. OSI has two major components: an abstract model of networking (the Basic Reference Model, or *seven-layer model*), and a set of concrete protocols. The standard documents that describe OSI are for sale and not currently available online.

Parts of OSI have influenced Internet protocol development, but none more than the abstract model itself, documented in OSI 7498 and its various addenda. In this model, a networking system is divided into layers. Within each layer, one or more entities implement its functionality. Each entity interacts directly only with the layer immediately beneath it, and provides facilities for use by the layer above it. Protocols enable an entity in one host to interact with a corresponding entity at the same layer in a remote host.



The seven layers of the OSI Basic Reference Model are (from bottom to top):

1. The **Physical Layer** describes the physical properties of the various communications media, as well as the electrical properties and interpretation of the exchanged signals. Ex: this layer defines the size of Ethernet coaxial cable, the type of BNC connector used, and the termination method.
2. The **Data Link Layer** describes the logical organization of data bits transmitted on a particular medium. Ex: this layer defines the framing, addressing and checksumming of Ethernet packets.

3. The **Network Layer** describes how a series of exchanges over various data links can deliver data between any two nodes in a network. Ex: this layer defines the addressing and routing structure of the Internet.
4. The **Transport Layer** describes the quality and nature of the data delivery. Ex: this layer defines if and how retransmissions will be used to ensure data delivery.
5. The **Session Layer** describes the organization of data sequences larger than the packets handled by lower layers. Ex: this layer describes how request and reply packets are paired in a remote procedure call.
6. The **Presentation Layer** describes the syntax of data being transferred. Ex: this layer describes how floating point numbers can be exchanged between hosts with different math formats.
7. The **Application Layer** describes how real work actually gets done. Ex: this layer would implement file system operations.

The original Internet protocol specifications defined a four-level model, and protocols designed around it (like TCP) have difficulty fitting neatly into the seven-layer model. Most newer designs use the seven-layer model.

The OSI Basic Reference Model has enjoyed a far greater acceptance than the OSI protocols themselves. There are several reasons for this. OSI's committee-based design process bred overgrown, unimaginative protocols that nobody ever accused of efficiency. Heavy European dominance helped protect their investments in X.25 (CONS is basically X.25 for datagram networks). Perhaps most importantly, X.25 data networks never caught people's imagination like the Internet, which, with a strong history of free, downloadable protocol specifications, has been loath to embrace yet another networking scheme where you have to pay to figure how things work.

And why should we? OSI's biggest problem is that doesn't really offer anything new. The strongest case for its implementation comes from its status as an "international standard", but we already have a de facto international standard - the Internet. OSI protocols will be around, but its most significant contribution is the philosophy of networking represented by its layered model.

If the Internet community has to worry about anything, it's the danger of IETF turning into another ISO - a big, overgrown standards organization run by committees, churning out

thousands of pages of rubbish, and dominated by big business players more interested in preserving investments than advancing the state of the art.

6.6 OSI Networking Protocols

Figure 12 illustrates the entire OSI protocol suite and its relation to the layers of the OSI reference model. Each component of this protocol suite is discussed briefly. The OSI routing protocols are addressed in more detail in "Open Shortest Path First (OSPF)." The OSI protocol suite maps to all layers of the OSI reference model.

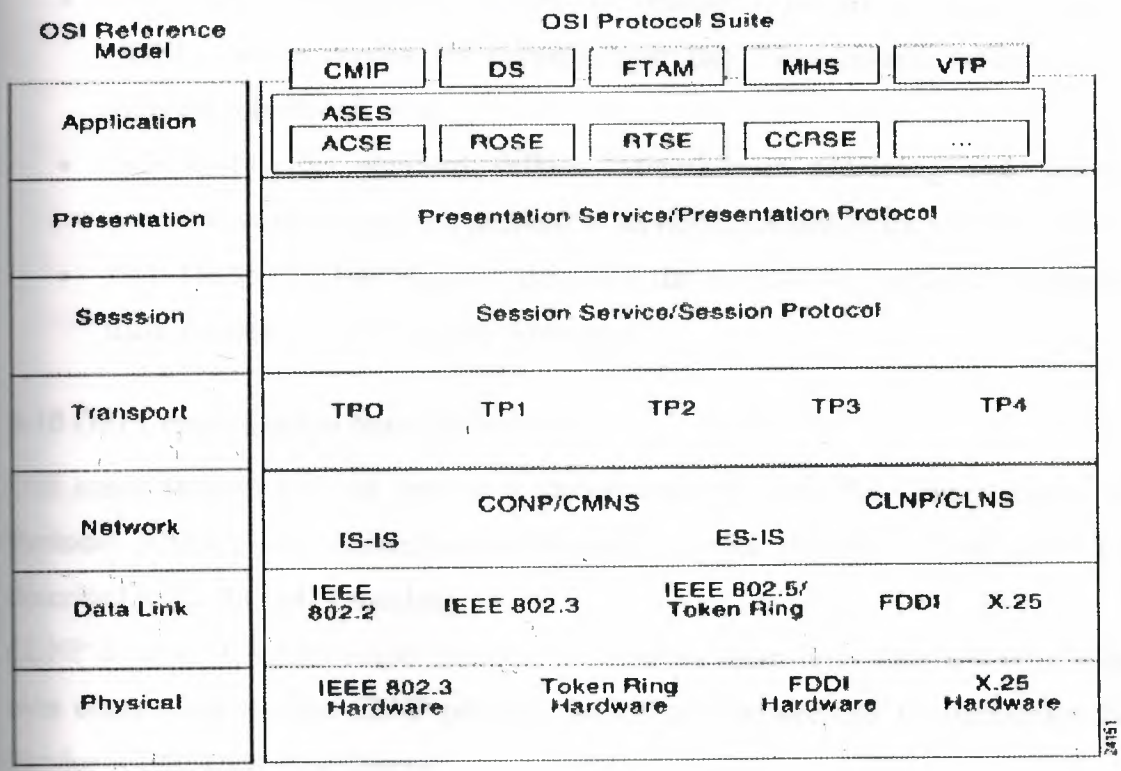


Figure 12

6.7 OSI Physical and Data Link layers

The OSI protocol suite supports numerous standard media-access protocols at the physical and data link layers. The wide variety of media-access protocols supported in the OSI protocol suite allows other protocol suites to exist easily alongside OSI on the same network

media. Supported media-access protocols include IEEE 802.2 LLC, IEEE 802.3, Token Ring/IEEE 802.5, Fiber Distributed Data Interface (FDDI), and X.25.

6.8 OSI Network Layer

The OSI protocol suite specifies two routing protocols at the network layer: End System-to-Intermediate System (ES-IS) and Intermediate System-to-Intermediate System (IS-IS). In addition, the OSI suite implements two types of network services: connectionless service and connection-oriented service.

6.9 OSI-Layer Standards

In addition to the standards specifying the OSI network-layer protocols and services, the following documents describe other OSI network-layer specifications:

- ISO 8648---This standard defines the *internal organization of the network layer* (IONL), which divides the network layer into three distinct sublayers to support different subnetwork types.
- ISO 8348---This standard defines network-layer addressing and describes the connection-oriented and connectionless services provided by the OSI network layer.
- ISO TR 9575---This standard describes the framework, concepts, and terminology used in relation to OSI routing protocols.

6.10 OSI Connectionless Network Service

OSI connectionless network service is implemented by using the Connectionless Network Protocol (CLNP) and Connectionless Network Service (CLNS). CLNP and CLNS are described in the ISO 8473 standard.

CLNP is an OSI network-layer protocol that carries upper-layer data and error indications over connectionless links. CLNP provides the interface between the Connectionless Network Service (CLNS) and upper layers.

CLNS provides network-layer services to the transport layer via CLNP.

CLNS does not perform connection setup or termination because paths are determined independently for each packet that is transmitted through a network. This contrasts with Connection-Mode Network Service (CMNS).

In addition, CLNS provides best-effort delivery, which means that no guarantee exists that data will not be lost, corrupted, misordered, or duplicated. CLNS relies on transport-layer protocols to perform error detection and correction.

6.11 OSI Connection-Oriented Network Service

OSI connection-oriented network service is implemented by using the Connection-Oriented Network Protocol (CONP) and Connection-Mode Network Service (CMNS).

CONP is an OSI network-layer protocol that carries upper-layer data and error indications over connection-oriented links. CONP is based on the X.25 Packet-Layer Protocol (PLP) and is described in the ISO 8208 standard, "X.25 Packet-Layer Protocol for DTE."

CONP provides the interface between CMNS and upper layers. It is a network-layer service that acts as the interface between the transport layer and CONP and is described in the ISO 8878 standard.

CMNS performs functions related to the explicit establishment of paths between communicating transport-layer entities. These functions include connection setup, maintenance, and termination, and CMNS also provides a mechanism for requesting a specific quality of service (QOS). This contrasts with CLNS.

6.11.1 Network-Layer Addressing

OSI network-layer addressing is implemented by using two types of hierarchical addresses: network service access-point addresses and network-entity titles.

A network service-access point (NSAP) is a conceptual point on the boundary between the network and the transport layers. The NSAP is the location at which OSI network services are provided to the transport layer. Each transport-layer entity is assigned a single NSAP, which is individually addressed in an OSI internetwork using NSAP addresses.

Figure 13 illustrates the format of the OSI NSAP address, which identifies individual NSAPs. The OSI NSAP address is assigned to each transport-layer entity.

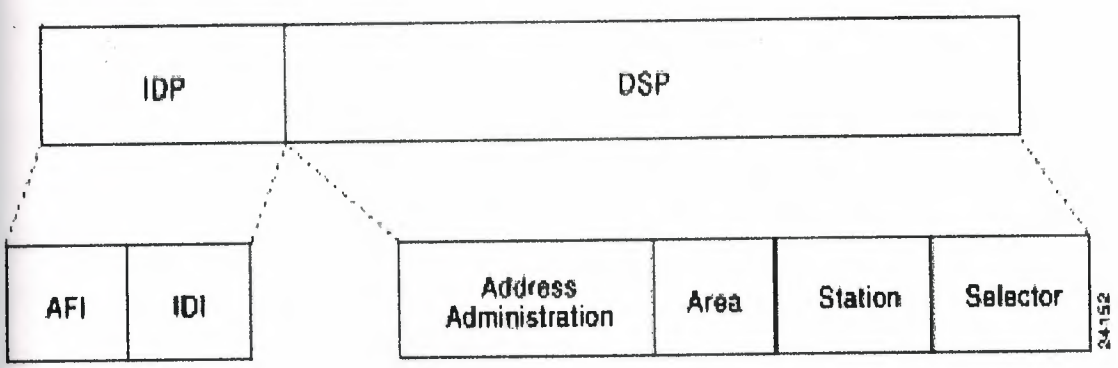


Figure 12

6.11.2 NSAP Address Fields

Two NSAP Address fields exist: the Initial Domain Part (IDP) and the Domain-Specific Part (DSP).

The IDP field is divided into two parts: the Authority Format Identifier (AFI) and the Initial Domain Identifier (IDI). The AFI provides information about the structure and content of the IDI and DSP fields, such as whether the IDI is of variable length and whether the DSP uses decimal or binary notation. The IDI specifies the entity that can assign values to the DSP portion of the NSAP address.

The DSP is subdivided into four parts by the authority responsible for its administration. The Address Administration fields allow for the further administration of addressing by adding a second authority identifier and by delegating address administration to subauthorities. The Area field identifies the specific area within a domain and is used for routing purposes. The Station field identifies a specific station within an area and also is used for routing purposes. The Selector field provides the specific n-selector within a station and, much like the other fields, is used for routing purposes. The reserved n-selector 00 identifies the address as a network entity title (NET).

6.11.3 End-System NSAPs

An OSI end system (ES) often has multiple NSAP addresses, one for each transport entity that it contains. If this is the case, the NSAP address for each transport entity usually differs only in the last byte (called the *n-selector*). Figure 13 illustrates the relationship between a transport entity, the NSAP, and the network service. The NSAP provides a linkage between a transport entity and a network service.

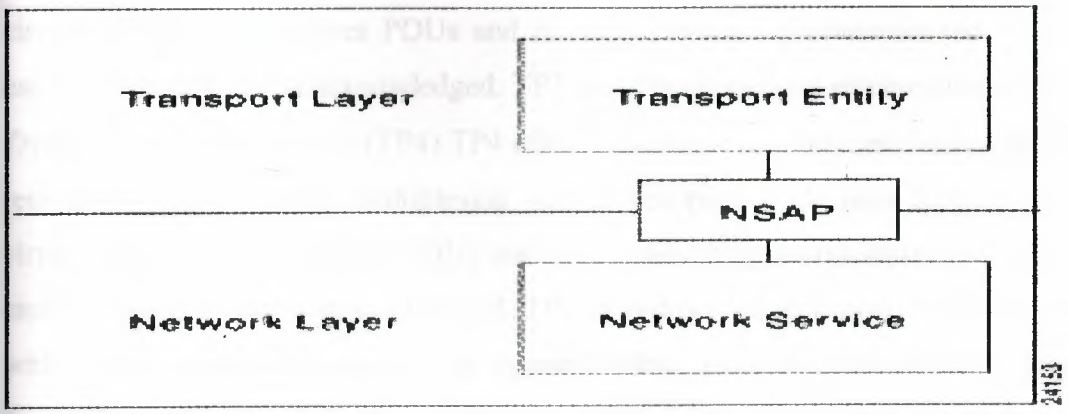


Figure 13

A network-entity title (NET) is used to identify the network layer of a system without associating that system with a specific transport-layer entity (as an NSAP address does). NETs are useful for addressing intermediate systems (ISs), such as routers, that do not interface with the transport layer. An IS can have a single NET or multiple NETs, if it participates in multiple areas or domains.

6.12 OSI Protocols Transport Layer

The OSI protocol suite implements two types of services at the transport layer: connection-oriented transport service and connectionless transport service.

Five connection-oriented transport-layer protocols exist in the OSI suite, ranging from Transport Protocol Class 0 through Transport Protocol Class 4. Connectionless transport service is supported only by Transport Protocol Class 4.

Transport Protocol Class 0 (TP0), the simplest OSI transport protocol, performs segmentation and reassembly functions. TP0 requires connection-oriented network service.

Transport Protocol Class 1 (TP1) performs segmentation and reassembly and offers basic error recovery. TP1 sequences protocol data units (PDUs) and will retransmit PDUs or reinitiate the connection if an excessive number of PDUs are unacknowledged. TP1 requires connection-oriented network service.

Transport Protocol Class 2 (TP2) performs segmentation and reassembly, as well as multiplexing and demultiplexing data streams over a single virtual circuit. TP2 requires connection-oriented network service.

Transport Protocol Class 3 (TP3) offers basic error recovery and performs segmentation and reassembly, in addition to multiplexing and demultiplexing data streams over a single virtual

circuit. TP3 also sequences PDUs and retransmits them or reinitiates the connection if an excessive number are unacknowledged. TP3 requires connection-oriented network service.

Transport Protocol Class 4 (TP4) TP4 offers basic error recovery, performs segmentation and reassembly, and supplies multiplexing and demultiplexing of data streams over a single virtual circuit. TP4 sequences PDUs and retransmits them or reinitiates the connection if an excessive number are unacknowledged. TP4 provides reliable transport service and functions with either connection-oriented or connectionless network service. It is based on the Transmission Control Protocol (TCP) in the Internet Protocols suite and is the only OSI protocol class that supports connectionless network service.

6.13 OSI Protocols Session Layer

The session-layer implementation of the OSI protocol suite consists of a session protocol and a session service. The session protocol allows session-service users (SS-users) to communicate with the session service. An *SS-user* is an entity that requests the services of the session layer. Such requests are made at Session-Service Access Points (SSAPs), and SS-users are uniquely identified by using an SSAP address. Figure 14 shows the relationship between the SS-user, the SSAP, the session protocol, and the session service.

Session service provides four basic services to SS-users. First, it establishes and terminates connections between SS-users and synchronizes the data exchange between them. Second, it performs various negotiations for the use of session-layer tokens, which must be possessed by the SS-user to begin communicating. Third, it inserts synchronization points in transmitted data that allow the session to be recovered in the event of errors or interruptions. Finally, it allows SS-users to interrupt a session and resume it later at a specific point. Session layer functions provide service to presentation layer functions via a SSAP.

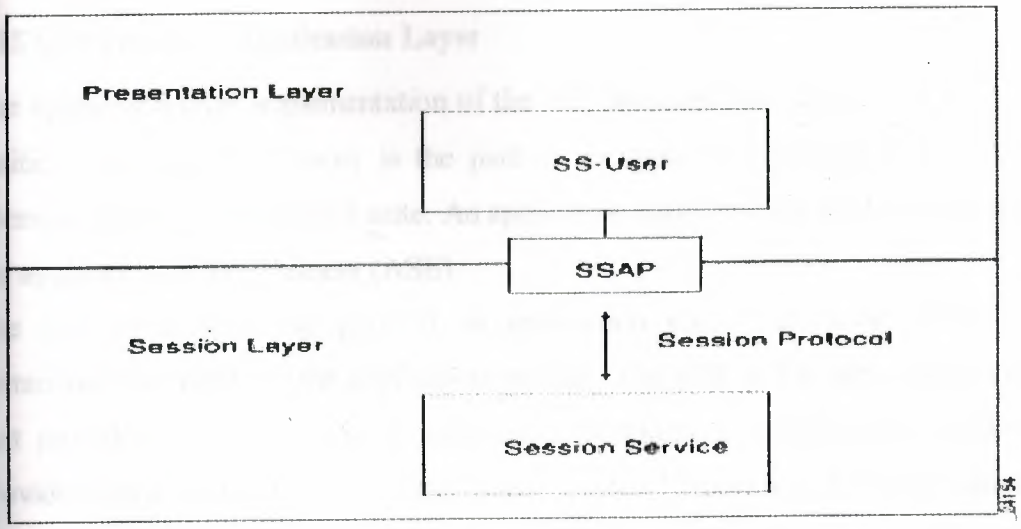


Figure 14

Session service is defined in the ISO 8326 standard and in the ITU-T X.215 recommendation. The session protocol is defined in the ISO 8327 standard and in the ITU-T X.225 recommendation. A connectionless version of the session protocol is specified in the ISO 9548 standard.

6.14 OSI Protocols Presentation Layer

The presentation-layer implementation of the OSI protocol suite consists of a presentation protocol and a presentation service. The presentation protocol allows presentation-service users (PS-users) to communicate with the presentation service.

A PS-user is an entity that requests the services of the presentation layer. Such requests are made at Presentation-Service Access Points (PSAPs). PS-users are uniquely identified by using PSAP addresses.

Presentation service negotiates transfer syntax and translates data to and from the transfer syntax for PS-users, which represent data using different syntaxes. The presentation service is used by two PS-users to agree upon the transfer syntax that will be used. When a transfer syntax is agreed upon, presentation-service entities must translate the data from the PS-user to the correct transfer syntax.

The OSI presentation-layer service is defined in the ISO 8822 standard and in the ITU-T X.216 recommendation. The OSI presentation protocol is defined in the ISO 8823 standard and in the ITU-T X.226 recommendation. A connectionless version of the presentation protocol is specified in the ISO 9576 standard.

6.15 OSI Protocols Application Layer

The application-layer implementation of the OSI protocol suite consists of various application entities. An application entity is the part of an application process that is relevant to the operation of the OSI protocol suite. An application entity is composed of the user element and the application service element (ASE).

The user element is the part of an application entity that uses ASEs to satisfy the communication needs of the application process. The ASE is the part of an application entity that provides services to user elements and, therefore, to application processes. ASEs also provide interfaces to the lower OSI layers. Figure 15 portrays the composition of a single application process (composed of the application entity, the user element, and the ASEs) and its relation to the PSAP and presentation service. An application process relies on the PSAP and presentation service.

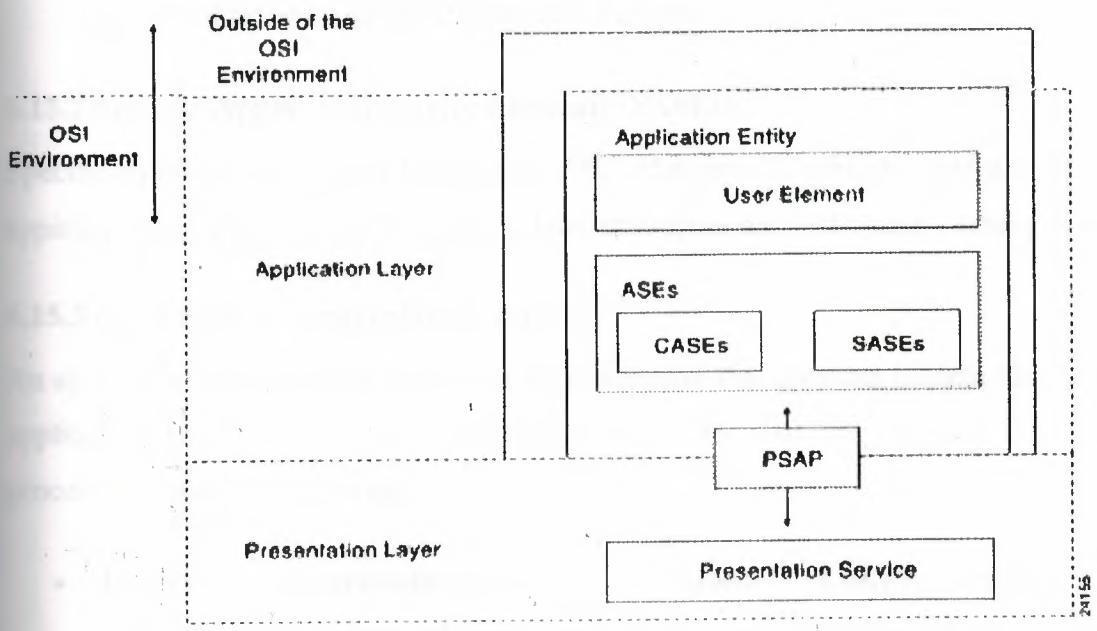


Figure 15

ASEs fall into one of the two following classifications: Common-Application Service Elements (CASEs) and Specific-Application Service Elements (SASEs). Both of these might be present in a single application entity.

6.15.1 Common-Application Service Elements (CASEs)

Common-Application Service Elements (CASEs) are ASEs that provide services used by a wide variety of application processes. In many cases, multiple CASEs are used by a single application entity. The following four CASEs are defined in the OSI specification:

- *Association Control Service Element (ACSE)*---Creates associations between two application entities in preparation for application-to-application communication
- *Remote Operations Service Element (ROSE)*---Implements a request-reply mechanism that permits various remote operations across an application association established by the ACSE
- *Reliable Transfer Service Element (RTSE)*---Allows ASEs to reliably transfer messages while preserving the transparency of complex lower-layer facilities
- *Commitment, Concurrence, and Recovery Service Elements (CCRSE)*---Coordinates dialogues between multiple application entities.

6.15.2 Specific-Application Service Elements (SASEs)

Specific-Application Service Elements are ASEs that provide services used only by a specific application process, such as file transfer, database access, and order-entry, among others.

6.15.3 OSI Protocols Application Processes

An application process is the element of an application that provides the interface between the application itself and the OSI application layer. Some of the standard OSI application processes include the following:

- *Common Management-Information Protocol (CMIP)*---Performs network management functions, allowing the exchange of management information between ESs and management stations. CMIP is specified in the ITU-T X.700 recommendation and is functionally similar to the Simple Network-Management Protocol (SNMP) and NetView.
- *Directory Services (DS)*---Serves as a distributed directory that is used for node identification and addressing in OSI internetworks. DS is specified in the ITU-T X.500 recommendation.
- *File Transfer, Access, and Management (FTAM)*---Provides file-transfer service and distributed file-access facilities.

- *Message Handling System* (MHS)---Provides a transport mechanism for electronic messaging applications and other applications by using store-and-forward services.
- *Virtual Terminal Protocol* (VTP)---Provides terminal emulation that allows a computer system to appear to a remote ES as if it were a directly attached terminal.

(1) Physical Layer

- Concerned with the transmission of *bits*.
- How many volts for 0, how many for 1?
- Number of bits of second to be transmitted.
- Two way or one-way transmission
- Standardized protocol dealing with electrical, mechanical and signaling interfaces.
- Many standards have been developed, e.g. RS-232 (for serial communication lines).
- Example : X.21

(2) Data Link Layer

- Handles errors in the physical layer.
- Groups bits into *frames* and ensures their correct delivery.
- Adds some bits at the beginning and end of each frame plus the checksum.
- Receiver verifies the checksum.
- If the checksum is not correct, it asks for retransmission. (send a control message).
- Consists of two sublayers:
    - Logical Link Control (LLC) defines how data is transferred over the cable and provides data link service to the higher layers.
    - Medium Access Control (MAC) defines who can use the network when multiple computers are trying to access it simultaneously (i.e. Token passing, Ethernet [CSMA/CD]).

(3) Network Layer

- Concerned with the transmission of *packets*.
- Choose the best path to send a packet (*routing*).

- It may be complex in a large network (e.g. Internet).
- Shortest (distance) route vs. route with least delay.
- Static (long term average) vs. dynamic (current load) routing.
- Two protocols are most widely used.
- **X.25**
    - Connection Oriented
    - Public networks, telephone, European PTT
    - Send a call request at the outset to the destination
    - If destination accepts the connection, it sends an connection identifier
- **IP (Internet Protocol)**
    - Connectionless
    - Part of Internet protocol suite.
    - An IP **packet** can be sent without a connection being established.
    - Each packet is routed to its destination independently.

(4) Transport Layer

- Network layer does not deal with lost messages.
- Transport layer ensures reliable service.
- Breaks the message (from sessions layer) into smaller packets, assigns sequence number and sends them.
- Reliable transport connections are built on top of X.25 or IP.
- In case IP, lost packets arriving out of order must be reordered.
- TCP : (Transport Control Protocol) Internet transport protocol.
- TCP/IP Widely used for network/transport layer (UNIX).
- UDP (Universal Datagram Protocol) : Internet connectionless transport layer protocol.
- Application programs that do not need connection-oriented protocol generally use UDP.

(5) Sessions Layer

- Just theory! Very few applications use it.
- Enhanced version of transport layer.
- Dialog control, synchronization facilities.
- Rarely supported (Internet suite does not).

(6) Presentation Layer

- Just theory! Very few applications use it.
- Concerned with the semantics of the bits.
- Define records and fields in them.
- Sender can tell the receiver of the format.
- Makes machines with different internal representations to communicate.
- If implemented, the best layer for cryptography.

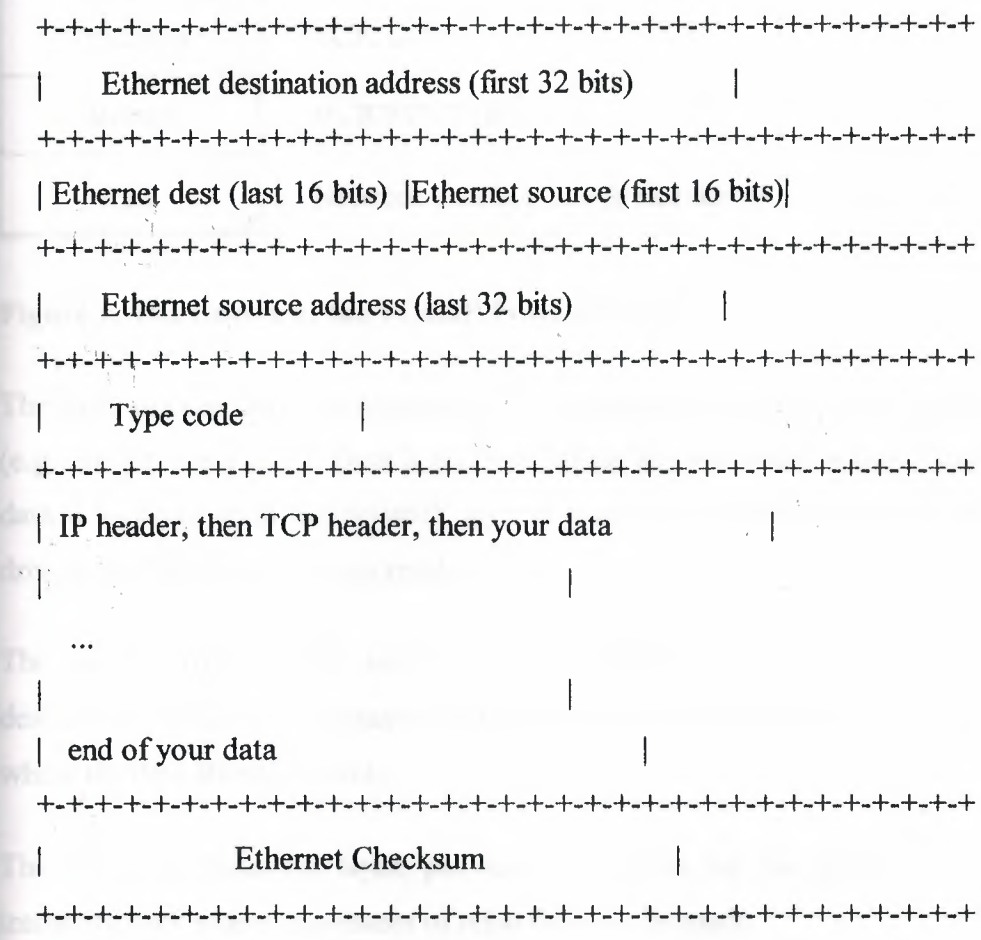
(7) Application Layer

- Collection of miscellaneous protocols for high level applications
- Electronic mail, file transfer, connecting remote terminals, etc.
- E.g. SMTP, FTP, Telnet, HTTP, etc.

7. The Ethernet level

However most of our networks these days use Ethernet. So now we have to describe Ethernet's headers. Unfortunately, Ethernet has its own addresses. The people who designed Ethernet wanted to make sure that no two machines would end up with the same Ethernet address. Furthermore, they didn't want the user to have to worry about assigning addresses. So each Ethernet controller comes with an address builtin from the factory. In order to make sure that they would never have to reuse addresses, the Ethernet designers allocated 48 bits for the Ethernet address. People who make Ethernet equipment have to register with a central authority, to make sure that the numbers they assign don't overlap any other manufacturer. Ethernet is a "broadcast medium". That is, it is in effect like an old party line telephone. When

you send a packet out on the Ethernet, every machine on the network sees the packet. So something is needed to make sure that the right machine gets it. As you might guess, this involves the Ethernet header. Every Ethernet packet has a 14-octet header that includes the source and destination Ethernet address, and a type code. Each machine is supposed to pay attention only to packets with its own Ethernet address in the destination field. (It's perfectly possible to cheat, which is one reason that Ethernet communications are not terribly secure.) Note that there is no connection between the Ethernet address and the Internet address. Each machine has to have a table of what Ethernet address corresponds to what Internet address. (We will describe how this table is constructed a bit later.) In addition to the addresses, the header contains a type code. The type code is to allow for several different protocol families to be used on the same network. So you can use TCP/IP, DECnet, Xerox NS, etc. at the same time. Each of them will put a different value in the type field. Finally, there is a checksum. The Ethernet controller computes a checksum of the entire packet. When the other end receives the packet, it recomputes the checksum, and throws the packet away if the answer disagrees with the original. The checksum is put on the end of the packet, not in the header. The final result is that your message looks like this:



If we represent the Ethernet header with "E", and the Ethernet checksum with "C", your file now looks like this:

EIT....C EIT....C EIT....C EIT....C EIT....C

When these packets are received by the other end, of course all the headers are removed. The Ethernet interface removes the Ethernet header and the checksum. It looks at the type code. Since the type code is the one assigned to IP, the Ethernet device driver passes the datagram up to IP. IP removes the IP header. It looks at the IP protocol field. Since the protocol type is TCP, it passes the datagram up to TCP. TCP now looks at the sequence number. It uses the sequence numbers and other information to combine all the datagrams into the original file.

The ends our initial summary of TCP/IP. There are still some crucial concepts we haven't gotten to, so we'll now go back and add details in several areas. (For detailed descriptions of the items discussed here see, RFC 793 for TCP, RFC 791 for IP, and RFC's 894 and 826 for sending IP over Ethernet.)

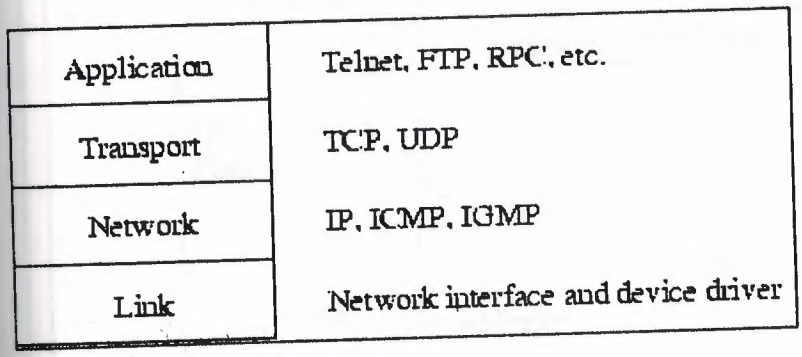


Figure 1. The Layers of the TCP/IP Protocol Suite

The first, the link layer, is responsible for communicating with the actual network hardware (e.g., the Ethernet card). Data it receives off the network wire it hands to the network layer; data it receives from the network layer it puts on the network wire. This is where device drivers for different interfaces reside.

The second, the network layer, is responsible for figuring out how to get data to its destination. Making no guarantee about whether data will reach its destination, it just decides where the data should be sent.

The third, the transport layer, provides data flows for the application layer. It is at the transport layer where guarantees of reliability may be made.

The fourth, the application layer, is where users typically interact with the network. This is where telnet, ftp, email, IRC, etc. reside.

Packets are the basic unit of transmission on the Internet. They contain both data and header information. Simply put, headers generally consist of some combination of checksums, protocol identifiers, destination and source addresses, and state information. Each layer may add its own header information, so it can interpret the data the lower layer is handing it. In Figure 2, we see a sample Ethernet frame. This is the product of a packet which has gone from that application layer all the way to the link layer. Each layer takes the previous layer's packet, viewing almost all of it as data, and puts its own header on it.

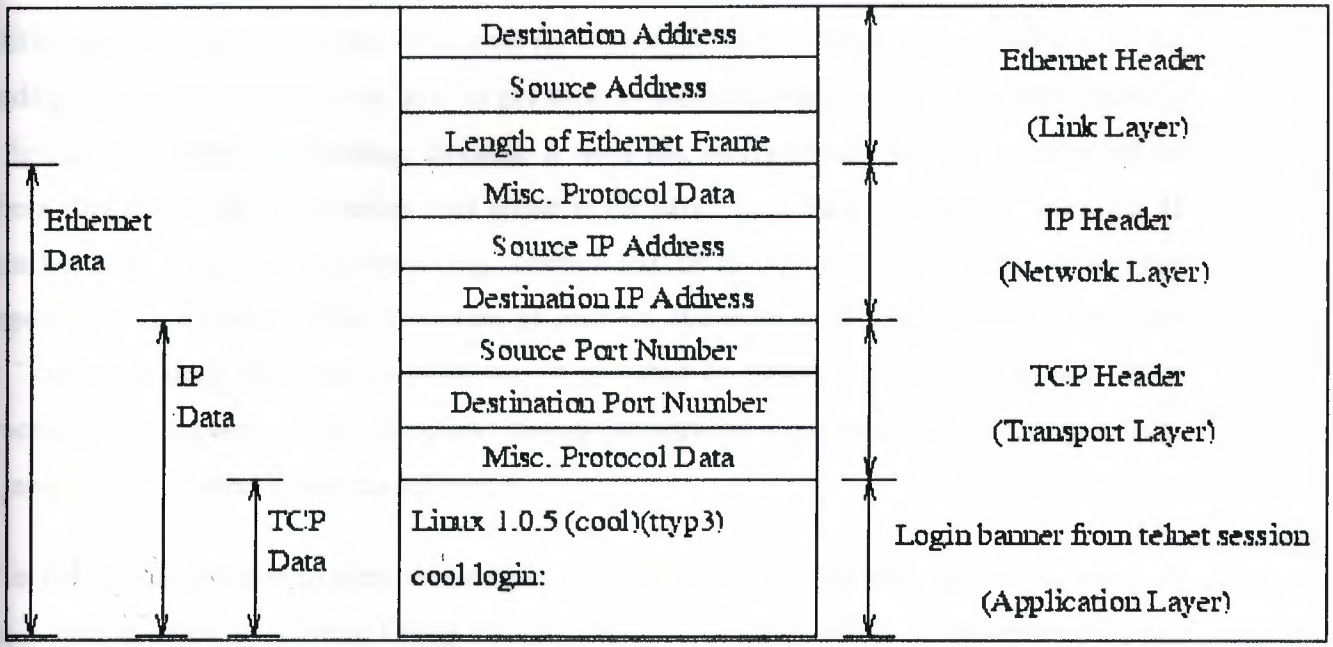


Figure 2. A Sample Ethernet Frame

We will now examine each part in turn, with a particular emphasis on the network and transport layers. In examples that follow, we'll refer to two machines: swell.cs.umass.edu and cool.alaska.edu. swell is the machine we are on, cool is the destination computer. We assume cool and swell are on Ethernets at their respective organizations. Most of our examples assume an Ethernet, but could work with any kind of network (e.g., token-ring).

7.1 The Link Layer

The link layer is the simplest layer to understand. Composed of the network hardware and the device drivers, the link layer is the lowest level of the protocol stack. When receiving data from the network, it takes packets from the network wire, strips away any link layer header

information, and hands it off to the network layer. When transmitting data onto the network, it takes packets from the network layer, sticks a link layer header on them, and send them out over the wire.

The benefit of separating out the hardware layer is that protocol implementors only have to write the network layer once. Then they provide a common interface to the network layer by writing different device drivers for each kind of network interface.

7.2 The Network Layer

This is where the Internet Protocol (IP) and the Internet Control Message Protocol (ICMP), among others, reside. ICMP is used both to provide network reliability information and by utilities like ping and traceroute. IP is used for almost all other Internet communication. When sending packets, it is figures out how to get them to their destination; when receiving packets, it figures out where they belong. Because it does not worry about whether packets get to where they are going nor whether they arrive in the order sent, its job is greatly simplified. If a packet arrives with any problems (e.g., corruption), IP silently discards it. Upper layers are responsible for insuring reliable reception of packets. We refer to IP's behavior as ``stateless" or ``connectionless" because the existence of previous or future packets is irrelevant when processing the current packet. We could unplug the network wire, wait a minute, plug it back in, and IP would never know the difference.

IP is able to get packets to their destinations because every network interface on the Internet has a unique, numeric address. Oddly enough, these numbers are called IP addresses. Notice, every *interface* has its own address. If a machine has multiple interfaces (as is the case with a router), each one has its own IP address. The Internic is responsible for assigning sets of addresses to organizations, thereby insuring uniqueness.

Because it's a pain to refer to machines with strings of numbers, the designers of TCP/IP allowed network administrators to associate names with IP addresses. Although this has nothing to do with the IP layer per se, we feel this is useful material. Originally, every host on the Internet maintained its own complete copy of this database (on Unix systems, it's in /etc/hosts). However, as the Internet reached its current size, this soon became unwieldly -- both in terms of raw size and the administrative nightmare of updating it. And so was born the domain name system (DNS). It is a distributed database of IP addresses and their natural language names, called host names. In fact one IP address can have multiple names associated

with it. When a network administrator adds a new machine to her network, she is responsible for updating her organization's nameserver table. Her changes quickly propagate. All communication with a machine is done via IP numeric addresses, so the hostname for a machine is only used at the beginning of a connection.

The steps IP takes to send a packet are simple: based on its IP address, figure out how to get it there and send it on its way.

Deciding out how to get the packet there, aka routing, is the critical task for IP. Fortunately, swell doesn't have to know how to get a packet all the way to Alaska, it just needs to figure out which local router is responsible for getting packets to Alaska. A router differs from a typical machine on the net because it has at least two network interfaces -- this allows it to connect to two or more networks. For a small organization, there will typically be a local network (e.g., Ethernet) and then a leased-line link to the Internet. The organization's router is connected to both the local network and the Internet link. All packets bound for the Internet are sent to the router, which then puts it on the leased line, bound for the next router.

Each router only needs to know about the routers to which it is connected. Those routers then know about all the routers to which they are connected. This allows swell's local router to say, ``Well, all packets bound for the West go to MIT, so I'll just send it there and let MIT figure out what to do next." MIT puts it on a T3 line to Cleveland, from there it goes to Chicago, San Francisco, Seattle, and into Alaska, where it goes from the organization's router to the Ethernet interface on swell. The router at each hop is only concerned with where to send it next. It doesn't try to determine the full path which the packet will take.

To determine where a given packet will go next, machines on the Internet maintain routing tables. They consist of three major items: addresses of routers, addresses they can handle, and the interface to which they are connected. In the case of a machine on a local net (like cool.cs.umass.edu), it probably has three entries: one for the loopback interface (which allows a host to connect to itself), one for the local network and a default entry.

The local network entry lets IP know that the machine is directly connected to a certain set of IP addresses. Rather than try and route those packets, IP figures out the hardware address of the Ethernet interface to which the IP address corresponds and sends the packet there. With

this entry, cool is essentially the router, the addresses it can handle are all the IP addresses on the local net, and the destination interface is an Ethernet card on the local net.

The default entry says ``for all other addresses, send it to this router." Instead of trying to deliver a packet for cool.alaska.edu to a machine on the local net, cool sends it to the router's interface, saying, ``Here, I don't know where this goes, you figure it out." The router then looks at its table, sees it doesn't have a direct connection to cool, so sends it to its default destination, MIT. And so the process continues.

At this point, the reader should have a rough idea of how packets are transmitted on the Internet. When receiving data, IP takes the packet from the link layer, checks for any blatant corruption, and hands the packet to the proper process at the transport layer. If there is any problem with the packet, IP silently discards it because it doesn't have to worry about whether a packet reaches its destination.

We have left a huge amount out of this picture. Here are just some of the issues we're ignoring: packet fragmentation, netmasks and other routing tricks, network error handling, and the interactions between the network and transport layer.

7.3 The Transport Layer

There are two protocols at the transport layer: the transmission control protocol (TCP) and the user datagram protocol (UDP). TCP provides end-to-end reliable communication and UDP doesn't. UDP is as unreliable as IP, but allows people to write user level software that creates its own packet formats, which is particularly helpful if you want to write new protocols, don't have the kernel sources, and don't want the overhead of TCP.

TCP creates a ``virtual circuit" between two processes. It insures that packets are received in the order they are sent and that lost packets are retransmitted. We won't go into the details of how it works, but interactive programs like ftp and telnet use it.

So far we have discussed addressing on the host level -- how to identify a particular machine. But once at a machine, we need a way to identify a particular service (e.g., mail). This is the function of ports -- identification numbers included with every UDP or TCP packet. TCP/IP ports are not hardware-based. They are a just a way of labeling packets. A process on a machine ``listens" on a particular port. When the transport layer receives a packet, it checks

the port number and sends the data to the corresponding process. When a process starts up, it registers a port number with the TCP/IP stack. Only one process per protocol can listen on a given port. So while a process using UDP and one using TCP can both listen on port 111, two processes that both used TCP could not. There are a number of ports which are reserved for standard services. For example, SMTP, the mail protocol, is always on port 25, and telnetd is always on port 23. To see a list of the reserved ports on a Unix system, look at /etc/services.

We've examined how ports work on the server end -- specific ports are reserved for set tasks. On the initiator end, port assignment is dynamic. When a telnet client on swell starts up, it gets a new port number (e.g., 1066). This is the source port which swell's TCP layer puts on every packet. This allows the telnet daemon (telnetd) on cool to responds to the correct telnet process on swell. The combination of source/destination IP addresses and ports provides a unique conversation identifier. Each conversation is called a flow.

UDP is essentially IP with port numbers (flows). It gives the user access to IP-style datagrams. The network file system (NFS) and talk are two examples of UDP-based protocols.

This has been an extremely cursory exploration of TCP and UDP. At this point, you should have a decent understanding of how the network (IP) and transport (TCP/UDP) layers interact. We now turn to the final layer.

7.4 The Application Layer

This is where the user interacts with the network. All network programs like telnet, ftp, mail, news, and WWW clients are at the application layer. They then use either TCP or UDP to communicate with other machines. To provide a clearer picture, I'll examine telnet in a little detail.

Telnet is used for remote login. It removes the need for hardwired terminals. A user on swell types ``telnet cool.alaska.edu" and he is rapidly connected to cool.alaska.edu, which asks him to login. He can then interact with cool. Here's a breakdown of the process:

1. The name address is turned into a numeric one (137.229.18.103), via a domain name server.

2. Telnet tells the transport layer it wants to start a TCP connection with 137.229.18.103, at port 23
3. TCP initiates a conversation with cool. IP is used to route packets. The telnet process on swell gets a port number of, say, 1096. TCP places the source and destination port numbers in its packet header, IP the IP address.
4. Packets are now handed to the IP layer, which sends them to the link layer. They proceed from the user's machine to his organization's router and out onto the Internet. They make their way to cool, one router at a time.
5. cool's TCP layer replies in a similar fashion.
6. The telnet daemon on cool and the telnet client on swell exchange terminal information and other parameters necessary for an interactive session. Control messages are sent in-band, as an escape byte of 255, followed by the control byte. Control messages include: echo, status, terminal type, terminal speed, flow control, linemode, and environmental variables.
7. The user sees a login prompt. After logging in, data is sent back and forth.

Once the task is broken into a number of steps, we see it's relatively simply. Because the transport layer provides a standard interface, network applications do not need to be rewritten or even recompiled if the transport layer code changes.

8. GOVERNMENT OPEN SYSTEMS INTERCONNECTION PROFILE (GOSIP) POLICY

Departments should migrate to OSI standards for their networks in a timely and cost effective manner.

8.1 BACKGROUND

The international OSI standards, developed under the auspices of the International Standards Organisation (ISO), were designed to provide the basis for achieving secure universal connectivity between heterogeneous computer systems. The business advantages of seeking to have a common system for communications between computing platforms are well recognised and documented. There are both efficiency and cost effectiveness benefits flowing

from a move to a more open computing environment. This is also important in promoting the international competitiveness of Australia's IT industry.

The Victorian Government's implementation of **GOSIP** is aimed at achieving objectives of facilitating information and/or data exchange between disparate computer systems. This is designed to promote access to and communication of information in a timely and cost effective manner. Other agencies, including Statutory Authorities and Government owned business enterprises were encouraged to follow the State Government's commitment to OSI in their development of IT systems and services. The implementation of international standards in IT will promote the achievement of greater efficiency and effectiveness in the use of IT to meet the needs of government programs. By introducing **OSI** standards the Government is seeking to ensure that there is a common IT applications architecture linking all its computer installations to facilitate access to, and transmission of, information within and between Departments.

The benefits of this approach are :

- by using a common communications architecture based on **GOSIP** standards, there will be greater capability to access data and to achieve significant savings in staff support and training costs
- departments avoid being locked-in to any particular supplier's communications architecture and move towards greater flexibility in choosing products and creating competition for products and services
- industry can develop software and hardware according to unambiguous specifications that are in the public domain with global benefits for local suppliers
  greater flexibility is achieved with the ability to readily move technology and skills between installations
- assists the Victorian Government in positioning itself for a future where universal connectivity and controlled and secure access to information, subject to privacy and security issues, will be the norm.

The original **GOSIP** policy, promulgated in 1991:

- required all suppliers to offer products which comply with **GOSIP** when responding to Victorian Government tenders from 1 October 1991
- required Victorian Government departments to implement **OSI** standards in all aspects of the network, unless there are compelling reasons to do otherwise

- did not seek retrospective conversion of existing networks to **OSI** standards
- took into account the availability and cost effectiveness of **OSI** compliant products in comparison to their proprietary equivalents
- sought to continuously test the market for the availability and cost effectiveness of products
- required all departments to develop migration plans as part of their IT plans for conversion of their existing networks to **OSI**
- placed responsibility with the department for implementing the policy and being accountable for it
- sought the conversion of all networks to **OSI** in a manner and time frame that is cost effective, taking into account the availability of products and the investment in existing systems.

These principles and requirements remain valid for this policy revision.

8.2 IMPLEMENTATION GUIDELINES

Future acquisitions and upgrades of Victorian Government IT&T architecture should support OSI standards for interconnectivity unless there are compelling reasons to do otherwise.

The following factors must be considered when implementing this policy :

- timing for, and manner of, conversion of a department's network(s) to **OSI** is entirely the responsibility of the department, which is accountable for the outcome
- there will be no supplementation of a department's budget for the conversion process. It is anticipated that each department will decide what is the appropriate time to convert taking into account the functionality, cost effectiveness and availability of **OSI** products versus their existing proprietary architectures. Market forces will influence this timing. It is expected that departments will use whole of life economic assessment methods taking into account the longer term benefits, as well as the short term costs, of adopting standards when making their decision
- it is recognised that cost effectiveness is an important element in determining when to convert a department network and that this will change as the market for **OSI** products develops and the technology changes. All Victorian Government tenders must seek the

provision of GOSIP compliant solutions, unless specific exemptions are justifiable. Departmental heads are responsible for ensuring that major acquisitions comply with government policies, including **GOSIP**, and that, where a proprietary solution is recommended in preference to a **GOSIP** compliant system, the reasons are sound and defensible

the preferred priority order for the application of standards is as follows

1. International standardised profiles and international standards including stable draft international standards.
2. National standards including stable draft national standards.
3. Industry/de facto standards where the specifications are publicly available and are not "owned" by a single supplier (eg TCP/IP, OSF and UI standards).
4. Industry/de facto standards where the specifications are publicly available and are widely accepted and adopted by many suppliers (eg MS-DOS, Micro soft Windows, UNIX, SNA).
5. remaining standards (proprietary).

The slow implementation of **OSI** standards in commercial products is acknowledged and, the widespread use of the competitive **Transmission Control Protocol/Internet Protocol (TCP/IP)** is recognised. Victorian Government policy is that OSI protocols are the preferred standard. The adoption of de facto/industry standards is recognised as providing a viable alternative in the absence of suitable **OSI** conformant products but with a clear commitment being given to migrate at a future date.

the use of **OSI** protocols for interdepartmental communication is preferred. It is recognised that there is a legacy of interconnection between departments based on proprietary protocols and more time must be allowed for their conversion. In addition, for the transfer of some selected applications, it may be more cost effective to continue to use a proprietary protocol in the short term. The longer term objective is to realise the overall benefits of standardisation on **GOSIP** protocols for all transmissions.

The strategy to move from the current proprietary based platforms to those based on standards encompassed in **GOSIP** should be incorporated in each department's migration plan to **OSI**. This plan, which should be part of the department's IT Strategic Plan is targeted at facilitating the above objectives and realising the very significant benefits of a move to a more open computing environment.

The timing of cost effective migration will vary from agency to agency and is dependent on the technological (especially **OSI** product availability) and business opportunities facing the organisation. Timing of the conversion will also be dependent upon

where the department is in the life cycle of its existing IT systems. There is a need for each department to review the continued appropriateness of their **OSI** migration plan against implementation experience.

The Commonwealth Government is examining a series of issues associated with **OSI** migration through separate working parties, eg **SNA to OSI, TCP/IP to OSI,** the introduction of standardised Cabling, and Naming and Addressing for inter-agency communications. These will be provided to all departments as they are progressively released.

8.3 OUTSOURCING

Since the **GOSIP** policy was announced, the Victorian Government has re-emphasised its commitment to outsourcing. Government IT policies will apply to all outsourced agreements and projects and thus outsourcing will not justify an exemption from the use of **GOSIP** standards.

8.4 CONFORMANCE AND INTEROPERABILITY

The interlinking of **OSI** conformant products developed by different suppliers requires users to have additional assurance on their level of conformance to standards and ability to interoperate. In this situation there are now risks for the user in having to take on more of the responsibility for the initial and continuing interoperability of such products and their integration into the agency computing environment, a responsibility largely taken on by the supplier in the proprietary environment. These new risks need to be recognised and managed and should be weighed against the benefits of implementation through a standards based architecture. The work of industry organisations and other consortia such as **X/OPEN**, the **Open Software Foundation (OSF), UNIX International** and the **Process to Support Interoperability (PSI)**, shows promise in reducing this risk by collectively agreeing and committing to the use of common standards, including **OSI** standards, and in some cases, code.

The issue of a conformance certificate for a product provides users and industry with a degree of assurance that such a product conforms to standards. This does not guarantee interoperability of products which requires additional guarantees and/or certification. Considerable progress is being made worldwide in the standardisation of conformance testing and the mutual recognition of certificates issued by accredited testing authorities. There is as

yet no universal system of recognition in place and there are clear risks of incompatibility when products are purchased from different suppliers.

In addition there are no third party testing centres in Australia at this time. Therefore consideration needs to be given to approaches which would ensure that products developed by Australian industry are not disadvantaged when competing with overseas suppliers for Victorian Government business. At present, there is little demand for local testing. Interoperability remains the key factor for the successful implementation of **GOSIP.** To address this issue, in the Victorian Government arena the following approach will apply in verifying the claims of products for **GOSIP** compliance :

- if a supplier offers a certificate from an accreditation agency showing successful completion by the product of independent interoperability tests, that will be accepted without further documentation, including the need for conformance evidence (recognising that conformance testing will be a prerequisite to formal interoperability testing and certification)
- a product which has a conformance certificate from a recognised accreditation agency covering testing performed at an accredited testing centre (including suppliers **[1st party]** centres) will be accepted as conforming to the standard without further documentation
- a product which has a manufacturer's declaration of conformance certification covering testing performed at an accredited testing centre (including suppliers **[1st party]** centre) will be accepted as conforming to the standard without further documentation
- a product which has a conformance test report from a non-accredited testing centre will be judged on the evidence on a case by case basis within the overall aim of making it as easy, and as practicable, to accept products (this makes it equivalent with current practices for procurement of proprietary products)
- if none of the conformance and interoperability certificates above apply and/or no 1st or third party conformance certificate or a manufacturers declaration of conformance is available, the supplier will be required to complete a **Protocol Implementation Conformance Statement (PICs)** demonstrating how they comply with the standard and contractually commit to supply product in conformance with the **PICS.**

Interoperability is recommended as a component of the acceptance testing procedures. Interoperability tests should cover existing systems in addition to new systems but care needs to be exercised in interpreting such test suites.

Given that the field of testing is changing rapidly, a degree of flexibility will be maintained and the rules amended as necessary, in consultation with departments and industry and in accord with accepted practice both in Australia and overseas. As this area matures, the ability to have accredited certification of **OSI** products may become a significant advantage in simplifying procurement as compared with the traditional individual procurement testing normally associated with proprietary solutions. Industry initiatives on interoperability testing through **OSIcom** and **OSIone** are noted and should be of assistance.

9. TCP/IP Real Time System

Some applications use real-time data, such as those in audio, radar, and sonar processing. For real-time applications, blocks of data must be processed within a set amount of time. This data must also be received from some source. To input real-time data in a GEDAE flow graph, a function box must be created which encapsulates the driver for the appropriate I/O device.

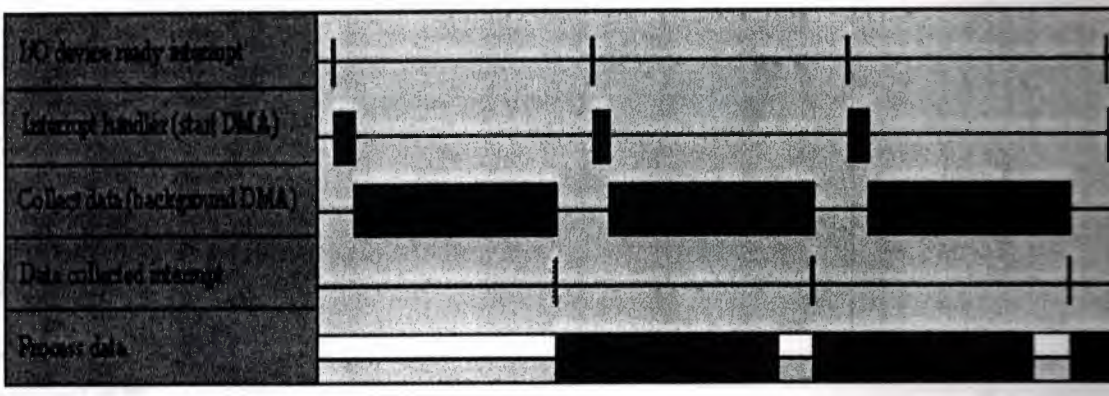
9.1 Problem:

For real-time applications, GEDAE has to collect and process a block of data within a certain time constraint avoiding errors or dropping data.

9.2 Solution:

9.2.1 Overview:

The following solution is based on a specific application created on a Sharc processor system and uses interrupts and background DMA's. When the I/O device has an available block of data, it sends an interrupt to the Sharc system. The interrupt handler starts the background DMA, which transfers the data from the I/O device to memory. When the DMA transfer is complete, another interrupt is sent to indicate that the data can now be processed. While processing, another block can be received from the I/O device. This is illustrated in the following time line diagram:



Maximizing processor usage results in minimizing system cost. By overlapping DMA data transfers and processing, processor usage is kept high.

This example works only on specific embedded systems. Slight system specific changes were made. Parts of the code examples were replaced with pseudo code.

9.3 Encapsulating the I/O device in a GEDAE function box

An embeddable GEDAE function box that reads real-time input is defined like any other function box. Parameters and other data inputs can be defined to process the real-time data. Users can build custom function boxes which provide a binding between interrupts and function calls via the primitive's Reset and Apply methods. The interrupt handlers are registered in the Reset method, and the data is received in the Apply method. The following is an example:

File location: FGlibraries/boxes/real_time/v_dev_input

```
Name: v_dev_input
Type: static
Comment: "Receive real-time input from an device.
      N_out == the vector size of the outputs.
      N_input == the vector size of the I/O data block"
Input: {
   int N_out;
   int N_input;
}
Local: {
   int needs_to_fire; /* number of vectors of real-time data to collect */
   int fired; /* current vector being collected */
}
Output: {
   stream float out1[N_out];
   stream float out2[N_out];
}
Include: {
#include < e_dev_input.h > /* found in ~/gedae/include/embeddable */
}
Reset: {
   init_dev_input(N_input); /* register interrupt handlers */
   needs_to_fire = 0;
   fired = 0;
}
Apply: {
  int progress = 0;
  if (needs_to_fire == 0) {
    /**** BEGINNING NEW EXECUTION OF FUNCTION ****/
    needs_to_fire = size(out1)/N_out; /* calculate the firing granularity */
    fired = 0;
}

  while (needs_to_fire) {
    /*
    ** if data is available then place it in out1 and out2,
    ** else return failure and try again later.
    */
    float *addr1 = out1 + (N_out * fired);
    float *addr2 = out2 + (N_out * fired);
    if (!read_dev_input(addr1, addr2, N_out)) {
    if (progress) OStaticProgressMade();
    else OStaticFailed("Not enough data available from the I/O device yet");
```

```
    break;
    }
    fired++;
    progress = 1;
    if (needs_to_fire == fired) {
       needs_to_fire = 0;
     }
    }
}
```

9.4 Notice there are no direct references to the Sharc system.

Hiding all the Sharc specific calls in the underlying functions (init_dev_input() and read_dev_input()) is recommended. These functions are prototyped in the header file e_dev_input.h (from the Include method). Also, in the Apply method the firing granularity is checked to see if multiple real-time data sets are to be processed as one large block. This allows for transparent scalability.

File location: ~/gedae/include/embeddable/e_dev_input.h

```
#ifndef __e_dev_input_h_
#define __e_dev_input_h_

void init_dev_input(int N_input);
int read_dev_input(float *out1, float *out2, int N_out);

#endif
```

9.5 The Reset Method

The Reset method registers the interrupt handlers and sets up any buffers and flags. The following code is used to register the interrupt handlers, allocate a double buffer where the data will be placed, and set a flag that represents the state of the real-time data. Global variables are used for handles to shared buffers between the interrupt handler and the

Apply function. Double buffering is used so while one buffer is being processed another may be filled with new data.

File location: ~/gedae/source/embeddable/e_dev_input.c

```
Static int BUFFER_SIZE;
static float *CURRENT_BUFFER;   /* handle to global buffer */
static float *DEVICE_INPUT_BUFFER1;   /* global buffer1 */
static float *DEVICE_INPUT_BUFFER2;  /* global buffer2 */
static int DATA_READY;   /* used to indicate if data is ready */

void init_dev_input(int N_input) {
 /**** ALLOCATE A DOUBLE BUFFER FOR OUTPUT OF THE INPUT
DEVICE
****/
DEVICE_INPUT_BUFFER1 = (float *)calloc(N_input, sizeof(float));
DEVICE_INPUT_BUFFER2 = (float *)calloc(N_input, sizeof(float));
CURRENT_BUFFER = DEVICE_INPUT_BUFFER1;
BUFFER_SIZE = N_input;

/**** SET TO: NO DATA IS AVAILABLE YET ****/
DATA_READY = 0;

/**** REGISTER "I/O HAS DATA READY" INTERRUPT HANDLER ****/
interrupt_handler(SIG_DEV_DATA_READY, read_iodev_handler);

/**** REGISTER "DMA COMPLETE" INTERRUPT HANDLER ****/
interrupt_handler(SIG_DMA_COMPLETE, dma_done_handler);
}
```

When the "SIG_DEV_DATA_READY" interrupt occurs, "read_iodev_handler" gets called. The handler starts a DMA that copies the real-time data into one of the double buffers. Since the DMA processes in the background, the handler can exit right away, allowing normal processing to continue. When the DMA transfers complete, the interrupt "SIG_DMA_COMPLETE" occurs. The "dma_done_handler" sets the "DATA_READY_FLAG". The following is the code for the interrupt handler:

```
/**** READ DATA FROM THE REAL-TIME INPUT DEVICE ****/
```

```
static void read_iodev_handler(void) {
/**** HANDLE THE DOUBLE BUFFERING ****/
if (CURRENT_BUFFER == DEVICE_INPUT_BUFFER1) {
CURRENT_BUFFER = DEVICE_INPUT_BUFFER2;
} else {
CURRENT_BUFFER = DEVICE_INPUT_BUFFER1;
}

/**** GET INPUT FROM REAL TIME IO DEVICE ****/
start_dma(input_device_addr, CURRENT_BUFFER, BUFFER_SIZE);
}

/**** DMA DONE INTERRUPT HANDLER ****/
static void dma_done_handler(void) {
/**** SET TO: DATA IS AVAILABLE ****/
DATA_READY = 1;
}
```

9.6 The Apply Method

The Apply method checks to see if the real-time data is ready. If not it returns a failure, giving any other schedules a chance to execute. If ready, the data is moved to the outputs. This function should be kept simple. It should just place the new data on the outputs, and set flags to indicate that the data has been consumed. Preliminary processing should be minimized. The following is the code of the function called by the Apply method:

```
/**** CALLED FROM THE RESET METHOD ****/
int read_dev_input(float *out1, float *out2, int N_out) {
 if (!DATA_READY) {
  /**** DATA NOT READY ****/
  return 0;
  } else {
  float *buffer = CURRENT_BUFFER;

  /**** DATA IS READY ****/
  DATA_READY = 0;

 /**** DO ANY PRELIMINARY PROCESSING HERE ****/
```

```
< preprocessing and place in out1 and out2 >

 return 1; /* success */
 }
}
```

List of pseudo functions:

The follow functions and data are definitions of the pseudo code used above.

interrupt_handler(SIGNAL, (*handler()));

Register an interrupt handler based on a signal.

SIG_DEV_DATA_READY

This is an interrupt signal flag indicating when the I/O device has data that is ready to be transferred to memory.

SIG_DMA_COMPLETE

This is an interrupt signal value that represent when the DMA engine has finished moving the requested data.

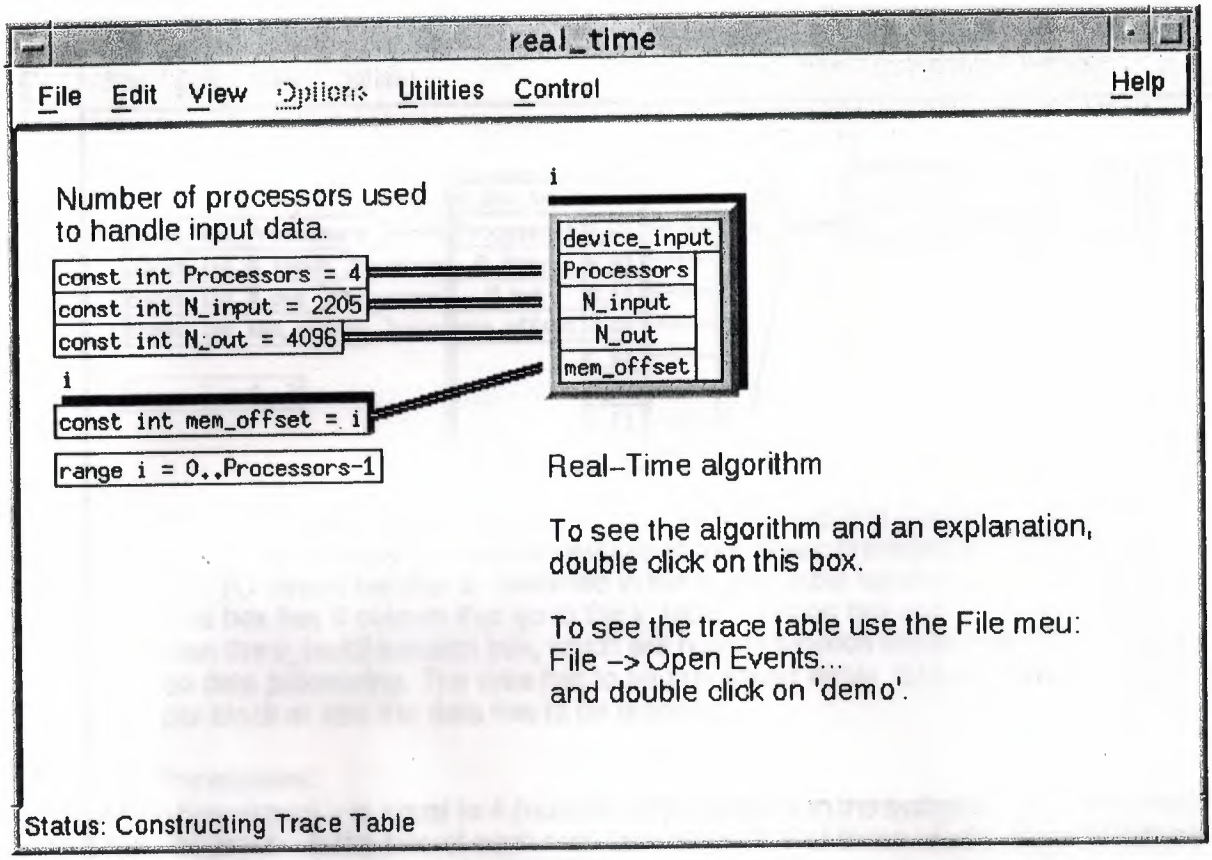
start_dma(source_addr, dest_addr, size);

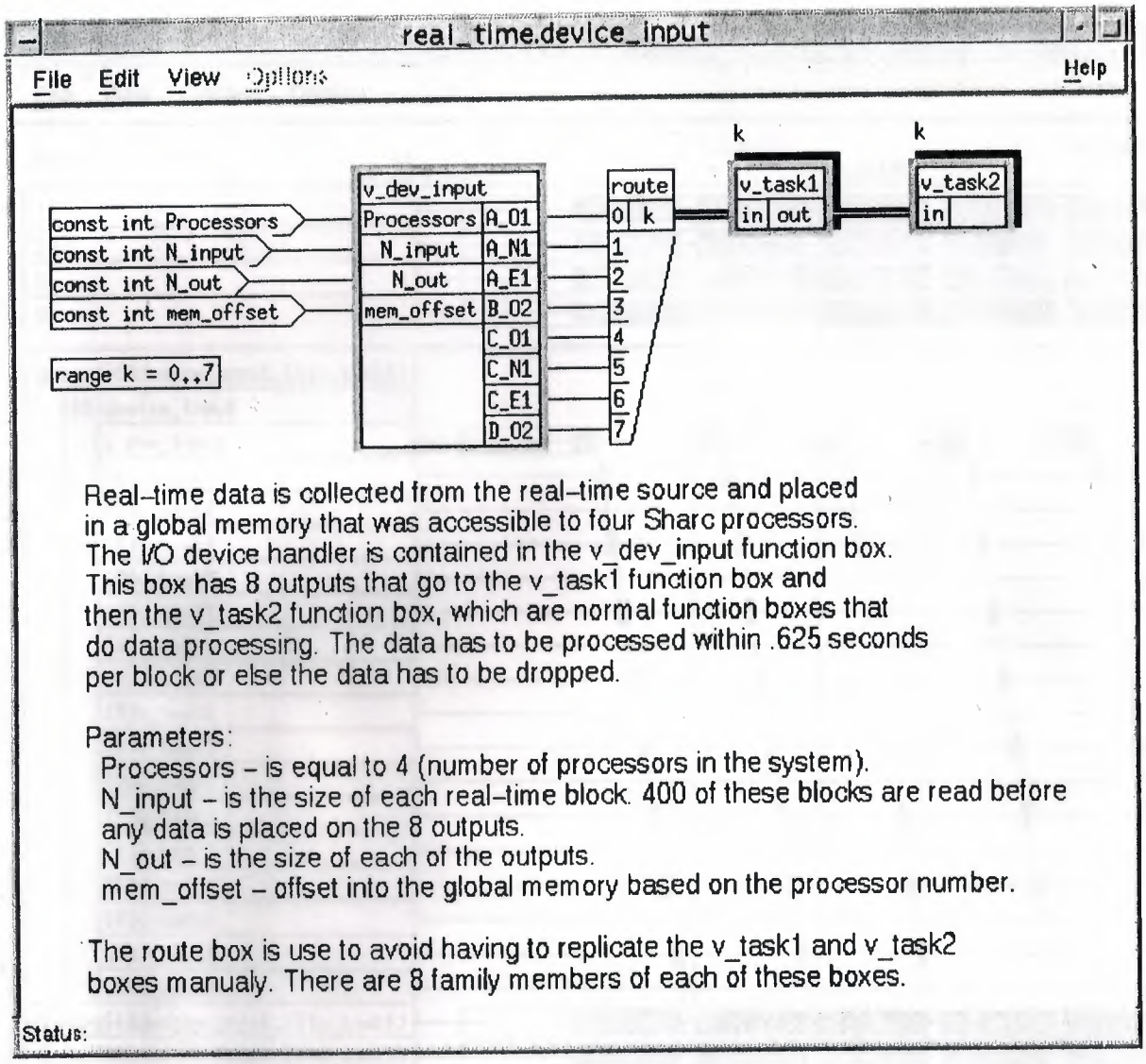
This is like memcpy() but uses a DMA in the background that runs in parallel with the Sharc processor.

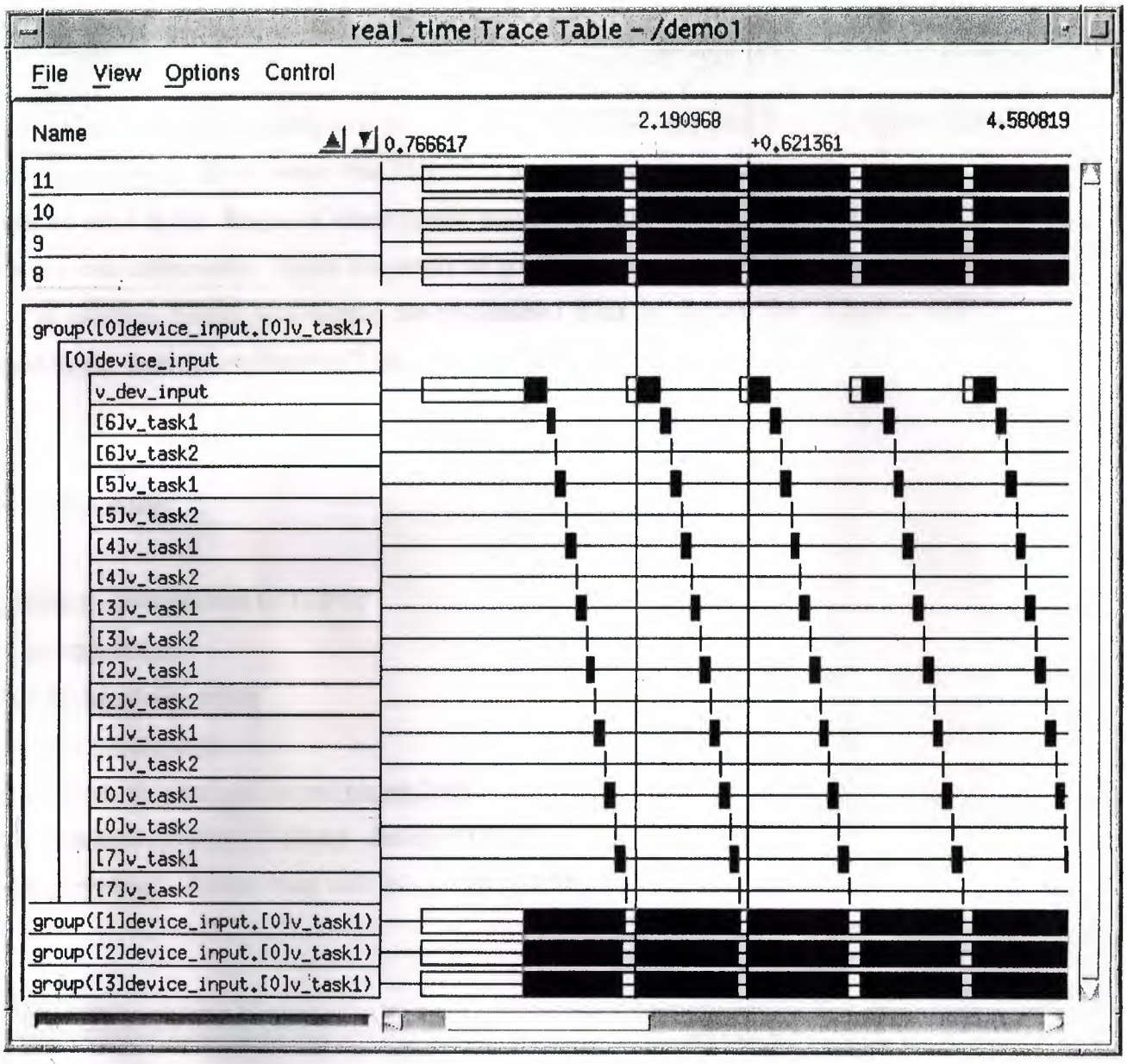
input_device_addr

This is a handle to the I/O device that contains the real-time data.

Example of a real-time algorithm designed with GEDAE™:


real_time
File Edit View Options Utilities Control Help
Number of processors used to handle input data.
const int Processors = 4
const int N_input = 2205
const int N_out = 4096
i
const int mem_offset = i
range i = 0..Processors-1
i
device_input
Processors
N_input
N_out
mem_offset
Real-Time algorithm
To see the algorithm and an explanation, double click on this box.
To see the trace table use the File meu:
File -> Open Events...
and double click on 'demo'.
Status: Constructing Trace Table


real_time.device_input
File Edit View Options Help
const int Processors
const int N_input
const int N_out
const int mem_offset
range k = 0..7
v_dev_input
Processors A_01
N_input A_N1
N_out A_E1
mem_offset B_02
C_01
C_N1
C_E1
D_02
route
0 k
1
2
3
4
5
6
7
k
v_task1
in out
k
v_task2
in
Real-time data is collected from the real-time source and placed
in a global memory that was accessible to four Sharc processors.
The I/O device handler is contained in the v_dev_input function box.
This box has 8 outputs that go to the v_task1 function box and
then the v_task2 function box, which are normal function boxes that
do data processing. The data has to be processed within .625 seconds
per block or else the data has to be dropped.
Parameters:
Processors – is equal to 4 (number of processors in the system).
N_input – is the size of each real-time block. 400 of these blocks are read before
any data is placed on the 8 outputs.
N_out – is the size of each of the outputs.
mem_offset – offset into the global memory based on the processor number.
The route box is use to avoid having to replicate the v_task1 and v_task2
boxes manualy. There are 8 family members of each of these boxes.
Status:


real_time Trace Table - /demo1
File View Options Control
Name
0.766617
2.190968
+0.621361
4.580819
11
10
9
8
group([0]device_input.[0]v_task1)
[0]device_input
v_dev_input
[6]v_task1
[6]v_task2
[5]v_task1
[5]v_task2
[4]v_task1
[4]v_task2
[3]v_task1
[3]v_task2
[2]v_task1
[2]v_task2
[1]v_task1
[1]v_task2
[0]v_task1
[0]v_task2
[7]v_task1
[7]v_task2
group([1]device_input.[0]v_task1)
group([2]device_input.[0]v_task1)
group([3]device_input.[0]v_task1)

10. Conclusion

The project has investigated the principles of TCP/IP and also its applications to real-time programming. It is shown that TCP/IP is a very versatile communications protocol and it can be used in the design of client-server based real time systems. Most of the detail has been left out deliberately. There are plenty of good book written on this subject, in particular [1]. In addition, further information can be obtained from the request for Comments (RFCs), issued by the Internet engineering Task Force (IETF).

11. Bibliography

H. Gilbert -Introduction to TCP/IP

ftp.internic.net:/rfc.

Ftp:// ftp.isi.edu/in- notes

http://www.cisco.com

http://tcpsat.lerc.nasa.gov/tcpsat/papers.html

Mike Coulombe,Emmett Dulaney -Inside TCP/IP

Drew Heywood - Networking with Microsoft TCP/IP