NW

Computer Networks

1 Introduction

Computer networks are an indispensable part of every research organization. Local networks tie together machines within a lab or over an entire campus, allowing users with common goals and interests to share resources such as disks, printers, or supercomputers. Wide area networks operated by the NSF, the Department of Energy, and other funding agencies are mainly used to exchange information but are also increasingly used to access remote resources.

This chapter is an introduction to computer networking for computational scientists. The emphasis is on how scientists are likely to use local and wide area networks. We will describe the types of resources that are available over networks and how to use common networking tools to take advantage of the resources. The emphasis will be on the user's view of networking, with only enough of a description of the hardware and software to give readers a sense of why networks evolved the way they have and where they are headed in the near future.

There are several good books on computer networks for those readers who want to learn more. Two textbooks on network concepts and local area network implementation are by Comer [3] and Perlman [6]. An informal, easy-to-read description of the Internet (the loosely organized collection of wide area networks that will be described in Section 3) is *The Whole Internet Catalog*, by Krol [4]. This book is highly recommended for anyone who will be more than a casual user of network resources.

2 Local Area Networks

Computers within a department or campus often communicate over a *local area network*, or LAN (see Figure 1). These networks are fairly small, in terms of physical distances, usually limited to a single building. The three basic categories of equipment that are found in a local network are workstations, servers, and gateways.

Users access networks through terminals and workstations. A terminal is simply an I/O device; the only processing it does is limited to presenting information for display on the screen. Simple ("dumb") terminals do little or no formatting, so everything that is displayed is arranged by a computer system and every keystroke is passed back to the computer. For example, if you make a mistake while typing a line and hit the backspace key, a terminal just sends the keystroke to the computer you are using. The computer will update an internal buffer and send the terminal a command that will cause it to move the cursor to the left by one space. Until a few years ago terminals were connected directly to computers, but now it is common for them to be connected to a communications server, as shown in Figure 1. The server then uses the local network to connect the terminal to any machine on the network, giving users more flexibility.

An "X terminal" is almost a stand-alone computer by itself, with an internal processor and a few megabytes of memory. The processor displays windows and handles mouse and keyboard input locally, relieving a compute server from doing these low-level tasks. An X terminal rarely, if ever, runs an application itself, though; instead it just draws windows and menus and handles the user interface for applications running on other machines. The name comes from the X window system, a common protocol used by window-based applications (Section 2.3).

Workstations are single-user computer systems. They range in complexity from simple personal computers to advanced processors with sophisticated graphics and vector processing capabilities. Their common attributes include the fact that they can operate as stand-alone computer systems and usually have a "point-and-click" (window-based) user interface. Most often the person using a workstation will sit at the console and use the main keyboard and mouse. Unix workstations allow several users to log in to the system. In a networked environment it is not uncommon for additional users to log in by sitting at a terminal and using the communication server to connect their terminal to the workstation.

The second category of machines on a network are servers. As their name suggests, these nodes provide services to other nodes and are shared by many users. Printers are a good example: rather than connect a printer directly to one computer, and forcing each person to use that computer in order to print a document, a printer with a built-in network interface can be connected directly to the network and used by any system. Other common types of servers are *file servers* and *compute servers*. A file server is often just a workstation with extra disk space, but most workstation manufacturers build special server configurations for higher performance and better reliability. Mainframes and supercomputers that are likely to be used by computational scientists are usually compute servers on a local network. The term "compute server" reflects the fact that mundane jobs like editing files and reading mail are done on workstations and microcomputers, but users can run computationally intense jobs on these special purpose high performance systems.

The third category of network equipment is an interface to another network. These interfaces are either *gateways* or *routers*. A gateway translates information between the formats used by two different types of networks. In the early days of local area networking there were few standards and a wide variety of performance levels, and thus different protocols and services evolved in different settings. For example Apple microcomputers could be connected via an Appletalk network. If these machines were to be linked to a set of Unix machines, one of the Unix systems typically ran a software package that translated Appletalk protocol packets into the format used by the Unix systems.

A router simply forwards packets of information between networks that use the same protocols. In a packet-switched network each packet of information contains the network address of its destination [6, 3]. The router monitors all network traffic and forwards all packets that are addressed for machines not on the local network. The most common use of a router is in building a hierarchical network. A local network within an academic department will have a router that connects to the campus-wide network, which in turn will have a router or gateway to a regional network. Routers are also used to improve performance within a local network. If network traffic is too high, the network administrator might split it into two separate networks connected by a router. Routers are invisible to users, so in most situations sending information to a machine elsewhere in the department or on campus is no different than sending information to the machine in the lab next door.

The remainder of this section is an overview of the types of services typically provided by a local area network and how they impact the way you work. The examples will typically be from networks of Unix systems because Unix is so predominant in university and government labs, but the main concepts also apply to most other types of local area networks.

2.1 Network File System (NFS)

For the most part interacting with the local network—accessing shared files, logging in to compute servers, or communicating with other users—is very simple. Networking software is invoked automatically, and it hides the physical layout of the network from you, giving you the illusion you are using one large computer system. You can access a file simply by giving its name, no matter which machine you are logged in to or where the file resides, or send mail to other users without worrying about which machine they use or are likely to log in to the next time they read their mail.

The Unix file system is a hierarchical collection of directories. In a stand-alone system the file system maps the directory tree onto local disks. In a networked system, however, some of the nodes correspond to files that reside elsewhere in the local network. Figure 2 shows part of a typical Unix file system and how it might be allocated on a combination of local disks and network nodes. The shaded sections of the tree correspond to parts of the file system that reside on other computers. The software used to implement nonlocal files is the *network file system*, or *NFS*. Nonlocal files are also called *remote files*.

If you are curious about where files reside, you can use the Unix command `df` (disk free) to print information about the files accessible from your workstation or computer. The output of `df` is a set of lines that describe the capacity and amount of free space on each local disk and each remote file system currently mounted on your computer (Figure 2). Local devices typically have names that begin `/dev/...`, but remote systems have names that identify the network node where the file resides. The name can either be the name of a system or a symbolic name. In the example in Figure 2 `fog` and `drizzle` are the names of other Unix systems in the network, but `cshost, faculty`, and `grads` are symbolic names. The advantage of symbolic names is that if the computer where these file systems reside goes down it is relatively easy to move the disks and change the definition of the symbolic name.

Communicating with other users in a local network is also handled automatically by network system software. In Unix, programs that communicate with other users (`mail, finger, talk,` etc.) are based on user login names. For a brief period as local area networks were evolving it was necessary to identify a user by the system they normally used. For example, suppose your lab had three systems, named `fog, drizzle`, and `mist`. You normally worked on `fog`, and your colleague named Fred worked on `mist`. To send mail to Fred, you had to type something like

```
% mail drizzle!mist!fred < msg
```
(obsolete)

Now, however, most Unix installations have a server dedicated to mail processing, and the mail boxes, user home directories, and password files are shared by all the workstations. To mail to a user who has a login anywhere in your local network you just type

```
% mail fred
```

as if `fred` is the login ID of another user who logs in to your workstation.

The NFS software is also used to provide a very flexible and more robust working environment. A common organization is to put users' home directories on a file server, i.e. every workstation mounts the home directory (e.g. `/home/users`) via NFS. Now wherever you log in, your home directory will "follow you" to that system. Suppose you normally work in your office, but one morning you come in and the CRT on your workstation is broken. You can go to a workstation or X terminal down the hall, log in there and get right to work.

Microcomputer operating systems have similar facilities for automatic access of remote files. For example, files are identified by icons on the Macintosh desktop. Networking software will display icons for files and folders that reside on other Macintosh systems in the local network. There are also programs that will translate between Mac O/S file structures and Unix file systems so the files can actually reside on Unix file servers. If you create and use files exclusively with Macintosh applications this won't pose any problems, but you need to be careful if you want to create a file on the Mac and read it with a Unix application or vice versa, since the file formats are radically different. You will probably need to use FTP or one of the other file transfer programs described in Section 3.1.3 because these programs translate file formats as they move data between systems.

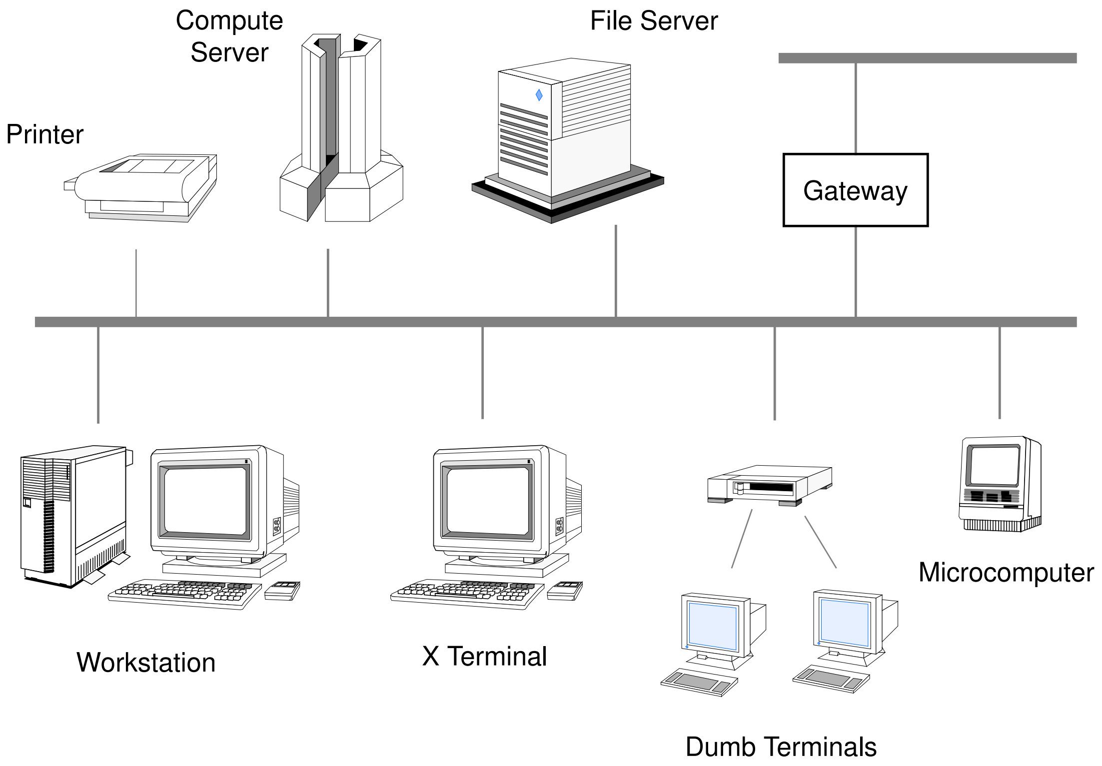
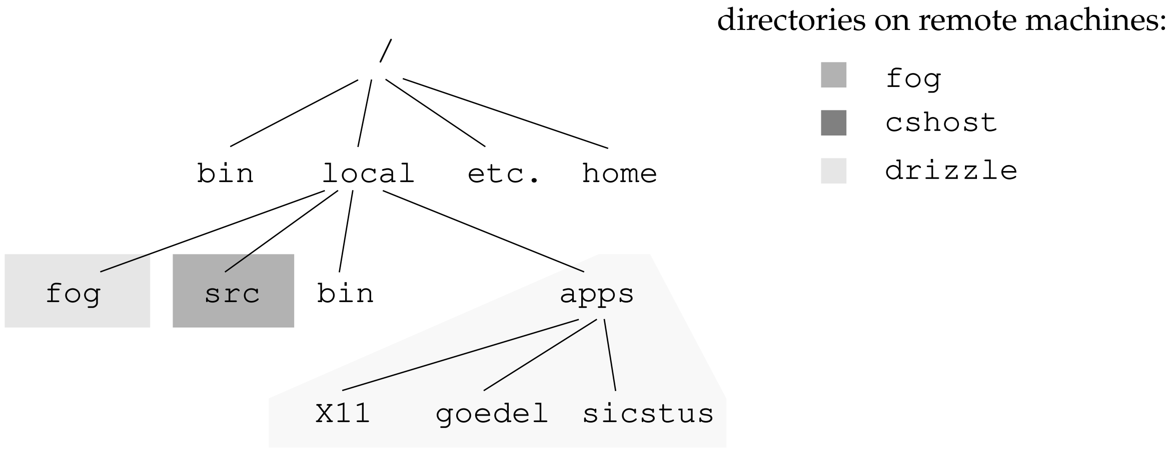


Figure 1: Local Area Network



% df

Filesystem		kbytes	used	avail	Mounted on
/dev/dsk/c4d0s13	484960	417413	19051	96%	/
fog:/local/fog	485288	322309	114450	74%	/local/fog
cshost:/local/src	551399	257917	238343	52%	/local/src
drizzle:/local/apps	484960	383897	52567	88%	/local/apps
faculty:/home/faculty	937177	708537	134923	84%	/home/faculty
grads:/home/grads	1571062	1311905	102051	93%	/home/grads

Figure 2: Network File System (NFS)

2.2 Access to Compute Servers: rlogin

Gaining access to computing services in a local network is not quite as transparent as accessing files. Ideally if you had an X terminal in your office you could go through the login process (type your name and password) and the terminal would connect you to a free or lightly used system. Most of the time, however, you have to identify the system you want to use by name. The same is true if you have a workstation in your office and want to use another system on the network. For example, suppose your department has licenses to run Mathematica on HP workstations, but the machine in your office is a Sun workstation. To run Mathematica you would log in to one of the HPs, specifying which system you want to use by its name, and then start Mathematica on that machine.

The program you run to log in to another computer is `rlogin` (remote login). To log in to another system, simply type `rlogin` and the name of the system:

```
% rlogin fog
```

`rlogin` also works for nonlocal systems, but in this case you need a more complex identifier for the other system. For example, if your workstation is in the Physics department and you want to use a machine in the Computer Science department, you might type

```
% rlogin fog.cs
```

or perhaps

```
% rlogin fog.cs.cascadia.edu
```

The format of these longer system names is described in the section on the Internet (section 3.1).

If your login name on the remote machine is different from the one you use locally, you need to give your login name as part of the command, e.g.

```
% rlogin fog.cs.cascadia.edu -l fjones
```

will log you in to the remote system with user name `fjones`. If you forget the -l option, you will still be able to log in. `rlogin` will first try to log you in with the wrong login name and prompt you for a password. Just type a return, and then you will see the usual prompt that asks you for both a login name and password and you can continue.

2.3 The X Window System

The idea of basing user interactions on a "mouse" or other pointer device and window panes that appear to be overlapping pieces of paper on a desktop can be traced to research projects carried out at Xerox Palo Alto Research Center (PARC) in the 1970s. Early commercial systems that used this style of "point and click" user interface for special purpose computing systems were developed by Xerox, Symbolics, and other companies. The Symbolics machines were advanced programming environments for developing applications in the LISP programming language, with editors and other software that organized information in window-based displays. Sun Microsystems was one of the first companies to use a windowing system as an interface for Unix, bringing this technology into the arena of general purpose computing.

By the mid 1980s there were several window-based Unix systems on the market, but the situation from a programmer's perspective was chaotic. Applications written for one window system were very difficult to "port" to another system, since each manufacturer had defined their own graphics languages and system routines for drawing and interacting with windows. A research project started at MIT and sponsored by the major workstation vendors addressed this problem by defining a programming interface to a generic (brand X) window system. Each vendor would then be responsible for implementing the X Window protocol on their own systems, optimizing and extending the system to take advantage of their special hardware as long as they provided the necessary basic services.

An important innovation that arose from the X Window project, in addition to providing a common interface, is the division of labor between *clients* and *servers*. A client in this model is an application program that is built around a window- based user interface. In order to display information, the client calls upon a server to do the operations that will actually display the windows, menus, etc. The advantage of this client/server model is that clients and servers do not have to run on the same machine. In most local networks, the server software runs on workstations. Many clients, for example editors, mail systems, calendars, and status displays, also run on workstations, but complex, compute-intensive applications can run on high performance machines. As long as the application software implements the X Window client protocol it can run by users from any system that is capable of being an X server.

The most straightforward way to run an X application on another machine in your local network is to use `rlogin` to connect to that machine. When you are logged in, you simply start the application.[1] Previously we used Mathematica as an example of a service you might find on another computer in your network. One way to use Mathematica is to log in to the system where it is installed, and then type `math` (the Unix command that starts Mathematica). Text input and output will go in the shell window you used for the *rlogin* command. When you ask Mathematica to draw a window, e.g. for a plot command, the window will appear on your workstation because Mathematica (the client) asks your workstation (the server) to make a window for it.

The connection to the remote host can be automated, but the process is a little tricky. You may be able to write a shell script for your local system and name it `math`. This would allow you to simply type `math` whenever you wanted to run Mathematica. The script could create a new shell for you, invoke `rlogin`, connect to the remote machine, and start up Mathematica. However, several steps in this procedure might fail, depending on the security

[1] X applications use an environment variable named `DISPLAY` to determine which X server to use. If you have trouble starting X clients on other machines (e.g. the application seems to start but you don't see and windows) check with your local administrator to make sure your `.login` and other startup files are configured properly

measures in place on your network. Another problem is that you might need to specify a password to log in to the remote system; you may be tempted to put this password in your script, but this is a security leak and is discouraged at most locations.

The definitions of X clients and servers provide a more concise definition of an X terminal: an X terminal is a device that implements the X server software. Often the server software is stored in ROM, along with common fonts and other data, and it runs the window manager and other server software. However, it does not normally run any client applications. It passes keystrokes, mouse clicks, and other events to the client software for processing, and then responds to client requests to move or redraw windows or update the contents of windows.

The definition of servers and clients may seem confusing at first. You may prefer to think of yourself as a client, and think of the machine that implements a shared software package like Mathematica as the server. However, this is the opposite of the X Window terminology, which calls Mathematica the client. Just remember that X clients and servers are defined from the application programmer's perspective. A client is an application that needs a window, and a server is a process running on a workstation that will draw the window for the client.

2.4 Electronic Mail

Electronic mail, or *e-mail*, is a fact of life in most working environments. People rely on it for technical discussions (if you have a problem using a system, you can ask an expert and get a response within minutes), sharing data and programs, working on papers with colleagues (mailing new drafts back and forth), and many other reasons. E-mail is fast—data files several thousand bytes long can be sent around the world in seconds—and it is also convenient. If the person you are communicating with is on the other side of the world your message will be waiting for them when they show up for work in the morning.

Since there are so many different mail systems, and often many different interfaces for each one, we will not try to describe any particular system here. Instead, we will just try to convince you to learn one of the e-mail systems on your local network if you have somehow avoided e-mail until now.

In addition to interacting with colleagues, e-mail is a default interface to many of the resources to be described in the section on Internet services. For example, one way to obtain technical reports from a university is to use the file transfer program FTP. If your machine is not on the Internet, FTP will not be able to connect to the remote machine. But machines that are not directly connected to the Internet can send mail to Internet sites, and it may be possible to retrieve a paper by sending e-mail to a server at the university. All you need to do is put the words "send X" in the body of your message, and the server will send you a copy of X within a few hours. Many other Internet services also have e-mail interfaces.

2.5 Local Network Technology

The predominant technology for local networks is *ethernet*, developed at Xerox PARC in the 1970s. The key innovation in ethernet was building a system in which a network node could sense when the network was in use. An earlier network known as the AlohaNet used radio to link terminals in the Hawaiian islands to a mainframe at the University of Hawaii. The organization of the network was a success, but performance suffered because a node could not tell when another node was transmitting. When two nodes transmitted at the same time their messages were garbled. Researchers at Xerox designed their system around a medium (the ether) which had the property that nodes could detect transmissions in progress and thus postpone their own transmissions. Analytical models predicted that a very high percentage of the network bandwidth could be exploited by a packet-switched network based on carrier-sense technology.

Logically the ether is a single transmission line connecting each node, similar to a bus used in the backplane of a computer system (Chapter "Computer Architecture"). Information put on the network by a node is broadcast to every other node. Message passing software in an Ethernet system is based on packet switching, in which a message is broken into small pieces that are transmitted individually and reassembled by the receiver. A communication protocol defines the structure of each packet. For example, the first few bits may contain the ID of the receiving node, followed by the packet number and size (e.g. "packet 4 of 12, length = 100 bytes"), followed by the data itself. Thus when node A wants to send a message to node B, it breaks it into packets, puts node B's address and the packet description in front of each packet, and broadcasts the packets using the ether. Every node continually monitors the network, looking for packets with their node ID. When B sees the packets addressed to it, it saves them in a buffer and reassembles the message as it arrives. The network consists of a single wire, so only one bit of information can be transmitted at a time. The theoretical peak performance of the network is defined by the number of bits per second that can be transmitted by a node. Actual performance depends on the number of bits used for overhead (packet IDs, etc.), the average packet size (including data plus overhead), the number of systems connected to the network, and the amount of time lost due to collisions when two or more nodes trying to transmit at the same time (see also problem 4).

Physically the first ethernet systems were thick, shielded coaxial cables up to 1 kilometer long. Workstations and other machines that wished to connect to the network could tap into the ether at arbitrary locations provided they were separated by a few meters. With cables of this length, there are situations where two nodes can transmit at the same time and interfere with each other. If two nodes at opposite ends of the ether see the network idle and decide to begin transmitting, it will take a few microseconds for the signals to reach the other end of the ether, so two nodes can begin transmitting only to discover a few microseconds later that another message was in the ether. To overcome this problem, each node continually monitors the network as it is transmitting, and if it ever sees its own transmission become garbled it will stop, assuming the problem was caused by another node that began transmitting at the same time. These thick-cable ethernet systems had a bandwidth of 10 million bits per second (Mbps).

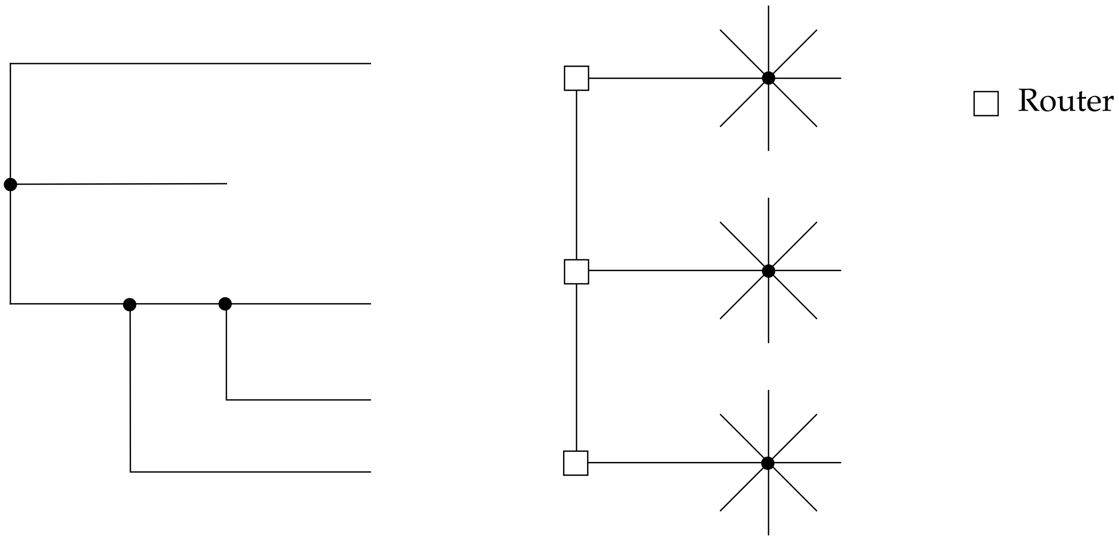


Figure 3: Ethernet Local Area Networks

As a result of improvements in the technology used to send and receive packets, modern ethernet systems can be simple shielded or unshielded "twisted pair" wiring. Many networks are now broken into independent segments connected by routers. Figure 3 shows two local networks. On the left is an older, single-segment ethernet (the black dots represent junctions where two or more cables are connected). A packet broadcast from any point on the network is broadcast to all other parts of the network. On the right is a typical modern ethernet which has several independent segments connected by routers. Packets originating in one segment are examined by the nearest router. If the packet is addressed to another node within the segment, the router ignores it, otherwise the router retransmits the packet to the other routers (and all the other nodes in the segment ignore the packet). The router improves performance by keeping packets destined for nodes within a segment from interfering with local traffic on other segments. Bandwidth within a segment in a modern ethernet can be as high as 100Mbps.

There are several alternatives to Ethernet as the basic technology for local area networks, but they are very recent and standards are still evolving so manufacturers have been slow to commit to any particular method. Fiber Distributed Data Interconnection (FDDI) and Asynchronous Transfer Mode (ATM) are two examples. FDDI is a fiber-optic based system that is a broadcast network, like Ethernet, but with higher bandwidth and other properties. ATM is a point-to-point network with very low overhead. Both are available currently and may become much more widely used in the near future.

3 Regional, National, and International Networks

Networks that connect machines situated over geographical distances larger than the 1 km covered by ethernet are known as *wide area networks*. Some wide area networks are regional networks which connect systems within a city, state, or region. There are also several national and international networks.

Membership in a regional or national network is usually extended to entire local networks. In other words, the nodes in a wide area network tend to be local networks themselves, as opposed to single machines. Connection to a regional or national network is through a gateway in the local network that links the local network to other networks in the regional network.

Wide area networks started up as a way of linking together sites with common interests. For example, the Energy Sciences Network (ESNET) connects university, academic, and industrial research labs funded by the US Department of Energy. NSFNET, sponsored by the National Science Foundation, is a hierarchical network with a very high bandwidth "backbone" connecting several regional NSF networks. Each of these networks is administered separately. If a department or research group wishes to join a regional or national network, it applies to the administrator for membership. Once connected, it pays dues to cover operational costs and follows guidelines set by the network administration.

The *Internet* is a loose federation of wide area networks. There is no central Internet administration to evaluate applications for membership, there are no dues, and there are no formal rules other than those of the constituent networks. If your workstation is part of a local area network that is connected to a regional or national network, chances are your machine is already a part of the Internet. Most national and international wide area networks, including NSFNET, ESNET, ARPANET, and BITNET, are members of the Internet. Internet services are not (yet) as transparent as local network services. For example, in order to copy a file in a local area network, you would just type

```
% cp x y
```

where `x` is the name of the existing file and `y` is the name you want to give the copy. You don't need to know whether `x` is on your machine, a different workstation, or a file server. In a wide area network, however, you start a special application that makes a remote connection, displays names of files you can access, and has many different transfer modes. In the near future, however, as bandwidth improves and the services described in this section evolve, you can expect the conceptual dividing line between local and wide area networks to blur. For example you should be able to start an X Window client application on a computer on the other side of the country and have it draw windows directly on your workstation.

3.1 Internet and the Domain Name System

What allows this federation of networks to cooperate with one another and exchange information is the fact that all are packet switched communications systems based on the TCP/IP protocol standards. TCP (transmission control protocol) and IP (Internet protocol) are two standards that together allow applications to communicate with one another over a wide range of physically different communication systems. Local networks can be based on a variety of technologies, but as long as the software layers on top of the network hardware implements the TCP/IP protocol operations a network can become part of the Internet.

TCP/IP communication depends on addresses being included in each packet. As described previously, routers use these addresses to decide whether to keep a packet in a local network or to forward it to a different network. Addresses have four numeric fields. For example, the address of the machine used by the Computational Science Education Project is 129.59.130.21. To make it easier to remember addresses, there is an equivalent symbolic form, in this case `compsci.cas.vanderbilt.edu`.

When you are using Internet software to communicate with another site, all you need to know about the site is its name. Most local networks either have a *name server* or a way to connect to a name server in order to translate symbolic names into internet addresses. If the translation fails, however, you can still make a connection by using the numeric address. For example, suppose you have an account on `compsci`, and you wish to log in. One way is to use the `rlogin` program (described in 2.2):

```
% rlogin compsci.cas.vanderbilt.edu
```

If your local network cannot translate the name, you will get an "unknown host" error message. If you know the numeric address, try it:

```
% rlogin 129.59.130.21
```

The symbolic name of a site is determined by a naming standard known as the *domain name system*. Each field in a symbolic address corresponds to a single domain. The first field is a host name, which identifies a single computer. The last field is a top level domain. In between are department names, organization names, etc. in order of increasing generality. In our example, `compsci.cas.vanderbilt.edu` is a host named `compsci` in the College of Arts and Sciences at Vanderbilt University, which is an educational institution. The six top level domains in the US and some sample addresses are listed in Figure 5. Symbolic addresses of sites outside the US end with a two-letter country code. Common country codes are also listed in Figure 5. There is a slow movement toward a naming standard that includes geographic information for sites within the US, also, but it is proceeding about as slowly as the conversion to the metric system. As new sites apply for Internet names many are given these new style names. For example, a name being considered for the Springfield Public School District is `sps.lane.or.us` (Springfield Public Schools, Lane County, Oregon, US).

At one point during the evolution of the Internet is was necessary to know which constituent network a site belonged to, and to use that network's naming scheme as part of its Internet name. For example, a network of Unix machines known as UUNET identified a site by the path a message traversed in reaching the site. A command of the form

```
% mail drizzle!fog!mist!fred
```

(obsolete)

meant "send a message from this machine to host `drizzle`, have it forward the message to `fog`, and have fog send it to the user named `fred` on `mist`." When `drizzle` was connected to the Internet, Fred's colleagues at other Internet sites could send him mail via the address "`fog!mist!fred@drizzle.cs.cascadia.edu`". Now, however, you would probably reach Fred simply by using the domain name of his organization, i.e.

```
% mail fred@cs.cascadia.edu
```

You will still see host names and user names that contain network- specific fields, but they are becoming increasingly rare.

Figure 6 shows a typical local area network and its relationship to a regional network. Every workstation and compute server supports TCP/IP and the applications built on top of this protocol. For example, users can log in to `fog` from `mist` using `rlogin`. Because there is a connection to the regional network (NorthWestNet) a user on `fog` can use `rlogin` to connect to `compsci`.

In many organizations the system that is connected to the regional network (e.g. `drizzle` in Figure 6) is the only one allowed to transfer packets outside the local network. The machine connected to the regional net is a *firewall* that isolates the other systems from the rest of the Internet. Given the recent history of Internet "worms" which, intentionally or not, invade local networks and waste resources, and the vulnerability of Unix systems to hackers who take it as a challenge to see how many systems they can log into, many system administrators prefer to connect only one system to the Internet and concentrate their efforts at maintaining security on this one system.

The presence of a firewall between your system and the rest of the Internet is a nuisance at times, but it does not prevent you from using `rlogin`, FTP, or any other TCP/IP application to access Internet resources. What you need to do is use `rlogin` to make a connection to the firewall, and then run FTP or other applications from there. Chances are your home directory is mounted on the firewall, so any file transfers you want to make can be done directly.

3.1.1 Testing Connections With 'ping'

A useful network application is a simple program known as *ping*. It sends a sequence of control packets to another host. Control packets are used to establish communication between two systems, but carry no data themselves. Every system that implements the TCP/IP protocol will respond to these control packets. The most common use of `ping` is to see if you can reach a remote host. For example, you may not know whether your system is behind a firewall. One way to find out is to try reaching a system you know about, such as `compsci` at Vanderbilt.

To invoke `ping` from a Unix command line, just type "ping" and the name of the system you want to reach, e.g.

```
% ping compsci.cas.vanderbilt.edu
```

`ping` will start sending a sequence of control packets at a rate of one per second. As the remote host responds, `ping` prints information about the amount of time it took to send the original packet and receive the response. Type `^C` to exit `ping` and return to the shell. Here is the output from a session that used `ping` to try to connect to `compsci`:

```
% ping compsci.cas.vanderbilt.edu
PING compsci.cas.vanderbilt.edu: (129.59.130.21): 56 data bytes
64 bytes from 129.59.130.21: icmp_seq=0 ttl=250 time=74 ms
64 bytes from 129.59.130.21: icmp_seq=1 ttl=250 time=46 ms
64 bytes from 129.59.130.21: icmp_seq=2 ttl=250 time=44 ms
^C
----compsci.cas.vanderbilt.edu PING Statistics----
3 packets transmitted, 3 packets received, 0% packet loss
round-trip min/avg/max = 44/54/74 ms
```

If `ping` does note produce any output within five seconds or so it probably means it was not able to establish a connection. Type `^C` to stop sending packets. `ping` will print its statistics about the number of packets sent and received. If it sent packets but received no responses then it failed to make a connection. Five seconds is probably enough time to wait. A host usually responds immediately to a control packet, and the round-trip time to Internet sites in Japan is around 250 ms, so if you do not see any output in five seconds it probably means you are not making a connection.

3.1.2 Network News

As is the case with e-mail, there are dozens of ways to interact with network newsgroups, and rather than explain one system here we will just give a broad overview of what newsgroups are and encourage you to learn about the newsreader software available on your local network.

Network news evolved out of mailing lists that allowed a group of people with common interests to share information. For example, if 20 people at different sites were all doing research on plankton layers in the ocean, each one could install a "mail alias" on their system so they could easily send mail to all their colleagues at once. A command of the form

```
% mail plankton-lovers < msg1
```

would have the system mail the file `msg1` to everyone on the mailing list named `plankton-lovers`. As the mailing list grows, however, there are two drawbacks to this approach: the e-mail system becomes needlessly overloaded when it sends the same message to several hundred users (many of whom might be at the same site), and users will be flooded with mail as more and more people start to send messages to the group.

The network news system is a central clearing house for these messages. Instead of mailing a message to everyone on the list, you post it to a newsgroup. The news software will store the message in the newsgroup on your local network, and also exchange postings generated at your site with postings from other sites. When people want to read the latest news, they use a news reader to find messages that have been posted since the last time they checked the newsgroup.

Mailing lists are still common, especially for small, specialized groups. As more people start subscribing to the mailing list (which is maintained by a volunteer member of the group) there are established procedures for converting the mailing list into an officially recognized newsgroup.

A word about mailing lists: there is a standard format for mail addresses for mailing lists. A name that ends in the string "`-request`" is the administrative mailing address. To subscribe to `plankton news`, you would send mail to `plankton-lovers-request`, *not* `plankton-lovers`.

An example of a mailing list that recently became a full newsgroup is the group for users of FrameMaker, a word processing system for Unix systems and personal computers. The mailing list was named `framers`. The Unix command

```
% mail framers < book.question
```

would send the contents of the file `book.question` via e-mail to every member of the newsgroup. New subscribers sent mail to `framers-request`; this mail would go only to the person who maintained the list. When the list grew to a few hundred members, a vote was taken, and the group applied to have a newsgroup formed. Now people who would like to share information about FrameMaker subscribe to the newsgroup named `comp.text.frame`.

As the previous example shows, newsgroups names are hierarchical and resemble domain names. One difference is that newsgroup names become more specific as you read from left to right. Here are some examples; the newsreader you choose will have a command to tell you which groups are received at your site:

One of the first things you will want to do when you subscribe to a newsgroup is find its list of frequently asked questions (*FAQ*). Almost every group has an FAQ list which is maintained by a volunteer. FAQ lists are posted to the newsgroup about once a month, and they are also available at various sites around the network.

3.1.3 Transferring Files

One of the most commonly used network applications is the file transfer program, or *FTP*, which copies files from remote systems. Examples of the kinds of things that are accessible via FTP are public domain software packages, technical reports and other research papers, newsgroup FAQ lists, and images (updated hourly) from weather satellites.

FTP is an interactive program. You start it by simply typing "`ftp`". After that you type commands which tell it which system you want to connect to, file transfer modes you want to use, and other options. Some of the more common FTP commands are listed in Figure 7.

To connect to a remote host, use the `open` command, e.g.

```
ftp> open compsci.cas.vanderbilt.edu
```

As a shortcut, you can specify the remote host on the command line when you start FTP.

Top Level	Subjects	Examples
comp	computers and computer science	comp.lang (computer languages)
		`comp.lang.prolog` (discussion of the language Prolog)
		`comp.lang.prolog.object` (object oriented programming in Prolog)
		`comp.sys.hp` (discussion of Hewlett-Packard computer systems)
sci	science and math	`sci.math` (general mathematics)
		`sci.math.num-analysis` (numerical analysis)
rec	recreation	`rec.sport.football.college` (US college football)

Table 1:

```
% ftp compsci.cas.vanderbilt.edu
```

is the same as starting FTP and then using `open`. When you have finished transferring files, use the `quit` command to terminate FTP.

In order to transfer files with FTP you need to have permission to use the remote machine. When FTP establishes a connection, it will ask you for a login name and password combination. If you have an account on the remote machine, use your login/password combination for that machine, just as if you were logging in via `rlogin`.

Machines that hold public domain software and other widely distributed files allow *anonymous logins*. Anonymous logins are like guest accounts: anybody can use them, but there are limits to what you can do. Simply type "anonymous" for the login name and then type your e-mail address when you are prompted for the password. The remote system will not verify your e-mail address, so don't worry if you make a typing mistake; giving your full e-mail address is just a courtesy to the people who maintain the public information in case they want to gather statistics about how widely their files are distributed.

One of the most important commands is the one that specifies the file transfer mode. FTP recognizes two different types of files, known as binary and ASCII. Binary files are things like compiled programs and large data sets. ASCII files are text files, for example sources of programs and messages posted to newsgroups.

A file transferred in binary mode is copied verbatim, bit for bit, and duplicated exactly on your system as it appeared on the remote system. Use this mode when you copy an executable program or some other file that your system recognizes. For example, if both your workstation and the remote host are Sun workstations, you can probably copy a working program from `/bin` of the remote host and install it in `/bin` on your computer and begin using it immediately. Another example is a compressed file, which is a collection of one or more files that has been run through a data compression algorithm. If you have the program that will decompress the file it makes sense to copy it, otherwise it will be useless to you.

Files transferred in ASCII mode are often transformed slightly. FTP knows about the text file formats of common systems and it translates among these formats as it copies a file. For example, lines of text in a source file are separated by a newline character (ASCII code 10) in Unix systems, but on an Apple Macintosh lines end with a carriage return (code 13) and on some systems lines are terminated by a carriage return and line feed combination. You should use ASCII mode when transferring text files because if you use binary mode and you copy a source file between two systems that have different file formats the copy will be unreadable.

A common mistake is to use the network file system to copy a program instead of FTP. For example, suppose you create a text file on a Macintosh which has an Appletalk link to the file server that also stores your Unix home directory. If you save the file in this disk directory (it will look like a folder on the Macintosh) the system will write the file OK, and you will be able to retrieve it later using the Macintosh again. However, if you now try to open the file with `emacs` on your Unix machine it will look like gibberish because the Macintosh application saved in Macintosh text format. If you want to open the file with a Unix application, copy it from the Mac to Unix with FTP and then open the copy.

It is often possible to recognize the type of a file (binary or text) by its name. This is especially true of files that have been placed in the public domain. The convention is to extend the name of a file with up to three letters. For example, documents that will be processed by the LaTeX text formatting program have file names that end in "`.tex`". Output files produced by LaTeX end in "`.aux`" (a text file with definitions of section numbers, etc. used in cross references) and "`.dvi`" (a binary file with the formatted document). Here are some common file extensions you will encounter:

.tar (binary) The output of the Unix utility named `tar` (tape archive). Use `tar` to extract the original files and directories (Section 3.1.4)

.z (binary) Output of a Unix text compression utility. Files that end in `.Z` are usually composites of other types. For example, it is a common practice to use `tar` to make an archive and then compress the archive, resulting in a file of the form `foo.tar.Z`. To extract information from this file, first decompress it and then use `tar` to extract files from the resulting archive.

.hqx (ascii) A text file created by `binhex`, a Macintosh application that creates text files from binary files. Use `binhex` to translate the `.hqx` file into binary. You will often need to use Stuffit or some other archive program to extract files from the new binary.

.dvi (binary) Device independent laser printer document generated by LaTeX, troff, and other text formatters. To print one of these documents, you need a program that translates the `dvi` format to a specific printer language. For example, `dvips` translates `dvi` to Postscript for the Apple LaserWriter and other Postscript printers.

.ps (ascii) A PostScript file. PostScript is a programming language, and `.ps` files can be created by a programmer using a text editor such as `emacs`, or by a document formatting system such as FrameMaker.

.tex (ascii) Source file (unformatted document) for the LaTeX text formatting system.

.c (ascii) C language source file.

.f (ascii) Fortran language source file.

.o (binary) Compiled output from a C or Fortran compiler (other languages also use the same naming convention for their output files).

.GIF (binary) Graphic image format recognized by a wide variety of programs, including `xv`.

Tools for compressing and uncompressing files are described in the next section. There are several other FTP commands worth knowing; refer to the manual pages for descriptions of these options. A transcript of a sample session with FTP can be found in Section 3.1.5, which also describes a tool for searching the Internet to find files.

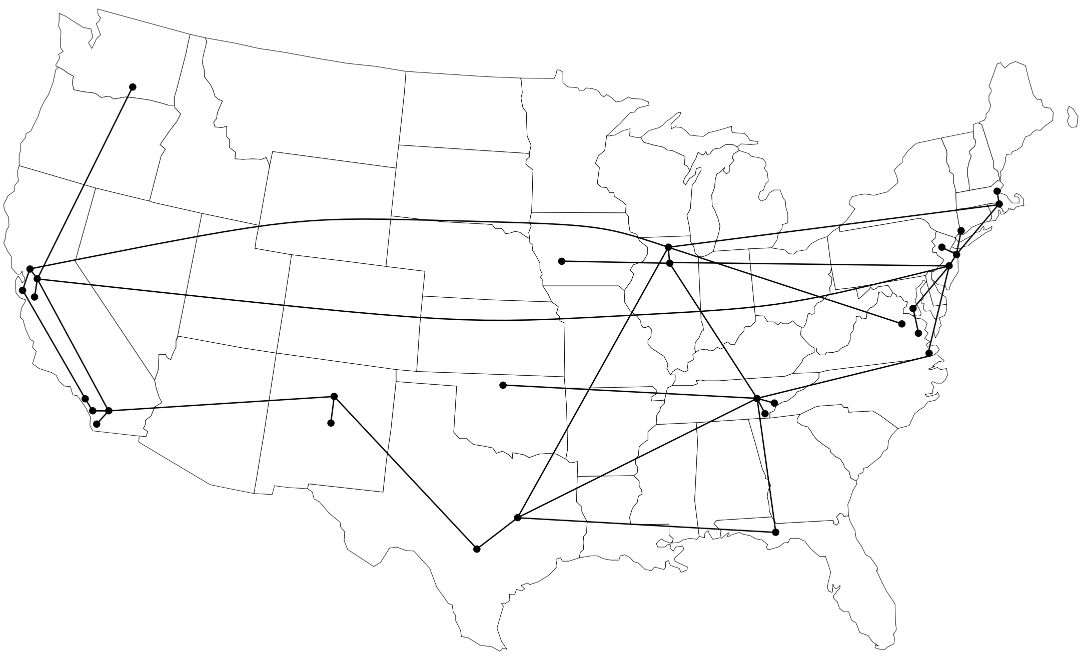


Figure 4: ESNET Sites in the US

Domain	Membership	Example
edu	Educational insitutions	mit.edu
gov	Government organizations	nsf.gov
com	Commercial organizations, corporations	hp.com
mil	Military organizations	navy.mil
net	Network administration	nic.es.net
org	Other types of organizations	aerospace.org

Common Country Codes

ca	Canada	jp	Japan
au	Australia	uk	United Kingdom
se	Sweden	de	Germany

Figure 5: Top Level Internet Domains

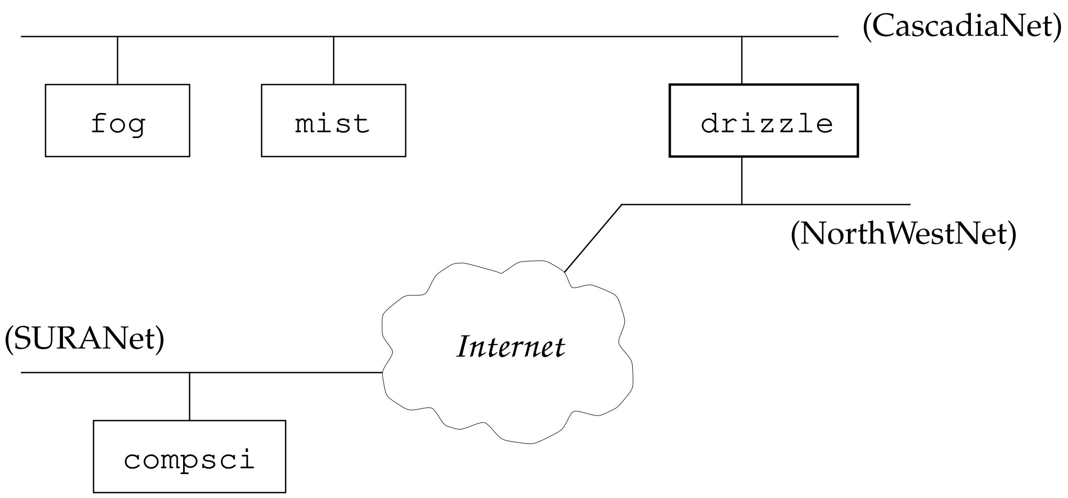


Figure 6: Local Network with Firewall

`ascii`	Use ascii mode transfer (text file)
`binary`	Use binary mode transfer (binary data, executable, or compressed file)
`bye` or `quit`	Exit FTP, return to shell
`cd`	Change directory (on remote machine)
`get <file>`	Transfer `<file>` from remote host to local host
`hash`	Print a hash mark (#) for every 1000 bytes transferred
`help` or `?`	Print the entire list of commands; type "help x" for help about command x.
`ls`	List files in current directory on remote host.
`open <host>`	Open a connection to `<host>`

Figure 7: Useful FTP Commands

3.1.4 Text Compression and File Archives

Much of the information available over the Internet has been collected in archives and compressed for more efficient storage and retrieval. In this section we will briefly mention the most common archiving and compressing programs. The commands we present here should be enough to help you decompress and extract most documents, but you should read the manual pages or other documentation if you end up using these programs frequently.

An archive is a collection of files that have been collected and stored in a single file (the archive) in order to make operations on the collection easier and more efficient. In many file systems a collection of several small files takes more space than one large file with the same data, so just making an archive and deleting the originals is worthwhile if you will not use the individual files for a while.

A compressed file is a file that has gone through a data compression algorithm. There is a lot of redundant information in most files, particularly text files. For example, characters in most systems are represented by 8 bits, and a text file with n characters occupies $8n$ bits on disk. It is possible to design a code that represents common letters with fewer bits. A coding scheme that uses a variable number of bits per character will take less space. The code for the letter e might be 01, and the code for the backquote character (‘) might be 00101000010101001. Even though the latter is much longer, the fact that it is so rare in most text files means files will be shorter.

The most common archiving program in Unix is `tar` (tape archive). Originally this program was used to write directories on tape, but now it is used to collect all the files in a directory (including all subdirectories) into a single file. To use tar, type “tar” followed by a set of letters that tell it what to do (with no spaces between the letters) followed by the name of the archive file.

When you first retrieve an archive, it is a good idea to find out what files are in it before you extract them. For example, suppose the archive file is named foo.tar. The command

```
%  tar  tf foo.tar
```

will print the names of every file in the archive. When you extract the files, they will be inserted in the current working directory, exactly as you see them. If a file has a name of the form `foo/bar`, then `tar` will create a directory `foo` and put a file named `bar` in it.

When you are ready to extract the files, use ‘`xvf`’ for the command, e.g.

```
%  tar  xvf foo.tar
```

Here is the transcript of a session with `tar`. Comments about Unix commands are in italics. Suppose we have just copied two archives, named `foo.tar` and `letters.tar`, and we want to extract the files in these archives. It is always a good idea to isolate files from an archive in their own subdirectory, so if the archive does not already have them in one directory we will make a new subdirectory, copy the archive to it, and then extract the files there.

```
% ls -l *.tar
-rw-r--r-- 1 csep users 10240 Aug 21 10:45 foo.tar
```

```
-rw-r--r-- 1 csep users 10240 Aug 21 10:46 letters.tar
% tar tf foo.tar
```
(see what's in foo.tar)
```
foo/
foo/foo
foo/bar
```
(foo.tar contains its own subdirectory)
```
foo/baz
% tar xvf foo.tar
```
(go ahead and extract the files)
```
x foo/foo, 4 bytes, 1 media blocks.
x foo/bar, 4 bytes, 1 media blocks.
x foo/baz, 4 bytes, 1 media blocks.
% tar tf letters.tar
```
(see what's in letters.tar)
```
alpha
beta
gamma                          (just a set of three files)
% mkdir letters
```
(make a new directory for these files)
```
% cd letters
% tar xvf ../letters.tar
```
(extract files in the new directory)
```
x alpha, 6 bytes, 1 media blocks.
x beta, 5 bytes, 1 media blocks.
x gamma, 6 bytes, 1 media blocks.
% cd ..
% ls -F
foo/                     letters/
%
ls letters
alpha                    beta                     gamma
% ls foo
bar                      baz                      foo
```

The easiest way to create an archive is to collect all the information in a single directory and then archive the entire directory. For example, suppose you have collected everything you want to save (simple files and other directories) in a directory named `oldstuff`. The following command will make an archive named `oldstuff.tar`:

```
tar  cvf  oldstuff.tar  oldstuff
```

Building a new archive can be tricky. A common mistake is to put the new archive in the directory you are trying to save. For example this command tells `tar` to collect everything in the current directory (named '.' in Unix) and put it in `oldstuff.tar`:

```
tar cvf oldstuff.tar .
```

(aaargggh! don't do this!)

Unfortunately `tar` will create the new archive file and then start collecting things to put in it. Since you told it to include everything in the current directory, and the current directory now includes the new archive file, tar will be caught in an infinite loop when it adds the archive to itself.

There are other `tar` commands, including commands to append files to existing archives. Carefully read the man pages for `tar` (and preferably talk to someone who has built archives) before you build your own archive file.

The Unix utilities that compress and decompress files are based on the Lempel-Zev text compression algorithm [7]. A common pair of programs to compress and expand files are called *compress* and *uncompress*, respectively. Both replace their input files by their output. For example, if you compress a file named `foo`, it will be deleted and in its place you will find `foo.Z`. Here is an example of how to compress and expand a file:

```
$ ls -l networks.*
-rw-r--r-- 1 conery faculty 564433 Aug 18 15:38 networks.mif
$ compress networks.mif
$ ls -l networks.*
-rw-r--r-- 1 conery faculty 129856 Aug 18 15:38 networks.mif.Z
$ uncompress networks.mif.Z
$ ls -l networks.*
-rw-r--r-- 1 conery faculty 564433 Aug 18 15:38 networks.mif
```

A newer and more flexible set of programs is available from the Free Software Foundation.[2] `gzip` and `gunzip` are similar to `compress` and `uncompress`, but do a better job of compressing files (they claim a typical reduction of 60% to 70% in the size of English text). `gunzip` will also expand files created by compress and other programs. Here is an example of how to use `gzip` and `gunzip`:

```
$ gzip networks.mif
$ ls -l networks.mif*
-rw-r--r-- 1 conery faculty 66571 Aug 18 15:38 networks.mif.z
$ gunzip networks.mif.z
$ ls -l networks.mif*
-rw-r--r-- 1 conery faculty 564433 Aug 18 15:38 networks.mif
```

`gzip` reduced the size of this text file from 564,433 bytes to 66,571 bytes.

[2] This is the same group that produced GNU emacs, the GNU C compiler(gcc), and several other widely used, high quality Unix programs. Use Archie () to find an anonymous FTP site for these other programs and for information about the foundation and its goals.

3.1.5 How to Find Files

Before you can use FTP you need to know exactly where to find the file you want to copy. If you only know a file's name but not where it is, there is a service that can help you find fit it. A program named archie will search anonymous FTP servers looking for files with names that match a search string.

For example, suppose you are interested in a programming language named SISAL. The following shell command will invoke archie and ask it to search for files with the string "sisal" in their titles:

```
% archie -s sisal
```

The -s option tells `archie` to ignore cases in the names, i.e. the search will match "sisal", "Sisal", and "SISAL". Refer to the man page for other options.

Here is the result of our search for titles on the SISAL language:

```
Host chook.cs.adelaide.edu.au
 Location: /pub
 FILE -rw-r--r-- 43933 Aug 14 1992 sisal_conf_1.ps.Z
 FILE -rw-r--r-- 105253 Aug 14 1992 sisal_conf_2.ps.Z
 FILE -rw-r--r-- 85395 Jul 28 1992 sisal_ref_man_1.2.tar.Z
 FILE -rw-r--r-- 581733 Jun 30 14:15 sisal_ref_man_2.0.ps.Z
 DIRECTORY drwxr-xr-x 512 Jun 29 19:35 sisal_reports
 DIRECTORY drwxr-xr-x 512 Aug 13 02:44 sisal_types
Host faui80.informatik.uni-erlangen.de
 Location: /pub
 DIRECTORY drwxr-xr-x 512 Apr 16 1992 sisal
 Location: /pub/sisal
 FILE -rw-r--r-- 7136 Apr 16 1992 sisal-mode.el.Z
 FILE -rw-r--r-- 5128 Apr 16 1992 sisal.info.Z
Host beethoven.cs.colostate.edu
 Location: /pub
 DIRECTORY drwxr-xr-x 512 Mar 9 18:49 SISAL
 Location: /pub/SISAL
 FILE -rw-r--r-- 5422 Aug 4 1989 sisal.README
 FILE -rw-r--r-- 587317 Aug 4 1989 sisal.tar.Z
Host cs.utk.edu
 Location: /pub/pozo/conferences
 FILE -rw-r--r-- 7819 Mar 10 1992 SISAL.04.92
```

Armed with this information, we can now use FTP to connect to the server at Colorado State University and see what is there. In the following transcript, prompts and other output from Unix, FTP, and `archie` are printed in italics. Comments are also printed in italics on the right side of the lines.

```
% ftp beethoven.colostate.edu
Connected to beethoven.CS.ColoState.EDU.
220 beethoven FTP server (SunOS 4.1) ready.
Name (beethoven.colostate.edu:fred): anonymous
331 Guest login ok,
send ident as password. Password:...
230 Guest login ok, access restrictions apply.
ftp> cd /pub/SISAL                    (archie said to look here)
250 CWD command successful.
ftp> ls                               (print a list of all files here)
200 PORT command successful.
150 ASCII data connection for /bin/ls (128.223.8.8,30398) (0 bytes).
TOC
TR.89-108.ps.Z
V2- man.ps.Z
biblio.ps                   (this looks interesting)
fe.tar.Z
gangd.README
gangd.tar.Z
s2.manual.ps.Z
seqb-fe.tar.Z
sisal.README                (always look at these README files)
sisal.tar.Z
sun3-fe.tar.Z
226 ASCII Transfer complete.
155 bytes received in 0.05 seconds (3 Kbytes/s)
ftp> get biblio.ps
200 PORT command successful.
150 ASCII data connection for biblio.ps (128.223.8.8,30391) (85761 bytes).
226 ASCII Transfer complete. local: biblio.ps remote:
biblio.ps 89235 bytes received in 2.7 seconds (32 Kbytes/s)
ftp> quit
221 Goodbye.
```

In this session we did an anonymous login to the server and then changed directories to `/pub/SISAL` (where `archie` said we'd find some things on SISAL). We then listed all the files in the directory. The PostScript file `biblio.ps` looked interesting (it's probably a list of papers on the language or tech reports on the implementation). Before getting anything else, though, we should probably get the README file and find out exactly what is in each file. No sense in transferring megabytes of information if it turns out it is not what we are looking for.

3.1.6 Browsing

Archie is a useful complement to FTP since it helps you find files. However, it is limited to matching search strings with the names of files. The next step in complexity is searching for files with contents that match a search string. There are three such services available to Internet users. We will give a very brief description of each one here; you will need to read the man pages or an Internet resource book (e.g. [4]) before you use them. Gopher is a system developed at the University of Minnesota. When you start `gopher` you will see a list of information services. When you select a service `gopher` will retrieved the information you want. One reason `gopher` is so useful is that it can retrieve and display a wide variety of types of information. For example, if you select a satellite image, `gopher` will start `xv` and use it to display the image when it arrives. Another interesting feature of `gopher` is that what you select can also be an interactive program, not just a static piece of information. You can select the on-line card catalog of your university library, for example, and be connected to the information service for the library. From that point on, until you exit the service, you would be interacting directly with the system at the library. Normally `gopher` is similar to `archie`, in that it displays a list of titles to you and you need to guess which one has the information you are looking for. However, there is a program named `veronica` that is available through `gopher`. You can give `veronica` a string and it will search files available through other `gopher` systems to find those that contain your search string.

WAIS (wide area information service) is based on indexed searches. Rather than searching documents, it searches indexes, much the way you would search a card catalog in a library instead of searching through each book. When you first use WAIS to begin a search, you will search a list of indexes to find one that would be likely to describe documents you are looking for. In the library analogy, this search would help you identify the library (main library, science library, public library...) that is most likely to have the information you want. Having found an index, you can then search it to find documents based on keywords. WAIS has several interesting features, including the ability to save previous searches so it becomes easier to use the more you use it, but the searches are rather limited (e.g. you can't say "search for papers with keywords 'logic' and 'programming' but not 'Prolog''"). WAIS is available through gopher, so if you already know the `gopher` interface it should be straightforward to expand your view of the network and use `gopher` to start searching "card catalogs" via WAIS.

The World Wide Web (WWW) is a new project that was originated at CERN, the European high energy research facility in Switzerland. WWW is based on the concept of *hypertext*. A hypertext document allows users to wander around in unstructured and unpredictable ways, instead of reading from top to bottom and one page to the next as in a normal linear text. A hypertext *browser* is a piece of software that lets users click their mouse button on a piece of text or part of a picture and then moves to a portion of the document that explains the selected item. For example, if you are looking at an equation in a science text, you could click on a variable in the equation and the browser would bring up a window that explains what the variable represents and its range of values.

There are several hypertext systems for small computer systems, including the well-known Hypercard for Macintosh systems. WWW is a combination of hypertext browser and Internet archive service. By clicking on portions of a document you will be able to initiate searches throughout the Internet for more information on the selected item.

3.2 Trouble Shooting

There are three general classes of error messages you might see when you use Internet services. Although the text of the error message might vary from one application to the next, they are generally similar, and you should be able to recognize the problem from the message.

The first type of error occurs when you try to make a connection. The message typically says "unknown host" and includes the name of the host you are trying to connect to. First double-check the spelling of the host name and domains to make sure you didn't type them wrong. If you have the name correct, there are two other reasons you might not be able to make a connection. The host name might not be known to the name server your system used, or the host might not exist. In the former case you might try again, but use the numeric Internet address if you know it.

The next type of error may or may not be accompanied by an error message. It occurs when the host name is valid but the connection is still not being made. If you wait long enough, the application might quit with an error message like "connection timed out" or "route to the remote host is not available." There are several reasons for this type of message: the remote host might be down for repairs, there might be network connection problems somewhere between your system and the remote host, or your system or the host might be behind a firewall . It may seem strange that hosts that are not directly connected to the Internet will have their names and addresses published in name server tables, but there are many reasons why the names might be there. Those hosts can still use TCP/IP applications within their local networks, and the local name servers might include those names when queried by global name servers, or the names might have been published in global name servers before the systems were placed behind a firewall.

The third type of error is a "connection refused" error. This will occur when you are able to connect to a remote host and it is up and running but you are not allowed to use it for some reason. Those reasons include the fact that you are not authorized (check to make sure you typed your login name and password correctly), or in the case of anonymous FTP servers there might be an upper limit on the number of guest users at any one time or the hours for public access might be restricted. In these cases the error message is usually accompanied by an explanation of why you have not been granted access.

See exercises 1, 2, 3, and 4.

---

4 Mosaic and the World Wide Web

The World Wide Web (WWW) is a growing network of information resources. Started at CERN, it has rapidly grown to include hundreds of information servers at Internet sites around the world. To use the Web you need to start a WWW *client* program on your workstation. The client will use FTP, Gopher, WAIS, or other Internet communications programs to get information from *servers* at known sites.

This appendix to the Computer Networks chapter describes NCSA Mosaic, a public domain client program for X window systems and the Macintosh system. There are several other Unix clients, including tkWWW, a program built using the Tk/Tcl interface kit. There are also several microcomputer clients, including versions of Mosaic for the Macintosh and IBM PC compatible systems. In this section we will assume you are using Mosaic to access the Internet from a Unix workstation.

4.1 Hypertext

The Web is based on the concept of *hypertext*. A hypertext document allows readers to access information in the document in almost any order, unlike a traditional linear text which is intended to be read from beginning to end. For example, suppose you are reading a research paper on computational fluid dynamics. A traditional paper might have an introduction, a definition of terms, some equations, some algorithms that implement the equations, and a discussion of the performance of the algorithms on one or more supercomputers. If you are an expert in this area, you might want to skip the definitions and equations and go right to the algorithms. As you are reading that section you might decide you shouldn't have skipped the equations after all, and want to go back and read them. Hypertext is intended to make these random movements through the document easier for the reader. The idea is to give the reader more of a role in deciding how to use the document and to make the document flexible enough to be used in several ways (e.g. as a textbook the first time it is read and later as a reference book).

Hypertext documents are made possible by computer technology. Documents are displayed on a computer console, and readers use a mouse and/or keyboard to navigate through the document. Random movements between sections are implemented by *links* which are activated when the reader clicks a mouse button. The program that displays a hypertext document and uses mouse clicks to move around in the document is sometimes called a *browser* or *navigator*.

A more interesting example of using links to navigate through text is a case where a person who is new to an area comes across an equation and needs to learn more about it; the browser could connect them to a completely separate document that has background information the reader could study before resuming the main text. The really interesting possibilities for hypertext will take full advantage of the underlying computer system. A text that introduces equations might have links to a symbolic math package such as Mathematica which could help students derive the equations from initial conditions or assumptions. Readers could ask the system to plot the equation for a given range of inputs, and then plot another equation for comparison. Complex figures could be scaled and rotated by computer aided design programs instead of lying flat on a page. Data sets could be fetched over the network from the authors, and readers could perform their own statistical analyses. In general, readers will be able to *interact* with the document rather than simply consume information.

Hypermedia generalizes the notion of hypertext to include sound and interactive video in documents. For example, a document on Mozart could include portions of his music which readers could listen to simply by clicking the mouse on a link in the main document. Several publishers have begun to produce hypermedia versions of their products. For example, there is a version of Grolier's Encyclopedia available on CD ROM that has sound and movie clips included with the text. If you read about US presidents, you will see pictures of the presidents. Some of these pictures have links that will play a movie (complete with sound) showing part of that president's inaugural address.

Mosaic is a hypermedia browser for X window systems. It displays information in a window on a workstation console, and uses keystrokes and mouse clicks to navigate through the document. Documents can be read as normal text files by using a scroll bar and mouse buttons to move back and forth in the text. Clicking the mouse on a link will either move you to another location within the current document or call up a new document. If necessary, Mosaic will use the Internet to fetch a document when you select it; for example, if you are viewing a document on an art exhibit that has a link that refers to a particular piece, when you click the mouse over the link Mosaic will invoke FTP to copy the image and then invoke `xv` to display it in another window. It is the ability to call external software (such as FTP and `xv`) that makes Mosaic a hypermedia software package: if you select a movie or piece of music, Mosaic will invoke software to play the movie or sound.

4.2 Starting Mosaic

Mosaic is available at ftp.ncsa.uiuc.edu in /Mosaic. Both source code and binaries (for Sun, SGI, IBM RS/6000, and DECstation Ultrix) are available. DEC Alpha binaries are now being distributed also. (You do not have to have Motif installed on your system to run Mosaic if you pick up a precompiled binary. You do, however, need Motif 1.1 to compile Mosaic.)

Before you start Mosaic, make sure `xv, ghostview,` and other auxiliary programs are installed on your system. The easiest way to check is with the `which` command:

```
% which xv
xv is /local/apps/X11/bin/xv
```

If `xv` is not installed (or not in your path) which will print something like "xv not found." The other programs Mosaic uses to display images are `ghostview` (to display Postscript files), `xdvi` (for LaTeX output), and `mpeg_play` for MPEG format movies. To see the complete list of these external programs, refer to the mosaic on-line documentation under the subject of "X resources."

If one of the auxiliary programs is not available on your system, but you do have an alternative program that does the same thing, you can change your `.Xdefaults` (X window resource database) file so Mosaic will use the other program. For example, Silicon Graphics workstations have a program named `ipaste` for viewing images in a format known as RGB. If you want to use `ipaste` instead of `xv` you can add a line to `.Xdefaults`. Refer to the on-line documentation for instructions.

If you cannot install some or all of the auxiliary programs you can still run Mosaic, but you will not be able to view all the information in every document. You will be limited to reading only text files and viewing in-lined images, but Mosaic can still be a valuable tool for finding information over the Internet.

To start Mosaic, just type "`xmosaic`" in a shell window. You will see a top level window displaying the initial startup document, which is known as the *home page*. The default home page from NCSA is described in the next section.

As Mosaic runs it needs to build temporary files. Some of these can be fairly large. By default Mosaic will build them in `/tmp` (a Unix directory that every user has access to). If there is not enough space in `/tmp`, Mosaic will not be able to display some images. One indication that you don't have enough space is that you will see the NCSA logo instead of a bitmap image in documents. The Mosaic Demo Document has several such bitmaps; if you see the NCSA logo instead of images in this document then you probably don't have enough free disk space in `/tmp`.

To fix this problem you can specify a different directory for Mosaic to use for its temporary files. One method is to specify the directory when you start Mosaic:

```
%  xmosaic  -tmpdir /home/users/fred/tmp
```

4.3 Home Page

The home page is the first document you see when you start Mosaic. It has links to commonly accessed items and other useful information. The default home page from NCSA is shown in Figure 8. This is the document you will see unless you tell Mosaic to use a different home page (either through a command line argument when you start Mosaic or a specification in your `.Xdefaults` file) or your system administrator has installed a special local home page.

A Mosaic window has four parts. At the top is the menu bar with common operations, such as opening new documents, searching for information, or accessing commonly used information. Below the menu bar is the title and source of the current document. Next is the document itself, and on the bottom are buttons that perform other common operations.

The document area of the window is the main display. Links are identified in one of two ways: either they are underlined and printed in blue or purple text, or they are images surrounded by a box. To activate a link, just move the mouse over it and click the left button. For example, on the NCSA home page there is a link named FAQ list (printed in blue near the bottom of the page). If you click on this link, Mosaic will fetch the list of frequently asked questions (FAQ) about Mosaic. When you have seen enough of this list and want to return to the home page, click the Home button at the bottom of the window.

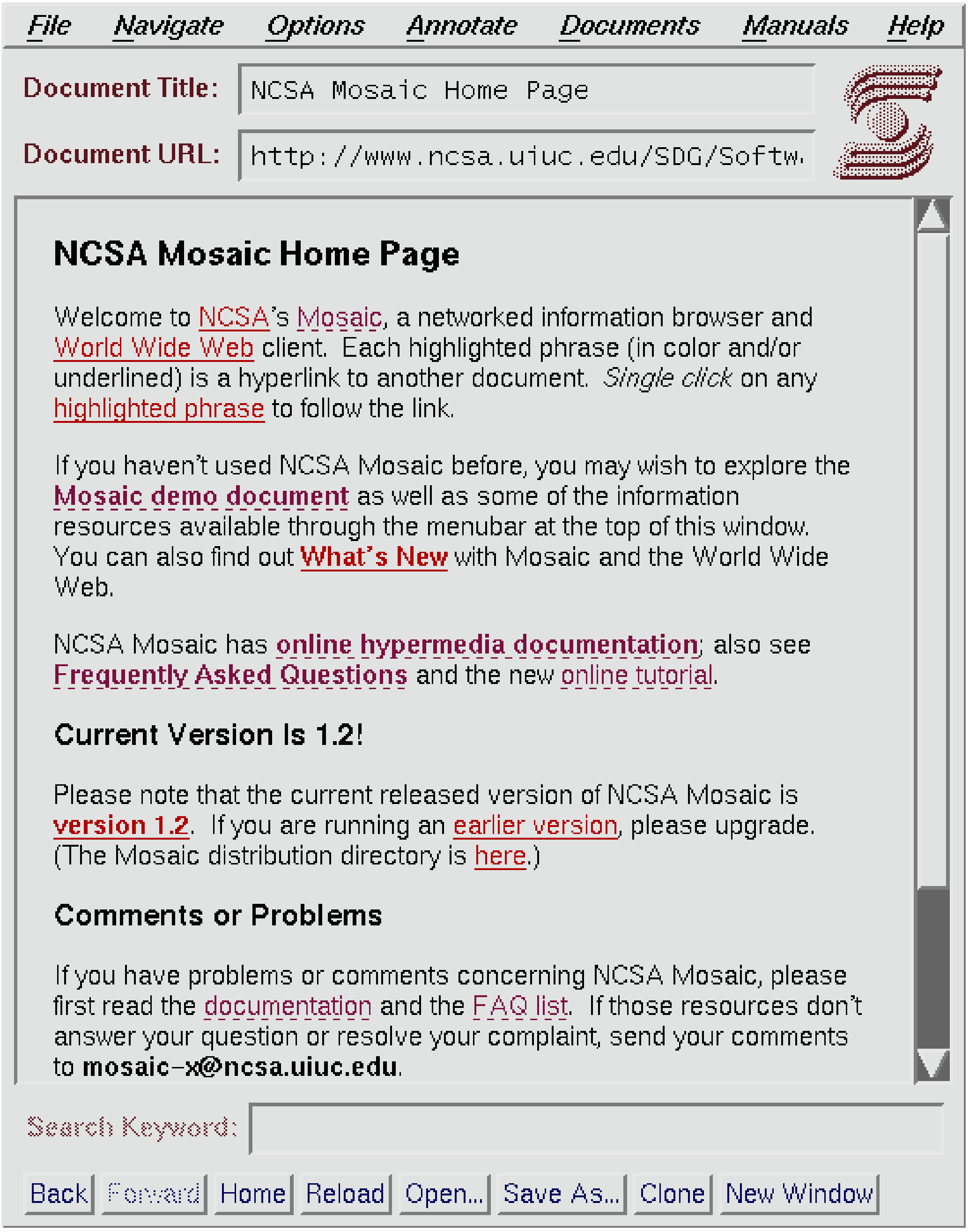


Figure 8: NCSA Home Page for version 1.2

(You might also see a link to something interesting in the FAQ document; clicking on that link will call up more information.)

Mosaic maintains a history list which is a record of all the documents you have accessed. To help you keep track of which links you have followed, if a link points to a location in a document in the history list it is displayed in purple; otherwise it is blue. Note that when you first open a document some links will be purple. That usually means those links point to sections of the same document. The distinction between blue and purple links is not important (other than the fact that blue links point to documents that might have to be fetched over the network, which might take some time). In practice blue links serve to identify documents you have not visited yet and thus help you navigate through more complex subjects.

To navigate with Mosaic, you need to know the following simple commands. They can be executed from either the Navigate menu or from the control panel at the bottom of the viewing window.

Back: Move back one document in this Document View's history list. This action is only possible if a previous document exists at the current position; otherwise, the menu bar entry (and corresponding bottom button) will be insensitive.

Forward: Move forward one document in this Document View's history list. This action is only possible if a subsequent document exists at the current position; otherwise, the menu bar entry (and corresponding bottom button) will be insensitive.

Home Document: Move to the home document. (This is the document that NCSA Mosaic accesses when it first starts up.) Instead of trying to explain any more about Mosaic's menus and buttons, we urge you to read the on-line documentation that is accessible through the Manuals and Help menus. The documentation is excellent, and it is much better to learn Mosaic by using it than by reading about it!

4.4 Uniform Resource Locators

One of the main reasons to learn about Mosaic and WWW is that it subsumes other Internet tools. In another words, you can use Mosaic to search Gopher databases and WAIS indexes, retrieve files with FTP, or log in to remote systems via Telnet. This flexibility to use a wide variety of information formats relies on the notion of a *uniform resource locator*, or URL.

You can think of a URL as a generalization of the concept of a file name. Within a given computer system you access information by its file name. To open a file for editing you tell the editor the name of the file you want to open, or to save data from a program you tell the system the name of a file in which to store the data. A simple file name will not suffice when specifying the location of information on the Internet because there are different kinds of computers, each with their own file name conventions, and there is a wide variety of types of data and methods for accessing it. For example, a WAIS server uses an index of files, sort of the electronic equivalent of a library's card catalog, but Gopher and FTP access files directly by name.

A URL can be thought of as an "Internet file name." It identifies the host that has the information, the format of the information, and a protocol to use when retrieving the information. Most URLs have the following structure:

```
<form>://<site>/<file>[#<anchor>]
```

The first field defines the format of the information and the protocol that must be used to access it. Examples are `file`, which means the information is simply in a file and it can be retrieved with FTP; `gopher`, which means the remainder of the URL is the ID of a Gopher server; and http, which stands for "hypertext transfer protocol." A URL that begins with `http` is the ID of a WWW server that runs a protocol specially designed for efficient transfer of hypertext documents.

The `<site>` field is the Internet domain name of a system that holds the information. Following the site is a file name (in the syntax of that machine), and finally there may be an anchor, which is a location within the file.

As an example, here is the URL for the NCSA home page document:

```
http://www.ncsa.uiuc.edu/SDG/Software/Mosaic/NCSAMosaicHome.html
```

When you start Mosaic it uses this URL to connect to the http server at NCSA (the machine named `www.ncsa.uiuc.edu`) and retrieve the current version of the home page, which is in the file named `/SDG/Software/Mosaic/NCSAMosaicHome.html`.

The reason you need to know about URLs is that they allow you to access information directly without traversing a path from the home page. If you hear about an interesting new document, for example a hypertext presentation of an art exhibit, you can view it by using the Open... command in the File menu. When you activate this menu option you will see a prompt to type in a URL name; when you are done Mosaic will open the document by using the URL to find the document on the Internet.

Under the Navigate menu, there is an option called the Hotlist. This brings up a dialogue box in which users can add documents to their person Hotlist. Later, a user can jump straight to these documents without remembering the URL or how to get there through other links. Refer to the on-line documentation for more information about URLs.

4.5 Making Your Own Documents with HTML

In the days before computerized word processing systems, authors used a standard set of symbols to mark drafts of books and papers to let typesetters know how to change the draft. Authors who use a computer-based text formatter also use a well-defined set of symbols. These symbols, which comprise a *mark-up language*, are embedded in the paper along with the text. A text formatter works very much like a compiler, reading a "program" written in the mark-up language and generating output for a laser writer or bitmap display. Well-known examples of text formatting systems based on mark-up languages are `nroff` and `troff` (early Unix programs) and LaTeX[3].

[3]Another way of formatting documents on a computer is the "what you see is what you get" (WHYSIWYG, pronounced "whizzy-wig") type of text processor, such as Microsoft Word or FrameMaker. In these systems authors make changes directly to the document and the results can be seen immediately.

As an example, suppose you want to produce the following sentence in a document:

> There are a^nn possible n-letter words constructed from
> an alphabet of a symbols.

The input to LaTeX that would generate this line is:

```
There are $a^n$ possible {\it n}-letter
words constructed from an alphabet of {\it a} symbols.
```

The dollar sign, caret, backslash, and braces are all part of the mark-up language. LaTeX interprets the symbols between dollar signs to be a mathematical equation, and within an equation the caret symbol means "the next character is a superscript." The string `it` means "print the following text (up to the next closing brace) in italics."

The mark-up language for Mosaic documents is the Hypertext Mark-up Language (HTML). HTML is extremely simple compared to LaTeX or Postscript. There are commands to print text in different styles (e.g. boldface or italics) and to create section headings, but little else in the way of formatting the output. The main distinction of HTML is that it has a way to define links to other documents.

To continue the example above, suppose you want to make the word "alphabet" a link to a glossary that defines the word. In HTML you would write

```
... from an <A>alphabet</A> of <I>a</I> symbols ...
```

HTML uses angle brackets to denote formatting commands. In this example, the item between `<A>` and `</A>` becomes an *anchor* (the beginning or ending of a link).

Just defining the text as an anchor is only half of what you want, however. What you need to do is tell Mosaic where to go when the user clicks on the word "alphabet." The following line says "fetch the document `grammars.html` from an HTTP server at `ftp.cascadia.edu` and position the entry for alphabet at the top of the window":

```
... from an
<A HREF=''http://ftp.cascadia.edu/grammars.html#alphabet">
alphabet</A> of ...
```

This `<A>` command has an "`href`" (hypertext reference) field, which identifies the anchor as the front part of a link. It will be printed in blue and underlined when the document is displayed. The argument of `href` is a URL for the document that should be opened when the user selects the link. There are also ways to specify link targets in local documents or in other sections of the current document.

Short and simple documents like a home page can be written directly in HTML. Just use your favorite text editor to write the HTML "program" and then open the program with the Open Local... command in the File menu.

For more complex projects you should look into one of the automatic translation program that are currently being developed. A group the University of Leeds in England is working on a program that will generate an HTML file from LaTeX input. This should be very useful for people who already have a large body of text in LaTeX and want to convert these documents to hypertext. We leave it as an exercise to use Mosaic to track down information on this and related projects.

See exercises 5, 6, 7, 8, and 9.

---

5 Tutorials in Using Xarchie and Xnetlib

Tutorials on using Xarchie and Xnetlib are available using Mosaic when to view this chapter.

6 Exercises

Exercise 1 *A map of your local network.*

Draw a map of the workstations, servers, and routers on your local network. For starters, you might see if there is a file named `/etc/hosts` on your workstation.

Exercise 2 *The use of an Internet service.*

Use any of the Internet services (`archie, gopher, WAIS, WWW`) to find the following information. Items marked with a bullet (•) are known to exist (i.e. we've found them ourselves). The other things should be out there, according to Krol [4] or pointers in the files we found.

• Find the names of the sites on ESNET. Fill in the map in Figure (4) with the names of each site.

Find information on Egypt in the *CIA World Factbook*.

• Get the latest weather for your area. You should be able to find (1) a text file with the latest report from the National Weather Service; (2) a weather map showing high and low pressure areas and thunderstorm activity, and (3) an image from a geostationary weather satellite. Use `xv` to display (2) and (3).

Get a copy of the *NorthWestNet User Services Guide* (much of the information in here is probably relevant for other networks, too).

Look up the definition of "hacker" in an on-line dictionary. There are several on the Internet, but your university or library may also have a local dictionary.

Look up the definition of "hacker" in the *The New Hacker's Dictionary*.

• Find the frequently asked questions list for `rec.crafts.brewing`.

Find a list of brewpubs in Eugene, Oregon (hint: start with the FAQ for `rec.crafts.brewing`).

Many campuses use `gopher` to organize on- line databases. Look around to see if you can find a course schedule, catalog, or campus phone book for your school.

Look up the e-mail addresses of two or three people who contributed chapters to the Computational Science Education Project.

- What resources are available at the San Diego Supercomputer Center?

Exercise 3 *The use of* `ping` *on the Internet.*

`ping` (section on testing connections with *ping*) prints a report that includes the time it takes to receive a control packet from the remote host. Can you use this information to estimate the network latency (*aka* response time)? Can you use it to estimate throughput (*aka* bandwidth)? If not, can you think of a way to modify `ping` so it could help you measure response time and/or bandwidth?
`[Click here to go to the section on testing connections with` *ping*
`(Section-3.1.1).]`

Exercise 4 *An evaluation of communication over the Internet.*

The authors of the original Ethernet paper [5] gave the following model to predict the performance of an ethernet in terms of the percentage of time data packets were on the net. Let Q be the number of systems connected to the net. A, the probability that one system gains exclusive access at any point in time, is

$$\begin{aligned} A &= Q \cdot \frac{1}{Q} \cdot (1 - \frac{1}{Q})^{Q-1} \\ &= (1 - \frac{1}{Q})^{Q-1} \end{aligned}$$

(Q systems have $1/Q$ probability of gaining access at a time when the other $Q-1$ systems, each with probability $1 - 1/Q$, decides not to access the net). This definition is used to define the mean time a station must wait before it gains access. The time unit, known as a *slot time*, is the amount of time a system waits before retransmitting when it finds the network busy. The probability of waiting 0 slots is equal to A; the probability of waiting one slot time is $A \cdot (1 - A)$, etc., so the probability of waiting exactly i slot times is

$$p = A \cdot (1 - A)^i$$

The mean of the probability distribution over all i gives the expected number of slot times a system will have to wait before acquiring exclusive access to the ether:

$$W = (1 - A)/A$$

Finally, the efficiency of the network in this model is the percentage of time there are packets on the net. Let P be the packet size, in bits (the ether is a bit-serial medium), let C be the peak capacity (in bits per second), and let T be the length of a time slot. Then the efficiency of the ethernet is

$$E = \frac{P/C}{W \cdot T + P/C}$$

i.e. the time to send a packet as a fraction of the total time.

(a) Construct a table that gives the efficiency of the ethernet as a function of the number of systems connected to the net (Q) and average packet size (P). Some reasonable values for C are 10,000,000 (10 Mbps, true of most current ethernet installations) or 100,000,000 (100 Mbps for new technology). For the slot time T use 16 microseconds, which is the value used by Metcalfe and Boggs.

(b) For a given number of systems on the network, is the network more efficient with long packets or short packets? Is this what you would expect?

(c) Is 16 microseconds a reasonable slot time? Explain.

(d) Given the way researchers actually use a local area network, with rlogins from an X terminal to an X client running on a compute server, e-mail, FTP transfers, etc., what do you think the actual distribution of packet sizes will be?

(e) If you have access to a network analyzer, see if you can record some data about packet sizes and efficiency for a segment of your local network. How do the measurements compare to the performance model?

Exercise 5 *Use Mosaic to find and retrieve copies of the two papers by Andreesen.*

Exercise 6 *Use Mosaic to find information about the LaTeX to HTML translator.*

Exercise 7 *Create your own home page using HTML.*

Design your own home page, and write an HTML document to implement it. Open it with Mosaic and verify that it works as you intended. Start with a simple text that has links to some commonly used information (for example your personal on-line phone list). Next try some fancier things like including a telephone icon next to the link for your phone list.

Exercise 8 *Modify* `.Xdefaults` *so your new home page is selected whenever you start Mosaic.*

Exercise 9 *Document a program using HTML.*

Use HTML to write the documentation for a program you are working on for this class. Your top level page should have a brief overview of what the program does and then an outline of major topics. Each item in the outline should be a link to another page in the document which goes into more detail. Put links from the hypertext to the actual code so readers can look at the code as they are reading your explanation. Can you design a link that will start the program and run it on a set of test data?

References

[1] Andreesen, M. *Getting Started with Mosaic.* National Center for Supercomputing Applications, May, 1993.

[2] Andreesen, M. *NCSA Mosaic Technical Summary.* National Center for Supercomputing Applications, May, 1993.

[3] Comer, D.E. *Internetworking with TCP/IP*, Vol. 1, Second Ed., Prentice Hall, 1991.

[4] Krol, E. *The Whole Internet User's Guide & Catalog.* O'Reilly & Associates, 1992.

[5] Metcalfe, R.M., and Boggs, D.R. Ethernet: distributed packet switching for local computer networks. *Comm.* ACM 19(7), July 1976, pp. 395–404.

[6] Perlman, R. *Interconnections: Bridges and Routers*, Addison Wesley, 1992

[7] Welch, T. A. A technique for high performance data compression. *IEEE Computer* 17(6), June 1984, pp. 8-19. [`A reference to the Lempel-Zev compression technique cited in the man pages for compress and uncompress.`]