A Dial-up Network of UNIX Systems

D. A. Nowitz

M. E. Lesk

Bell Laboratories
Murray Hill, New Jersey 07974

ABSTRACT

A network of over one hundred UNIX† computer systems has been established using the telephone system as its primary communication medium. The network was designed to meet the growing demands for software distribution and exchange. There are several features that helped make it a successful system:

- The start-up cost is low. A system needs only a dial-up port, but systems with automatic calling units have much more flexibility.
- No operating system changes are required to install or use the system.
- The communication is basically over dial-up lines, but hard-wired communication lines can be used to increase speed.
- The command for sending/receiving files is simple to use.
- A general remote execution facility is part of the system; remote mail is one use of this feature.
- Adequate security facilities are available, so the site administrators feel comfortable about being on the network.

Introduction

The UNIX operating system[1] is a time-sharing system that runs on small to large minicomputers. A typical user gets access to the system through the telephone network. Since each computer is connected to the telephone network, they are potentially connected together; all that is needed to connect the machines is an automatic dialer and a program that can emulate a terminal (see Figure 1).

Many UNIX systems are used within the Bell System. Although there are some differences between them, a large set of common software modules exist: compilers, text editors, assemblers, linking loaders, debuggers, phototypesetting programs, and so on. An informal network emerged out of the need to exchange, deliver and maintain software. The telephone network provides the transmission path and the computer programs described in this paper implement the necessary protocols.

Over the past year, the network has grown to over one hundred machines throughout the country. There are several major uses of the network:

† UNIX is a trademark of Bell Laboratories.

- distribution of software
- distribution of documentation
- personal communication (mail)
- data transfer between closely sited machines
- transmission of debugging dumps and data exposing bugs
- production of hard-copy output on remote printers.

Design

In keeping with the style of UNIX commands, the user interface is quite simple. To copy a file from the local system to a remote system, the user would execute the command:

```
uucp file1 sysx!file2
```

where *file1* and *file2* are file names, and *sysx* is the remote system name. All details of the connection (e.g., device(s) used for the connection, phone number, times the remote system is available, its login sequence and password, retries for unavailable device or busy phone) are hidden from the user. The file transfer takes place when the connection is made.

We had to adapt to a community of systems that are independently operated and resistant to suggestions that they should all buy particular hardware or install particular operating system modifications. Therefore, we make minimal demands on the local sites in the network. Our implementation requires no operating system changes; in fact, the transfer programs look like any other user entering the system through the normal dial-up login ports, obeying all local protection rules.

We distinguish between "active" and "passive" systems on the network. Active systems have an automatic calling unit or a hard-wired line to another system: they can initiate a connection. Passive systems do not have the hardware to initiate a connection. However, an active system can be assigned the job of calling passive systems and executing work found there: this makes a passive system the functional equivalent of an active system except for an additional delay while it waits to be polled.

Several groups, both inside and outside Bell Laboratories, have constructed networks using hard-wired connections exclusively.[2], [3] Our network, however, uses both dial-up and hard-wired connections so that service can be provided to as many sites as possible and so the slower dial-up paths can be automatically used if a hard-wired link is out of service. Dial-up connections are made at either 300 or 1200 baud; hard-wired connections are asynchronous up to 9600 baud and might run even faster on special-purpose communications hardware.[4], [5] Thus, systems typically join our network first as passive systems. When they find the service important, acquire automatic calling units and become active systems; eventually, they may install high-speed links to particular machines with which they handle a great deal of traffic. At no time, however, must users change their programs or procedures.

The basic operation of the network is simple. Each participating system has a spool directory in which work (files to be moved or commands to be executed remotely) is placed. The **UUCICO** program performs all transfers. This program is started periodically to perform the file transfers; it selects devices, establishes the connection to the remote machine, performs the required login sequence, performs file security checks, transfers data files, logs results, and notifies specified users of transfer completions.

After the calling program completes all the work for the called system, the called machine sends any files that have been queued for the calling system. In this way, all services are available from all sites; passive sites, however, must wait until called. A variety of line protocols may be used; this gives users some flexibility and provides a mechanism for distributing other protocols in the future.* As long as the caller and called programs have a protocol in common,

* There is only one known system that has implemented an additional protocol. The protocol is used for high-speed transfers using a shared disk.

they can communicate. Furthermore, each caller knows the hours when each destination system should be called. If a destination is unavailable, the data intended for it remains in the spool directory until the destination machine can be reached.

In addition to the **UUCP** command, the user has the **UUX** command which allows execution of programs that require resources of remote machines. A common use of this facility is to format a printout on the local machine and send the result to a remote machine which has a hard copy output device. Remote mail is implemented using the **UUX** command but its execution is embedded in the standard *mail* command.

Processing

The user has two commands that set up communications, **UUCP** to set up file copying, and **UUX** to set up command execution where some of the required resources (system and/or files) are not on the local machine. Each of these commands will put work and data files into the spool directory for execution by the **UUCICO** program. Figure 2 shows the major blocks of the file transfer process.

The file names in the spool directory are constructed to allow the **UUCICO** program to identify the work and data files, the remote machines that should be called, and the order that the files for a particular system should be processed.

The call is made using information from several files. A single conversation between a pair of systems is ensured by the use of a lock file. A "systems" file contains information for making a connection to a remote machine:

[1] system name,

[2] system access time (days-of-week and times-of-day),

[3] device or device type to be used for the call,

[4] line speed,

[5] phone number,

[6] login information (multiple fields).

The *phone number* may contain abbreviations (e.g. "nyc", "boston") that get translated into dial sequences using a "dial-codes" file. This permits the same *phone number* to be stored at every site, despite local variations in telephone services and dialing conventions.

A "devices" file is scanned using fields [3] and [4] from the "systems" file to find an available device for the connection. If the connection fails after two attempts, an alternate path can be tried. (The presence of more than one entry for a system in the "systems" file indicates alternate paths.) If the connection is complete, the *login information* is used to log into the remote system. The conversation between the two **UUCICO** programs begins with a handshake started by the called (or *SLAVE*) system. The *SLAVE* sends a message to let the *MASTER* know that it is ready to receive the system identification and conversation sequence number. The response from the *MASTER* is verified by the *SLAVE* and if acceptable, protocol selection begins.

The remote system sends a message:

P*proto-list*

where *proto-list* is a string of characters, each representing a line protocol. The calling program checks the *proto-list* for a letter corresponding to an available line protocol and returns a *use-protocol* message. The *use-protocol* message is:

U*code*

where *code* is either a one character protocol letter or *N*, which means no common protocol.

During the processing, one program is the *MASTER* and the other is the *SLAVE*. Initially, the calling program is the *MASTER*. These roles may switch one or more times during the conversation.

There are five messages used during the work processing, each specified by the first character of the message. They are:

S	send a file,
R	receive a file,
X	get files whose names are determined on the remote system,
C	copy complete,
H	hangup.

The *MASTER* uses the first three to request file transfers. The *SLAVE* responds to each with a *yes* or *no*.

The send, receive and execute replies are based on permission to access the requested file/directory. After each file is copied to the receiving system, a copy-complete message is sent by the receiver of the file. The requests and results are logged on both systems, and, if requested, mail is sent to the user reporting completion. A failure message is sent by mail to the requester.

The *MASTER* executes all the work for the remote, followed by an *H* message. The *SLAVE* checks its spool directory for work. If work for the remote system exists, an *HN* message is sent and the programs switch roles; otherwise, an *HY* is sent. When the *MASTER* receives an *HY* message, it echoes it back to the *SLAVE* and the protocols are turned off. Each program sends a final *OO* (close) message to the other. Figure 3 shows a sample conversation.

Security

The implementation of this network between independent sites, many of which store proprietary programs and data, illustrates the pervasive need for security and administrative controls over file access. A number of security features evolved during the development of the system:

- file directory access restrictions,
- file monitoring,
- call back,
- call sequence checking,
- limited commands for the **UUCP** logins,
- restricted commands available to the **UUX** command,
- limited access to remote login information.

Each site, when configuring its programs and system files, limits and monitors transmissions. The administrator can give some remote systems limited file access while others have the same access privileges as the local users. Each system establishes a public directory for the **UUCP** program. A degree of file security can be achieved if the administrator allows the remote **UUCP** programs to access only this public directory. This requires a local user for remote sites to get or send files. Records are kept identifying all files that are moved into and out of the local system, and also of how the requester of such accesses identified themselves.

A site can arrange to permit users to call up and request that work be done; the calling users are then called back before the work is actually performed. This makes it possible to verify that the requester is legitimate. Masquerading is difficult even if the necessary password is known.

Each machine can optionally maintain a sequence count for conversations with other machines and require a verification of the count at the start of each conversation. A would-be impersonator must steal not only the correct phone number, user name, and password, but also the sequence count, and must call in promptly before the next legitimate request from either side. Even a successful masquerade will be detected on the next correct conversation.

The "systems" file, which is described in the Processing Section, contains information to allow the **UUCICO** program to login to the remote machines. This information would usually permit almost complete access to the system. The normal file system protections are used to restrict access of the "systems" file to the **UUCP** programs and the administrator. This gives some security, but it depends on the remote system administrator. To minimize this problem, we set up the system so that the only program that can be executed with the **UUCP** login is the **UUCICO** program. The system administrator can use the directory access restrictions to protect the local system without depending on a remote system to protect the login information.

The **UUX** command allows users to execute commands on remote systems. To protect a system, the administrator has to specify a list of commands that the **UUCP** system can execute. All commands received are checked against the list.

Present Uses

One application of this software is remote mail. Normally, a UNIX system user writes "mail dan" to send mail to user "dan". By writing "mail pwba!dan" the mail is sent to user "dan" on system "pwba".

The primary uses of our network to date have been in software maintenance. New programs (or new program versions) are sent to users, and potential bugs are returned to authors. A "stockroom" has been established at two sites. This allows remote users to call in and request software without bothering the author. A "stock list" of available programs, new bug fixes, and utilities is updated regularly.

Test cases are retrieved from other systems to determine whether errors on remote systems are caused by local misconfigurations or old versions of software, or whether they are bugs that must be fixed at the home site. This helps identify errors rapidly.*

The **UUX** command has been useful for providing remote output. There are some machines that do not have hard-copy devices, but that are connected over 9600 baud communication lines to machines with printers. The **UUX** command allows the formatting of a printout on a local machine and printing on a remote machine using standard UNIX commands.

Performance

Throughput, of course, is primarily dependent on transmission speed. The table below shows the real throughput of characters on communication links of different speeds. These numbers represent actual data transferred; they do not include bytes used by the line protocol for data validation such as checksums and messages. At the higher speeds, contention for the processors on both ends prevents the network from driving the line at full speed. The range of speeds represents the difference between light and heavy loads on the two systems. If desired, operating system modifications can be installed that permit full use of fast links.

Nominal speed	Characters/sec.
300 baud	27
1200 baud	100-110
9600 baud	200-850

In addition to the transfer time, there is some overhead for making the connection and logging in, ranging from a few seconds to 1 minute. Even at 300 baud, however, a typical 5,000 byte source program can be transferred in four minutes instead of the 2 days that might be required to mail a tape.

Traffic between systems is variable. During a typical week for a group a three co-located systems, 30 users made about 300 requests resulting in the transfer of about 3 million bytes. (These transfers took place over 9600 baud hard-wired lines.) On a system that distributes and

* For one set of test programs maintained by us, over 70% of the bugs reported from remote sites were due to old software and were fixed merely by distributing the current version.

maintains standard system software, a typical week consists of transferring about 1500 files (10 million bytes): this includes the dial-up network at 300 or 1200 baud and hard-wired local lines.

Presently, the total number of sites in the network is about 120. This includes most of the Bell Laboratories full-size machines that run the UNIX operating system, many operating telephone companies, some Western Electric sites, and several universities. Geographically, the machines range from Andover, Massachusetts to Berkeley, California.

Further Goals

Eventually, we would like to develop a full system of remote software maintenance. Conventional maintenance (a support group that mails tapes) has many well-known disadvantages.[6] There are distribution errors and delays, resulting in old software running at remote sites and old bugs continually reappearing. These difficulties are aggravated when there are 100 different small systems, instead of a few large ones.

The availability of file transfer on a network of compatible operating systems makes it possible to send programs directly to the end user who wants them. This avoids the bottleneck of negotiation and packaging in the central support group. The "stockroom" provides this function for new utilities and fixes to old utilities. However, it is still likely that distributions will not be sent and installed as often as needed. Users are justifiably suspicious of the "latest version" that has just arrived; all too often it features the "latest bug." What is needed is to address both problems simultaneously:

1. send distributions whenever programs change.
2. have sufficient quality control so that users will install them.

To do this, we recommend systematic regression testing both on the distributing and receiving systems. Acceptance testing on the receiving systems can be automated permitting the local system to ensure that its essential work can continue despite the constant installation of changes sent from elsewhere. The work of writing the test sequences should be recovered by lower counseling and distribution costs.

Some slow-speed network services have been implemented. We now have inter-system *mail* plus the many implied commands represented by UUX. However, we still need inter-system *write* (real-time inter-user communication) and *who* (list of people logged in on different systems). A slow-speed network of this sort may be very useful for speeding up counseling and education, even if not fast enough for the distributed data base applications that attract many users to networks. Effective use of remote execution over slow-speed lines, however, must await the general installation of multiplexed channels so that long file transfers do not lock out short inquiries.

Lessons

What follows is a summary of the lessons we learned in building this system.

1. By starting the network in a way that requires no hardware or operating system changes, one can get going quickly.
2. Support will follow use. Since the network existed and was being used, system maintainers were easily persuaded to help keep it operating, including purchasing additional hardware to speed traffic.
3. Make the network commands look like local commands. Our users have a resistance to learning anything new; all the inter-system commands look similar to standard UNIX system commands so that little training cost is involved.
4. In the first version, we made the mistake of using dial-up communications exclusively. The second implementation of the system permits the use of different connecting fabric, such as hard-wired, asynchronous lines: this adds flexibility to the network.

5. Security presented a bigger problem than we anticipated. We had to give the administrators features that enabled them to protect their systems. These features, however, made it difficult to do useful work. The creation of a public directory on each system alleviated some of the problem, but the casual users of **UUCP** are often unpleasantly surprised when their requests are rejected by remote systems.

Acknowledgements

We thank G. L. Chesson for his design and implementation of the packet driver line protocol, and A. S. Cohen, A. G. Fraser, J. Lions, and P. F. Long for their suggestions and assistance.

References

[1] D. M. Ritchie and K. Thompson, "The UNIX Time-Sharing System," *Bell Sys. Tech. J.* **57**(6), pp. 1905-1929 (1978).

[2] T. A. Dolotta, R. C. Haight, and J. R. Mashey, "UNIX Time-Sharing System: The Programmer's Workbench," *Bell Sys. Tech. J.* **57**(6), pp. 2177-2200 (1978).

[3] G. L. Chesson, "The Network UNIX System," *Operating Systems Review* **9**(5), pp. 60-66 (1975). Also in *Proc. 5th Symp. on Operating Systems Principles,* 1975.

[4] A. G. Fraser, "Spider—An Experimental Data Communications System," *Proc. IEEE Conf. on Communications,* p. 21F (June 1974). IEEE Cat. No. 74CH0859-9-CSCB.

[5] A. G. Fraser, "A Virtual Channel Network," *Datamation,* pp. 51-56 (February 1975).

[6] F. P. Brooks, Jr., *The Mythical Man-Month,* Addison-Wesley, Reading, MA (1975).

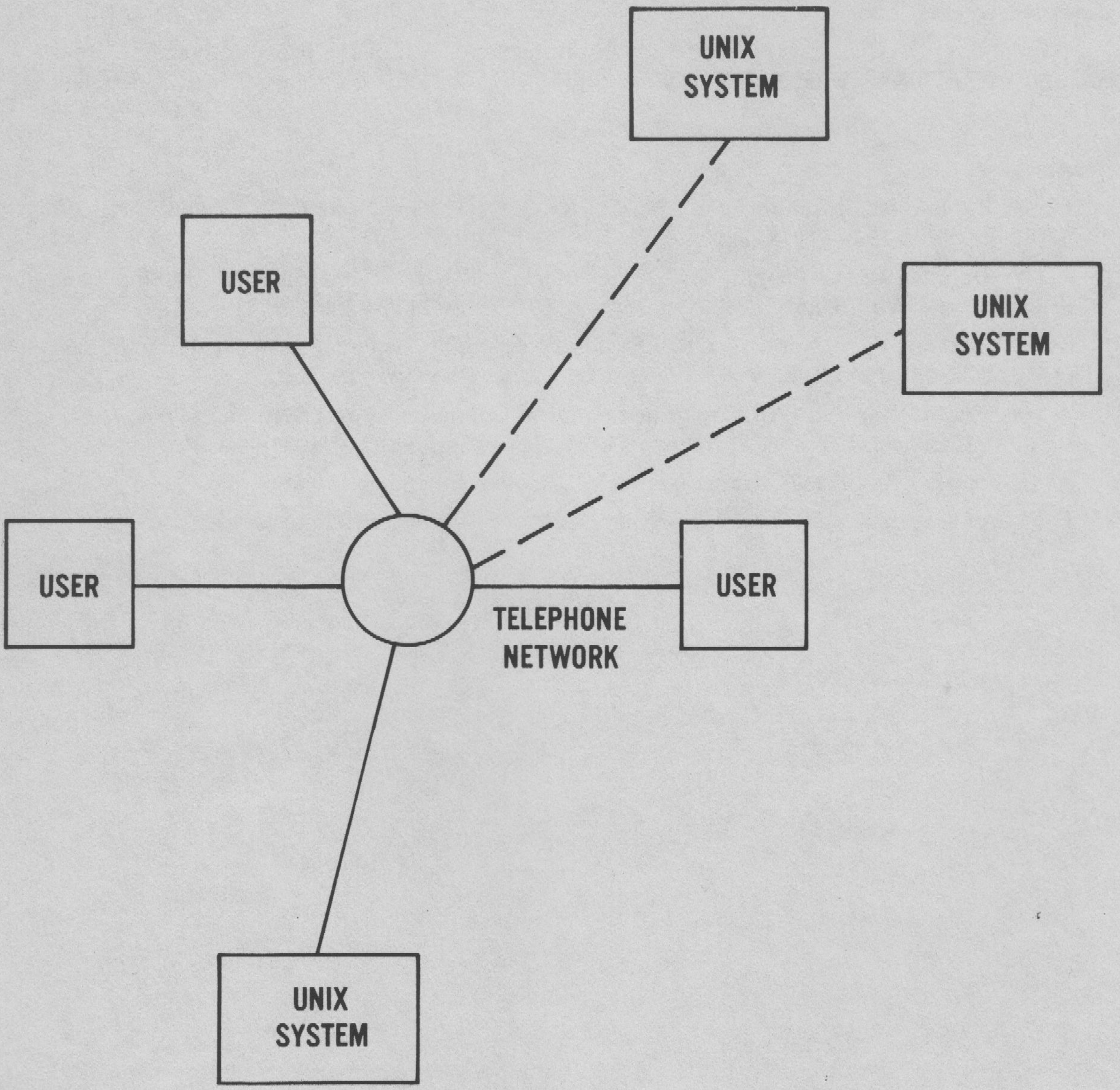

FIGURE 1
UUCP NETWORK - ACCESS TO REMOTE UNIX SYSTEM THROUGH DIALUP TELEPHONE NETWORK

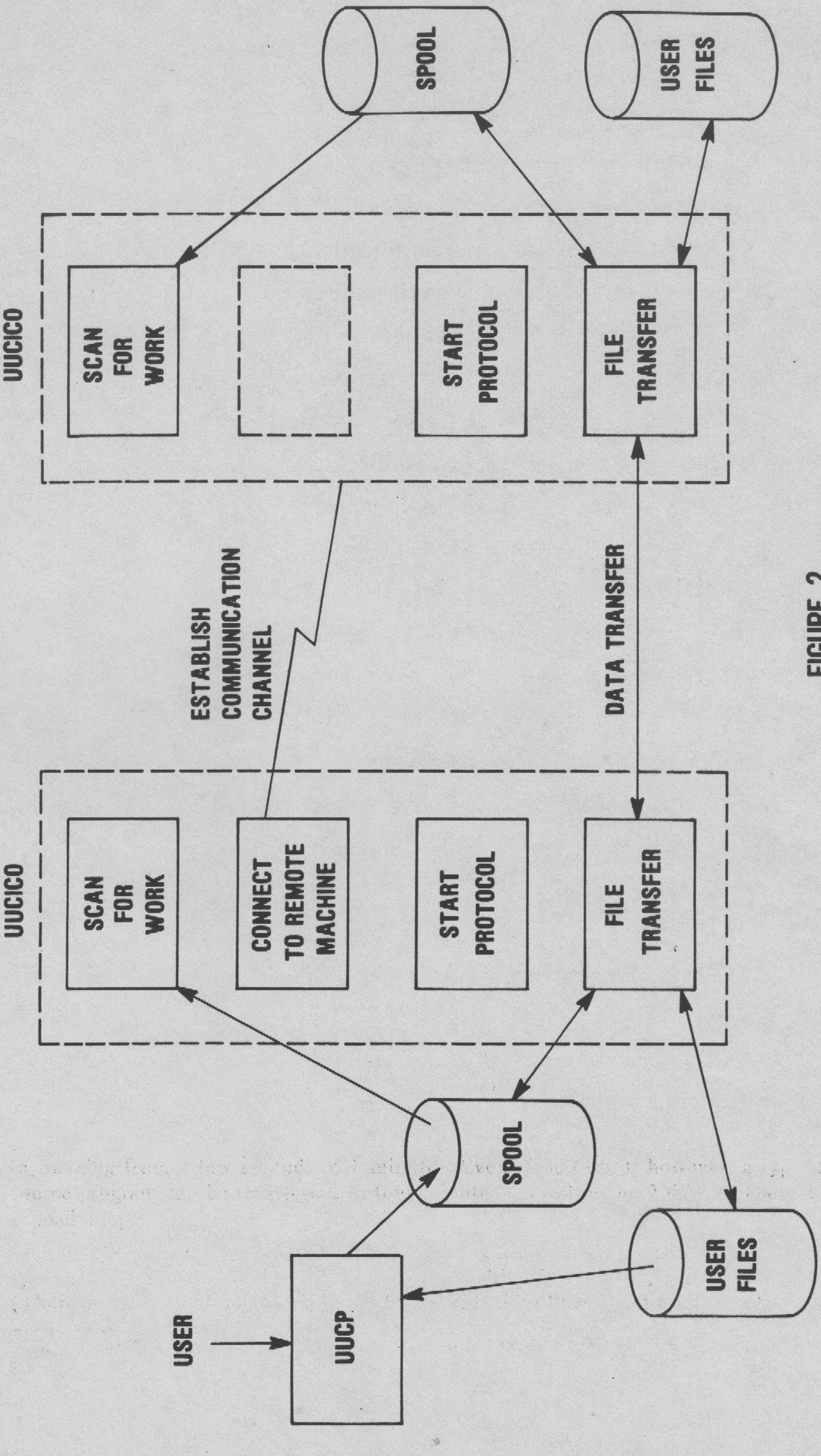

FIGURE 2
FILE TRANSFER PROCESS

```
CALLING        MESSAGES       CALLED
SYSTEM         & DATA         SYSTEM
               555-6789 ---------->  [ Dial up ]
     <----     login:
               user-id, password ->  [ Identification ]
     <----     S                     [ Handshake and
               S 36 myname ------>     sequence check ]
     <----     P a, b, c             [ Offer protocols ]
               U b --------------->  [ Choose protocol ]
               S /abc/def/ghi ---->  [ Send file ]
     <----     SY                    [ OK to send]
      ====     DATA =============>
               R /abc/def/jkl        [ Get file ]
     <----     RY                    [ OK to receive]
     <=====    DATA ====
               H ----------------->  [ Offer to stop ]
     <----     HN                    [ Refused ]
     <----     S /abc/mno/prs        [ Called system sends ]
               SY ---------------->  [ OK to send]
     <=====    DATA ====
     <----     H                     [ Offer to stop ]
               HY ---------------->  [ Accepted ]
     <----     (OO)                  [ Close]
               (OO) -------------->
               Hangup
```

FIGURE 3
SAMPLE CONVERSATION