ESCUELA LATINOAMERICANA DE REDES

UNIVERSIDAD DE LOS ANDES
MERIDA-VENEZUELA
(2 AL 20 NOVIEMBRE. 1992)

B2

TCP/P
NETWORKING PROTOCOLS

BEN M. SEGAL

CERN
GINEBRA - SUIZA

Apdo. Postal 49, Mérida 5251-A. Venezuela Telf:+58 74 401379. Fax:+58 74 401286

TCP/IP NETWORKING PROTOCOLS

Ben M. Segal

(ben@cernvax.cern.ch)

CERN, Computing & Networks Division
1211 Geneva 23, Switzerland

August, 1990

Introduction

This presentation gives an elementary overview of the "TCP/IP" networking protocols, which permit logical unification of networking over heterogeneous computer systems and multiple dissimilar media. Much of this material may be familiar to Unix users already, but VMS or VM users may only just be starting to use TCP/IP.

TCP/IP Overview

The TCP/IP or "Internet" protocols were developed as part of the US DoD ARPANET project, starting in 1969, and reached their modern form in the late 1970s and early 1980s. They were designed according to a layered model, similar to the OSI/OSI model but with fewer layers, to connect heterogeneous computer systems across heterogeneous composite networks. User-transparent routing across multiple networks and media is performed by the "Internet Protocol" (IP) datagram layer; an open-ended set of network services and applications is then built upon IP, using either the connectionless "User Datagram Protocol" (UDP) or the connection-oriented "Transmission Control Protocol" (TCP) as appropriate.

Apart from "classical" applications like FTP (File Transfer Protocol) and Telnet (remote login), many newer services like the NFS distributed file system and the X remote windowing system have been added to the Internet suite, making it an extremely active and developing area of networking practice.

The Relation to UNIX

In the early 1980s, UC Berkeley was contracted by DARPA to incorporate TCP/IP into the BSD release of Unix, and this was first offered in 4.1 and then improved in 4.2 and 4.3BSD. Happily, their design of TCP/IP was done in a generic way which permitted the addition of other protocols within the same basic framework, and provided a uniform and elegant programming interface to all these protocols, the so-called "BSD socket interface". The inclusion of a user-level programming interface into a networking standard was such a significant step that this socket interface, despite certain defects, has become a de facto standard for modern networking, and it is found today in many non-BSD (and non-Unix) environments. This greatly simplifies the porting of user-written applications between Unix and the non-Unix world: a C compiler, a socket application library, and the desired protocol support are all that is needed. (At CERN we commonly write applications distributed over Unix, UNICOS, VMS, VM/CMS, MSDOS, Xenix, OS-9 and Apollo systems).

UC Berkeley also wrote a set of Unix-style TCP/IP applications commonly known as the "r" commands ("rsh", "rcp" and "rlogin"), to complement the more basic FTP and Telnet applications. When present on a system, these are normally preferred by users. They are also found sometimes on non-Unix systems.

Unifying LANs and WANs

In the old days (i.e. just yesterday) quite different approaches to networking were taken, depending upon the type and speed of the line which happened to connect a client and its target host: a slow serial line would rapidly condemn the user to manipulations with Kermit or some other terminal emulator; a faster synchronous line would probably restrict him/her to using those proprietary protocols found at his (and, hopefully, at the other) side of the line; and Ethernet or a Satellite would again change the rules and possibly prevent any communication at all.

Even among applications, and even within UNIX, there was (and is) an enormous variety of differing protocols, mostly non-standard, to achieve the same basic aims. Why should mail require the use of "uucp" when it runs over one sort of line, and RSCS over another? Of course the answer is: it doesn't.

Certain manufacturers (notably DEC with DECnet) successfully overcame this sort of difficulty, but at the price of permitting communication only among systems equipped with their proprietary protocols. Certain nations tried to develop open standards (notably the UK with its "Coloured Books") in advance of the (still-awaited..) ISO standards, but these did not find wide international acceptance. Today, the only approximation to a universally supported suite of network protocols, which addresses the whole spectrum of speeds and media, is the Internet suite.

LAN/WAN Media Gatewaying

How in practice do the Internet protocols allow us to carry out data communications over an existing mixture of LANs and WANs? The answer is via IP gateways, which are implemented commercially in many forms. Sometimes the gateway function is performed simply within an IP host computer which happens to have interfaces on more than one IP network; sometimes it is done in a dedicated box; and sometimes it is done in both of these ways at once.

The result is an "internet": an interconnected set of networks. Consider a laboratory which is wired up on its principal site with both Ethernet and one or more Token Ring LAN technologies (as is CERN). Suppose that it also has departments or collaborators with whom the only link is an X.25 WAN, plus some others with whom the only link is a satellite circuit. Then all of the machines at all of those locations which choose to participate in the common Internet logical network will be able to intercommunicate, and only the link speeds (and perhaps satellite delays) will affect the quality of network service. The LOGICAL network services will be the same, and there will not be any requirement to install only one manufacturer's proprietary hardware or software.

Application Bridging

In the present world of multiple networking standards we often require bridges at the protocol and/or application level. This brings us nicely to ULTRIX, which is a bridger "par excellence". ULTRIX is endowed not only with the Internet protocol suite with all modern conveniences (the "r" commands, NFS, X, and so on), but also has DECnet (with a socket interface included) and will one day have OSI. It also has explicitly designed Application Bridges between Telnet and FTP and their DECnet equivalents.

The "VMS ULTRIX Connection" is not really a bridge (although one of its prerelease names implied this). It is simply an implementation under VMS of the server side of the NFS protocols, allowing VMS file systems to be accessed by NFS clients. Other TCP/IP functionality is also present.

How To Set Up a TCP/IP Network

1. Run Unix (either BSD4.x, Ultrix, Sun, Apollo, HP or other system WITH BSD NETWORKING). Note that a "pure" AT&T System III or V will NOT have this; however more and more manufacturers (or third party suppliers) are adding on the missing parts.

2. If not Unix, add on the BSD networking. This is routinely available for Vax VMS, IBM PC DOS/Xenix, IBM VM/MVS, etc. (For VMS there is the "VMS ULTRIX Connection" from DEC, or a choice of third party products such as Multinet or WIN/TCP).

3. Set up the basic IP routing and name servers across the physical media in your network. Ethernet is always available, but gateways are available for IBM Token Ring, Apollo Domain, high and low speed serial lines, X.25, and via direct channel attachments for some systems. An interesting cheap option for PC's is to run over RS232 at 9600bps, via a gateway running "SLIP" (Serial Line IP), in the case that Ethernet is not available.

What Services Do You Get From All This?

1. Remote login to/from all systems (Telnet or rlogin protocols).

2. File transfer (FTP or rcp protocols).

3. Distributed file system (NFS).

4. Distributed window system (X).

5. Remote procedure call support (Apollo NCS or Sun RPC).

6. Task-to-task programming interface (BSD socket library).

7. Remote command execution (rsh protocol).

8. Remote printing (lpr protocol).

9. Electronic mail/conferencing/news (sendmail).

10. Remote file backup.

.... etc... etc...

INTRODUCTION TO THE INTERNET PROTOCOLS

Computer Science Facilities Group
RUTGERS
The State University of New Jersey

3 July 1987

This is an introduction to the Internet networking protocols (TCP/IP). It includes a summary of the facilities available and brief descriptions of the major protocols in the family.

This document is a brief introduction to TCP/IP, followed by advice on what to read for more information. This is not intended to be a complete description. It can give you a reasonable idea of the capabilities of the protocols. But if you need to know any details of the technology, you will want to read the standards yourself. Throughout the text, you will find references to the standards, in the form of *IRC* or *IEN* numbers. These are document numbers. The final section of this document tells you how to get copies of those standards.

1. 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.

[illegible]t 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 described above were designed by Berkeley, Sun, or other organizations. Thus they 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. However if you need a facility that is not listed here, we encourage you to look through the current edition of Internet Protocols (currently RFC 1011), which lists all of the available protocols, and also to look at some of the major TCP/IP implementations to see what various vendors have added.

2. 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
- IF ~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 i. "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.

2.1. 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: lp 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.

Source Port			Destination Port
Sequence Number			
Acknowledgment Number			
Data Offset	Reserved	U A P R S F R C S S Y I G K H T N N	Window
Checksum			Urgent Pointer
your data ... next 500 octets			

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.

2.2. 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:

Version	IHL	Type of Service	Total Length	
Identification			Flags	Fragment Offset
Time to Live		Protocol	Header Checksum	
Source Address				
Destination Address				
TCP header, then your data				

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).

2.3. 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:

Ethernet destination address (first 32 bits)	
Ethernet dest (last 16 bits)	Ethernet source (first 16 bits)
Ethernet source address (last 32 bits)	
Type code	
IP header, then TCP header, then your data ... end of your data	
Ethernet Checksum	

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.)

2.3.1. Well-known sockets and the Applications layer

So far, we have described how a stream of data is broken up into datagrams, sent to another computer, and put back together. However something more is needed in order to accomplish anything useful. There has to be a way for you to open a connection to a specified computer, log into it, tell it what file you want, and control the transmission of the file. (If you have a different application in mind, e.g. computer mail, some analogous protocol is needed.) This is done by *application protocols*. The application protocols run *on top* of TCP/IP. That is, when they want to send a message, they give the message to TCP. TCP makes sure it gets delivered to the other end. Because TCP and IP take care of all the networking details, the applications protocols can treat a network connection as if it were a simple byte stream, like a terminal or phone line.

Before going into more details about applications programs, we have to describe how you find an application. Suppose you want to send a file to a computer whose Internet address is 128.6.4.7. To start the process, you need more than just the Internet address. You have to connect to the FTP server at the other end. In general, network programs are specialized for a specific set of tasks. Most systems have separate programs to handle file transfers, remote terminal logins, mail, etc. When you connect to 128.6.4.7, you have to specify that you want to talk to the FTP server. This is done by having *well-known sockets* for each server. Recall that TCP uses port numbers to keep track of individual conversations. User programs normally use more or less random port numbers. However specific port numbers are assigned to the programs that sit waiting for requests. For example, if you want to send a file, you will start a program called "ftp". It will open a connection using some random number, say 1234, for the port number on its end. However it will specify port number 21 for the other end. This is the official port number for the FTP server. Note that there are two different programs involved. You run ftp on your side. This is a program designed to accept commands from your terminal and pass them on to the other end. The program that you talk to on the other machine is the FTP server. It is designed to accept commands from the network connection, rather than an interactive terminal. There is no need for your program to use a well-known socket number for itself. Nobody is trying to find it. However the servers have to have well-known numbers, so that people can open connections to them and start sending them commands. The official port numbers for each program are given in *Assigned Numbers.*

Note that a connection is actually described by a set of 4 numbers: the Internet address at each end, and the TCP port number at each end. Every datagram has all four of those numbers in it. (The Internet addresses are in the IP header, and the TCP port numbers are in the TCP header.) In order to keep things straight, no two connections can have the same set of numbers. However it is enough for any one number to be different. For example, it is perfectly possible for two different users on a machine to be sending files to the same other machine. This could result in connections with the following parameters:

	Internet addresses	TCP ports
connection 1	128.6.4.194, 128.6.4.7	1234, 21
connection 2	128.6.4.194, 128.6.4.7	1235, 21

Since the same machines are involved, the Internet addresses are the same. Since they are both doing file transfers, one end of the connection involves the well-known port number for FTP. The only thing that differs is the port number for the program that the users are running. That's enough of a difference. Generally, at least one end of the connection asks the network software to assign it a port number that is guaranteed to be unique. Normally, it's the user's end, since the server has to use a well-known number. Now that we know how to open connections, let's get back to the applications programs. As mentioned earlier, once TCP has opened a connection, we have something that might as well be a simple wire. All the hard parts are handled by TCP and IP. However we still need some agreement as to what we send over this connection. In effect this is simply an agreement on what set of commands the application will understand, and the format in which they are to be sent. Generally, what is sent is a combination of commands and data. They use context to differentiate. For example, the mail protocol works like this: Your mail program opens a connection to the mail server at the other end. Your program gives it your machine's name, the sender of the message, and the recipients you want it sent to. It then sends a command saying that it is starting the message. At that point, the other end stops treating what it sees as commands, and starts accepting the message. Your end then starts sending the text of the message. At the end of the message, a special mark is sent (a dot in the first column). After that, both ends understand that your program is again sending commands. This is the simplest way to do things, and the one that most applications use.

File transfer is somewhat more complex. The file transfer protocol involves two different connections. It starts out just like mail. The user's program sends commands like :

- log me in as this user
- here is my password
- send me the file with this name

However once the command to send data is sent, a second connection is opened for the data itself. It would certainly be possible to send the data on the same connection, as mail does. However file transfers often take a long time. The designers of the file transfer protocol wanted to allow the user to continue issuing commands while the transfer is going on. For example, the user might make an inquiry, or he might abort the transfer. Thus the designers felt it was best to use a separate connection for the data and leave the original command connection for commands. (It is also possible to open command connections to two different computers, and tell them to send a file from one to the other. In that case, the data couldn't go over the command connection.)

Remote terminal connections use another mechanism still. For remote logins, there is just one connection. It normally sends data. When it is necessary to send a command (e.g. to set the terminal type or to change some mode), a special character is used to indicate that the next character is a command. If the user happens to type that special character as data, two of them are sent.

We are not going to describe the application protocols in detail in this document. It's better to read the RFC's yourself. However there are a couple of common conventions used by applications that will be described here. First, the common network representation: TCP/IP is intended to be usable on any computer. Unfortunately, not all computers agree on how data is represented. There are differences in character codes (ASCII vs. EBCDIC), in end of line conventions (carriage return, line feed, or a representation using counts), and in whether terminals expect characters to be sent individually or a line at a time. In order to allow computers of different kinds to communicate, each applications protocol defines a standard representation. Note that TCP and IP do not care about the representation. TCP simply sends octets. However the programs at both ends have to agree on how the octets are to be interpreted. The RFC for each application specifies the standard representation for that application. Normally it is *net ASCII*.This characters, with end of line denoted by a carriage return followed by a line feed. For remote login, there is also a definition of a *standard terminal* which turns out to be a half-duplex terminal with echoing happening on the local machine. Most applications also make provisions for the two computers to agree on other representations that they may find more convenient. For example, PDP-10's have 36-bit words. There is a way that two PDP-10's can agree to send a 36-bit binary file. Similarly, two systems that prefer full-duplex terminal conversations can agree on that. However each application has a standard representation, which every machine must support.

2.4. An example application: SMTP

In order to give a bit better idea what is involved in the application protocols, I'm going to show an example of SMTP, which is the mail protocol. (SMTP is *simple mail transfer protocol.*) We assume that a computer called TOPAZ.RUTGERS.EDU wants to send the following message.

Date: Sat, 27 Jun 87 13:26:31 EDT
From: hedrick@topaz.rutgers.edu
To: levy@red.rutgers.edu
Subject: meeting

Let's get together Monday at 1pm.

First, note that the format of the message itself is described by an Internet standard (RFC 822). The standard specifies the fact that the message must be transmitted as net ASCII (i.e. it must be ASCII, with carriage return/linefeed to delimit lines). It also describes the general structure, as a group of header lines, then a blank line, and then the body of the message. Finally, it describes the syntax of the header lines in detail. Generally they consist of a keyword and then a value.

Note that the addressee is indicated as LEVY@RED.RUTGERS.EDU. Initially, addresses were simply *person at machine*. However recent standards have made things more flexible. There are now provisions for systems to handle other systems' mail. This can allow automatic forwarding on behalf of computers not connected to the Internet. It can be used to direct mail for a number of systems to one central mail server. Indeed there is no requirement that an actual computer by the name of RED.RUTGERS.EDU even exist. The name servers could be set up so that you mail to department names, and each department's mail is routed automatically to an appropriate computer. It is also possible that the part before the @ is something other than a user name. It is possible for programs to be set up to process mail. There are also provisions to handle mailing lists, and generic names such as *postmaster* or *operator*

The way the message is to be sent to another system is described by RFC's 821 and 974. The program that is going to be doing the sending asks the name server several queries to determine where to route the message. The first query is to find out which machines handle mail for the name RED.RUTGERS.EDU. In this case, the server replies that RED.RUTGERS.EDU handles its own mail. The program then asks for the address of RED.RUTGERS.EDU, which is 128.6.4.2. Then the mail program opens a TCP connection to port 25 on 128.6.4.2. Port 25 is the well-known socket used for receiving mail. Once this connection is established, the mail program starts sending commands. Here is a typical conversation. Each line is labelled as to whether it is from TOPAZ or RED. Note that TOPAZ initiated the connection:

```
RED      220 RED.RUTGERS.EDU SMTP Service at 29 Jun 87 05:17:18 EDT
TOPAZ    HELO topaz.rutgers.edu
RED      250 RED.RUTGERS.EDU - Hello, TOPAZ.RUTGERS.EDU
TOPAZ    MAIL From:<hedrick@topaz.rutgers.edu>
RED      250 MAIL accepted
TOPAZ    RCPT To:<levy@red.rutgers.edu>
RED      250 Recipient accepted
TOPAZ    DATA
RED      354 Start mail input; end with <CRLF>.<CRLF>
TOPAZ    Date: Sat, 27 Jun 87 13:26:31 EDT
TOPAZ    From: hedrick@topaz.rutgers.edu
TOPAZ    To: levy@red.rutgers.edu
TOPAZ    Subject: meeting
TOPAZ
TOPAZ    Let's get together Monday at 1pm.
TOPAZ    .
RED      250 OK
TOPAZ    QUIT
RED      221 RED.RUTGERS.EDU Service closing transmission channel
```

First, note that commands all use normal text. This is typical of the Internet standards. Many of the protocols use standard ASCII commands. This makes it easy to watch what is going on and to diagnose problems. For example, the mail program keeps a log of each conversation. If something goes wrong, the log file can simply be mailed to the postmaster. Since it is normal text, he can see what was going on. It also allows a human to interact directly with the mail server, for testing. (Some newer protocols are complex enough that this is not practical. The commands would have to have a syntax that would require a significant parser. Thus there is a tendency for newer protocols to use binary formats. Generally they are structured like C or Pascal record structures.) Second, note that the responses all begin with numbers. This is also typical of Internet protocols. The allowable responses are defined in the protocol. The numbers allow the user program to respond unambiguously. The rest of the response is text, which is normally for use by any human who may be watching or looking at a log. It has no effect on the operation of the programs. (However there is one point at which the protocol uses part of the text of the response.) The commands themselves simply allow the mail program on one end to tell the mail server the information it needs to know in order to deliver the message. In this case, the mail server could get the information by looking at the message itself. But for more complex cases, that would not be safe. Every session must begin with a HELO, which gives the name of the system that initiated the connection. Then the sender and recipients are specified. (There can be more than one RCPT command, if there are several recipients.) Finally the data itself is sent. Note that the text of the message is terminated by a line containing just a period. (If such a line appears in the message, the period is doubled.) After the message is accepted, the sender can send another message, or terminate the session as in the example above.

Generally, there is a pattern to the response numbers. The protocol defines the specific set of responses that can be sent as answers to any given command. However programs that don't want to analyze them in detail can just look at the first digit. In general, responses that begin with a 2 indicate success. Those that begin with 3 indicate that some further action is needed, as shown above. 4 and 5 indicate errors. 4 is a temporary error, such as a disk filling. The message should be saved, and tried again later. 5 is a permanent error, such as a non-existent recipient. The message should be returned to the sender with an error message.

(For more details about the protocols mentioned in this section, see RFC's 821/822 for mail, RFC 959 for file transfer, and RFC's 854/855 for remote logins. For the well-known port numbers, see the current edition of Assigned Numbers, and possibly RFC 814.)

3. PROTOCOLS OTHER THAN TCP: UDP AND ICMP

So far, we have described only connections that use TCP. Recall that TCP is responsible for breaking up messages into datagrams, and reassembling them properly. However in many applications, we have messages that will always fit in a single datagram. An example is name lookup. When a user attempts to make a connection to another system, he will generally specify the system by name, rather than Internet address. His system has to translate that name to an address before it can do anything. Generally, only a few systems have the database used to translate names to addresses. So the user's system will want to send a query to one of the systems that has the database. This query is going to be very short. It will certainly fit in one datagram. So will the answer. Thus it seems silly to use TCP. Of course TCP does more than just break things up into datagrams. It also makes sure that the data arrives, resending datagrams where necessary. But for a question that fits in a single datagram, we don't need all the complexity of TCP to do this. If we don't get an answer after a few seconds, we can just ask again. For applications like this, there are alternatives to TCP.

The most common alternative is UDP *User Datagram Protocol*. UDP is designed for applications where you don't need to put sequences of datagrams together. It fits into the system much like TCP. There is a UDP header. The network software puts the UDP header on the front of your data, just as it would put a TCP header on the front of your data. Then UDP sends the data to IP, which adds the IP header, putting UDP's protocol number in the protocol field instead of TCP's protocol number. However UDP doesn't do as much as TCP does. It doesn't split data into multiple datagrams. It doesn't keep track of what it has sent so it can resend if necessary. About all that UDP provides is port numbers, so that several programs can use UDP at once. UDP port numbers are used just like TCP port numbers. There are well-known port numbers for servers that use UDP. Note that the UDP header is shorter than a TCP header. It still has source and destination port numbers, and a checksum, but that's about it. No sequence number, since it is not needed. UDP is used by the protocols that handle name lookups (see IEN 116, RFC 882, and RFC 883), and a number of similar protocols.

Another alternative protocol is ICMP *Internet Control Message Protocol*. ICMP is used for error messages, and other messages intended for the TCP/IP software itself, rather than any particular user program. For example, if you attempt to connect to a host, your system may get back an ICMP message saying *host unreachable*. ICMP can also be used to find out some information about the network. See RFC 792 for details of ICMP. ICMP is similar to UDP, in that it handles messages that fit in one datagram. However it is even simpler than UDP. It doesn't even have port numbers in its header. Since all ICMP messages are interpreted by the network software itself, no port numbers are needed to say where a ICMP message is supposed to go.

4. KEEPING TRACK OF NAMES AND INFORMATION: THE DOMAIN SYSTEM

As we indicated earlier, the network software generally needs a 32-bit Internet address in order to open a connection or send a datagram. However users prefer to deal with computer names rather than numbers. Thus there is a database that allows the software to look up a name and find the corresponding number. When the Internet was small, this was easy. Each system would have a file that listed all of the other systems, giving both their name and number. There are now too many computers for this approach to be practical. Thus these files have been replaced by a set of name servers that keep track of host names and the corresponding Internet addresses. (In fact these servers are somewhat more general than that. This is just one kind of information stored in the domain system.) Note that a set of interlocking servers are used, rather than a single central one. There are now so many different institutions connected to the Internet that it would be impractical for them to notify a central authority whenever they installed or moved a computer. Thus naming authority is delegated to individual institutions. The name servers form a tree, corresponding to institutional structure. The names themselves follow a similar structure. A typical example is the name BORAX.LCS.MIT.EDU. This is a computer at the Laboratory for Computer Science (LCS) at MIT. In order to find its Internet address, you might potentially have to consult 4 different servers. First, you would ask a central server (called the root) where the EDU server is. EDU is a server that keeps track of educational institutions. The root server would give you the names and Internet addresses of several servers for EDU. (There are several servers at each level, to allow for the possibly that one might be down.) You would then ask EDU where the server for MIT is. Again, it would give you names and Internet addresses of several servers for MIT. Generally, not all of those servers would be at MIT, to allow for the possibility of a general power failure at MIT. Then you would ask MIT where the server for LCS is, and finally you would ask one of the LCS servers about BORAX. The final result would be the Internet address for BORAX.LCS.MIT.EDU. Each of these levels is referred to as a *domain*. The entire name, BORAX.LCS.MIT.EDU, is called a *domain name*. (So are the names of the higher-level domains, such as LCS.MIT.EDU, MIT.EDU, and EDU.)

Fortunately, you don't really have to go through all of this most of the time. First of all, the root name servers also happen to be the name servers for the top-level domains such as EDU. Thus a single query to a root server will get you to MIT. Second, software generally remembers answers that it got before. So once we look up a name at LCS.MIT.EDU, our software remembers where to find servers for LCS.MIT.EDU, MIT.EDU, and EDU. It also remembers the translation of BORAX.LCS.MIT.EDU. Each of these pieces of information has a *time to live* associated with it. Typically this is a few days. After that, the information expires and has to be looked up again. This allows institutions to change things.

The domain system is not limited to finding out Internet addresses. Each domain name is a node in a database The node can have records that define a number of different properties. Examples are Internet address, computer type, and a list of services provided by a computer. A program can ask for a specific piece of information, or all information about a given name. It is possible for a node in the database to be marked as an "alias" (or nickname) for another node. It is also possible to use the domain system to store information about users, mailing lists, or other objects.

There is an Internet standard defining the operation of these databases, as well as the protocols used to make queries of them. Every network utility has to be able to make such queries, since this is now the official way to evaluate host names. Generally utilities will talk to a server on their own system. This server will take care of contacting the other servers for them. This keeps down the amount of code that has to be in each application program.

The domain system is particularly important for handling computer mail. There are entry types to define what computer handles mail for a given name, to specify where an individual is to receive mail, and to define mailing lists. (See RFC's 882, 883, and 973 for specifications of the domain system. RFC 974 defines the use of the domain system in sending mail.)

5. ROUTING

The description above indicated that the IP implementation is responsible for getting datagrams to the destination indicated by the destination address, but little was said about how this would be done. The task of finding how to get a datagram to its destination is referred to as routing. In fact many of the details depend upon the particular implementation. However some general things can be said.

First, it is necessary to understand the model on which IP is based. IP assumes that a system is attached to some local network. We assume that the system can send datagrams to any other system on its own network. (In the case of Ethernet, it simply finds the Ethernet address of the destination system, and puts the datagram out on the Ethernet.) The problem comes when a system is asked to send a datagram to a system on a different network. This problem is handled by gateways. A gateway is a system that connects a network with one or more other networks. Gateways are often normal computers that happen to have more than one network interface. For example, we have a Unix machine that has two different Ethernet interfaces. Thus it is connected to networks 128.6.4 and 128.6.3. This machine can act as a gateway between those two networks. The software on that machine must be set up so that it will forward datagrams from one network to the other. That is, if a machine on network 128.6.4 sends a datagram to the gateway, and the datagram is addressed to a machine on network 128.6.3, the gateway will forward the datagram to the destination. Major communications centers often have gateways that connect a number of different networks. (In many cases, special-purpose gateway systems provide better performance or reliability than general-purpose systems acting as gateways. A number of vendors sell such systems.)

Routing in IP is based entirely upon the network number of the destination address. Each computer has a table of network numbers. For each network number, a gateway is listed. This is the gateway to be used to get to that network. Note that the gateway doesn't have to connect directly to the network. It just has to be the best place to go to get there. For example at Rutgers, our interface to NSFnet is at the John von Neuman Supercomputer Center (JvNC). Our connection to JvNC is via a high-speed serial line connected to a gateway whose address is 128.6.3.12. Systems on net 128.6.3 will list 128.6.3.12 as the gateway for many off-campus networks. However systems on net 128.6.4 will list 128.6.4.1 as the gateway to those same off-campus networks. 128.6.4.1 is the gateway between networks 128.6.4 and 128.6.3, so it is the first step in getting to JvNC.

When a computer wants to send a datagram, it first checks to see if the destination address is on the system's own local network. If so, the datagram can be sent directly. Otherwise, the system expects to find an entry for the network that the destination address is on. The datagram is sent to the gateway listed in that entry. This table can get quite big. For example, the Internet now includes several hundred individual networks. Thus various strategies have been developed to reduce the size of the routing table. One strategy is to depend upon *default routes*. Often, there is only one gateway out of a network.

This gateway might connect a local Ethernet to a campus-wide backbone network. In that case, we don't need to have a separate entry for every network in the world. We simply define that gateway as a *default*. When no specific route is found for a datagram, the datagram is sent to the default gateway. A default gateway can even be used when there are several gateways on a network. There are provisions for gateways to send a message saying :

I'm not the best gateway -- use this one instead.

(The message is sent via ICMP. See RFC 792.) Most network software is designed to use these messages to add entries to their routing tables. Suppose network 128.6.4 has two gateways, 128.6.4.59 and 128.6.4.1. 128.6.4.59 leads to several other internal Rutgers networks. 128.6.4.1 leads indirectly to the NSFnet. Suppose we set 128.6.4.59 as a default gateway, and have no other routing table entries. Now what happens when we need to send a datagram to MIT? MIT is network 18. Since we have no entry for network 18, the datagram will be sent to the default, 128.6.4.59. As it happens, this gateway is the wrong one. So it will forward the datagram to 128.6.4.1. But it will also send back an error saying in effect:

to get to network 18, use 128.6.4.1

Our software will then add an entry to the routing table. Any future datagrams to MIT will then go directly to 128.6.4.1. (The error message is sent using the ICMP protocol. The message type is called *ICMP redirect.)*

Most IP experts recommend that individual computers should not try to keep track of the entire network. Instead, they should start with default gateways, and let the gateways tell them the routes, as just described. However this doesn't say how the gateways should find out about the routes. The gateways can't depend upon this strategy. They have to have fairly complete routing tables. For this, some sort of routing protocol is needed. A routing protocol is simply a technique for the gateways to find each other, and keep up to date about the best way to get to every network. RFC 1009 contains a review of gateway design and routing. However rip.doc is probably a better introduction to the subject. It contains some tutorial material, and a detailed description of the most commonly-used routing protocol.

6. DETAILS ABOUT INTERNET ADDRESSES: SUBNETS AND BROADCASTING

As indicated earlier, Internet addresses are 32-bit numbers, normally written as 4 octets (in decimal), e.g. 128.6.4.7. There are actually 3 different types of address. The problem is that the address has to indicate both the network and the host within the network. It was felt that eventually there would be lots of networks. Many of them would be small, but probably 24 bits would be needed to represent all the IP networks. It was also felt that some very big networks might need 24 bits to represent all of their hosts. This would seem to lead to 48 bit addresses. But the designers really wanted to use 32 bit addresses. So they adopted a kludge. The assumption is that most of the networks will be small. So they set up three different ranges of address. Addresses beginning with 1 to 126 use only the first octet for the network number. The other three octets are available for the host number. Thus 24 bits are available for hosts. These numbers are used for large networks. But there can only be 126 of these very big networks. The Arpanet is one, and there are a few large commercial networks. But few normal organizations get one of these *class A* addresses. For normal large organizations, *class B* addresses are used. Class B addresses use the first two octets for the network number. Thus network numbers are 128.1 through 191.254. (We avoid 0 and 255, for reasons that we see below. We also avoid addresses beginning with 127, because that is used by some systems for special purposes.) The last two octets are available for host addesses, giving 16 bits of host address. This allows for 64516 computers, which should be enough for most organizations. (It is possible to get more than one class B address, if you run out.) Finally, class C addresses use three octets, in the range 192.1.1 to 223.254.254. These allow only 254 hosts on each network, but there can be lots of these networks. Addresses above 223 are reserved for future use, as class D and E (which are currently not defined).

Many large organizations find it convenient to divide their network number into *subnets*. For example, Rutgers has been assigned a class B address, 128.6. We find it convenient to use the third octet of the address to indicate which Ethernet a host is on. This division has no significance outside of Rutgers. A computer at another institution would treat all datagrams addressed to 128.6 the same way. They would not look at the third octet of the address. Thus computers outside Rutgers would not have different routes for 128.6.4 or 128.6.5. But inside Rutgers, we treat 128.6.4 and 128.6.5 as separate networks. In effect, gateways inside Rutgers have separate entries for each Rutgers subnet, whereas gateways outside Rutgers just have one entry for 128.6. Note that we could do exactly the same thing by using a separate class C address for each Ethernet. As far as Rutgers is concerned, it would be just as convenient for us to have a number of class C addresses. However using class C addresses would make things inconvenient for the rest of the world. Every institution that wanted to talk to us would have to have a separate entry for each one of our networks. If every institution did this, there would be far too many networks for any reasonable gateway to keep track of. By subdividing a class B network, we hide our internal structure from everyone else, and save them trouble. This subnet strategy requires special provisions in the network software. It is described in RFC 950.

0 and 255 have special meanings. 0 is reserved for machines that don't know their address. In certain circumstances it is possible for a machine not to know the number of the network it is on, or even its own host address. For example, 0.0.0.23 would be a machine that knew it was host number 23, but didn't know on what network.

255 is used for broadcast. A broadcast is a message that you want every system on the network to see. Broadcasts are used in some situations where you don't know who to talk to. For example, suppose you need to look up a host name and get its Internet address. Sometimes you don't know the address of the nearest name server. In that case, you might send the request as a broadcast. There are also cases where a number of systems are interested in information. It is then less expensive to send a single broadcast than to send datagrams individually to each host that is interested in the information. In order to send a broadcast, you use an address that is made by using your network address, with all ones in the part of the address where the host number goes. For example, if you are on network 128.6.4, you would use 128.6.4.255 for broadcasts. How this is actually implemented depends upon the medium. It is not possible to send broadcasts on the Arpanet, or on point to point lines. However it is possible on an Ethernet. If you use an Ethernet address with all its bits on (all ones), every machine on the Ethernet is supposed to look at that datagram.

Although the official broadcast address for network 128.6.4 is now 128.6.4.255, there are some other addresses that may be treated as broadcasts by certain implementations. For convenience, the standard also allows 255.255.255.255 to be used. This refers to all hosts on the local network. It is often simpler to use 255.255.255.255 instead of finding out the network number for the local network and forming a broadcast address such as 128.6.4.255. In addition, certain older implementations may use 0 instead of 255 to form the broadcast address. Such implementations would use 128.6.4.0 instead of 128.6.4.255 as the broadcast address on network 128.6.4. Finally, certain older implementations may not understand about subnets. Thus they consider the network number to be 128.6. In that case, they will assume a broadcast address of 128.6.255.255 or 128.6.0.0. Until support for broadcasts is implemented properly, it can be a somewhat dangerous feature to use.

Because 0 and 255 are used for unknown and broadcast addresses, normal hosts should never be given addresses containing 0 or 255. Addresses should never begin with 0, 127, or any number above 223. Addresses violating these rules are sometimes referred to as *Martians* because of rumors that the Central University of Mars is using network 225.

7. DATAGRAM FRAGMENTATION AND REASSEMBLY

TCP/IP is designed for use with many different kinds of network. Unfortunately, network designers do not agree about how big packets can be. Ethernet packets can be 1500 octets long. Arpanet packets have a maximum of around 1000 octets. Some very fast networks have much larger packet sizes. At first, you might think that IP should simply settle on the smallest possible size. Unfortunately, this would cause serious performance problems. When transferring large files, big packets are far more efficient than small ones. So we want to be able to use the largest packet size possible. But we also want to be able to handle networks with small limits. There are two provisions for this. First, TCP has the ability to negotiate about datagram size. When a TCP connection first opens, both ends can send the maximum datagram size they can handle. The smaller of these numbers is used for the rest of the connection. This allows two implementations that can handle big datagrams to use them, but also lets them talk to implementations that can't handle them. However this doesn't completely solve the problem. The most serious problem is that the two ends don't necessarily know about all of the steps in between. For example, when sending data between Rutgers and Berkeley, it is likely that both computers will be on Ethernets. Thus they will both be prepared to handle 1500-octet datagrams. However the connection will at some point end up going over the Arpanet. It can't handle packets of that size. For this reason, there are provisions to split datagrams up into pieces. (This is referred to as "fragmentation".) The IP header contains fields indicating the a datagram has been split, and enough information to let the pieces be put back together. If a gateway connects an Ethernet to the Arpanet, it must be prepared to take 1500-octet Ethernet packets and split them into pieces that will fit on the Arpanet. Furthermore, every host implementation of TCP/IP must be prepared to accept pieces and put them back together. This is referred to as *reassembly*.

TCP/IP implementations differ in the approach they take to deciding on datagram size. It is fairly common for implementations to use 576-byte datagrams whenever they can't verify that the entire path is able to handle larger packets. This rather conservative strategy is used because of the number of implementations with bugs in the code to reassemble fragments. Implementors often try to avoid ever having fragmentation occur. Different implementors take different approaches to deciding when it is safe to use large datagrams. Some use them only for the local network. Others will use them for any network on the same campus. 576 bytes is a *safe* size, which every implementation must support.

8. ETHERNET ENCAPSULATION: ARP

There was a brief discussion earlier about what IP datagrams look like on an Ethernet. The discussion showed the Ethernet header and checksum. However it left one hole: It didn't say how to figure out what Ethernet address to use when you want to talk to a given Internet address. In fact, there is a separate protocol for this, called ARP *Address Resolution Protocol.* (Note by the way that ARP is not an IP protocol. That is, the ARP datagrams do not have IP headers.) Suppose you are on system 128.6.4.194 and you want to connect to system 128.6.4.7. Your system will first verify that 128.6.4.7 is on the same network, so it can talk directly via Ethernet. Then it will look up 128.6.4.7 in its ARP table, to see if it already knows the Ethernet address. If so, it will stick on an Ethernet header, and send the packet. But suppose this system is not in the ARP table. There is no way to send the packet, because you need the Ethernet address. So it uses the ARP protocol to send an ARP request. Essentially an ARP request says :

I need the Ethernet address for 128.6.4.7

Every system listens to ARP requests. When a system sees an ARP request for itself, it is required to respond. So 128.6.4.7 will see the request, and will respond with an ARP reply saying in effect :

128.6.4.7 is 8:0:20:1:56:34

(Recall that Ethernet addresses are 48 bits. This is 6 octets. Ethernet addresses are conventionally shown in hex, using the punctuation shown.) Your system will save this information in its ARP table, so future packets will go directly. Most systems treat the ARP table as a cache, and clear entries in it if they have not been used in a certain period of time.

Note by the way that ARP requests must be sent as *broadcasts.* There is no way that an ARP request can be sent directly to the right system. After all, the whole reason for sending an ARP request is that you don't know the Ethernet address. So an Ethernet address of all ones is used, i.e. ff:ff:ff:ff:ff:ff. By convention, every machine on the Ethernet is required to pay attention to packets with this as an address. So every system sees every ARP requests. They all look to see whether the request is for their own address. If so, they respond. If not, they could just ignore it. (Some hosts will use ARP requests to update their knowledge about other hosts on the network, even if the request isn't for them.) Note that packets whose IP address indicates broadcast (e.g. 255.255.255.255 or 128.6.4.255) are also sent with an Ethernet address that is all ones.

9. GETTING MORE INFORMATION

This directory contains documents describing the major protocols. There are literally hundreds of documents, so we have chosen the ones that seem most important. Internet standards are called RFC's. RFC stands for Request for Comment. A proposed standard is initially issued as a proposal, and given an RFC number. When it is finally accepted, it is added to Official Internet Protocols, but it is still referred to by the RFC number. We have also included two IEN's. (IEN's used to be a separate classification for more informal documents. This classification no longer exists -- RFC's are now used for all official Internet documents, and a mailing list is used for more informal reports.) The convention is that whenever an RFC is revised, the revised version gets a new number. This is fine for most purposes, but it causes problems with two documents: Assigned Numbers and Official Internet Protocols. These documents are being revised all the time, so the RFC number keeps changing. You will have to look in rfc-index.txt to find the number of the latest edition. Anyone who is seriously interested in TCP/IP should read the RFC describing IP (791). RFC 1009 is also useful. It is a specification for gateways to be used by NSFnet. As such, it contains an overview of a lot of the TCP/IP technology. You should probably also read the description of at least one of the application protocols, just to get a feel for the way things work. Mail is probably a good one (821/822). TCP (793) is of course a very basic specification. However the spec is fairly complex, so you should only read this when you have the time and patience to think about it carefully. Fortunately, the author of the major RFC's (Jon Postel) is a very good writer. The TCP RFC is far easier to read than you would expect, given the complexity of what it is describing. You can look at the other RFC's as you become curious about their subject matter.

Here is a list of the documents you are more likely to want:

rfc-index	list of all RFC's
rfc1012	somewhat fuller list of all RFC's
rfc1011	Official Protocols. It's useful to scan this to see what tasks protocols have been built for. This defines which RFC's are actual standards, as opposed to requests for comments.
rfc1010	Assigned Numbers. If you are working with TCP/IP, you will probably want a hardcopy of this as a reference. It's not very exciting to read. It lists all the offically defined well-known ports and lots of other things.
rfc1009	NSFnet gateway specifications. A good overview of IP routing and gateway technology.
rfc1001/2	netBIOS: networking for PC's
rfc973	update on domains
rfc959	FTP (file transfer)
rfc950	subnets
rfc937	POP2: protocol for reading mail on PC's
rfc894	how IP is to be put on Ethernet, see also rfc825
rfc882/3	domains (the database used to go from host names to Internet address and back -- also used to handle UUCP these days). See also rfc973
rfc854/5	telnet - protocol for remote logins
rfc826	ARP - protocol for finding out Ethernet addresses
rfc821/2	mail
rfc814	names and ports - general concepts behind well-known ports
rfc793	TCP
rfc792	ICMP
rfc791	IP
rfc768	UDP
rip.doc	details of the most commonly-used routing protocol
ien-116	old name server (still needed by several kinds of system)
ien-48	the Catenet model, general description of the philosophy behind TCP/IP

The following documents are somewhat more specialized.

rfc813	window and acknowledgement strategies in TCP
rfc815	datagram reassembly techniques
rfc816	fault isolation and resolution techniques
rfc817	modularity and efficiency in implementation
rfc879	the maximum segment size option in TCP
rfc896	congestion control
rfc827,888,904,975,985	EGP and related issues

To those of you who may be reading this document remotely instead of at Rutgers: The most important RFC's have been collected into a three-volume set, the DDN Protocol Handbook. It is available from the DDN Network Information Center, SRI International, 333 Ravenswood Avenue, Menlo Park, California 94025 (telephone: 800-235-3155). You should be able to get them via anonymous FTP from sri-nic.arpa. File names are:

RFC's:
 rfc:rfc-index.txt
 rfc:rfcxxx.txt
IEN's:

ien:ien-index.txt
ien:ien-xxx.txt

rip.doc is available by anonymous FTP from topaz.rutgers.edu, as /puo/tcp-ip-docs/rip.doc.

Si. .: with access to UUCP but not FTP may be able to retreive them via UUCP from UUCP host rutgers. The file names would be

RFC's:
/topaz/pub/pub/tcp-ip-docs/rfc-index.txt
/topaz/pub/pub/tcp-ip-docs/rfcxxx.txt
IEN's:
/topaz/pub/pub/tcp-ip-docs/ien-index.txt
/topaz/pub/pub/tcp-ip-docs/ien-xxx.txt
/topaz/pub/pub/tcp-ip-docs/rip.doc

Note that SRI-NIC has the entire set of RFC's and IEN's, but rutgers and topaz have only those specifically mentioned above.

CONTENTS

1 WHAT IS TCP/IP ? 2
2 GENERAL DESCRIPTION OF THE TCP/IP PROTOCOLS 4
  2.1 The TCP Level 5
  2.2 The IP Level 7
  2.3 The ETHERNET Level 8
    2.3.1 Well-known sockets and the Applications layer 8
  2.4 An example application: SMTP 10
3 PROTOCOLS OTHER THAN TCP: UDP AND ICMP 11
4 KEEPING TRACK OF NAMES AND INFORMATION: THE DOMAIN SYSTEM 12
5 ROUTING 13
6 DETAILS ABOUT INTERNET ADDRESSES: SUBNETS AND BROADCASTING 14
7 DATAGRAM FRAGMENTATION AND REASSEMBLY 15
8 ETHERNET ENCAPSULATION: ARP 16
9 GETTING MORE INFORMATION 16

Introduction to Administration of an Internet-based Local Network

```
C                        R

        C       S
Computer Science Facilities Group
        C       I

L                        S
```

RUTGERS
The State University of New Jersey
Center for Computers and Information Services
Laboratory for Computer Science Research

24 July 1988

This is an introduction for people who intend to set up or administer a network based on the Internet networking protocols (TCP/IP).

Table of Contents

1. The problem 1
2. Routing and Addressing 2
3. Choosing an addressing structure 3
  3.1 Should you subdivide your address space? 5
  3.2 Subnets vs. multiple network numbers 5
  3.3 How to allocate subnet or network numbers 6
    3.3.1 Dealing with multiple "virtual" subnets on one network 7
  3.4 Choosing an address class 8
4. Setting up routing for an individual computer 9
  4.1 How datagrams are routed 11
  4.2 Fixed routes 13
  4.3 Routing redirects 14
  4.4 Other ways for hosts to find routes 16
    4.4.1 Spying on Routing 16
    4.4.2 Proxy ARP 17
    4.4.3 Moving to New Routes After Failures 22
5. Bridges and Gateways 24
  5.1 Alternative Designs 25
    5.1.1 A mesh of point to point lines 26
    5.1.2 Circuit switching technology 27
    5.1.3 Single-level networks 27
    5.1.4 Mixed designs 28
  5.2 An introduction to alternative switching technologies 29
    5.2.1 Repeaters 29
    5.2.2 Bridges and gateways 30
    5.2.3 More about bridges 32
    5.2.4 More about gateways 34
  5.3 Comparing the switching technologies 34
    5.3.1 Isolation 35
    5.3.2 Performance 36
    5.3.3 Routing 37
    5.3.4 Network management 39
    5.3.5 A final evaluation 41
  5.4 Configuring routing for gateways 44

This document is intended to help people who planning to set up a new network based on the Internet protocols, or to administer an existing one. It assumes a basic familiarity with the TCP/IP protocols, particularly the structure of Internet addresses. A companion paper, "Introduction to the Internet Protocols", may provide a convenient introduction. This document does not attempt to replace technical documentation for your specific TCP/IP implementation. Rather, it attempts to give overall background that is not specific to any particular implementation. It is directed specifically at networks of "medium" complexity. That is, it is probably appropriate for a network involving several dozen buildings. Those planning to manage larger networks will need more preparation than you can get by reading this document.

In a number of cases, commands and output from Berkeley Unix are shown. Most computer systems have commands that are similar in function to these. It seemed more useful to give some actual examples that to limit myself to general talk, even if the actual output you see is slightly different.

1. The problem

This document will emphasize primarily "logical" network architecture. There are many documents and articles in the trade press that discuss actual network media, such as Ethernet, Token Ring, etc. What is generally not made clear in these articles is that the choice of network media is generally not all that critical for the overall design of a network. What can be done by the network is generally determined more by the network protocols supported, and the quality of the implementations. In practice, media are normally chosen based on purely pragmatic grounds: what media are supported by the particular types of computer that you have to connect. Generally this means that Ethernet is used for medium-scale systems, Ethernet or a network based on twisted-pair wiring for micro networks, and specialized high-speed networks (typically token ring) for campus-wide backbones, and for local networks involving super-computer and other very high-performance applications.

Thus this document assumes that you have chosen and installed individual networks such as Ethernet or token ring, and your vendor has helped you connect your computers to these network. You are now faced with the interrelated problems of

- configuring the software on your computers
- finding a way to connect individual Ethernets, token rings, etc., to form a single coherent network
- connecting your networks to the outside world

My primary thesis in this document is that these decisions require a bit of advance thought. In fact, most networks need an "architecture". This consists of a way of assigning addresses, a way of doing routing, and various choices about how hosts interact with the network. These decisions need to be made for the entire network, preferably when it is first being installed.

2. Routing and Addressing

Many of the decisions that you need to make in setting up TCP/IP depend upon routing, so it will be best to give a bit of background on that topic now. I will return to routing in a later section when discussing gateways and bridges. In general, IP datagrams pass through many networks while they are going between the source and destination. Here's a typical example. (Addresses used in the examples are taken from Rutgers University.)

```
          network 1                 network 2     network 3
           128.6.4                   128.6.21      128.121
    ==============================  ==========  ================
       |               |          |     |       |     |          |
    ___|_______   _____|____   ___|_____|__  ___|_____|____  ____|_________
    128.6.4.2     128.6.4.3    128.6.4.1     128.6.21.1      128.121.50.2
                               128.6.21.2    128.121.50.1
    ___________   __________   ____________  ____________   ______________
    computer A    computer B    gateway R     gateway S       computer C
```

This diagram shows three normal computer systems, two gateways, and three networks. The networks might be Ethernets, token rings, or any other sort. Network 2 could even be a single point to point line connecting gateways R and S.

Note that computer A can send datagrams to computer B directly, using network 1. However it can't reach computer C directly, since they aren't on the same network. There are several ways to connect separate networks. This diagram assumes that gateways are used. (In a later section, we'll look at an alternative.) In this case, datagrams going between A and C must be sent through gateway R, network 2, and gateway S. Every computer that uses TCP/IP needs appropriate information and algorithms to allow it to know when datagrams must be sent through a gateway, and to choose an appropriate gateway.

Routing is very closely tied to the choice of addresses. Note that the address of each computer begins with the number of the network that it's attached to. Thus 128.6.4.2 and 128.6.4.3 are both on network 128.6.4. Next, notice that gateways, whose job is to connect networks, have an address on each of those networks. For example, gateway R connects networks 128.6.4 and 128.6.21. Its connection to network 128.6.4 has the address 128.6.4.1. Its connection to network 128.6.21 has the address 128.6.21.2.

Because of this association between addresses and networks, routing decisions can be based strictly on the network number of the destination. Here's what the routing information for computer A might look like:

network	gateway	metric
128.6.4	none	0
128.6.21	128.6.4.1	1
128.121	128.6.4.1	2

>From this table, computer A can tell that datagrams for computers on network 128.6.4 can be sent directly, and datagrams for computers on networks 128.6.21 and 128.121 need to be sent to gateway R for forwarding. The "metric" is used by some routing algorithms as a measure of how far away the destination is. In this case, the metric simply indicates how many gateways the datagram has to go through. (This is often referred to as a "hop count".)

When computer A is ready to send a datagram, it examines the destination address. The network number is taken from the beginning of the address and looked up in the routing table. The table entry indicates whether the packet should be sent directly to the destination or to a gateway.

Note that a gateway is simply a computer that is connected to two different networks, and is prepared to forward packets between them. In many cases it is most efficient to use special-purpose equipment designed for use as a gateway. However it is perfectly possible to use ordinary computers as gateways, as long as they have more than one network interface, and their software is prepared to forward datagrams. Most major TCP/IP implementations (even for microcomputers) are designed to let you use your computer as a gateway. However some of this software has limitations that can cause trouble for your network.

3. Choosing an addressing structure

The first comment to make about addresses is a warning: Before you start using a TCP/IP network, you must get one or more official network numbers. TCP/IP addresses look like this: 128.6.4.3. This address is used by one computer at Rutgers University. The first part of it, 128.6, is a network number, allocated to Rutgers by a central authority. Before you start allocating addresses to your computers, you must get an official network number. Unfortunately, some people set up networks using either a randomly-chosen number, or a number taken from examples in vendor documentation. While this may work in the short run, it is a very bad idea for the long run. Eventually, you will want to connect your network to some other organization's network. Even if your organization is highly secret and very concerned about security, somewhere in your organization there is going to be a research computer that ends up being connected to a nearby university. That university will probably be connected to a large-scale national network. As soon as one of your datagrams escapes your local network, the organization you are talking to is going to become very confused, because the addresses that appear in your datagrams are probably officially allocated to someone else.

The solution to this is simple: get your own network number from the beginning. It costs nothing. If you delay it, then sometime years from now you are going to be faced with the job of changing every address on a large network. Network numbers are currently assigned by the DDN Network Information Center, SRI International, 333 Ravenswood Avenue, Menlo Park, California 94025 (telephone: 800-235-3155). You can get a network number no matter what your network is being used for. You do not need authorization to connect to the Defense Data Network in order to get a number. The main piece of information that will be needed when you apply for a network number is that address class that you want. See below for a discussion of this.

In many ways, the most important decision you have to make in setting up a network is how you will assign Internet addresses to your computers. This choice should be made with a view of how your network is likely to grow. Otherwise, you will find that you have to change addresses. When you have several hundred computers, address changes can be nearly impossible.

Addresses are critical because Internet datagrams are routed on the basis of their address. For example, addresses at Rutgers University have a 2-level structure. A typical address is 128.6.4.3. 128.6 is assigned to Rutgers University by a central authority. As far as the outside world is concerned, 128.6 is a single network. Other universities send any packet whose address begins with 128.6 to the nearest Rutgers gateway. However within Rutgers, we divide up our address space into "subnets". We use the next 8 bits of address to indicate which subnet a computer belongs to. 128.6.4.3 belongs to subnet 128.6.4. Generally subnets correspond to physical networks, e.g. separate Ethernets, although as we will see later there can be exceptions. Systems inside Rutgers, unlike those outside, contain information about the Rutgers subnet structure. So once a packet for 128.6.4.3 arrives at Rutgers, the Rutgers network will route it to the departmental Ethernet, token ring, or whatever, that has been assigned subnet number 128.6.4.

When you start a network, there are several addressing decisions that face you:

- Do you subdivide your address space?

- If so, do you use subnets or class C addresses?

- You do you allocate subnets or class C networks?

- How big an address space do you need?

3.1 Should you subdivide your address space?

It is not necessary to use subnets at all. There are network technologies that allow an entire campus or company to act as a single large logical Ethernet, so that no internal routing is necessary. If you use this technology, then you do not need to subdivide your address space. In that case, the only decision you have to make is what class address to apply for. However we recommend using either a subnet approach or some other method of subdividing your address space in all cases:

- In section 5.2 we will argue that internal gateways are desirable for networks of any degree of complexity.

- Even if you do not need gateways now, you may find later that you need to use them. Thus it probably makes sense to assign addresses as if each Ethernet or token ring was going to be a separate subnet. This will allow for conversion to real subnets later if it proves necessary.

- For network maintenance purposes, it is convenient to have addresses whose structure corresponds to the structure of the network. That is, when you see a stray packet from system 128.6.4.3, it is nice to know that all addresses beginning with 128.6.4 are in a particular building.

3.2 Subnets vs. multiple network numbers

Suppose that you have been convinced that it's a good idea to impose some structure on your addresses. The next question is what that structure should be. There are two basic approaches. One is subnets. The other is multiple network numbers.

The Internet standards specify what constitutes a network number. For numbers beginning with 128 through 191 (the most common numbers these days), the first two octets form the network number. E.g. in 140.3.50.1, 140.3 is the network number. Network numbers are assigned to a particular organization. What you do with the next two octets is up to you. You could choose to make the next octet be a subnet number, or you could use some other scheme entirely. Gateways within your organization will be set up to know the subnetting scheme that you are using. However outside your organization, no one will know that 140.3.50 is one subnet and 140.3.51 is another. They will simply know that 140.3 is your organization. Unfortunately, this ability to add additional structure to the address via subnets was not present in the original TCP/IP specifications. Thus some software is incapable of being told about subnets.

If enough of the software that you are using has this problem, it may be impractical for you to use subnets. Some organizations have used a different approach. It is possible for an organization to apply for several network numbers. Instead of dividing a single network number, say 140.3, into several subnets, e.g. 140.3.1 through 140.3.10, you could apply for 10 different network numbers. Thus you might be assigned the range 140.3 through 140.12. All TCP/IP software will know that these are different network numbers.

While using separate network numbers will work just fine within your organization, it has two very serious disadvantages. The first, and less serious, is that it wastes address space. There are only about 16,000 possible class B addresses. We cannot afford to waste 10 of them on your organization, unless it is very large. This objection is less serious because you would normally ask for class C addresses for this purpose, and there are about 2 million possible class C addresses.

The more serious problem with using several network numbers rather than subnets is that it overloads the routing tables in the rest of the Internet. As mentioned above, when you divide your network number into subnets, this division is known within your organization, but not outside it. Thus systems outside your organization need only one entry in their tables in order to be able to reach you. E.g. other universities have entries in their routing tables for 128.6, which is the Rutgers network number. If you use a range of network numbers instead of subnets, that division will be visible to the entire Internet. If we used 128.6 through 128.16 instead of subdividing 128.6, other universities would need entries for each of those network numbers in their routing tables. As of this writing the routing tables in many of the national networks are exceeding the size of the current routing technology. It would be considered extremely unfriendly for any organization to use more than one network number. This may not be a problem if your network is going to be completely self-contained, or if only one small piece of it will be connected to the outside world. Nevertheless, most TCP/IP experts strongly recommend that you use subnets rather than multiple networks. The only reason for considering multiple networks is to deal with software that cannot handle subnets. This was a problem a few years ago, but is currently less serious. As long as your gateways can handle subnets, you can deal with a few individual computers that cannot by using "proxy ARP" (see below).

3.3 How to allocate subnet or network numbers

Now that you have decided to use subnets or multiple network numbers, you have to decide how to allocate them. Normally this is fairly easy. Each physical network, e.g. Ethernet or token ring, is assigned a separate subnet or network number. However you do have some options.

In some cases it may make sense to assign several subnet numbers to a single physical network. At Rutgers we have a single Ethernet that spans three buildings, using repeaters. It is very clear to us that as computers are added to this Ethernet, it is going to have to be split into several separate Ethernets. In order to avoid having to change addresses when this is done, we have allocated three different subnet numbers to this Ethernet, one per building. (This would be handy even if we didn't plan to split the Ethernet, just to help us keep track of where computers are.) However before doing this, make very sure that the software on all of your computers can handle a network that has three different network numbers on it.

You also have to choose a "subnet mask". This is used by the software on your systems to separate the subnet from the rest of the address. So far we have always assumed that the first two octets are the network number, and the next octet is the subnet number. For class B addresses, the standards specify that the first two octets are the network number. However we are free to choose the boundary between the subnet number and the rest of the address. It's very common to have a one-octet subnet number, but that's not the only possible choice. Let's look again at a class B address, e.g. 128.6.4.3. It is easy to see that if the third octet is used for a subnet number, there are 256 possible subnets and within each subnet there are 256 possible addresses. (Actually, the numbers are more like 254, since it is generally a bad idea to use 0 or 255 for subnet numbers or addresses.) Suppose you know that you will never have more than 128 computers on a given subnet, but you are afraid you might need more than 256 subnets. (For example, you might have a campus with lots of small buildings.) In that case, you could define 10 bits for the subnet number, leaving 6 bits for addresses within each subnet. This choice is expressed by a bit mask, using ones for the bits used by the network and subnet number, and 0's for the bits used for individual addresses. Our normal subnet choice is given as 255.255.255.0. If we chose 10 bit subnet numbers and 6 bit addresses, the subnet mask would be 255.255.255.192.

Generally it is possible to specify the subnet mask for each computer as part of configuring its TCP/IP software. The TCP/IP protocols also allow for computers to send a query asking what the subnet mask is. If your network supports broadcast queries, and there is at least one computer or gateway on the network that knows the subnet mask, it may be unnecessary to set it on the other computers. (This capability brings with it a whole new set of possible problems. One well-known TCP/IP implementation would answer with the wrong subnet mask when queried, thus leading causing every other computer on the network to be misconfigured.)

3.3.1 Dealing with multiple "virtual" subnets on one network

Most software is written under the assumption that every computer on the local network has the same subnet number. When traffic is being sent to a machine with a different subnet number, the software will generally expect to find a gateway to handle forwarding to that subnet. Let's look at the implications. Suppose subnets 128.6.19 and 128.6.20 are on the same Ethernet. Consider the way things look from the point of view of a computer with address 128.6.19.3. It will have no problem sending to other machines with addresses 128.6.19.x. They are on the same subnet, and so our computer will know to send directly to them on the local Ethernet. However suppose it is asked to send a packet to 128.6.20.2. Since this is a different subnet, most software will expect to find a gateway that handles forwarding between the two subnets. Of course there isn't a gateway between subnets 128.6.19 and 128.6.20, since they are on the same Ethernet. Thus it must be possible to tell your software that 128.6.20 is actually on the same Ethernet.

For the most common TCP/IP implementations, it is possible to deal with more than one subnet on a network. For example, Berkeley Unix allows you to define gateways using a command "route add". Suppose that you get from subnet 128.6.19 to subnet 128.6.4 using a gateway whose address is 128.6.19.1. You would use the command

```
  route add 128.6.4.0 128.6.19.1 1
```

This says that to reach subnet 128.6.4, traffic should be sent via the gateway at 128.6.19.1, and that the route only has to go through one gateway. The "1" is referred to as the "routing metric". If you use a metric of 0, you are saying that the destination subnet is on the same network, and no gateway is needed. In our example, on system 128.6.19.3, you would use

```
  route add 128.6.20.0 128.6.19.1 0
```

The actual address used in place of 128.6.19.1 is irrelevant. The metric of 0 says that no gateway is actually going to be used, so the gateway address is not used. However it must be a legal address on the local network.

Note that the commands in this section are simply examples. You should look in the documentation for your particular implementation to see how to configure your routing.

3.4 Choosing an address class

When you apply for an official network number, you will be asked what class of network number you need. The possible answers are A, B, and C. This affects how large an address space you will be allocated. Class A addresses are one octet long, class B addresses are 2 octets, and class C addresses are 3 octets. This represents a tradeoff: there are a lot more class C addresses than class A addresses, but the class C addresses don't allow as many hosts. The idea was that there would be a few very large networks, a moderate number of medium-size ones, and a lot of mom-and-pop stores that would have small networks. Here is a table showing the distinction:

class	range of first octet	network	rest	possible addresses
A	1 - 126	p	q.r.s	16777214
B	128 - 191	p.q	r.s	65534

```
    C        192 - 223              p.q.r       s    254
```

For example network 10, a class A network, has addresses between 10.0.0.1 and 10.255.255.254. So it allows 254**3, or about 16 million possible addresses. (Actually, network 10 has allocated addresses where some of the octets are zero, so there are a few more networks possible.) Network 192.12.88, a class C network has hosts between 192.12.88.1 and 128.12.88.254, i.e. 254 possible hosts.

In general, you will be expected to choose the lowest class that will provide you with enough addresses to handle your growth over the next few years. In general organizations that have computers in many buildings will probably need and be able to get a class B address, assuming that they are going to use subnetting. (If you are going to use many separate network numbers, you would ask for a number of class C addresses.) Class A addresses are normally used only for large public networks and for a few very large corporate networks.

4. Setting up routing for an individual computer

All TCP/IP implementations require some configuration for each host. In some cases this is done in a "system generation". In other cases, various startup and configuration files must be set up on the system. Still other systems get configuration information across the network from a "server". While the details differ, the same kinds of information need to be supplied for most implementations. This includes

- parameters describing the specific machine, such as its Internet address.

- parameters describing the network, such as the subnet mask (if any)

- routing software and the tables that drive it

- startup of various programs needed to handle network tasks

Before a machine is installed on your network, a coordinator should assign it a host name and Internet address. If the machine has more than one network interface, you must assign a separate Internet address for each. (In most cases, the same host name can be used. The name goes with the machine as a whole, whereas the address is associated with the connection to a specific network.) The address should begin with the network number for the network to which it is to be attached. We recommend that you assign addresses starting from 1. Should you find that you need more subnets than your current subnet mask allows, you may later need to expand the subnet mask to use more bits. If all addresses use small numbers, this will be possible.

Generally the Internet address must be specified individually in a configuration file on each computer. However some computers (particularly those without permanent disks on which configuration information could be stored) find out their Internet address by sending a broadcast request over the network. In that case, you will have to make sure that some other system is configured to answer the request. When a system asks for its Internet address, enough information must be put into the request to allow another system to recognize it and to send a response back. For Ethernet systems, generally the request will include the Ethernet address of the requesting system. Ethernet addresses are assigned by the computer manufacturers, and are guaranteed to be unique. Thus they are a good way of identifying the computer. And of course the Ethernet address is also needed in order to send the response back. If it is used as the basis for address lookup, the system that is to answer the request will need a table of Ethernet addresses and the corresponding Internet address. The only problem in constructing this table will be finding the Ethernet address for each computer. Generally, computers are designed so that they print the Ethernet address on the console shortly after being turned on. However in some cases you may have to type a command that displays information about the Ethernet interface.

Generally the subnet mask should be specified in a configuration file associated with the computer. (For Unix systems, the "ifconfig" command is used to specify both the Internet address and subnet mask.) However there are provisions in the IP protocols for a computer to broadcast a request asking for the subnet mask. The subnet mask is an attribute of the network. That is, it is the same for all computers on a given subnet. Thus there is no separate subnet table corresponding to the Ethernet/Internet address mapping table used to answer address queries. Generally any machine on the network that believes it knows the subnet mask will answer any query about the subnet mask. For that reason, an incorrect subnet mask setting on one machine can cause confusion throughout the network.

Normally the configuration files do roughly the following things:

- enable each of the network interfaces (Ethernet interface, serial lines, etc.) Normally this involves specifying an Internet address and subnet mask for each, as well as other options that will be described in your vendor's documentation.

- establish network routing information, either by commands that add fixed routes, or by starting a program that obtains them dynamically.

- turn on the name server (used for looking up names and finding the corresponding Internet address -- see the section on the domain system in the Introduction to TCP/IP).

- set various other information needed by the system software, such as the name of the system itself.

- start various "daemons". These are programs that provide network services to other systems on the network, and to users on this system.

It is not practical to document these steps in detail, since they differ for each vendor. This section will concentrate on a few issues where your choice will depend upon overall decisions about how your network is to operate. These overall network policy decisions are often not as well documented by the vendors as the details of how to start specific programs. Note that some care will be necessary to integrate commands that you add for routing, etc., into the startup sequence at the right point. Some of the most mysterious problems occur when network routing is not set up before a program needs to make a network query, or when a program attempts to look up a host name before the name server has finished loading all of the names from a master name server.

4.1 How datagrams are routed

If your system consists of a single Ethernet or similar medium, you do not need to give routing much attention. However for more complex systems, each of your machines needs a routing table that lists a gateway and interface to use for every possible destination network. A simple example of this was given at the beginning of this section. However it is now necessary to describe the way routing works in a bit more detail. On most systems, the routing table looks something like the following. (This example was taken from a system running Berkeley Unix, using the command "netstat -n -r". Some columns containing statistical information have been omitted.)

```
Destination          Gateway              Flags        Interface

128.6.5.3            128.6.7.1            UHGD         il0
128.6.5.21           128.6.7.1            UHGD         il0
127.0.0.1            127.0.0.1            UH           lo0
128.6.4              128.6.4.61           U            pe0
128.6.6              128.6.7.26           U            il0
128.6.7              128.6.7.26           U            il0
128.6.2              128.6.7.1            UG           il0
10                   128.6.4.27           UG           pe0
128.121              128.6.4.27           UG           pe0
default              128.6.4.27           UG           pe0
```

The example system is connected to two Ethernets:

controller	network	address	other networks
il0	128.6.7	128.6.7.26	128.6.6
pe0	128.6.4	128.6.4.61	none

The first column shows the designation for the controller hardware that connects the computer to that Ethernet. (This system happens to have controllers from two different vendors. The first one is made by Interlan, the second by Pyramid.) The second column is the network number for the network. The third column is this computer's Internet address on that network. The last column shows other subnets that share the same physical network.

Now let's look at the routing table. For the moment, let us ignore the first 3 lines. The majority of the table consists of a set of entries describing networks. For each network, the other three columns show where to send datagrams destined for that network. If the "G" flag is present in the third column, datagrams for that network must be sent through a gateway. The second column shows the address of the gateway to be used. If the "G" flag is not present, the computer is directly connected to the network in question. So datagrams for that network should be sent using the controller shown in the third column. The "U" flag in the third column simply indicates that the route specified by that line is up, i.e. that no errors have occured indicating that the path is unusable.

The first 3 lines show "host routes", indicated by the "H" flag in column three. Routing tables normally have entries for entire networks or subnets. For example, the entry

```
    128.6.2              128.6.7.1            UG         il0
```

indicates that datagrams for any computer on network 128.6.2 (i.e. addresses 128.6.2.1 through 128.6.2.254) should be sent to gateway 128.6.7.1 for forwarding. However sometimes routes apply only to a specific computer, rather than to a whole network. In that case, a host route is used. The first column then shows a complete address, and the "H" flag is present in column 3. E.g. the entry

```
    128.6.5.21           128.6.7.1            UHGD       il0
```

indicates that datagrams for the specific address 128.6.5.21 should be sent to the gateway 128.6.7.1. As with network routes, the "G" flag is used for routes that involve a gateway. The "D" flag indicates that the route was added dynamically, based on an ICMP redirect message from a gateway. (See below for details.)

The following route is special:

```
    127.0.0.1            127.0.0.1            UH         lo0
```

127.0.0.1 is the address of the "loopback device". This is a dummy software module. Any datagram sent out through that "device" appears immediately as input. It can be used for testing. The loopback address is also handy for sending queries to programs that are designed to respond to network queries, but happen to be running on the same computer. (Why bother to use your network to talk to a program that is on the same machine you are?)

Finally, there are "default" routes, e.g.

```
    default              128.6.4.27           UG         pe0
```

This route is used for datagrams that don't match any other entry. In this case, they are sent to a gateway with address 128.6.4.27.

In most systems, datagrams are routed by looking up the destination address in a table such as the one just described. If the address matches a specific host route, then that is used. Otherwise, if it matches a network route, that is used. If no other route works, the default is used. If there is no default, normally the user gets an error message such as "network is unreachable".

The following sections will describe several ways of setting up these routing tables. Generally, the actual operation of sending packets doesn't depend upon which method you use to set up the routes. When a packet is to be sent, its destination is looked up in the table. The different routing methods are simply more and less sophisticated ways of setting up and maintaining the tables.

4.2 Fixed routes

The simplest way of doing routing is to have your configuration contain commands to set up the routing table at startup, and then leave it alone. This method is practical for relatively small networks, particularly if they don't change very often.

Most computers automatically set up some routing entries for you. Unix will add an entry for the networks to which you are directly connected. For example, your startup file might contain the commands

```
      ifconfig ie0 128.6.4.4 netmask 255.255.255.0
      ifconfig ie1 128.6.5.35 netmask 255.255.255.0
```

These specify that there are two network interfaces, and your addresses on them. The system will automatically create routing table entries

```
    128.6.4              128.6.4.4            U           ie0
    128.6.5              128.6.5.35           U           ie1
```

These specify that datagrams for the local subnets, 128.6.4 and 128.6.5, should be sent out the corresponding interface.

In addition to these, your startup files would contain commands to define routes to whatever other networks you wanted to reach. For example,

```
      route add 128.6.2.0 128.6.4.1  1
      route add 128.6.6.0 128.6.5.35 0
```

These commands specify that in order to reach network 128.6.2, a gateway at address 128.6.4.1 should be used, and that network 128.6.6 is actually an additional network number for the physical network connected to interface 128.6.5.35. Some other software might use different commands for these cases. Unix differentiates them by the "metric", which is the number at the end of the command. The metric indicates how many gateways the datagram will have to go through to get to the destination. Routes with metrics of 1 or greater specify the address of the first gateway on the path. Routes with metrics of

0 indicate that no gateway is involved -- this is an additional network number for the local network.

Finally, you might define a default route, to be used for destinations not listed explicitly. This would normally show the address of a gateway that has enough information to handle all possible destinations.

If your network has only one gateway attached to it, then of course all you need is a single entry pointing to it as a default. In that case, you need not worry further about setting up routing on your hosts. (The gateway itself needs more attention, as we will see.) The following sections are intended to provide help for setting up networks where there are several different gateways.

4.3 Routing redirects

Most Internet experts recommend leaving routing decisions to the gateways. That is, it is probably a bad idea to have large fixed routing tables on each computer. The problem is that when something on the network changes, you have to go around to many computers and update the tables. If changes happen because a line goes down, service may not be restored until someone has a chance to notice the problem and change all the routing tables.

The simplest way to keep routes up to date is to depend upon a single gateway to update your routing tables. This gateway should be set as your default. (On Unix, this would mean a command such as "route add default 128.6.4.27 1", where 128.6.4.27 is the address of the gateway.) As described above, your system will send all datagrams to the default when it doesn't have any better route. At first, this strategy does not sound very good if you have more than one gateway. After all, if all you have is a single default entry, how will you ever use the other gateways in the cases where they are better? The answer is that most gateways are able to send "redirects" when they get datagrams for which there is a better route. A redirect is a specific kind of message using the ICMP (Internet Control Message Protocol). It contains information that generally translates to "In the future, to get to address XXXXX, please use gateway YYYYY instead of me". Correct TCP/IP implementations use these redirects to add entries to their routing table. Suppose your routing table starts out as follows:

```
    Destination          Gateway            Flags       Interface

    127.0.0.1            127.0.0.1          UH          lo0
    128.6.4              128.6.4.61         U           pe0
    default              128.6.4.27         UG          pe0
```

This contains an entry for the local network, 128.6.4, and a default pointing to the gateway 128.6.4.27. Suppose there is also a gateway 128.6.4.30, which is the best way to get to network 128.6.7. How do you find it? Suppose you have datagrams to send to 128.6.7.23. The first datagram will go to the default gateway, since that's the only thing in the routing table. However the default gateway, 128.6.4.27, will notice that 128.6.4.30 would really be a better route. (How it does that is up to the gateway. However there are some fairly simple methods for a gateway to determine that you would be better off using a different one.) Thus 128.6.4.27 will send back a redirect specifying that packets for 128.6.7.23 should be sent via 128.6.4.30. Your TCP/IP software will add a routing entry

```
    128.6.7.23            128.6.4.30            UDHG          pe0
```

Any future datagrams for 128.6.7.23 will be sent directly to the appropriate gateway.

This strategy would be a complete solution, if it weren't for three problems:

- It requires each computer to have the address of one gateway "hardwired" into its startup files, as the initial default.

- If a gateway goes down, routing table entries using it may not be removed.

- If your network uses subnets, and your TCP/IP implementation does not handle them, this strategy will not work.

How serious the first problem is depends upon your situation. For small networks, there is no problem modifying startup files whenever something changes. But some organizations can find it very painful. If network topology changes, and a gateway is removed, any systems that have that gateway as their default must be adjusted. This is particularly serious if the people who maintain the network are not the same as those maintaining the individual systems. One simple appoach is to make sure that the default address never changes. For example, you might adopt the convention that address 1 on each subnet is the default gateway for that subnet. For example, on subnet 128.6.7, the default gateway would always be 128.6.7.1. If that gateway is ever removed, some other gateway is given that address. (There must always be at least one gateway left to give it to. If there isn't, you are completely cut off anyway.)

The biggest problem with the description given so far is that it tells you how to add routes but not how to get rid of them. What happens if a gateway goes down? You want traffic to be redirected back to a gateway that is up. Unfortunately, a gateway that has crashed is not going to issue Redirects. One solution is to choose very reliable gateways. If they crash very seldom, this may not be a problem. Note that Redirects can be used to handle some kinds of network failure. If a line goes down, your current route may no longer be a good one. As long as the gateway to which you are talking is still up and talking to you, it can simply issue a Redirect to the gateway that is now the best one. However you still need a way tc detect failure of one of the gateways that you are talking to directly.

The best approach for handling failed gateways is for your TCP/IP implementation to detect routes that have failed. TCP maintains various timers that allow the software to detect when a connection has broken. When this happens, one good approach is to mark the route

down, and go back to the default gateway. A similar approach can also be used to handle failures in the default gateway. If you have mark two gateways as default, then the software should be capable of switching when connections using one of them start failing. Unfortunately, some common TCP/IP implementations do not mark routes as down and change to new ones. (In particular Berkeley 4.2 Unix does not.) However Berkeley 4.3 Unix does do this, and as other vendors begin to base products on 4.3 rather than 4.2, this ability is expected to be more common.

4.4 Other ways for hosts to find routes

As long as your TCP/IP implementations handle failing connections properly, establishing one or more default routes in the configuration file is likely to be the simplest way to handle routing. However there are two other routing approaches that are worth considering for special situations:

- spying on the routing protocol
- using proxy ARP

4.4.1 Spying on Routing

Gateways generally have a special protocol that they use among themselves. Note that redirects cannot be used by gateways. Redirects are simply ways for gateways to tell "dumb" hosts to use a different gateway. The gateways themselves must have a complete picture of the network, and a way to compute the optimal route to each subnet. Generally they maintain this picture by exchanging information among themselves. There are several different routing protocols in use for this purpose. One way for a computer to keep track of gateways is for it to listen to the gateways' messages. There is software available for this purpose for most of the common routing protocols. When you run this software, it maintains a complete picture of the network, just as the gateways do. The software is generally designed to maintain your computer's routing tables dynamically, so that datagrams are always sent to the proper gateway. In effect, the routing software issues the equivalent of the Unix "route add" and "route delete" commands as the network topology changes. Generally this results in a complete routing table, rather than one that depends upon default routes. (This assumes that the gateways themselves maintain a complete table. Sometimes gateways keep track of your campus network completely, but use a default route for all off-campus networks, etc.)

Running routing software on each host does in some sense "solve" the routing problem. However there are several reasons why this is not normally recommended except as a last resort. The most serious problem is that this reintroduces configuration options that must be kept up to date on each host. Any computer that wants to participate in the protocol among the gateways will need to configure its software compatibly with the gateways. Modern gateways often have configuration options that are complex compared with those of an individual host. It is undesirable to spread these to every host.

There is a somewhat more specialized problem that applies only to diskless computers. By its very nature, a diskless computer depends upon the network and file servers to load programs and to do swapping. It is dangerous for diskless computers to run any software that listens to network broadcasts. Routing software generally depends upon broadcasts. For example, each gateway on the network might broadcast its routing tables every 30 seconds. The problem with diskless nodes is that the software to listen to these broadcasts must be loaded over the network. On a busy computer, programs that are not used for a few seconds will be swapped or paged out. When they are activated again, they must be swapped or paged in. Whenever a broadcast is sent, every computer on the network needs to activate the routing software in order to process the broadcast. This means that many diskless computers will be doing swapping or paging at the same time. This is likely to cause a temporary overload of the network. Thus it is very unwise for diskless machines to run any software that requires them to listen to broadcasts.

4.4.2 Proxy ARP

Proxy ARP is an alternative technique for letting gateways make all the routing decisions. It is applicable to any broadcast network that uses ARP or a similar technique for mapping Internet addresses into network-specific addresses such as Ethernet addresses. This presentation will assume Ethernet. Other network types can be acccomodated if you replace "Ethernet address" with the appropriate network-specific address, and ARP with the protocol used for address mapping by that network type.

In many ways proxy ARP it is similar to using a default route and redirects, however it uses a different mechanism to communicate routes to the host. With redirects, a full routing table is used. At any given moment, the host knows what gateways it is routing datagrams to. With proxy ARP, you dispense with explicit routing tables, and do everything at the level of Ethernet addresses. Proxy ARP can be used for all destinations, only for destinations within your network, or in various combinations. It will be simplest to explain it as used for all addresses. To do this, you instruct the host to pretend that every computer in the world is attached directly to your local Ethernet. On Unix, this would be done using a command

```
route add default 128.6.4.2 0
```

where 128.6.4.2 is assumed to be the Internet address of your host. As explained above, the metric of 0 causes everything that matches this route to be sent directly on the local Ethernet.

When a datagram is to be sent to a local Ethernet destination, your computer needs to know the Ethernet address of the destination. In order to find that, it uses something generally called the ARP table. This is simply a mapping from Internet address to Ethernet address. Here's a typical ARP table. (On our system, it is displayed using the command "arp -a".)

```
    FOKKER.RUTGERS.EDU (128.6.5.16) at 8:0:20:0:8:22 temporary
    CROSBY.RUTGERS.EDU (128.6.5.48) at 2:60:8c:49:50:63 temporary
    CAIP.RUTGERS.EDU (128.6.4.16) at 8:0:8b:0:1:6f temporary
    DUDE.RUTGERS.EDU (128.6.20.16) at 2:7:1:0:eb:cd temporary
    W20NS.MIT.EDU (18.70.0.160) at 2:7:1:0:eb:cd temporary
    OBERON.USC.EDU (128.125.1.1) at 2:7:1:2:18:ee temporary
    gatech.edu (128.61.1.1) at 2:7:1:0:eb:cd temporary
    DARTAGNAN.RUTGERS.EDU (128.6.5.65) at 8:0:20:0:15:a9 temporary
```

Note that it is simply a list of Internet addresses and the corresponding Ethernet address. The "temporary" indicates that the entry was added dynamically using ARP, rather than being put into the table manually.

If there is an entry for the address in the ARP table, the datagram is simply put on the Ethernet with the corresponding Ethernet address. If not, an "ARP request" is broadcast, asking for the destination host to identify itself. This request is in effect a question "will the host with Internet address 128.6.4.194 please tell me what your Ethernet address is?". When a response comes back, it is added to the ARP table, and future datagrams for that destination can be sent without delay.

This mechanism was originally designed only for use with hosts attached directly to a single Ethernet. If you need to talk to a host on a different Ethernet, it was assumed that your routing table would direct you to a gateway. The gateway would of course have one interface on your Ethernet. Your computer would then end up looking up the address of that gateway using ARP. It would generally be useless to expect ARP to work directly with a computer on a distant network. Since it isn't on the same Ethernet, there's no Ethernet address you can use to send datagrams to it. And when you send an ARP request for it, there's nobody to answer the request.

Proxy ARP is based on the concept that the gateways will act as proxies for distant hosts. Suppose you have a host on network 128.6.5, with address 128.6.5.2. (computer A in diagram below) It wants to send a datagram to host 128.6.4.194, which is on a different Ethernet (subnet 128.6.4). (computer C in diagram below) There is a gateway connecting the two subnets, with address 128.6.5.1 (gateway R):

```
          network 1                  network 2
           128.6.5                    128.6.4
    ============================    ==================
          |              |        |    |      |    |
       ___|_______  _____|___  ___|____|__  __|____|____
    128.6.5.2     128.6.5.3  128.6.5.1   128.6.4.194
                             128.6.4.1
    __________    __________  __________  ____________
    computer A    computer B  gateway R   computer C
```

Now suppose computer A sends an ARP request for computer C. C isn't able to answer for itself. It's on a different network, and never even sees the ARP request. However gateway R can act on its behalf. In effect, your computer asks "will the host with Internet address 128.6.4.194 please tell me what your Ethernet address is?", and the gateway says "here I am, 128.6.4.194 is 2:7:1:0:eb:cd", where 2:7:1:0:eb:cd is actually the Ethernet address of the gateway. This bit of illusion works just fine. Your host now thinks that 128.6.4.194 is attached to the local Ethernet with address 2:7:1:0:eb:cd. Of course it isn't. But it works anyway. Whenever there's a datagram to be sent to 128.6.4.194, your host sends it to the specified Ethernet address. Since that's the address of a gateway R, the gateway gets the packet. It then forwards it to the destination.

Note that the net effect is exactly the same as having an entry in the routing table saying to route destination 128.6.4.194 to gateway 128.6.5.1:

```
    128.6.4.194          128.6.5.1          UGH          pe0
```

except that instead of having the routing done at the level of the routing table, it is done at the level of the ARP table.

Generally it's better to use the routing table. That's what it's there for. However here are some cases where proxy ARP makes sense:

- when you have a host that does not implement subnets

- when you have a host that does not respond properly to redirects

- when you do not want to have to choose a specific default gateway

- when your software is unable to recover from a failed route

The technique was first designed to handle hosts that do not support subnets. Suppose that you have a subnetted network. For example, you have chosen to break network 128.6 into subnets, so that 128.6.4 and 128.6.5 are separate. Suppose you have a computer that does not understand subnets. It will assume that all of 128.6 is a single network. Thus it will be difficult to establish routing table entries to handle the configuration above. You can't tell it about the gateway explicitly using "route add 128.6.4.0 128.6.5.1 1" Since it thinks all of 128.6 is a single network, it can't understand that you are trying to tell it where to send one subnet. It will instead interpret this command as an attempt to set up a host route to a host who address is 128.6.4.0. The only thing that would work would be to establish explicit host routes for every individual host on every other subnet. You can't depend upon default gateways and redirects in this situation either. Suppose you said "route add default 128.6.5.1 1". This would establish the gateway 128.6.5.1 as a default. However the system wouldn't use it to send packets to other subnets. Suppose the host is 128.6.5.2, and wants to send a datagram to 128.6.4.194. Since the destination is part of 128.6, your computer considers it to be on the same network as itself, and doesn't bother to look for a gateway.

Proxy ARP solves this problem by making the world look the way the defective implementation expects it to look. Since the host thinks all other subnets are part of its own network, it will simply issue ARP requests for them. It expects to get back an Ethernet address that can be used to establish direct communications. If the gateway is practicing proxy ARP, it will respond with the gateway's Ethernet address. Thus datagrams are sent to the gateway, and everything works.

As you can see, no specific configuration is need to use proxy ARP with a host that doesn't understand subnets. All you need is for your gateways to implement proxy ARP. In order to use it for other purposes, you must explicitly set up the routing table to cause ARP to be used. By default, TCP/IP implementations will expect to find a gateway for any destination that is on a different network. In order to make them issue ARP's, you must explicitly install a route with metric 0, as in the example "route add default 128.6.5.2 0".

It is obvious that proxy ARP is reasonable in situations where you have hosts that don't understand subnets. Some comments may be needed on the other situations. Generally TCP/IP implementations do handle ICMP redirects properly. Thus it is normally practical to set up a default route to some gateway, and depend upon the gateway to issue redirects for destinations that should use a different gateway. However in case you ever run into an implementation that does not obey redirects, or cannot be configured to have a default gateway, you may be able to make things work by depending upon proxy ARP. Of course this requires that you be able to configure the host to issue ARP's for all destinations. You will need to read the documentation carefully to see exactly what routing features your implementation has.

Sometimes you may choose to depend upon proxy ARP for convenience. The problem with routing tables is that you have to configure them. The simplest configuration is simply to establish a default route, but even there you have to supply some equivalent to the Unix command "route add default ...". Should you change the addresses of your gateways, you have to modify this command on all of your hosts, so that they point to the new default gateway. If you set up a default route that depends upon proxy ARP (i.e. has metric 0), you won't have to change your configuration files when gateways change. With proxy ARP, no gateway addresses are given explicitly. Any gateway can respond to the ARP request, no matter what its address.

In order to save you from having to do configuration, some TCP/IP implementations default to using ARP when they have no other route. The most flexible implementations allow you to mix strategies. That is, if you have specified a route for a particular network, or a default route, they will use that route. But if there is no route for a destination, they will treat it as local, and issue an ARP request. As long as your gateways support proxy ARP, this allows such hosts to reach any destination without any need for routing tables.

Finally, you may choose to use proxy ARP because it provides better recovery from failure. This choice is very much dependent upon your implementation. The next section will discuss the tradeoffs in more detail.

In situations where there are several gateways attached to your network, you may wonder how proxy ARP allows you to choose the best one. As described above, your computer simply sends a broadcast asking for the Ethernet address for a destination. We assumed that the gateways would be set up to respond to this broadcast. If there is more than one gateway, this requires coordination among them. Ideally, the gateways will have a complete picture of the network topology. Thus they are able to determine the best route from your host to any destination. If the gateway coordinate among themselves, it should be possible for the best gateway to respond to your ARP request. In practice, it may not always be possible for this to happen. It is fairly easy to design algorithms to prevent very bad routes. For example, consider the following situation:

```
        1              2            3
     -------  A  ----------  B ----------
```

1, 2, and 3 are networks. A and B are gateways, connecting network 2 to 1 or 3. If a host on network 2 wants to talk to a host on network 1, it is fairly easy for gateway A to decide to answer, and for gateway B to decide not to. Here's how: if gateway B accepted a datagram for network 1, it would have to forward it to gateway A for delivery. This would mean that it would take a packet from network 2 and send it right back out on network 2. It is very easy to test for routes that involve this sort of circularity. It is much harder to deal with a situation such as the following:

```
                 1
          ---------------
            A        B
            |        | 4
            |        |
          3 |        C
            |        |
            |        | 5
            D        E
          ---------------
                 2
```

Suppose a computer on network 1 wants to send a datagram to one on network 2. The route via A and D is probably better, because it goes through only one intermediate network (3). It is also possible to go via B, C, and E, but that path is probably slightly slower. Now suppose the computer on network 1 sends an ARP request for a destination on 2. It is likely that A and B will both respond to that request. B is not quite as good a route as A. However it is not so bad as the case above. B won't have to send the datagram right back out onto network 1. It is unable to determine there is a better alternative route without doing a significant amount of global analysis on the network. This may not be practical in the amount of time available to process an ARP request.

4.4.3 Moving to New Routes After Failures

In principle, TCP/IP routing is capable of handling line failures and gateway crashes. There are various mechanisms to adjust routing tables and ARP tables to keep them up to date. Unfortunately, many major implementations of TCP/IP have not implemented all of these mechanisms. The net result is that you have to look carefully at the documentation for your implementation, and consider what kinds of failures are most likely. You then have to choose a strategy that will work best for your site. The basic choices for finding routes have all been listed above: spying on the gateways' routing protocol, setting up a default route and depending upon redirects, and using proxy ARP. These methods all have their own limitations in dealing with a changing network.

Spying on the gateways' routing protocol is theoretically the cleanest solution. Assuming that the gateways use good routing technology, the tables that they broadcast contain enough information to maintain optimal routes to all destinations. Should something in the network change (a line or a gateway goes down), this information will be reflected in the tables, and the routing software will be able to update the hosts' routing tables appropriately. The disadvantages are entirely practical. However in some situations the robustness of this approach may outweight the disadvantages. To summarize the discussion above, the disadvantages are:

- If the gateways are using sophisticated routing protocols, configuration may be fairly complex. Thus you will be faced with setting up and maintaining configuration files on every host.

- Some gateways use proprietary routing protocols. In this case, you may not be able to find software for your hosts that understands them.

- If your hosts are diskless, there can be very serious performance problems associated with listening to routing broadcasts.

Some gateways may be able to convert from their internal routing protocol to a simpler one for use by your hosts. This could largely bypass the first two disadvantages. Currently there is no known way to get around the third one.

The problems with default routes/redirects and with proxy ARP are similar: they both have trouble dealing with situations where their table entries no longer apply. The only real difference is that different tables are involved. Suppose a gateway goes down. If any of your current routes are using that gateway, you may be in trouble. If you are depending upon the routing table, the major mechanism for adjusting routes is the redirect. This works fine in two situations:

- where the default gateway is not the best route. The default gateway can direct you to a better gateway
- where a distant line or gateway fails. If this changes the best route, the current gateway can redirect you to the gateway that is now best

The case it does not protect you against is where the gateway that you are currently sending your datagrams to crashes. Since it is down, it is unable to redirect you to another gateway. In many cases, you are also unprotected if your default gateway goes down, since there routing starts by sending to the default gateway.

The situation with proxy ARP is similar. If the gateways coordinate themselves properly, the right one will respond initially. If something elsewhere in the network changes, the gateway you are currently issuing can issue a redirect to a new gateway that is better. (It is usually possible to use redirects to override routes established by proxy ARP.) Again, the case you are not protected against is where the gateway you are currently using crashes. There is no equivalent to failure of a default gateway, since any gateway can respond to the ARP request.

So the big problem is that failure of a gateway you are using is hard to recover from. It's hard because the main mechanism for changing routes is the redirect, and a gateway that is down can't issue redirects. Ideally, this problem should be handled by your TCP/IP implementation, using timeouts. If a computer stops getting response, it should cancel the existing route, and try to establish a new one. Where you are using a default route, this means that the TCP/IP implementation must be able to declare a route as down based on a timeout. If you have been redirected to a non-default gateway, and that route is declared down, traffic will return to the default. The default gateway can then begin handling the traffic, or redirect it to a different gateway. To handle failure of a default gateway, it should be possible to have more than one default. If one is declared down, another will be used. Together, these mechanisms should take care of any failure.

Similar mechanisms can be used by systems that depend upon proxy ARP. If a connection is timing out, the ARP table entry that it uses should be cleared. This will cause a new ARP request, which can be handled by a gateway that is still up. A simpler mechanism would simply be to time out all ARP entries after some period. Since making a new ARP request has a very low overhead, there's no problem with removing an ARP entry even if it is still good. The next time a datagram is to be sent, a new request will be made. The response is normally fast enough that users will not even notice the delay.

Unfortunately, many common implementations do not use these strategies. In Berkeley 4.2, there is no automatic way of getting rid of any kind of entry, either routing or ARP. They do not invalidate routes on timeout nor ARP entries. ARP entries last forever. If gateway crashes are a significant problem, there may be no choice but to run software that listens to the routing protocol. In Berkeley 4.3, routing entries are removed when TCP connections are failing. ARP entries are still not removed. This makes the default route strategy more attractive for 4.3 than proxy ARP. Having more than one default route may also allow for recovery from failure of a default gateway. Note however that 4.3 only handles timeout for connections using TCP. If a route is being used only by services based on UDP, it will not recover from gateway failure. While the "traditional" TCP/IP services use TCP, network file systems generally do not. Thus 4.3-based systems still may not always be able to recover from failure.

In general, you should examine your implementation in detail to determine what sort of error recovery strategy it uses. We hope that the discussion in this section will then help you choose the best way of dealing with routing.

There is one more strategy that some older implementations use. It is strongly discouraged, but we mention it here so you can recognize it if you see it. Some implementations detect gateway failure by taking active measure to see what gateways are up. The best version of this is based on a list of all gateways that are currently in use. (This can be determined from the routing table.) Every minute or so, an echo request datagram is sent to each such gateway. If a gateway stops responding to echo requests, it is declared down, and all routes using it revert to the default. With such an implementation, you normally supply more than one default gateway. If the current default stops responding, an alternate is chosen. In some cases, it is not even necessary to choose an explicit default gateway. The software will randomly choose any gateway that is responding. This implementation is very flexible and recovers well from failures. However a large network full of such implementations will waste a lot of bandwidth on the echo datagrams that are used to test whether gateways are up. This is the reason that this strategy is discouraged.

5. Bridges and Gateways

This section will deal in more detail with the technology used to construct larger networks. It will focus particularly on how to connect together multiple Ethernets, token rings, etc. These days most networks are hierarchical. Individual hosts attach to local-area networks such as Ethernet or token ring. Then those local networks are connected via some combination of backbone networks and point to point links. A university might have a network that looks in part like this:

```
 _______________________________
|  net 1      net 2    net 3  |        net 4          net 5
| ---------X---------X-------- |       --------        --------
|                          |   |          |               |
|  Building A              |   |          |               |
|           ----------X--------------X-----------------X
|                              | campus backbone network  :
|______________________________|                          :
                                                   serial :
                                                     line :
                                                   -------X-----
                                                      net  6
```

Nets 1, 2 and 3 are in one building. Nets 4 and 5 are in different buildings on the same campus. Net 6 is in a somewhat more distant location. The diagram above shows nets 1, 2, and 3 being connected directly, with switches that handle the connections being labelled as "X". Building A is connected to the other buildings on the same campus by a backbone network. Note that traffic from net 1 to net 5 takes the following path:

- from 1 to 2 via the direct connection between those networks

- from 2 to 3 via another direct connection

- from 3 to the backbone network

- across the backbone network from building A to the building in which net 5 is housed

- from the backbone network to net 5

Traffic for net 6 would additionally pass over a serial line. With the setup as shown, the same switch is being used to connect the backbone network to net 5 and to the serial line. Thus traffic from net 5 to net 6 would not need to go through the backbone, since there is a direct connection from net 5 to the serial line.

This section is largely about what goes in those "X"'s.

5.1 Alternative Designs

Note that there are alternatives to the sort of design shown above. One is to use point to point lines or switched lines directly to each host. Another is to use a single-level of network technology that is capable of handling both local and long-haul networking.

5.1.1 A mesh of point to point lines

Rather than connecting hosts to a local network such as Ethernet, and then interconnecting the Ethernets, it is possible to connect long-haul serial lines directly to the individual computers. If your network consists primarily of individual computers at distant locations, this might make sense. Here would be a small design of that type.

```
          computer 1                computer 2              computer 3
              |                         |                       |
              |                         |                       |
              |                         |                       |
          computer 4 ------------- computer 5 ----------- computer 6
```

In the design shown earlier, the task of routing datagrams around the network is handled by special-purpose switching units shown as "X"'s. If you run lines directly between pairs of hosts, your hosts will be doing this sort of routing and switching, as well as their normal computing. Unless you run lines directly between every pair of computers, some systems will end up handling traffic for others. For example, in this design, traffic from 1 to 3 will go through 4, 5 and 6. This is certainly possible, since most TCP/IP implementations are capable of forwarding datagrams. If your network is of this type, you should think of your hosts as also acting as gateways. Much of the discussion below on configuring gateways will apply to the routing software that you run on your hosts. This sort of configuration is not as common as it used to be, for two reasons:

- Most large networks have more than one computer per location. In this case it is less expensive to set up a local network at each location than to run point to point lines to each computer.

- Special-purpose switching units have become less expensive. It often makes sense to offload the routing and communications tasks to a switch rather than handling it on the hosts.

It is of course possible to have a network that mixes the two kinds of techology. In this case, locations with more equipment would be handled by a hierarchical system, with local-area networks connected by switches. Remote locations with a single computer would be handled by point to point lines going directly to those computers. In this case the routing software used on the remote computers would have to be compatible with that used by the switches, or there would need to be a gateway between the two parts of the network.

Design decisions of this type are typically made after an assessment of the level of network traffic, the complexity of the network, the quality of routing software available for the hosts, and the ability of the hosts to handle extra network traffic.

5.1.2 Circuit switching technology

Another alternative to the hierarchical LAN/backbone approach is to use circuit switches connected to each individual computer. This is really a variant of the point to point line technique, where the circuit switch allows each system to have what amounts to a direct line to every other system. This technology is not widely used within the TCP/IP community, largely because the TCP/IP protocols assume that the lowest level handles isolated datagrams. When a continuous connection is needed, higher network layers maintain it using datagrams. This datagram-oriented technology does not match a circuit-oriented environment very closely. In order to use circuit switching technology, the IP software must be modified to be able to build and tear down virtual circuits as appropriate. When there is a datagram for a given destination, a virtual circuit must be opened to it. The virtual circuit would be closed when there has been no traffic to that destination for some time. The major use of this technology is for the DDN (Defense Data Network). The primary interface to the DDN is based on X.25. This network appears to the outside as a distributed X.25 network. TCP/IP software intended for use with the DDN must do precisely the virtual circuit management just described. Similar techniques could be used with other circuit-switching technologies, e.g. ATT's DataKit, although there is almost no software currently available to support this.

5.1.3 Single-level networks

In some cases new developments in wide-area networks can eliminate the need for hierarchical networks. Early hierarchical networks were set up because the only convenient network technology was Ethernet or other LAN's, and those could not span distances large enough to cover an entire campus. Thus it was necessary to use serial lines to connect LAN's in various locations. It is now possible to find network technology whose characteristics are similar to Ethernet, but where a single network can span a campus. Thus it is possible to think of using a single large network, with no hierarchical structure.

The primary limitations of a large single-level network are performance and reliability considerations. If a single network is used for the entire campus, it is very easy to overload it. Hierarchical networks can handle a larger traffic volume than single-level networks if traffic patterns have a reasonable amount of locality. That is, in many applications, traffic within an individual department tends to be greater than traffic among departments.

Let's look at a concrete example. Suppose there are 10 departments, each of which generate 1 Mbit/sec of traffic. Suppose futher than 90% of that traffic is to other systems within the department, and only 10% is to other departments. If each department has its own network, that network only needs to handle 1 Mbit/sec. The backbone network connecting the department also only needs 1 Mbit/sec capacity, since it is handling 10% of 1 Mbit from each department.. In order to handle this situation with a single wide-area network, that network would have to be able to handle the simultaneous load from all 10 departments, which would be 10 Mbit/sec.

The second limitation on single-level networks is reliability, maintainability and security. Wide-area networks are more difficult to diagnose and maintain than local-area networks, because problems can be introduced from any building to which the network is connected. They also make traffic visible in all locations. For these reasons, it is often sensible to handle local traffic locally, and use the wide-area network only for traffic that actually must go between buildings. However if you have a situation where each location has only one or two computers, it may not make sense to set up a local network at each location, and a single-level network may make sense.

5.1.4 Mixed designs

In practice, few large networks have the luxury of adopting a theoretically pure design.

It is very unlikely that any large network will be able to avoid using a hierarchical design. Suppose we set out to use a single-level network. Even if most buildings have only one or two computers, there will be some location where there are enough that a local-area network is justified. The result is a mixture of a single-level network and a hierachical network. Most buildings have their computers connected directly to the wide-area network, as with a single-level network. However in one building there is a local-area network which uses the wide-area network as a backbone, connecting to it via a switching unit.

On the other side of the story, even network designers with a strong commitment to hierarchical networks are likely to find some parts of the network where it simply doesn't make economic sense to install a local-area network. So a host is put directly onto the backbone network, or tied directly to a serial line.

However you should think carefully before making ad hoc departures from your design philosophy in order to save a few dollars. In the long run, network maintainability is going to depend upon your ability to make sense of what is going on in the network. The more consistent your technology is, the more likely you are to be able to maintain the network.

5.2 An introduction to alternative switching technologies

This section will discuss the characteristics of various technologies used to switch datagrams between networks. In effect, we are trying to fill in some details about the black boxes assumed in previous sections. There are three basic types of switches, generally referred to as repeaters, bridges, and gateways, or alternatively as level 1, 2 and 3 switches (based on the level of the ISO model at which they operate). Note however that there are systems that combine features of more than one of these, particularly bridges and gateways.

The most important dimensions on which switches vary are isolation, performance, routing and network management facilities. These will be discussed below.

The most serious difference is between repeaters and the other two types of switch. Until recently, gateways provided very different services from bridges. However these two technologies are now coming closer together. Gateways are beginning to adopt the special-purpose hardware that has characterized bridges in the past. Bridges are beginning to adopt more sophisticated routing, isolation features, and network management, which have characterized gateways in the past. There are also systems that can function as both bridge and gateway. This means that at the moment, the crucial decision may not be to decide whether to use a bridge or a gateway, but to decide what features you want in a switch and how it fits into your overall network design.

5.2.1 Repeaters

A repeater is a piece of equipment that connects two networks that use the same technology. It receives every data packet on each network, and retransmits it onto the other network. The net result is that the two networks have exactly the same set of packets on them. For Ethernet or IEEE 802.3 networks there are actually two different kinds of repeater. (Other network technologies may not need to make this distinction.)

A simple repeater operates at a very low level indeed. Its primary purpose is to get around limitations in cable length caused by signal loss or timing dispersion. It allows you to construct somewhat larger networks than you would otherwise be able to construct. It can be thought of as simply a two-way amplifier. It passes on individual bits in the signal, without doing any processing at the packet level. It even passes on collisions. That is, if a collision is generated on one of the networks connected to it, the repeater generates a collision on the other network. There is a limit to the number of repeaters that you can use in a network. The basic Ethernet design requires that signals must be able to get from one end of the network to the other within a specified amount of time. This determines a maximum allowable length. Putting repeaters in the path does not get around this limit. (Indeed each repeater adds some delay, so in some ways a repeater makes things worse.) Thus the Ethernet configuration rules limit the number of repeaters that can be in any path.

A "buffered repeater" operates at the level of whole data packets. Rather than passing on signals a bit at a time, it receives an entire packet from one network into an internal buffer and then retransmits it onto the other network. It does not pass on collisions. Because such low-level features as collisions are not repeated, the two networks continue to be separate as far as the Ethernet specifications are concerned. Thus there are no restrictions on the number of buffered repeaters that can be used. Indeed there is no requirement that both of the networks be of the same type. However the two networks must be sufficiently similar that they have the same packet format. Generally this means that buffered repeaters can be used between two networks of the IEEE 802.x family (assuming that they have chosen the same address length), or two networks of some other related family. A pair of buffered repeaters can be used to connect two networks via a serial line.

Buffered repeaters share with simple repeaters the most basic feature: they repeat every data packet that they receive from one network onto the other. Thus the two networks end up with exactly the same set of packets on them.

5.2.2 Bridges and gateways

A bridge differs from a buffered repeater primarily in the fact that it exercizes some selectivity as to what packets it forwards between networks. Generally the goal is to increase the capacity of the system by keeping local traffic confined to the network on which it originates. Only traffic intended for the other network (or some other network accessed through it) goes through the bridge. So far this description would also apply to a gateway. Bridges and gateways differ in the way they determine what packets to forward. A bridge uses only the ISO level 2 address. In the case of Ethernet or IEEE 802.x networks, this is the 6-byte Ethernet or MAC-level address. (The term MAC-level address is more general. However for the sake of concreteness, examples in this section will assume that Ethernet is being used. You may generally replace the term "Ethernet address" with the equivalent MAC-level address for other similar technologies.) A bridge does not examine the packet itself, so it does not use the IP address or its equivalent for routing decisions. In contrast, a gateway bases its decisions on the IP address, or its equivalent for other protocols.

There are several reasons why it matters which kind of address is used for decisions. The most basic is that it affects the relationship between the switch and the upper layers of the protocol. If forwarding is done at the level of the MAC-level address (bridge), the switch will be invisible to the protocols. If it is done at the IP level, the switch will be visible. Let's give an example. Here are two networks connected by a bridge:

```
               network 1           network 2
                128.6.5             128.6.4
          ==================  ================================
               |            |       |        |             |
           ____|______    __|_______|__   ___|___      ____|___
          128.6.5.2         bridge       128.6.4.3     128.6.4.4
          __________     ____________   __________    __________
          computer A                    computer B    computer C
```

Note that the bridge does not have an IP address. As far as computers A, B, and C are concerned, there is a single Ethernet (or other network) to which they are all attached. This means that the routing tables must be set up so that computers on both networks treat both networks as local. When computer A opens a connection to computer B, it first broadcasts an ARP request asking for computer B's Ethernet address. The bridge must pass this broadcast from network 1 to network 2. (In general, bridges must pass all broadcasts.) Once the two computers know each other's Ethernet addresses, communications use the Ethernet address as the destination. At that point, the bridge can start exerting some selectivity. It will only pass packets whose Ethernet destination address is for a machine on the other network. Thus a packet from B to A will be passed from network 2 to 1, but a packet from B to C will be ignored.

In order to make this selection, the bridge needs to know which network each machine is on. Most modern bridges build up a table for each network, listing the Ethernet addresses of machines known to be on that network. They do this by watching all of the packets on both networks. When a packet first appears on network 1, it is reasonable to conclude that the Ethernet source address corresponds to a machine on network 1.

Note that a bridge must look at every packet on the Ethernet, for two different reasons. First, it may use the source address to learn which machines are on which network. Second, it must look at the destination address in order to decide whether it needs to forward the packet to the other network.

As mentioned above, generally bridges must pass broadcasts from one network to the other. Broadcasts are often used to locate a resource. The ARP request is a typical example of this. Since the bridge has no way of knowing what host is going to answer the broadcast, it must pass it on to the other network. Some newer bridges have user-selectable filters. With them, it is possible to block some broadcasts and allow others. You might allow ARP broadcasts (which are essential for IP to function), but confine less essential broadcasts to one network. For example, you might choose not to pass rwhod broadcasts, which some systems use to keep track of every user logged into every other system. You might decide that it is sufficient for rwhod to know about the systems on a single segment of the network.

Now let's take a look at two networks connected by a gateway

```
          network 1                     network 2
           128.6.5                       128.6.4
   ====================          =====================================
         |              |          |            |              |
      ___|______    ____|__________|____   _____|___      _____|___
     128.6.5.2    128.6.5.1  128.6.4.1    128.6.4.3      128.6.4.4
     _________    ____________________    _________      _________
     computer A          gateway          computer B     computer C
```

Note that the gateway has IP addresses assigned to each interface. The computers' routing tables are set up to forward through appropriate address. For example, computer A has a routing entry saying that it should use the gateway 128.6.5.1 to get to subnet 128.6.4.

Because the computers know about the gateway, the gateway does not need to scan all the packets on the Ethernet. The computers will send packets to it when appropriate. For example, suppose computer A needs to send a message to computer B. Its routing table will tell it to use gateway 128.6.5.1. It will issue an ARP request for that address. The gateway will respond to the ARP request, just as any host would. >From then on, packets destinated for B will be sent with the gateway's Ethernet address.

5.2.3 More about bridges

There are several advantages to using the Mac-level address, as a bridge does. First, every packet on an Ethernet or IEEE network has such an address. The address is in the same place for every packet, whether it is IP, DECnet, or some other protocol. Thus it is relatively fast to get the address from the packet. A gateway must decode the entire IP header, and if it is to support protocols other than IP, it must have software for each such protocol. This means that a bridge automatically supports every possible protocol, whereas a gateway requires specific provisions for each protocol it is to support.

However there are also disadvantages. The one that is intrinsic to the design of a bridge is

- A bridge must look at every packet on the network, not just those addressed to it. Thus it is possible to overload a bridge by putting it on a very busy network, even if very little traffic is actually going through the bridge.

However there are another set of disadvantages that are based on the way bridges are usually built. It is possible in principle to design bridges that do not have these disadvantages, but I don't know of any plans to do so. They all stem from the fact that bridges do not have a complete routing table that describes the entire system of networks.

They simply have a list of the Ethernet addresses that lie on each of its two networks. This means

- A bridge can handle only two network interfaces. At a central site, where many networks converge, this normally means that you set up a backbone network to which all the bridges connect, and then buy a separate bridge to connect each other network to that backbone. Gateways often have between 4 and 8 interfaces.

- Networks that use bridges cannot have loops in them. If there were a loop, some bridges would see traffic from the same Ethernet address coming from both directions, and would be unable to decide which table to put that address in. Note that any parallel paths to the same direction constitute a loop. This means that multiple paths cannot be used for purposes of splitting the load or providing redundancy.

There are some ways of getting around the problem of loops. Many bridges allow configurations with redundant connections, but turn off links until there are no loops left. Should a link fail, one of the disabled ones is then brought back into service. Thus redundant links can still buy you extra reliability. But they can't be used to provide extra capacity. It is also possible to build a bridge that will make use of parallel point to point lines, in the one special case where those lines go between a single pair of bridges. The bridges would treat the two lines as a single virtual line, and use them alternately in round-robin fashion.

The process of disabling redundant connections until there are no loops left is called a "spanning tree algorithm". This name comes from the fact that a tree is defined as a pattern of connections with no loops. Thus one wants to disable connections until the connections that are left form a tree that "spans" (includes) all of the networks in the system. In order to do this, all of the bridges in a network system must communicate among themselves. There is an IEEE proposal to standardize the protocol for doing this, and for constructing the spanning tree.

Note that there is a tendency for the resulting spanning tree to result in high network loads on certain parts of the system. The networks near the "top of the tree" handle all traffic between distant parts of the network. In a network that uses gateways, it would be possible to put in an extra link between parts of the network that have heavy traffic between them. However such extra links cannot be used by a set of bridges.

5.2.4 More about gateways

Gateways have their own advantages and disadvantages. In general a gateway is more complex to design and to administer than a bridge. A gateway must participate in all of the protocols that it is designed to forward. For example, an IP gateway must respond to ARP requests. The IP standards also require it to completely process the IP header, decrementing the time to live field and obeying any IP options.

Gateways are designed to handle more complex network topologies than bridges. As such, they have a different (and more complex) set of decisions to make. In general a bridge has only a binary decision to make: does it or does it not pass a given packet from one network to the other? However a gateway may have several network interfaces. Furthermore, when it forwards a packet, it must decide what host or gateway to send the packet to next. It is even possible for a gateway to decide to send a packet back onto the same network it came from. If a host is using the gateway as its default, it may send packets that really should go to some other gateway. In that case, the gateway will send the packet on to the right gateway, and send back an ICMP redirect to the host. Many gateways can also handle parallel paths. If there are several equally good paths to a destination, the gateway will alternate among them in round-robin fashion.

In order to handle these decisions, a gateway will typically have a routing table that looks very much like a host's. As with host routing tables, a gateway's table contains an entry for each possible network number. For each network, there is either an entry saying that that network is connected directly to the gateway, or there is an entry saying that traffic for that network should be forwarded through some other gateway or gateways. We will describe the "routing protocols" used to build up this information later, in the discussion on how to configure a gateway.

5.3 Comparing the switching technologies

Repeaters, buffered repeaters, bridges, and gateways form a spectrum. Those devices near the beginning of the list are best for smaller networks. They are less expensive, and easier to set up, but less general. Those near the end of the list are suitable for building more complex networks. Many networks will contain a mixture of switch types, with repeaters being used to connect a few nearby network segments, bridges used for slightly larger areas (particularly those with low traffic levels), and gateways used for long-distance links.

Note that this document so far has assumed that only gateways are being used. The section on setting up a host described how to set up a routing table listing the gateways to use to get to various networks. Repeaters and bridges are invisible to IP. So as far as previous sections are concerned, networks connected by them are to be considered a single network. Section 3.3.1 describes how to configure a host in the case where several subnets are carried on a single physical network. The same configuration should be used when several subnets are connected by repeaters or bridges.

As mentioned above, the most important dimensions on which switches vary are isolation, performance, routing, network management, and performing auxilliary support services.

5.3.1 Isolation

Generally people use switches to connect networks to each other. So they are normally thinking of gaining connectivity, not providing isolation. However isolation is worth thinking about. If you connect two networks and provide no isolation at all, then any network problems on other networks suddenly appear on yours as well. Also, the two networks together may have enough traffic to overwhelm your network. Thus it is well to think of choosing an appropriate level of protection.

Isolation comes in two kinds: isolation against malfunctions and traffic isolation. In order to discuss isolation of malfunctions, we have to have a taxonomy of malfunctions. Here are the major classes of malfunctions, and which switches can isolate them:

- Electrical faults, e.g. a short in the cable or some sort of fault that distorts the signal. All types of switch will confine this to one side of the switch: repeater, buffered repeater, bridge, gateway. These are worth protecting against, although their frequency depends upon how often your cables are changed or disturbed. It is rare for this sort of fault to occur without some disturbance of the cable.

- Transceiver and controller problems that general signals that are valid electrically but nevertheless incorrect (e.g. a continuous, infinitely long packet, spurious collisions, never dropping carrier). All except the simple repeater will confine this: buffered repeater, bridge, gateway. (Such problems are not very common.)

- Software malfunctions that lead to excessive traffic between particular hosts (i.e. not broadcasts). Bridges and gateways will isolate these. (This type of failure is fairly rare. Most software and protocol problems generate broadcasts.)

- Software malfunctions that lead to excessive broadcast traffic. Gateways will isolate these. Generally bridges will not, because they must pass broadcasts. Bridges with user-settable filtering can protect against some broadcast malfunctions. However in general bridges must pass ARP, and most broadcast malfunctions involve ARP. This problem is not severe on single-vendor networks where software is under careful control. However research sites generally see problems of this sort regularly.

Traffic isolation is provided by bridges and gateways. The most basic decision is how many computers can be put onto a network without overloading its capacity. This requires knowledge of the capacity of the network, but also how the hosts will use it. For example, an Ethernet may support hundreds of systems if all the network is used for is remote logins and an occasional file transfer. However if the computers are diskless, and use the network for swapping, an Ethernet will support between 10 and 40, depending upon their speeds and I/O rates.

When you have to put more computers onto a network than it can handle, you split it into several networks and put some sort of switch between them. If you do the split correctly, most of the traffic will be between machines on the same piece. This means putting clients on the same network as their servers, putting terminal servers on the same network as the hosts that they access most commonly, etc.

Bridges and gateways generally provide similar degrees of traffic isolation. In both cases, only traffic bound for hosts on the other side of the switch is passed. However see the discussion on routing.

5.3.2 Performance

This is becoming less of an issue as time goes on, since the technology is improving. Generally repeaters can handle the full bandwidth of the network. (By their very nature, a simple repeater must be able to do so.) Bridges and gateways often have performance limitations of various sorts. Bridges have two numbers of interest: packet scanning rate and throughput. As explained above, a bridge must look at every packet on the network, even ones that it does not forward. The number of packets per second that it can scan in this way is the packet scanning rate. Throughput applies to both bridges and gateways. This is the rate at which they can forward traffic. Generally this depends upon packet size. Normally the number of packets per second that a unit can handle will be greater for short packets than long ones. Early models of bridge varied from a few hundred packets per second to around 7000. The higher speeds are for equipment that uses special-purpose hardware to speed up the process of scanning packets. First-generation gateways varied from a few hundred packets per second to 1000 or more. However second-generation gateways are now available, using special-purpose hardware of the same sophistication as that used by bridges. They can handle on the order of 10000 packets per second. Thus at the moment high-performance bridges and gateways can switch most of the bandwidth of an Ethernet. This means that performance should no longer be a basis for choosing between types of switch. However within a given type of switch, there are still specific models with higher or lower capacity.

Unfortunately there is no single number on which you can base performance estimates. The figure most commonly quoted is packets per second. Be aware that most vendors count a packet only once as it goes through a gateway, but that one prominent vendor counts packets twice. Thus their switching rates must be deflated by a factor of 2. Also, when comparing numbers make sure that they are for packets of the same size. A simple performance model is

```
    processing time = switching time + packet size * time per byte
```

That is, the time to switch a packet is normally a constant switching time, representing interrupt latency, header processing, routing table lookup, etc., plus a component proportional to packet size, representing the time needed to do any packet copying. One reasonable approach to reporting performance is to give packets per second for minimum and maximum size packets. Another is to report limiting switching speed in packets per second and throughput in bytes per second, i.e. the two terms of the equation above.

5.3.3 Routing

Routing refers to the technology used to decide where to send a packet next. Of course for a repeater this is not an issue, since repeaters forward every packet.

Bridges are almost always constructed with exactly two interfaces. Thus routing turns into two decisions: (1) whether the bridge should function at all, and (2) whether it should forward any particular packet. The second decision is usually based on a table of MAC-level addresses. As described above, this is built up by scanning traffic on both sides of the bridge. The goal is to forward those packets whose destination is on the other side of the bridge. This algorithm requires that the network configuration have no loops or redundant lines. Less sophisticated bridges leave this up to the system designer. With these bridges, you must set up your network so that there are no loops in it. More sophisticated bridges allow arbitrary topology, but disable links until no loops remain. This provides extra reliability. If a link fails, an alternative link will be turned on automatically. Bridges that work this way have protocol that allows them to detect when a unit must be disabled or reenabled, so that at any instant the set of active links forms a "spanning tree". If you require the extra reliability of redundant links, make sure that the bridges you use can disable and enable themselves in this way. There is currently no official standard for the protocol used among bridges, although there is a standard in the proposal stage. If you buy bridges from more than one vendor, make sure that their spanning-tree protocols will interoperate.

Gateways generally allow arbitrary network topologies, including loops and redundant links. Because gateways may have more than two interfaces, they must decide not only when to forward a packet, but where to send it next. They do this by maintaining a model of the entire network topology. Different routing techniques maintain models of greater or lesser complexity, and use the data with varying degrees of sophistication. Gateways that handle TCP/IP should generally support the two Internet standard routing protocols: RIP (Routing

Information Protocol) and EGP (External Gateway Protocol). EGP is a special-purpose protocol for use in networks where there is a backbone under a separate administration. It allows exchange of reachability information with the backbone in a controlled way. If you are a member of such a network, your gateway must support EGP. This is becoming common enough that it is probably a good idea to make sure that all gateways support EGP.

RIP is a protocol designed to handle routing within small to moderate size networks, where line speeds do not differ radically. Its primary limitations are:

- It cannot be used with networks where any path goes through more than 15 gateways. This range may be further reduced if you use an optional feature for giving a slow line a weight larger than one.

- It cannot share traffic between parallel lines (although some implementations allow this if the lines are between the same pair of gateways).

- It cannot adapt to changes in network load.

- It is not well suited to situations where there are alternative routes through lines of very different speeds.

- It may not be stable in networks where lines or gateways change a lot.

Some vendors supply proprietary modifications to RIP that improve its operation with EGP or increase the maximum path length beyond 15, but do not otherwise modify it very much. If you expect your network to involve gateways from more than one vendor, you should generally require that all of them support RIP, since this is the only routing protocol that is generally available. If you expect to use a more sophisticated protocol in addition, the gateways must have some ability to translate between their own protocol and RIP. However for very large or complex networks, there may be no choice but to use some other protocol throughout.

More sophisticated routing protocols are possible. The primary ones being considered today are cisco System's IGRP, and protocols based on the SPF (shortest-path first) algorithms. In general these protocols are designed for larger or more complex networks. They are in general stable under a wider variety of conditions, and they can handle arbitrary combinations of line type and speed. Some of them allow you to split traffic among parallel paths, to get better overall throughput. Some newer technologies may allow the network to adjust to take into account paths that are overloaded. However at the moment I do not know of any commercial gateway that does this. (There are very serious problems with maintaining stable routing when this is done.) There are enough variations among routing technology, and it is changing rapidly enough, that you should discuss your proposed network topology in detail with all of the vendors that you are considering. Make sure that their technology can handle your topology, and can support any special requirements that you have for sharing traffic among parallel lines, and for adjusting topology to take into account failures. In the long run, we expect one or more of these newer routing protocols to attain the status of a standard, at least on a de facto basis. However at the moment, there is no generally implemented routing technology other than RIP.

One additional routing topic to consider is policy-based routing. In general routing protocols are designed to find the shortest or fastest possible path for every packet. In some cases, this is not desired. For reasons of security, cost accountability, etc., you may wish to limit certain paths to certain uses. Most gateways now have some ability to control the spread of routing information so as to give you some administrative control over the way routes are used. Different gateways vary in the degree of control that they support. Make sure that you discuss any requirements that you have for control with all prospective gateway vendors.

5.3.4 Network management

Network management covers a wide variety of topics. In general it includes gathering statistical data and status information about parts of your network, and taking action as necessary to deal with failures and other changes. There are several things that a switch can do to make this process easier. The most basic is that it should have a way of gathering and reporting statistics. These should include various sorts of packet counts, as well as counts of errors of various kinds. This data is likely to be most detailed in a gateway, since the gateway classifies packets using the protocols, and may even respond to certain types of packet itself. However bridges and even buffered repeaters can certainly have counts of packets forwarded, interface errors, etc. It should be possible to collect this data from a central monitoring point.

There is now an official Internet approach to network monitoring. The first stages use a related set of protocols, SGMP and SNMP. Both of these protocols are designed to allow you to collect information and to make changes in configuration parameters for gateways and other entities on your network. You can run the matching interface programs on any host in your network. SGMP is now available for several commercial gateways, as well as for Unix systems that are acting as gateways. There is a limited set of information which any SGMP implementation is required to supply, as well as a uniform mechanism for vendors to add information of their own. By late 1988, the second generation of this protocol, SNMP, should be in service. This is a slightly more sophisticated protocol. It has with it a more complete set of information that can be monitored, called the MIB (Management Information Base). Unlike the somewhat ad hoc collection of SGMP variables, the MIB is the result of numerous committee deliberations involving a number of vendors and users. Eventually it is expected that there will be a TCP/IP equivalent of CMIS, the ISO network monitoring service. However CMIS, and its protocols, CMIP, are not yet official ISO standards, so they are still in the experimental stages.

In general terms all of these protocols accomplish the same thing: They al ow you to collect criticial information in a uniform way from all vendors' equipment. You send commands as UDP datagrams from a network management program running on some host in your network. Generally the interaction is fairly simple, with a single pair of datagrams exchanged: a command and a response. At the moment security is fairly simple. It is possible to require what amounts to a password in the command. (In SGMP it is referred to as a "session name", rather than a password.) More elaborate, encryption-based security is being developed.

You will probably want to configure the network management tools at your disposal to do several different things. For short-term network monitoring, you will want to keep track of switches crashing or being taken down for maintenance, and of failure of communications lines and other hardware. It is possible to configurate SGMP and SNMP to issue "traps" (unsolited messages) to a specified host or list of hosts when some of these critical events occur (e.g. lines up and down). However it is unrealistic to expect a switch to notify you when it crashes. It is also possible for trap messages to be lost due to network failure or overload. Thus you should also poll your switches regularly to gather information. Various displays are available, including a map of your network where items change color as their status changes, and running "strip charts" that show packet rates and other items through selected switches. This software is still in its early stages, so you should expect to see a lot of change here. However at the very least you should expect to be notified in some way of failures. You may also want to be able to take actions to reconfigure the system in response to failures, although security issues make some mangers nervous about doing that through the existing management protocols.

The second type of monitoring you are likely to want to do is to collect information for use in periodic reports on network utilization and performance. For this, you need to sample each switch perodically, and retrieve numbers of interest. At Rutgers we sample hourly, and get the number of packets forwarded for IP and DECnet, a count of reloads, and various error counts. These are reported daily in some detail. Monthly summaries are produced giving traffic through each gateway, and a few key error rates chosen to indicate a gateway that is being overloaded (packets dropped in input and output).

It should be possible to use monitoring techniques of this kind with most types of switch. At the moment, simple repeaters do not report any statistics. Since they do not generally have processors in them, doing so would cause a major increase in their cost. However it should be possible to do network management for buffered repeaters, bridges, and gateways. Gateways are the most likely to contain sophisticated network management software. Most gateway vendors that handle TCP/IP are expected to implement the monitoring protocols described above. Many bridge vendors make some provisions for collecting performance data. Since bridges are not protocol-specific, most of them do not have the software necessary to implement TCP/IP-based network management protocols. In some cases, monitoring can be done only by typing commands to a directly-attached console. (We have seen one case where it is necessary to take the bridge out of service to gather this data.) In other cases, it is possible to gather data via the network, but the monitoring protocol is ad hoc or even proprietary.

Except for very small networks, you should probably insist that all of the devices on your network collect statistics and provide some way of querying them remotely. In the long run, you can expect the most software to be available for standard protocols such as SGMP/SNMP and CMIS. However proprietary monitoring tools may be sufficient as long as they work with all of the equipment that you have.

5.3.5 A final evaluation

Here is a summary of the places where each kind of switch technology is normally used:

- Repeaters are normally confined to a single building. Since they provide no traffic isolation, you must make sure that the entire set of networks connected by repeaters can carry the traffic from all of the computers on it. Since they generally provide no network monitoring tools, you will not want to use repeaters for a link that is likely to fail.

- Bridges and gateways should be placed sufficiently frequently to break your network into pieces for which the traffic volume is manageable. You may want to place bridges or gateways in places where traffic would not require them for network monitoring reasons.

- Because bridges must pass broadcast packets, there is a limit to the size network you can construct using them. It is probably a good idea to limit the network connected by bridges to a hundred systems or so. This number can be increased somewhat for bridges with good facilities for filtering.

- Because certain kinds of network misbehavior will be passed, bridges should be used only among portions of the network where a single group is responsible for diagnosing problems. You have to be crazy to use a bridge between networks owned by different organizations. Portions of your network where experiments are being done in network technology should always be isolated from the rest of the network by gateways.

- For many applications it is more important to choose a product with the right combination of performance, network management tools, and other features than to make the decision between bridges and gateways.

@section(Configuring Gateways)

This section deals with configuration issues that are specific to gateways. Gateways than handle TCP/IP are themselves Internet hosts. Thus the discussions above on configuring addresses and routing information apply to gateways as well as to hosts. The exact way you configure a gateway will depend upon the vendor. In some cases, you edit files stored on a disk in the gateway itself. However for reliability reasons most gateways do not have disks of their own. For them, configuration information is stored in non-volatile memory or in configuration files that are uploaded from one or more hosts on the network.

At a minimum, configuration involves specifying the Internet address and address mask for each interface, and enabling an appropriate routing protocol. However generally a few other options are desirable. There are often parameters in addition to the Internet address that you should set for each interface.

One important parameter is the broadcast address. As explained above, older software may react badly when broadcasts are sent using the new standard broadcast address. For this reason, some vendors allow you to choose a broadcast address to be used on each interface. It should be set using your knowledge of what computers are on each of the networks. In general if the computers follow current standards, a broadcast address of 255.255.255.255 should be used. However older implementations may behave better with other addresses, particularly the address that uses zeros for the host number. (For the network 128.6 this would be 128.6.0.0. For compatibility with software that does not implement subnets, you would use 128.6.0.0 as the broadcast address even for a subnet such as 128.6.4.) You should watch your network with a network monitor and see the results of several different broadcast address choices. If you make a bad choice, every time the gateway sends a routing update broadcast, many machines on your network will respond with ARP's or ICMP errors. Note that when you change the broadcast address in the gateway, you may need to change it on the individual computers as well. Generally the idea is to change the address on the systems that you can configure to give behavior that is compatible with systems that you can't configure.

Other interface parameters may be necessary to deal with peculiarities of the network it is connected to. For example, many gateways test Ethernet interfaces to make sure that the cable is connected and the transceiver is working correctly. Some of these tests will not work properly with the older Ethernet version 1 transceivers. If you are using such a transceiver, you would have to disable this keepalive testing. Similarly, gateways connected by a serial line normally do regular testing to make sure that the line is still working. There can be situations where this needs to be disabled.

Often you will have to enable features of the software that you want to use. For example, it is often necessary to turn on the network management protocol explicitly, and to give it the name or address of a host that is running software to accept traps (error messages).

Most gateways have options that relate to security. At a minimum, this may include setting password for making changes remotely (and the "session name" for SGMP). If you need to control access to certain parts of your network, you will also need to define access control lists or whatever other mechanism your gateway uses.

Gateways that load configuration information over the network present special issues. When such a gateway boots, it sends broadcast requests of various kinds, attempting to find its Internet address and then to load configuration information. Thus it is necessary to make sure that there is some computer that is prepared to respond to these requests. In some cases, this is a dedicated micro running special software. In other cases, generic software is available that can run on a variety of machines. You should consult your vendor to make sure that this can be arranged. For reliability reasons, you should make sure that there is more than one host with the information and programs that your gateways need. In some cases you will have to maintain several different files. For example, the gateways used at Rutgers use a program called "bootp" to supply their Internet address, and they then load the code and configuration information using TFTP. This means that we have to maintain a file for bootp that contains Ethernet and Internet addresses for each gateway, and a set of files containing other configuration information for each gateway. If your network is large, it is worth taking some trouble to make sure that this information remains consistent. We keep master copies of all of the configuration information on a single computer, and distribute it to other systems when it changes, using the Unix utilities make and rdist. If your gateway has an option to store configuration information in non-volatile memory, you will eliminate some of these logistical headaches. However this presents its own problems. The contents of non-volatile memory should be backed up in some central location. It will also be harder for network management personnel to review configuration information if it is distributed among the gateways.

Starting a gateway is particularly challenging if it loads configuration information from a distant portion of the network. Gateways that expect to take configuration information from the network generally issue broadcast requests on all of the networks to which they are connected. If there is a computer on one of those networks that is prepared to respond to the request, things are straightforward. However some gateways may be in remote locations where there are no nearby computer systems that can support the necessary protocols. In this case, it is necessary to arrange for the requests to be routed back to network where there are appropriate computers. This requires what is strictly speaking a violation of the basic design philosophy for gateways. Generally a gateway should not allow broadcasts from one network to pass through to an adjacent network. In order to allow a gateway to get information from a computer on a different network, at least one of the gateways in between will have to be configured to pass the particular class of broadcasts used to retrieve this information. If you have this sort of configuration, you should test the loading process regularly. It is not unusual to find that gateways do not come up after a power failure because someone changed the configuration of another gateway and made it impossible to load some necessary information.

5.4 Configuring routing for gateways

The final topic to be considered is configuring routing. This is more complex for a gateway than for a normal host. Most Internet experts recommend that routing be left to the gateways. Thus hosts may simply have a default route that points to the nearest gateway. Of course the gateways themselves can't get by with this. They need to have complete routing tables.

In order to understand how to configure a gateway, we have to look in a bit more detail at how gateways communicate routes. When you first turn on a gateway, the only networks it knows about are the ones that are directly connected to it. (They are specified by the configuration information.) In order to find out how to get to more distant parts of the network, it engages in some sort of "routing protocol". A routing protocol is simply a protocol that allows each gateway to advertise which networks it can get to, and to spread that information from one gateway to the next. Eventually every gateway should know how to get to every network. There are different styles of routing protocol. In one common type, gateways talk only to nearby gateways. In another type, every gateway builds up a database describing every other gateway in the system. However all of the protocols have some way for each gateway in the system to find out how to get to every destination.

A metric is some number or set of numbers that can be used to compare routes. The routing table is constructed by gathering information from other gateways. If two other gateways claim to be able to get to the same destination, there must be some way of deciding which one to use. The metric is used to make that decision. Metrics all indicate in some general sense the "cost" of a route. This may be a cost in dollars of sending packets over that route, the delay in milliseconds, or some other measure. The simplest metric is just a count of the number of gateways along the path. This is referred to as a "hop count". Generally this metric information is set in the gateway configuration files, or is derived from information appearing there.

At a minimum, routing configuration is likely to consist of a command to enable the routing protocol that you want to use. Most vendors will have a prefered routing protocol. Unless you have some reason to choose another, you should use that. The normal reason for choosing another protocol is for compatibility with other kinds of gateway. For example, your network may be connected to a national backbone network that requires you to use EGP (exterior gateway protocol) to communicate routes with it. EGP is only appropriate for that specific case. You should not use EGP within your own network, but you may need to use it in addition to your regular routing protocol to communicate with a national network. If your own network has several different types of gateway, then you may need to pick a routing protocol that all of them support. At the moment, this is likely to be RIP (Routing Information Protocol). Depending upon the complexity of your network, you could use RIP throughout it, or use a more sophisticated protocol among the gateways that support it, and use RIP only at the boundary between gateways from different vendors.

Assuming that you have chosen a routing protocol and turned it on, there are some additional decisions that you may need to make. One of the more basic configuration options has to do with supplying metric information. As indicated above, metrics are numbers which are used to decide which route is the best. Unsophisticated routing protocols, e.g. RIP, normally just count hops. So a route that passes through 2 gateways would be considered better than one that passes through 3. Of course if the latter route used 1.5Mbps lines and the former 9600 bps lines, this would be the wrong decision. Thus most routing protocols allow you to set parameters to take this sort of thing into account. With RIP, you would arrange to treat the 9600 bps line as if it were several hops. You would increase the effective hop count until the better route was chosen. More sophisticated protocols may take the bit rate of the line into account automatically. However you should be on the lookout for configuration parameters that need to be set. Generally these parameters will be associated with the particular interface. For example, with RIP you would have to set a metric value for the interface connected to the 9600 bps line. With protocols that are based on bit rate, you might need to specify the speed of each line (if the gateway cannot figure it out automatically).

Most routing protocols are designed to let each gateway learn the topology of the entire network, and to choose the best possible route for each packet. In some cases you may not want to use the "best" route. You may want traffic to stay out of a certain portion of the network for security or cost reasons. One way to institute such controls is by specifying routing options. These options are likely to be different for different vendors. But the basic strategy is that if the rest of the network doesn't know about a route, it won't be used. So controls normally take the form of limiting the spread of information about routes whose use you want to control.

Note that there are ways for the user to override the routing decisions made by your gateways. If you really need to control access to a certain network, you will have to do two separate things: Use routing controls to make sure that the gateways use only the routes you want them to. But also use access control lists on the gateways that are adjacent to the sensitive networks. These two mechanisms act at different levels. The routing controls affect what happens to most packets: those where the user has not specified routing manually. Your routing mechanism must be set up to choose an acceptable route for them. The access control list provides an additional limitation which prevents users from supplying their own routing and bypassing your controls.

For reliability and security reasons, there may also be controls to allow you to list the gateways from which you will accept information. It may also be possible to rank gateways by priority. For example, you might decide to listen to routes from within your own organization before routes from other organizations or other parts of the organization. This would have the effect of having traffic use internal routes in preference to external ones, even if the external ones appear to be better.

If you use several different routing protocols, you will probably have some decisions to make regarding how much information to pass among them. Since multiple routing protocols are often associated with multiple organizations, you must be sure to make these decisions in consultation with management of all of the relevant networks. Decisions that you make may have consequences for the other network which are not immediately obvious. You might think it would be best to configure the gateway so that everything it knows is passed on by all routing protocols. However here are some reasons why you may not want to do so:

- The metrics used by different routing protocols may not be comparable. If you are connected to two different external networks, you want to specify that one should always be used in preference to the other, or that the nearest one should be used, rather than attempting to compare metric information received from the two networks to see which has the better route.

- EGP is particularly sensitive, because the EGP protocol cannot handle loops. Thus there are strict rules governing what information may be communicated to a backbone that uses EGP. In situations where EGP is being used, management of the backbone network should help you configure your routing.

- If you have slow lines in your network (9600 bps or slower), you may prefer not to send a complete routing table throughout the network. If you are connected to an external network, you may prefer to treat it as a default route, rather than to inject all of its routing information into your routing protocol.