

**NETWORK TRANSPARENCY IN AN INTERNET
ENVIRONMENT**

Alan Sheltzer

**August 1985
Report No. CSD-850028**

UNIVERSITY OF CALIFORNIA

Los Angeles

Network Transparency in an Internetwork Environment

A dissertation submitted in partial satisfaction of the
requirement for the degree of Doctor of Philosophy

in Computer Science

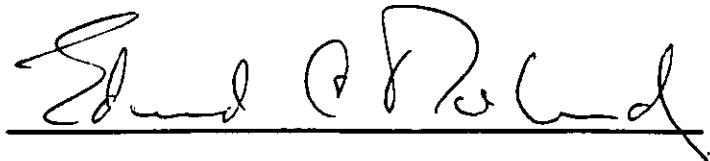
by

Alan Brian Sheltzer

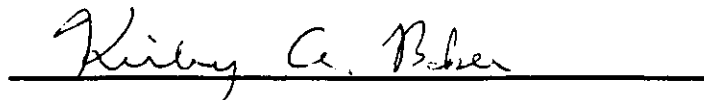
1985

© Copyright by
Alan Brian Sheltzer
1985

The dissertation of Alan Brian Sheltzer is approved.

A handwritten signature in cursive script, appearing to read "Edward Deland", written above a solid horizontal line.

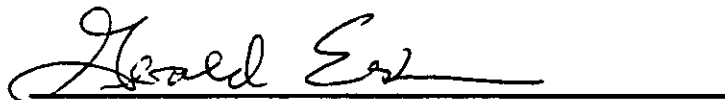
Edward Deland

A handwritten signature in cursive script, appearing to read "Kirby A. Baker", written above a solid horizontal line.

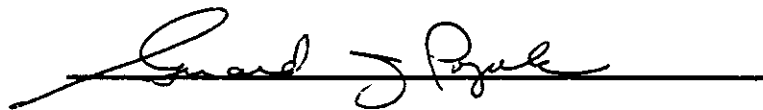
Kirby Baker

A handwritten signature in cursive script, appearing to read "Mario Gerla", written above a solid horizontal line.

Mario Gerla

A handwritten signature in cursive script, appearing to read "Gerald Estrin", written above a solid horizontal line.

Gerald Estrin

A handwritten signature in cursive script, appearing to read "Gerald J. Popek", written above a solid horizontal line.

Gerald Popek, Committee Chair

University of California, Los Angeles

1985

TABLE OF CONTENTS

page

1 Introduction	1
1.1 Transparency in Distributed Systems	1
1.2 A New Computing Environment: LANs Interconnected by Long Haul Networks	2
1.3 The Thesis	3
1.4 Related Research	5
1.4.1 National Software Works	5
1.4.2 RSEXEC	6
1.4.3 Grapevine	7
1.4.4 NewCastle Connection	7
1.4.5 The V-System	8
1.4.6 Accent	9
1.4.7 Cronus	10
1.5 Body of the Dissertation	11
2 The Internetwork Environment	12
2.1 Network Interconnection	12
2.2 The Darpa Internetwork	14
2.3 Services Provided by Current Internet Protocols	15
2.4 Distributed Operating Systems	16
2.4.1 Locus	16
2.4.2 An Example of Accessing Remote Resources in the IP and Locus Environments	17
2.5 Communication Characteristics of the Internet	20
2.6 Summary	21
3 Principles of Distributed System Design	23
3.1 Introduction	23
3.2 A Hierarchical View of Distributed Systems	23
3.3 Benefits of Internet Transparency	26
3.4 Arguments Against Internet Transparency	28
3.5 Approaches to Network Transparency	29
3.5.1 Remote Procedure Call	29
3.5.2 Message Passing	30
3.5.3 Connection Layer	31
3.5.4 Kernel to Kernel Communication	32
3.6 Strategies for Achieving Internet Transparency	33
3.6.1 Locality in Distributed Systems	36
3.6.2 A Semantics-based Approach to the Design of Network Protocols	40
3.6.3 Remote Execution	46
3.7 Summary	47
4 Name Management in an Internet DOS	49
4.1 Introduction	49
4.2 Name Service in a Distributed Environment	49
4.3 Pathname Expansion in Unix and Locus	50

4.4	Issues in Distributed Cache Design	54
4.5	Directory Reference Measurements	56
4.5.1	Measurement Results	57
4.5.2	Locality in Directory Referencing	59
4.5.3	Trace-driven Simulation Results	61
4.6	Operation of the Distributed Directory Cache	64
4.7	Summary	65
5	A Protocol for an Internet Operating System	67
5.1	Introduction	67
5.2	Methodology	67
5.3	Client Behavior	69
5.4	Message Primitives for Internet Locus	71
5.4.1	List a Directory	72
5.4.2	Examine a File	73
5.4.3	Change Working Directory	75
5.4.4	Edit a File	76
5.4.5	Remove a File	79
5.4.6	Copy a File to an Existing File	80
5.4.7	Copy a File to a New File	82
5.4.8	Remote Execution: Make and Grep	83
5.5	Protocol Support for Message Primitives	85
5.5.1	Piggyback Acknowledgements	85
5.5.2	Variable Length Messages	87
5.5.3	Data Compression	87
5.5.4	Data Stream Protocol	89
5.5.5	Summary	91
6	The Performance of an Internet Operating System	93
6.1	Introduction	93
6.2	Performance Evaluation of Distributed Operating Systems ...	93
6.3	The Internet Locus Testbed	95
6.4	Performance of Interactive Commands in an Internet Environment	98
6.4.1	List a Directory	99
6.4.2	Examine a File	100
6.4.3	Edit a File	100
6.4.4	Copy a File	100
6.4.5	Discussion	101
6.5	Performance of Commands Requiring Large Data Transfers	102
6.5.1	FTP and dd Performance Across a LAN	103
6.5.2	FTP and dd Performance Across the Internet	104
6.5.3	The Effect of Window Size on Throughput	105
6.5.4	Discussion	106
6.5.5	Compile a Program	107
6.6	Evaluation of Internet Locus Strategies	109
6.7	Use of Internet Locus Strategies in Other Environments	113
7	Conclusions	115
7.1	Summary of Goals and Accomplishments	115
7.2	Future Work	116
7.3	Contribution to Computer Science	118

Appendix A The Implementation of Internet Locus	119
Appendix B Locus Network Measurements	121

LIST OF FIGURES

	page
Figure 2.1: The Darpa Internet (1985)	14
Figure 2.2: Editing a file in the Internet and Locus environments	20
Figure 2.3: Internet Network Characteristics	21
Figure 3.1: Hierarchical View of the Internet Environment	24
Figure 3.2: Locus Remote System Call Processing	45
Figure 3.3: Internet-Locus Remote System Call Processing	46
Figure 4.1: Message Traffic for Pathname Expansion	53
Figure 4.2: Results of Directory Reference Event Collection	59
Figure 4.3: Average locality $\overline{L(\tau)}$ as a function of window size	60
Figure 4.4: Miss ratio versus window size for a directory cache	62
Figure 4.5: Distribution of # of sites that require cache invalidation	63
Figure 4.6: Plot of miss ratio with and without cache invalidation	64
Figure 5.1: The 10 Most Frequently Executed Unix Commands	71
Figure 5.2: Message Traffic for the List Directory Command	73
Figure 5.3: Message Traffic for the Examine File Command	75
Figure 5.4: Message Traffic for the Change Working Directory Command	76
Figure 5.5: Message Traffic for the VI Command	78
Figure 5.6: Message Traffic for the Remove Command	80
Figure 5.7: Message Traffic for the Copy File Command	81
Figure 5.8: Message Traffic for the Copy File Command	83
Figure 5.9: Total # of Messages Transferred	84
Figure 5.10: Distribution of Message Types	86
Figure 6.1: Internet Locus Testbed	96

Figure 6.2: Internet Locus Packet Format	97
Figure 6.3: Estimated Delay (in milliseconds) to Transmit Messages ...	98
Figure 6.4: Elapsed Time for the List Directory Command	99
Figure 6.5: Elapsed Time for the Examine File Command	100
Figure 6.6: Elapsed Time for the Edit File Command	100
Figure 6.7: Elapsed time for the Copy File Command	101
Figure 6.8: Throughput of FTP versus the <i>dd</i> Command	105
Figure 6.9: Effect of Window Size on Throughput	106
Figure 6.10: Elapsed Time for the Compile Command	108
Figure 6.11: Performance Effect of Internet Locus Strategies	110
Figure 6.12: Performance Effect of Internet Locus Strategies	112
Figure 6.13: Round Trip Delay	113

ACKNOWLEDGEMENTS

I would like to thank the members of my committee, Professors Gerald Estrin, Mario Gerla, Edward Deland and Kirby Baker. I am especially grateful to my advisor, Gerald Popek, for his valuable technical guidance and unfailing enthusiasm. In addition, I would like to thank the members of the Locus research group, especially Robert Lindell and Alan Downing, for their helpful suggestions and contributions to Internet Locus. Finally, I wish to acknowledge the Defense Advanced Research Projects Agency, which provided continuing support for the Locus project (contract DSS-MDA-903-82-C-0189).

VITA

- 1975 B.S. (Physics), Brandeis University, Waltham, MA.
- 1977-1978 Computer Applications Engineer, Beckman Instruments
Fullerton, CA.
- 1978-1979 Research Assistant, University of Southern
California, Los Angeles, CA.
- 1979 M.S. (Computer Science), University of Southern
California, Los Angeles, CA.
- 1979-1982 Computer Scientist, Bolt Beranek and Newman, Inc.
Cambridge, MA.
- 1983-1985 Post Graduate Research Assistant, University of
California, Los Angeles, CA.

PUBLICATIONS

R. Hinden, J. Haverty, and A. Sheltzer, "The DARPA Internet",
IEEE Computer, Vol. 16, No. 9, Sept. 1983.

A. Sheltzer, R. Hinden, and M. Brescia, "Connecting Different
Types of Networks with Gateways", Data Communications,
Vol. 11, No. 8, Aug. 1982, pp.111-121.

ABSTRACT OF THE DISSERTATION

Network Transparency in an Internetwork Environment

by

Alan Brian Sheltzer

Doctor of Philosophy in Computer Science

University of California, Los Angeles, 1985

Professor Gerald Popek, Chair

Network transparency refers to the ability of a distributed system to hide machine boundaries from users and programs; all resources are accessed in the same manner, independent of their location. Network transparency has been shown to be highly valuable and achievable in the local area network environment. In contrast, access to remote resources in long haul networks traditionally is not transparent; substantially different access methods are required and only a limited set of operations is available.

In this dissertation, we demonstrate that transparency across a system of local area networks connected by long haul links is both highly desirable and technically feasible. Three strategies are proposed to overcome the performance limitations of the communications media: exploitation of locality, semantics-based protocol design, and remote process execution. A distributed name cache is shown to dramatically reduce the overhead of name management for an internetwork operating system. New, higher semantic level message primitives for the most frequent user commands are described and their impact on reducing network traffic is demonstrated. A testbed based on the Locus operating system, extended to operate across an internetwork consisting of LANs connected by long haul links, was implemented. Performance

measurements from the testbed were used to evaluate the effectiveness of the three strategies and to demonstrate the viability of transparency in an internet environment.

CHAPTER 1

Introduction

1.1 Transparency in Distributed Systems

The focus of research in computer science is constantly shifting. As the hardware for computing engines and communications evolve, the set of opportunities and problems to be solved likewise evolves. Interest in the efficient sharing of a single mainframe machine has been followed by interest in the problems of communication among machines. Today, the abundance of inexpensive computing power, coupled with well developed local area and long haul network technologies brings the issue of transparent, distributed computing to the forefront of computer science research.

It is well known that computing environments are improved when complex details of the underlying machine architecture are hidden from the user. In a virtual memory system, for instance, the user has the illusion of addressing a large, contiguous memory space even though the corresponding real memory locations may be scattered throughout primary or secondary memory. In an operating system such as Unix, the filesystem conceals the physical properties of file storage devices and provides the user with the simple view of a file as a sequence of randomly addressable bytes.

This principle of hiding unnecessary detail from the user can be extended to a distributed system of computers. In the distributed environ-

ment, hiding machine boundaries from the user is known as *network transparency*. Network transparency provides opportunities for greater data sharing, easier application software development and increased reliability when compared to non-transparent distributed environments. In the ideal network transparent environment, users and programs are presented with a unified, location independent view of resources so they do not have to worry about network protocols, network failures, or resource location.

1.2 A New Computing Environment: LANs Interconnected by Long Haul Networks

Increasingly, one expects that computer environments will be composed of workstations, locally connected by high speed local area networks (LANs), with those networks linked by lower speed more conventional wide area or "long haul" networks. Within a LAN, it is becoming clear that it is important to make machine boundaries logically invisible both to users and applications software. This *local network transparency* has been successfully demonstrated and is in routine use in a number of systems, including Tandem's NonStop [Bar 81], Apollo's Domain [Lea 82], and Locus [Pop 81].

In contrast, computer systems connected by long distance communications networks have traditionally only provided a limited set of specialized interfaces by which remote resources (files, programs, devices, services) are accessed by the local user or program. Examples are the file transfer protocol (FTP) of the Arpanet, site naming requirements of Decnet, or the specialized interfaces of SNA. As single host sites within these networks are replaced by clusters of local *networks* of workstations and other computers, users are faced with transparent access to resources stored within a cluster but non-

transparent access to resources stored at all other sites. Furthermore, this dichotomy forces processes running on sites that are geographically separated to interact in a non-transparent fashion, regardless of how closely related the processes are.

The major reason that interfaces to remote resources have differed from local interfaces is that the performance quality of long haul networks makes remote access potentially orders of magnitude slower than local access. For example, a local disk may run at a several megabyte transfer rate with tens of milliseconds delay, and a local area network may provide a bandwidth of greater than one megabyte with delay less than a millisecond. By contrast, long haul networks such as the Arpanet, Tymnet, and Telenet provide a maximum of approximately 50 kilobit bandwidths throughout most of their links with delay commonly in the tenths of seconds. This dramatic difference between long haul communications media and LANs or directly connected devices has generally meant that it was not feasible to provide transparent access to remote resources in the long haul case.

Furthermore, a wide variety of machines and operating systems have been historically represented in long haul networks. The goal of developing a standard for information exchange between a heterogeneous set of systems, where it was often impractical to modify vendor supplied software, has also contributed to restricting the interface to remote resources.

1.3 The Thesis

The fundamental issue to be addressed in this dissertation is:

- Can a distributed operating system provide network transparency with

satisfactory performance in an environment of interconnected networks where some of the communication links have poor throughput, delay, and error characteristics?

We will show that transparency *is* feasible, and highly desirable, in an internet environment. That is, the conventional view of internet access is incorrect, at least when the computer systems at either end of the internet link are similar (e.g. both Unix), and the links are comparable in quality to the Arpanet. Remote access *can* be provided with precisely the same interface as local access, often with similar performance.

A proper defense of the thesis requires a case study that demonstrates transparency in an internet environment. A case study based on Internet Locus, which is a version of Locus extended to run on sites that communicate across the Darpa Internet, has been implemented. Many of the lessons described in this dissertation are a direct outgrowth of the transformation of LAN based Locus into Internet Locus.

The underlying network model for the dissertation is a system of machines running the same operating system, interconnected by a variety of network technologies, including long haul networks and LANs. The topology of most interest is an environment of LAN clusters connected by a long haul network such as the Arpanet, with the LAN clusters separated by a maximum of 5 or 6 packet switching nodes.

The dissertation does not specifically address the issues of file replication in an internet environment, distributed database systems, or internet security, but provides a foundation for the study of these issues. Support for

heterogeneous operating systems and the problems of distributed systems with very large numbers of sites are discussed elsewhere ([Loc 83] and [Rei 85]).

1.4 Related Research

A number of researchers have developed distributed operating systems that support varying degrees of network transparency. Several systems have provided a unified user view of files and tools located on different Arpanet hosts but the performance of these systems has been disappointing. A mail system, Grapevine, developed at Xerox is of interest because the system provides an internetwork-wide name service. A network transparent distributed operating system developed at Stanford called the V-System, was originally built for sites on a LAN, and has been extended to an internet environment. The number of distributed operating systems that provide some degree of transparency is quite large so only a few of the more important systems will be described.

1.4.1 National Software Works

One of the earliest systems that supported limited network transparency was the National Software Works (NSW) [Hol 81]. The goal of the project was to integrate tools available on different Arpanet hosts into a single tool kit, accessible through a single monitor and filesystem. Network transparent access was limited to those tools or files that were registered in the NSW database so only a small subset of a host's files or tools were available through the NSW.

The NSW designers realized that the existing Arpanet connection-oriented host to host protocol (NCP) was not appropriate for achieving network transparency, so a new protocol that handled request-response communication was developed. The NSW project had the ambitious aim of providing service to machines with heterogeneous operating systems without modifying operating system level code. Requests for NSW service, even local requests, were first intercepted by a NSW process which sent the request to a centralized NSW monitor (usually located on a remote site) who then forwarded the request to the appropriate tool. The implementation of network transparency on top of the host's native operating system, instead of building transparency at the operating system level, resulted in unsatisfactory performance for access to both remote and local resources.

1.4.2 RSEXEC

Another early system that provided limited transparent access to remote resources was RSEXEC [Tho 73]. Like the NSW system, RSEXEC was built on top of an existing operating system and suffered the resulting performance overhead. Files stored on a remote RSEXEC site were specified by prefixing the remote site name to the normal file name. A user could build a limited form of name transparency by setting up a user profile that listed the corresponding storage site for RSEXEC files so the site portion of a filename could be omitted.

RSEXEC provided a limited set of distributed functions. Only users, not programs could name remote files. Remote login and connection establishment were required before a remote resource could be accessed. All non-local file access was achieved by first copying the file in its entirety to the

local machine. Non-local processes and devices could not be accessed.

1.4.3 Grapevine

Grapevine [Bir 82] is a system that provides mail service, resource location, and authentication services in an internetwork. A distributed, replicated database is used by application programs to map the name of an object, such as a user's mailbox into its internet address. The name database is replicated at several sites so the loss of a database site will not shut down the name service. The Grapevine mail service is used to propagate updates to insure eventual consistency among copies of the distributed database but multi-copy atomic update is not supported. Clients of the name service occasionally observe inconsistent copies of the database at different sites since the propagation mechanism may take several minutes. The Grapevine approach to a replicated internet name service is sufficient for applications such as electronic mail where an inconsistent name database may be inconvenient but not disastrous. The loose consistency constraint of the Grapevine name service is not appropriate for an operating system name service. For instance, if an operating system uses an out of date name to location entry to find the storage or synchronization site for a file, any file locking policy can no longer be enforced.

1.4.4 NewCastle Connection

The Newcastle Connection [Bro 82] was developed at the University of Newcastle Upon Tyne to connect loosely coupled Unix systems. It provides remote access to files, using what appear to the application to be the same system calls as used locally. Each system's file tree and root is explicitly visi-

ble, so a file's location cannot be changed without altering the file's name. The implementation is via extension to the language libraries of applications; consequently local access is slowed down. Remote access requires application level daemons at both the requesting and serving sites, with the attendant performance costs. No transparent remote processes are provided. However, because the Newcastle Connection is implemented at the user level, no kernel changes are required, making it interesting for practical reasons.

1.4.5 The V-System

The V-System [Che 83] is a distributed operating system that provides network transparency in an environment of workstations and server machines connected by an Ethernet. The operating system provides uniform, message based, inter-process communication between processes on a single machine or on different machines. Dedicated server processes provide file access, printer access and other services.

The V-system has been extended to support communication between processes in an internetwork environment. For a process on one network to communicate with a process on another network, an intelligent gateway first sets up a local alias process to mirror the remote process. Although the alias process mechanism masks the fact that the actual remote process is not on the same net as the sender, the process identifier name space is local to each network so process identifiers cannot be used transparently between networks. Four processes are actually involved in any internetwork communication: the local process, the local alias process running in the local gateway, the remote alias process running in the remote gateway, and the destination process.

The V-system relies heavily on broadcast and multicast communication which are not currently supported by internet architectures. Consequently, the use of the V-system across an existing internetwork such as the Darpa Internet is seriously limited. One expects the V-system internet design, with its high process overhead, to exhibit performance problems, but measurements for internet service have not yet been reported.

1.4.6 Accent

Accent is a communication oriented operating system kernel used to support distributed personal computing at Carnegie-Mellon University [Ras 81]. The basic communication abstraction for interprocess communication (IPC) in Accent is the *port* which is a protected kernel object into which messages are placed and removed by processes. A system call corresponds to the notion of sending and receiving messages on ports.

Since messages are sent to ports rather than specific processes, a single process can act as an intermediary for communication between processes. Network transparent IPC is accomplished by having an intermediary network process fabricate a local, "alias" port that corresponds to a remote port. Two network servers can each allocate an alias port and provide inter-machine communication without the communicating processes being aware that the communication is crossing machines boundaries.

A message-based operating system such as Accent may exhibit slow performance since all references to resources, even local resources, suffer from high context-switch overhead. Microcode support for context switching is being used by the researchers at CMU to help reduce the performance

difficulties.

1.4.7 Cronus

Cronus [Gur 85] is a distributed operating system used to interconnect heterogeneous computers across local area networks. The Cronus implementation centers on an object-oriented, interprocess communication mechanism where all resources, including processes, files, and devices are viewed as abstract objects under the control of a manager process on a Cronus host. Every object is an instance of some abstract type which defines the operations that may be invoked on the object.

Cronus is structured to run on top of existing single site operating systems such as Unix, VMS, and CMOS. Existing operating system tools are integrated into Cronus via a library-level trapping facility. Messages are encoded in a host independent form to facilitate information exchange between heterogeneous systems. External representations are defined for common data types such as integers and character strings as well as aggregates of types such as arrays and structures.

The extension of Cronus to the internet environment faces three obstacles. First, Cronus, like the V-system, makes extensive use of broadcast communication which is not currently available in any internetwork. Second, the complex object-oriented IPC mechanism and the multilayered structure of Cronus may lead to performance problems when Cronus is used across long haul links. Third, the need for data conversion to support heterogeneous operation can potentially lead to unacceptable processing overhead on communicating hosts.

1.5 Body of the Dissertation

In the body of the dissertation we first examine characteristics of the internet environment. In chapter 2, current internet services available to users and programs are described and contrasted to the services provided by a distributed operating system such as Locus. In chapter 3, approaches to extending network transparency beyond the local area network are discussed. Three useful strategies emerge from the discussion:

- Exploitation of locality in distributed systems
- Semantics-based protocol design
- Remote Execution

These strategies are discussed in detail in chapters 4 and 5. A case study based on Internet Locus is described in chapter 6. The results of performance measurements of Internet Locus are presented and used to defend the validity of the thesis. Conclusions and suggestions for future work follow in chapter 7.

CHAPTER 2

The Internetwork Environment

2.1 Network Interconnection

In their pioneering article on computer communications in 1974 [Cer 74], Cerf and Kahn present "a protocol design and philosophy that supports the sharing of resources that exist in *different* packet switching networks". They foresee the desirability of interconnecting networks to facilitate data and program sharing and remote access to resources. Five potential differences between individual networks are identified which must be resolved by a common protocol for network intercommunication. These differences are:

- Each network may have a distinct way of addressing the receiver of messages
- Each network may accept data of different maximum size
- The delay to transfer packets across a network may vary
- The reliability characteristics of networks may vary
- Each network may have different facilities for supporting routing, fault detection and status reporting

Two protocols are proposed to resolve these issues: the Internet Protocol (IP) and the Transmission Control Protocol (TCP). The IP defines a datagram-based service that allows any host in a system of interconnected networks (an *internet*) to communicate with any other host in the system. Special sites called *gateways*, which have interfaces on two or more networks, are responsible for routing datagrams between networks and breaking datagrams into smaller fragments if the datagrams exceed a network's maximum size. Although datagrams are delivered with high probability, the IP does not provide a reliable transport service. Datagrams may be occasionally lost, duplicated, delayed for long periods of time, or contain bit errors.

Reliable communication between processes is provided by the TCP. End-to-end state information such as sequence numbers and checksums are used by the TCP to insure reliable delivery of messages. An additional protocol, the Internet Control Message Protocol (ICMP) has been developed to provide a mechanism whereby local network status information can be transported across an internet.

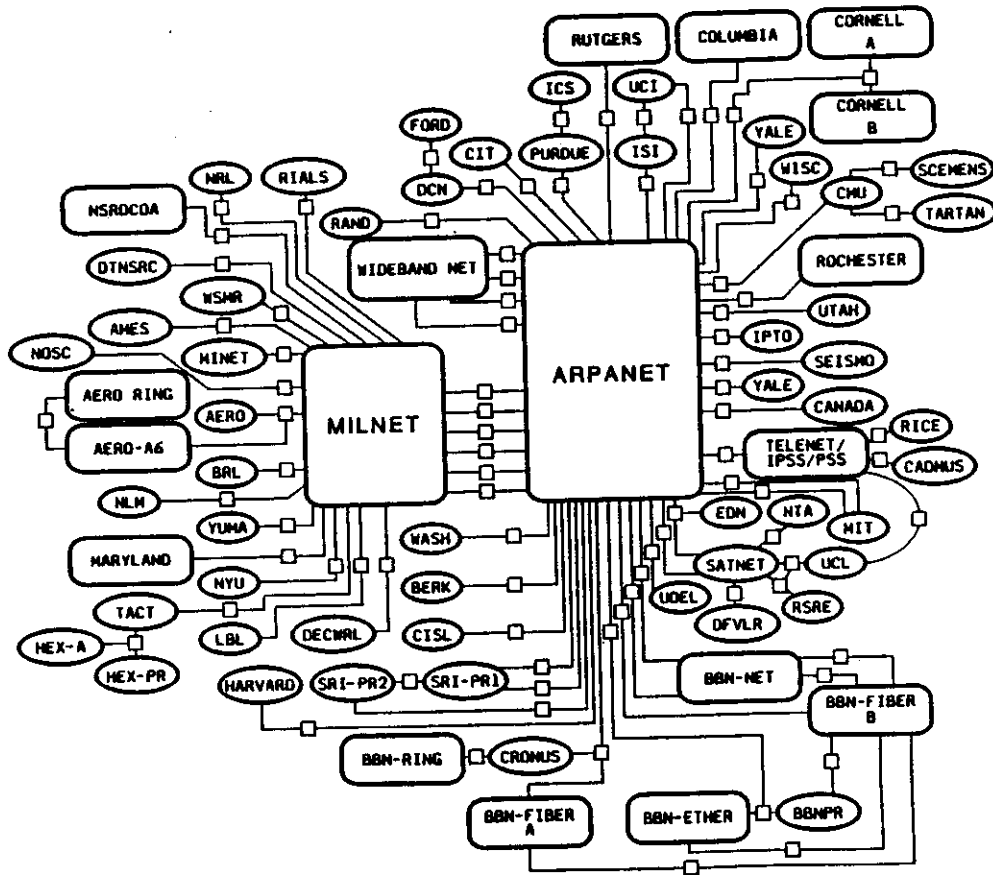


Figure 2.1: The Darpa Internet (1985)

2.2 The Darpa Internetwork

The early research on network interconnection, sponsored by the Defense Advanced Research Project Agency (Darpa), has evolved dramatically. Today, the Darpa Internet is a system of hundreds of host computers interconnected by a variety of network technologies including local area networks, satellite networks, packet radio networks, and terrestrial long haul networks. The current Internet system is shown in figure 2.1. Internet Gateways are represented by small squares and networks are represented by ovals

or rectangles. Individual hosts, such as the UCLA Arpanet hosts, are not shown.

The Darpa Internet is only one of several network interconnection efforts. An internet architecture called *PUP* [Bog 80], which is similar to the Darpa Internet architecture is in widespread use within Xerox. Public data networks which adhere to the CCITT recommendation X.25 can be interconnected by an interface specified in recommendation X.75 [Pos 80]. While the Darpa and Xerox interconnection architecture provides a datagram based service, X.75 specifies a virtual circuit service. An X.75 connection is a series of concatenated virtual circuits so once an internet connection is established, no alternate routing between gateways is possible. X.75 packets carry a logical channel number that identifies the internet connection instead of a full destination address so X.75 packet headers are smaller than Darpa Internet or PUP packet headers.

2.3 Services Provided by Current Internet Protocols

Although there has been tremendous growth in the number of sites that communicate across the Darpa Internet, the number of services available to users and programs has remained relatively stagnant since the early days of the Arpanet. The following three services dominate use of the Internet:

- Remote Terminal Service (Telnet) - a process, usually acting on behalf of a user, logs into a remote site and sends and receives characters as if it were a terminal on the remote system.
- File Service (FTP) - a process sends and retrieves data files from a remote site.

- Mail Service - Text messages addressed to individuals are distributed to designated machines. Individuals query a process on the designated machine to receive any mail that has been sent to them.

2.4 Distributed Operating Systems

While the development of internetworks was stretching the geographic boundaries of communicating hosts, another communication development was taking place that brought computers closer together. By the 1970s, it was possible to connect machines via a local area network with bandwidth and delay characteristics that were close to those of a local disk. A number of researchers developed distributed operating systems whereby a collection of sites connected by a LAN appeared to the user as a single site. In a system that exhibited full network transparency, any resource in the network could be accessed by the entire repertoire of single site commands and system calls. This contrasts sharply with the limited access to remote resources available to an Internet host.

2.4.1 Locus

One such distributed operating system that provides network transparency is a distributed version of Unix, called Locus, which runs on high speed, low delay, local area networks. Locus provides a fully transparent, distributed file system as well as transparent support for distributed processes. Networks of heterogeneous cpu types are also handled transparently.

The file system appears to the user and applications software as a single, tree structured name space with one root, across the entire collection of machines. Changing the storage site of a file does not require changing its

name. Important files are replicated below the user visible level, and the system is responsible for keeping the copies mutually consistent.

Process execution in Locus is similarly transparent. It is possible to create (i.e. fork) processes locally or remotely, with exactly the same semantics. Processes may migrate to a similar cpu type while in the midst of execution without effect on continued correct execution. Processes interact with one another across machine boundaries in the same manner as if they were co-located (e.g. Unix signals and IPC operate transparently networkwide). Besides providing access to remote resources through the same interfaces as local ones, Locus also achieves a substantial degree of *performance transparency*. The delay in access to remote resources typically is little different than the delay in access to local resources.

2.4.2 An Example of Accessing Remote Resources in the IP and Locus Environments

A vivid comparison between accessing a remote resource using the traditional internetwork protocols and using Locus protocols is shown in figure 2.2. In the example, the *emacs* editor is invoked at a user's site and the file to be edited is stored at a remote site across an Internet link. When the File Transfer Protocol is used, the user must first login to the remote site. The user copies the file to the using site and edits the file. The new version of the file is then copied by the user back to the storage site.

An alternative approach is to access the file using the Telnet protocol. In this case, the terminal handling is performed by the remote site instead of the local site so in a full duplex system such as Unix, characters must be

echoed across the Internet link. Furthermore, if the file is to be used by a program running on the user's site, the file still must be transferred by the user to the local site.

When the emacs editor is invoked under Locus, terminal handling is performed by the local site so characters do not have to be echoed across the Internet link. The editor issues normal read system calls and the operating system transparently transfers the file to the user's site from the storage site. When the file is written, it is transferred back to the storage site without any extra user commands. A program running on the user's site may access the file stored at the remote site without any user intervention in exactly the same manner as if the file was stored locally.

IP Environment	Locus Environment
<pre> ftp 8.2.0.7 Trying "8.2.0.7" (8.2.0.7) ... Open 220 bbnccq Server FTP >user sheltzer 331 Enter PASS command >pass Password: xxxxx 230 user Sheltzer logged in >get file1 local file: file1.ftp 150 Retrieval of "file1" started okay 226 File Transfer completed okay >bye 221 disconnect received, closing connections, (1736 bps, 217 bytes/sec) Transferred 217 bytes in 1 second emacs file1.ftp ftp 8.2.0.7 Trying "8.2.0.7" (8.2.0.7) ... Open 220 bbnccq Server FTP >user sheltzer 331 Enter PASS command >pass Password: xxxxx 230 user Sheltzer logged in >send file1.ftp remote file: file1 200 OK 125 Storing "file1" started okay 226 File Transfer completed okay >bye 221 disconnect received, closing connections, Transferred 202 bytes in 0 second [sic] </pre>	<pre> emacs file1 </pre>

Figure 2.2: Editing a file in the Internet and Locus environments

2.5 Communication Characteristics of the Internet

Unfortunately, the Internet communications environment presents a number of difficulties for a transparent distributed operating system. First, the typical bandwidth of an Internet path that includes a long haul network does not usually exceed 50 kilobits/sec. Second, the effective delay between two computers that communicate across an Internet link can be several hundred milliseconds or more. This delay occurs in land based lines largely because of the time required to clock data in and out of switching computers, and in satellite based communications because of the transmission delays to and from geosynchronous orbit. While land based optical fibers will reduce terrestrial transmission delays, the necessary switching computers are still expected to introduce delays far greater than those experienced in local networks today. Further, an Internet path that contains several gateways and possible satellite links will usually exhibit a much higher error rate than a local area network.

The wide range of characteristics of the underlying networks within the Darpa Internet are shown in figure 2.3.

Network	Max. Message Size (bytes)	Bandwidth	Delay	Guaranteed Delivery	Network Type
Arpanet	1008	Medium	Medium	Yes	Terrestrial
Satnet	256	Low	High	No	Satellite Network
Pronet	2048	High	Low	Yes	LAN
Ethernet	1500	High	Low	Yes	LAN
Telenet	128	Low	Medium	Yes	Public X.25 Network
Packet Radio	254	Medium	Medium	No	Varying Topology
Wideband	2000	High	High	No	Satellite Network

Low bandwidth: less than 50K bits/sec Low delay: less than 20ms.
 Medium bandwidth: 50K bits/sec Medium delay: 50ms. to 500ms.
 High bandwidth: greater than 1M bits/sec High delay: greater than 500ms.

Figure 2.3: Internet Network Characteristics

These bandwidth, delay and error characteristics mean that distributed operating system designs that are dependent on the low delay, megabyte bandwidths, and low error rates of local area networks will not extend to the internet environment. For example, Locus sends a round trip message for listing each file entry in a remote directory, and explicitly acknowledges each page written (as done in a remote procedure call oriented structure) during file operations before transferring another page. Unfortunately, these strategies, which simplify implementation in the local area network case, will perform unacceptably in the Internet case. New approaches to computer intercommunication, which are more suitable to the internet environment, must be developed if transparency is to be extended beyond local area networks.

2.6 Summary

The idea of building systems of interconnected computer networks to facilitate data and program sharing and remote access to resources was introduced by researchers in the 1970s. One such internet, the Darpa Internet,

has grown significantly in terms of the number of interconnected sites, but the available services have remained mostly limited to Telnet, FTP and mail service. In contrast, distributed operating systems such as Locus, developed for high speed, low delay, local area networks provide users and programs with the entire repertoire of single site commands and system calls to access any resource in the network. The bandwidth, delay, and error characteristics of the internet environment prevent direct application of protocols developed for LAN based distributed operating systems to the Internet. If network transparency is to be extended beyond the local area network, new protocols must be developed which are suitable for the communication characteristics of the Internet.

CHAPTER 3

Principles of Distributed System Design

3.1 Introduction

A new, hierarchical view of the internet environment is presented in this chapter. Virtual memory and local area network transparency are viewed as instances of the general application of transparency across all levels in the distributed system hierarchy. Several approaches to extending transparency beyond local area networks are described and the choice of kernel to kernel communication is defended. Strategies for overcoming the performance limitations of long haul networks are explored.

3.2 A Hierarchical View of Distributed Systems

The internet environment is often viewed as a set of loosely coupled, single site hosts, each controlled by a separate administration. Resources located on remote hosts are accessed via a different interface than local resources and the set of operations available for remote resources is restricted to a few applications such as file transfer or remote terminal service. Explicit permission must be granted each time a user or program accesses a resource located on another host.

In contrast, the *hierarchical view* of the internet environment, illustrated in figure 3.1, views an internet as a single, unified collection of processors and storage devices, available to any user or program (subject to access control). Resources are accessed in the same manner, regardless of their level in the hierarchy. As one moves from the top of the hierarchy downward, the number of available resources and services vastly increases while the cost of accessing resources may also increase in terms of higher delay and lower throughput.

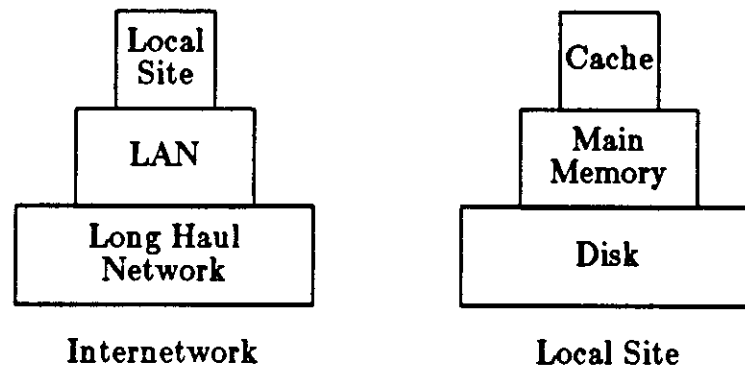


Figure 3.1: Hierarchical View of the Internet Environment

The hierarchical view of distributed systems exposes the similarities between a memory hierarchy within a single site and the hierarchy of processors and storage devices in an internet. In a memory hierarchy, each memory level provides a larger, slower resource than the preceding level. Memory management enforces information hiding by providing the user with a single, global name space and a single interface to resources, regardless of where in the memory hierarchy resources are stored. The user views a storage facility that has the high throughput and low delay characteristics of the highest level in the hierarchy with the storage capacity of the lowest level in the hierarchy. Performance transparency is achieved by applying the principle of

locality to reduce the number of accesses that must be serviced by the lower levels.

In the memory hierarchy, a short access time can only be obtained at a high cost per byte. Therefore, capacity is limited at the higher levels of the hierarchy and expands as the cost per byte decreases. For instance, the relationship between the access times (T) for cache (c), main memory (mm), and disk (d) is given approximately by [Sto 80]:

$$T_c = 0.1 * T_{mm} = 0.000001 * T_d$$

For a Vax 11/780 site configuration with 8K bytes cache, 2M bytes main memory, and a 500M byte disk, the storage capacity (C) grows as:

$$C_d = 250 * C_{mm} = 62500 * C_c$$

Similarly, in the internet hierarchy, as the distance to the local site decreases, access time also decreases. The time to access a 1K byte page from the local site (s), a LAN (l), a long haul network through 1 packet switching node (lh1) and 5 packet switching nodes (lh5) is approximately (see section 6.4):

$$T_s = 0.9 * T_l = 0.075 * T_{lh1} = 0.015 * T_{lh5}$$

Of course, the capacity of the system in terms of the number of available resources and services can greatly increase when the geographic boundaries of the system are expanded.

These access time ratios lend support to the claim that transparency in an internet environment is achievable since the differences in access time between levels in the internet hierarchy are far less severe than the access

time differences in the memory hierarchy. Furthermore, in a memory hierarchy, data traditionally must move up to the CPU for processing while in the internet hierarchy data not only moves to the using site, processing can also migrate to the data storage site.

Transparent access to resources has long been provided within a single site memory hierarchy. Recently, transparent access to resources has been extended to sites connected by local area networks. When a distributed system is viewed as a hierarchy of processors and storage devices, single site memory transparency and local area network transparency can be viewed as instances of the general application of transparency across all levels of a distributed system hierarchy.

3.3 Benefits of Internet Transparency

The benefits of organizing memory into a transparent hierarchy are well known. The benefits of network transparency on a local area network are discussed in detail in [Wal 83a]. Certain benefits are especially important when network transparency is applied across all levels in the distributed system hierarchy.

1. *There is a single interface to all resources in the distributed system, regardless of resource location.*

The user interface is simplified since a single set of commands is used for all sites. Program development is simplified because a single set of system calls is used and the knowledge of resource location is not necessary for correct program execution. Furthermore, certain integrated applications are naturally geographically distributed, such

as the collection and analysis of radar information or database operations on a database with relations stored at many locations. The task of writing software for distributed applications is simplified because the details of network management are hidden from the application.

Data and programs are easily shared between sites when remote resources are accessed in the same manner as local resources. In an internet environment that only offers a file transfer service, for example, sharing a file between sites is very difficult because the file must be explicitly copied to a using site before it is accessed.

2. *Extensibility*

As additional processing power and storage capacity are added to the internet, users have immediate access to the new resources. For example, if a powerful but scarce resource such as a supercomputer is added to an internet system, users located in different geographic regions can take advantage of the increased computing power without rewriting their applications.

3. *High availability and reliability*

In an internet system with an abundance of processors and storage devices, availability can be greatly improved if it is possible to substitute equivalent resources for one another. A uniform interface that hides the binding of resource names to specific locations is an important building block for achieving improved availability.

3.4 Arguments Against Internet Transparency

There are two common arguments against extending network transparency to an internetwork environment. First, since it is not possible nor desirable to force all sites in an internet to run the same operating system, it is argued that internet transparency is impractical. Second, since long haul networks present far higher delay and much lower bandwidth than local networks, full performance transparency is not possible. It is argued, therefore, that this difference in performance should not be masked by forcing the same interface to local and remote resources.

We claim, however, that clusters of sites running the same operating system, connected internally by LANs and interconnected by long haul networks is becoming an increasingly important network configuration. Even though it may not be practical to extend network transparency to all sites in a *physical internet*, extending transparency to selected groups of geographically distributed sites within an internet is a desirable goal.

Furthermore, we will show that for many internet topologies the performance of remote access *can* approach the performance of local access. When performance transparency is not possible, such as when large amounts of data must be transferred across long haul networks, it is still useful to have a single interface to all resources rather than force the user to have knowledge of resource location and use specialized programs for remote access.

3.5 Approaches to Network Transparency

Researchers have pursued four approaches to network transparency: remote procedure call [Bir 84], message passing [Ras 81], connection layer [Bro 82], and kernel to kernel communication [Pop 81]. Distributed operating systems across LANs have been built using each of these approaches with varying success. The approaches are described below and reasons why the fourth approach, kernel to kernel communication, was chosen as a basis for extending transparency to an internet is explained.

3.5.1 Remote Procedure Call

Remote procedure call (RPC) is a means of providing a programming language interface to remote service through the familiar syntax and semantics of local subroutine calls. In most RPC schemes, a programming language preprocessor generates client and server "stubs" which package subroutine invocations and arguments into network messages. The client stub sends messages to a remote server process which executes the subroutine, while the client awaits a response which often includes a return value.

There are several reasons why RPC is not an appropriate paradigm for remote service in an internet operating system. RPC limits remote service to synchronous behavior since procedure call semantics require the caller to block until the call returns. However, asynchronous behavior is often desirable. The ability to send and receive asynchronous signals across machine boundaries is crucial to certain applications. Furthermore, if remote service is only provided at the subroutine level, remote performance may suffer. In many instances it is better to move the data to the using site rather than run

each subroutine remotely at the data storage site.

Another drawback of the RPC model is that only those applications written in the RPC-based language can take advantage of distributed operation. Existing applications such as a database management system or operating system command interpreter must either be rewritten in the RPC-based language or must reimplement the basic mechanisms for distributed operation such as transactions, a global name service, reliable communications, etc.

3.5.2 Message Passing

In a message-based distributed operating system, all local and remote services are provided by passing messages between processes. Processes communicate by sending and receiving messages on ports or queues managed by the kernel. Since all messages go through the kernel, only the kernel needs to know where processes are located. Therefore, the application interface for local and remote services can be identical. Both asynchronous and synchronous behavior can be supported. A client can either send a message and block until it receives a reply or send a message, continue executing a task, and receive notification from the kernel when the reply is ready.

A major disadvantage of the message-passing approach is that performance, especially local performance, is likely to be poor unless process overhead is very small. The server process competes with other processes for CPU resources and must wait to be scheduled by the operating system before it can service a request. The completion of all services, including local system calls, pay the penalty of server process scheduling. In addition, message-passing usually forces data to be copied from the client process address space

to the server process address space and back to the client address space. The performance characteristics of message-passing are unattractive unless there is hardware support for fast context switching, process scheduling, and efficient sharing between process address spaces.

3.5.3 Connection Layer

A third approach to network transparency is to introduce a new layer of software between the kernel and application layer. This approach is used in the Unix United system where the additional layer of software is called the Newcastle Connection (NC). In this system, distributed operation is achieved without modification to the kernel. Instead, the location of resources are embedded in their user-visible names. The NC determines whether a local or remote operation is appropriate by intercepting all system calls and examining their arguments. If the argument indicates that a local resource is to be accessed, the NC issues a normal system call on behalf of the user. If a remote resource is to be accessed, the local NC communicates with a remote NC which sets up a remote server process to mimic the local user process.

The connection layer approach has the advantage that it is easy to provide distributed operation to existing systems because the kernel is unchanged. However, to achieve distributed operation at the program level, programs must be linked with new library routines. Local performance will likely suffer with the addition of a new layer of software. Remote performance may also be inadequate since remote service is provided by user-level processes. Perhaps the major disadvantage of this approach is that users must be aware of the location of resources. Without location transparency, it is difficult to replicate resources for reliability and to move resources to new

locations when names are embedded in programs.

3.5.4 Kernel to Kernel Communication

The fourth approach, kernel to kernel communication, offers a number of advantages over the other three approaches. In this approach, as in message-passing, the location of a resource is determined by the kernel, not by a layer of software between the application and kernel. However, in message-passing a server process must be scheduled even for local services. In the kernel to kernel communication approach, service provided by a local system call on a single site is still provided by a low overhead, local system call when the site is part of a distributed system. In addition, data copying between a client and server process on the same site is not needed because the operating system shares an address space with all processes.

In kernel to kernel communication, remote service is efficient because much of the communication protocol can be run at interrupt level. However, the ability to suspend execution pending the completion of an event, which is a requirement to support a service such as reading a page from disk, is not possible at interrupt level. In one approach to this problem, a set of special kernel processes are used for remote service. The code, stack area, and global variables of the kernel processes are resident in the operating system nucleus. Kernel processes call internal system routines directly. As network requests arrive, they are placed on a kernel queue, and when a kernel process finishes an operation it looks on the queue for more work to do. Each kernel process serially serves a request. An important distinction between kernel to kernel communication versus process to process communication used in most message-passing systems is that any kernel process can service any request.

The overhead of scheduling a matching server process to handle a given client request is eliminated.

In summary, the kernel to kernel communication approach to network transparency is attractive because local service is not penalized when a site becomes part of a distributed system, remote service is efficient because service is provided by a combination of interrupt level code and lightweight kernel processes and single site applications do not have to be reimplemented in a distributed environment because the network is hidden in the kernel.

3.6 Strategies for Achieving Internet Transparency

The implementation of network transparency across a local area network in a system such as Locus has relied on two basic assumptions that must be relaxed when the internet environment is considered. First, the local area network is assumed to support a small number of sites that are usually fully connected. Second, in a local area network, sites communicate via a high bandwidth, low delay link.

In a closely coupled distributed operating system such as Locus it is convenient for all sites to maintain detailed state information about all other sites in the network. When a single site joins or leaves the network all sites participate in recovery and merge algorithms. Furthermore, all sites maintain a globally replicated data structure (called the mount table) which records the connection between individual parts of the network-wide name space. Whenever membership in the network changes, all sites alter copies of their mount table so a consistent name space is always maintained.

As clusters of sites are interconnected by long haul networks, the number of sites in the system may grow from less than one hundred sites to hundreds or thousands of sites. In a very large distributed system the requirement that sites maintain detailed, up to date, information about all other sites in the system cannot be met. To solve this problem, Reiher proposes developing a system of domain name servers that keep replicated name information for specific sets of sites [Rei 85]. Only domain name servers, instead of all sites in a region, need to maintain a consistent view of the name space. Non-server sites request remote naming information from domain name servers and store the responses in "lazy evaluation" caches to reduce the number of requests to servers. Cache entries contain enough information to determine if the entry is up to date when the entry is used. If the entry is incorrect, the non-server site requests the correct information from a server.

When sites communicate over a high bandwidth, low delay link, performance transparency is easily achieved with relatively unsophisticated protocols. Since the delay in transmitting a message from one site to another is very low, the number of messages that are needed to service a user or system task is not critical. A simple addressing scheme whereby each site address corresponds to a single physical local area network address is sufficient for packets to be routed between sites.

In the internet environment, the bandwidth and delay characteristics of long haul networks make remote access potentially orders of magnitude slower than local access. Simple request response protocols designed for local area networks do not efficiently use the available bandwidth of long haul links. If many messages must be sent to a remote site to achieve a given task,

remote performance will be dramatically inferior to local performance.

An addressing scheme is needed that allows all sites in an internet to communicate with each other, not just those connected by a single local area network. A fragmentation and reassembly strategy must be implemented to handle messages that traverse networks that enforce different maximum packet sizes. The internetwork addressing and packet fragmentation problems are easily solved by encapsulating messages within standard internet headers such as those specified by the PUP protocol or the Darpa Internet Protocol. A solution to the performance problem is more difficult.

If performance transparency is to be achieved in the internet environment, two goals must be met. First, the number of times a using site communicates with a remote site must be minimized without losing the functionality of network transparency. Second, when a using site must communicate with a remote site, the number of messages and number of bits per message that are transmitted to achieve a given task must be minimized.

In a memory hierarchy, the principle of locality is used to reduce the frequency of access to the lower levels of the hierarchy. Similarly, the principle of locality can be applied to a large distributed system to reach the first goal in achieving performance transparency which is to reduce the frequency of communication between sites.

A new methodology for protocol development called *semantics-based protocol design*, is applied to reach the second goal. Semantics-based protocol design attempts to minimize the number of messages that must be sent to accomplish frequent operating system and user tasks.

In the internet environment, each level in the hierarchy not only contains memory resources but also processor resources. The ability to move a process to the data, which is not available in a memory hierarchy, can also be used to reduce the information that must be transmitted across a link.

The combination of these three strategies, exploitation of locality, semantics-based protocol design and moving the process to the data are applied to reduce the difference in the performance of remote and local access in the internet environment. Ideally, we wish to have the access time of the highest level in the distributed system hierarchy (the using site) and the capacity of the lowest level (all sites and storage devices in the internet system). Each of the strategies used to reach this goal is discussed in the following sections.

3.6.1 Locality in Distributed Systems

Distributed operating systems constantly search for information that is stored at remote sites (e.g. the translation from string name to object) and then discard this information when in fact, the information may be used again in the near future. If locality exists, caching recently used information can help reach the first goal towards achieving performance transparency which is to reduce the number of times that remote sites must be referenced for information.

Caches are typically placed between a large, relatively slow and expensive source of information and a much faster consumer of that information. The cache capacity is small and expensive, but quickly accessible. The goal is for the cache behavior to dominate performance but the large storage

facility to dominate costs, thus giving the illusion of a large, fast, inexpensive storage system. Successful operation depends both on a substantial level of locality being exhibited by the consumer, and careful strategies being chosen for cache operation - disciplines for replacement of contents, update synchronization, etc. The value of successful caches often is enormous, representing the difference between satisfactory cost/performance and failure.

Caches have been used for many years as part of the single site memory hierarchy between main memory and the central processor. Recently, distributed caches have appeared in integrated hardware systems constructed of multiple processors. In these multiprocessor architectures, each processor has a private cache, and a mechanism exists to prevent the simultaneous existence of different versions of the same data block in different caches. The considerations in the design of multiprocessor hardware caches are present, in analogous ways, in a distributed operating system, and their proper resolution is of similar importance.

The property of "locality of reference" has been observed in program execution [Den 72] [Mad 76], single site file access [Maj 84], as well as database access [Kea 83]. We are interested in the degree to which locality is also exhibited by distributed operating system functions such as name to object translation, remote file access, and remote site access. We must identify those objects which have sufficient locality to indicate that caching will be beneficial and determine at which level in the distributed system hierarchy the cache should be built.

The property of "locality of reference" has two components: temporal locality and spatial locality. Temporal locality refers to the observation that information which will be in use in the near future is likely to be in use currently. Spatial locality means that the locations of references in the near future are likely to be near the locations of current references. By observing temporal locality in the patterns of referencing remote information we can determine what information should be cached. By observing spatial locality in reference patterns, we can determine how large each cache entry should be and whether it is advantageous to prefetch certain data. For example, entries in a name service cache might be single name to object mappings if little spatial locality exists or full directory pages if sufficient spatial locality exists whereby several directory entries within the same directory page are accessed within a short time period. The definition of spatial locality can be broadened to include logical relationships between objects in addition to physical "nearness". It might be useful to bring an entire set of logically related objects into a cache whenever a single object in the set is accessed.

The following objects in a distributed system may show sufficient locality to indicate that caching is beneficial:

File pages

Name to object translations

Directory pages

Files

Each of these objects is described below. A full discussion of name service caching is given in chapter 4. An investigation into the logical relationships between objects remains a subject for future research.

Many operating systems maintain a buffer cache in main memory of pages recently read from disk. If sufficient locality exists, the number of accesses to disk is greatly reduced as most pages are found in the buffer cache. A distributed operating system can take advantage of the buffer cache by entering pages read from remote sites into the cache. Subsequent access to that data will not require any network traffic. Cached entries are invalidated when a file is closed if the file is stored remotely. The number of remote accesses is further reduced if the first data page of a remotely stored file is returned with the response to open for read message. This can be viewed as an example of spatial locality where the first page of a file is logically related to the information returned in the response to open message. By prefetching the first page of the file, network traffic is reduced.

It is important to note that the distributed operating system protocol must be specifically designed to exploit existing kernel buffer caching. If messages are sent at the application process to process level rather than the kernel level, it may not be possible to take advantage of buffer caching within the kernel.

Another important opportunity for exploiting locality in a distributed system occurs in the mapping of a name to the object it represents. In an operating system such as Locus, where the name space is tree-structured, a significant amount of network traffic results from translating a pathname to an object. Even in a single machine Unix system such as Berkeley Unix, release 4.2, name mapping consumes approximately 40% of system overhead [Kar 84]. By caching name to object mappings at the using site, the number of messages sent to remote sites during pathname expansion can be reduced.

Instead of caching individual name to object mappings, it may be advantageous to cache entire directory pages because many common tasks such as listing the contents of a directory, expanding wildcard arguments, and accessing parent directories use information that is stored in directory pages.

A using site can cache some small number of pages for remote files and directories and still stay within the limits of reasonable main memory cost. However, it may be desirable to cache entire files to decrease message traffic further. If files are cached at each using site, main memory in each site would be quickly exhausted. Instead, a site in each LAN cluster can be configured as a file cache server and relieve individual sites from their inherent cache size limitations. When a file on another cluster is referenced, the file cache server is first queried. If the file is stored at the file cache server, the file is accessed without expensive intercluster traffic. A file cache machine has been demonstrated at Cambridge University for a system of LANs connected by a high delay satellite link [Ric 83].

3.6.2 A Semantics-based Approach to the Design of Network Protocols

Network protocols are traditionally designed as a series of layers. Each layer offers a well defined service to the layer above it, while hiding the implementation details of the service. The purpose of building layered protocols is to simplify network design by partitioning the full network design problem into a small number of independent, manageable pieces. Layered design also facilitates the substitution of different implementations for a given layer.

In the seven layer ISO model, the *physical layer* is concerned with the transmission of bits over a communication channel. The *data link layer* provides an error free transmission facility between a site and a network or between nodes in a network. Data is transmitted in sequential units called frames and lost or damaged frames are retransmitted at this layer. The *network layer* provides a facility for one site to communicate with another site in a network. The issue of how to route data between sites is solved in this layer. The *transport layer* is responsible for providing an error-free channel between sites. Messages sent over the channel must be delivered in sequence; duplicate messages must be detected and discarded; lost or damaged messages must be retransmitted; and the rate of message transmission must be flow controlled so the sender does not flood the receiver.

The distinction between the remaining three layers, the *session*, *presentation*, and *application layers* is not well defined. User services such as remote terminal service, distributed database access, and file access and common functions used by several user services such as text compression or data encryption fall within these three layers.

The interface between adjacent network protocol layers defines the set of primitive operations the lower layer offers the higher layer. The amount of information that flows across this interface is kept to a minimum so modules for a given layer can be designed and implemented independently.

The bottom up, layered approach to the design of network protocols has been successfully used in the design of network architectures such as SNA, DECNET, and the Arpanet, and correct network operation has been demonstrated. However, correct operation alone does not insure adequate

performance for applications.

Three objections to the traditional protocol design methodology can be raised. First, by isolating the details of each network layer, the performance of certain applications that use the network may suffer. For example, the Arpanet handles single packet messages of less than 1008 bits much more efficiently than multipacket messages but this implementation detail is hidden from protocol layers above the network layer. For example, the measured mean delay of transmitting a message of 1440 bits (which is the Locus message header plus Internet header size) across 5 IMP* hops is 318 ms while the mean delay for transmitting a message that fits within a single Arpanet packet is only 150 ms. If an application sends small, fixed size messages there is a clear performance advantage to reducing the size of messages that travel across the Arpanet to less than 1008 bits. Second, by concentrating on the lower layers of the network protocol, the services provided by the lower layers do not always match the services required by applications. An application may require rapid, reliable delivery of single messages. If the only service offered by the transport layer is a dynamic virtual circuit, the delay in establishing the circuit each time a message is to be delivered may be unacceptable. Third, by designing each protocol layer independently, many functions such as flow control, duplicate message detection, message sequencing and guaranteed message delivery may appear in several layers and interact in undesirable ways.

*The IMP or Interface Message Processor is the packet switching node in the Arpanet

A new approach to protocol design, *semantics-based protocol design*, is proposed as a methodology to reach the second step towards performance transparency which is to reduce the number of messages that are sent to achieve a given task. Historically, network protocol development has centered on the lower layers of the network architecture. Instead of focusing on the lower protocol layers, the semantics-based approach centers on defining what activity should be represented in messages to support applications. Once the proper semantics of messages are understood, the lower levels of the protocol are built to efficiently support the reliable delivery of messages. The semantics-based methodology is application-driven or *top down* while the traditional protocol design methodology is data communications driven or *bottom up*.

The activities that must be supported efficiently in a distributed operating system protocol are system calls and frequent user commands. The communication style that is predominant in these activities is a request followed by a response. We must determine whether connection-oriented protocols used in long haul networks or LAN request-response protocols can efficiently support these activities in an internetwork environment.

In a typical connection-oriented protocol the communication path is between a user or application program on a local site and a server process on a remote site. The local user or application program first sets up a session with the remote site to guarantee that the requester has permission to use the service; to set up any session specific parameters such as byte order; to guarantee that the remote site has adequate resources to provide the service; and to establish sequence numbers for flow control, duplicate message detec-

tion, etc.

If sites have disjoint process and file name spaces the server process on a remote machine must be identified. Usually the service has a well known address where a process always listens for requests. When a request comes in, the server creates a new process to handle the request and returns the identification of the new process to the requester. The initial session is closed and the requester opens a new session with the new remote server process. When the requester has finished using the remote service, the session is closed and the remote site frees any resources that were dedicated to provide the service.

A connection-oriented protocol is clearly not well matched to a request-response communication style. The overhead and delay of connection establishment preclude acceptable support for distributed operating system functions in an internet.

In contrast to connection-oriented protocols, the request-response protocol used between sites in a transparent local area network does not suffer from the overhead and delay of connection establishment. In a Locus network, for example, all sites share a global name space so no initial connection is needed to establish the address of a remote server process. The communication path is from the operating system on one site to the operating system on another site. No service authentication is necessary because the client is the operating system rather than a user. A permanent, logical connection between sites is established when a site first comes up, so no session specific parameters or sequence numbers need to be set up each time a remote service is used.

The model for remote service in Locus is shown in figure 3.2. Initial system call processing is performed at the local site. When a point is reached in the execution of the system call where remote access is needed, a message is sent to the appropriate site. A response is returned from the remote site and the execution of the system call is continued locally. Unfortunately, remote service may require several requests and responses to be exchanged during the execution of a single system call. While this has been acceptable in the LAN environment, the increase in message traffic will degrade performance over a higher delay, internet link.

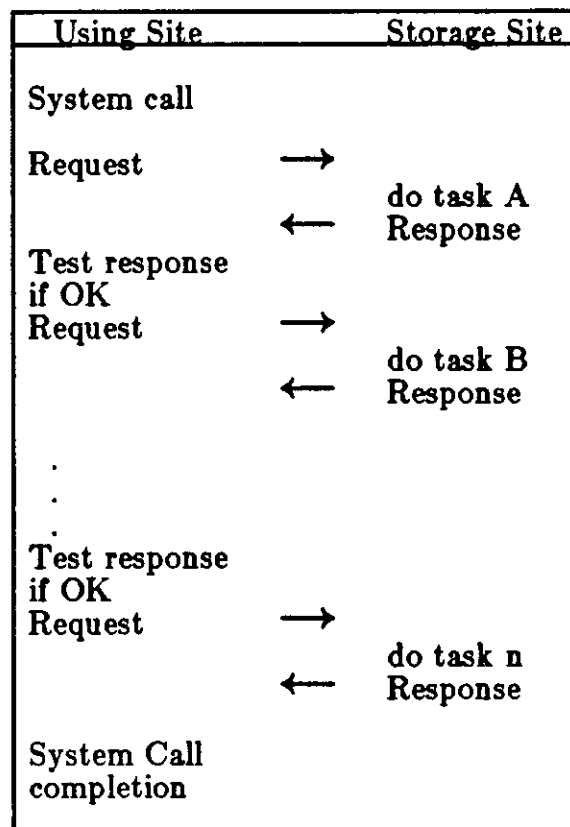


Figure 3.2: Locus Remote System Call Processing

It is often possible to raise the semantic level of the initial request message so more of the system call processing is performed at the storage site (see figure 3.3). The test of the outcome of the request is executed at the storage

site instead of the using site. In the likely case that the outcome is successful, execution of the system call continues at the storage site. If the outcome is failure, a response is immediately returned to the using site.

For example, in LAN-Locus, the last close of a remotely stored file (open for modification) often requires a Commit request and response message exchange. If the Commit succeeds, a Close request is sent by the using site and a response is returned. The semantics of the initial request can be raised to mean: "Commit this file and if the commit succeeds, close the file". Therefore the same task can be accomplished with a single Commit_and_Close message exchange.

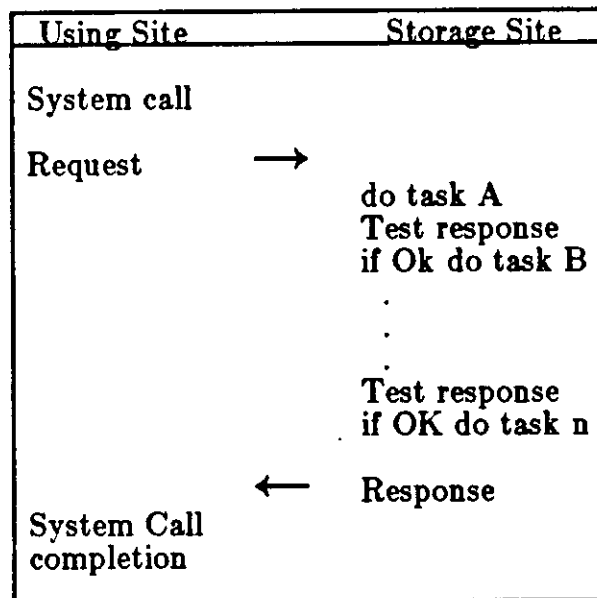


Figure 3.3: Internet-Locus Remote System Call Processing

3.6.3 Remote Execution

When the data needed by a given process does not reside at the site on which the process exists, either the data can be moved to the initial process site or the process can be moved to the data storage site. The second alterna-

tive is only possible in a system that provides network transparency, i.e. one where the execution of a process at the data storage site is semantically equivalent to the execution of the process at the initial process site. Such items as the file naming hierarchy, current open file descriptors, process identifiers used for IPC, pending interrupts, device addresses, etc. as seen by the relocated process must all be correctly interpreted.

Locus permits programs to be executed at any site in the network, subject to permission control, just as if they were executed locally. Furthermore, a process can change its site of execution even while in the midst of execution. This ability for a process to move to the storage site(s) of its target data is invaluable as a means of reducing message traffic due to data movement in the internet environment.

3.7 Summary

A new, hierarchical view of the internet environment is presented in this chapter. The benefits of extending network transparency across all levels of the hierarchy include a consistent user interface, the ability to use the full repertoire of single site commands on all remote resources, ease of distributed application development, enhanced sharing of data and programs, and the potential for high availability and reliability.

If the performance of remote access, even across long haul networks, is to approach the performance of local access, two goals must be achieved. The frequency of communication between sites must be reduced and the number of messages transmitted to achieve a given task must be minimized. Three strategies are proposed to reach these goals: exploitation of locality,

semantics-based protocol design, and remote execution.

CHAPTER 4

Name Management in an Internet DOS

4.1 Introduction

Name service is an important component in the overall behavior of distributed operating systems. The use of a distributed name cache to improve system performance and reduce the elapsed time to access remote resources is explored. A classical cache design evaluation, applied to the problem of distributed name management, is presented. Reference strings, collected from a production distributed system, are used as input to a trace-driven simulation to determine the degree of locality exhibited in name to object translation, evaluate cache parameters, and justify the utility of a distributed name cache.

4.2 Name Service in a Distributed Environment

In a system with a transparent name space,* it is the system's responsibility to find the resource given its name, and set up all necessary bindings for subsequent access. The cost of this distributed name lookup, and the associated update of relevant (perhaps partially replicated) tables as resources are created, destroyed, and moved, can be a major cost of the distributed system's operation. In a production Locus installation for example, it is not uncommon for half of total network traffic to be in support of name service.

*Strictly speaking, we refer here to *location transparency*.

An immediate question is whether this key function is susceptible to cache based speedup methods. If so, the performance improvement can be remarkable. However, one must first determine whether name lookup exhibits the requisite degree of locality, and whether the necessity to invalidate other systems' cache entries when a name service update occurs represents significant cost, complexity and delay. If these characteristics are satisfactory, one can then proceed with the determination of the other relevant cache design parameters.

In order to address these issues, over 15 million "name-service" reference string entries were collected during normal operation of a production Locus system. Each such entry represents a path name element in the distributed Unix directory hierarchy. These measurements serve as a basis for much of the discussion in this chapter. The reference strings were input to a trace-driven simulation which was used to determine the degree of locality in directory referencing, determine directory cache parameters, and evaluate the overhead of maintaining multicache consistency.

4.3 Pathname Expansion in Unix and Locus

In a hierarchically structured file system such as Unix or Locus, each file in the system is either a data file or a directory. Directories contain entries of the form <string name, file descriptor pointer> where the file descriptor pointer is an index into a table of file descriptors (called "inodes" in Unix) and each inode contains the device addresses of the actual data pages for the target file, plus status information. An object is referred to by a pathname which is a sequence of directory names separated by slashes and ending in a file name. A pathname starts either at the root directory or the current

working directory.

The system maps a name to an object by reading the first directory component in the pathname and then sequentially searching the entries within the directory for a match on the string name of the next pathname component. If an entry is found, the file descriptor pointer of the entry is used to locate the device addresses for the next pathname component. If the next component is the last component in the pathname, the target object has been found. If not, the pathname component is a directory and searching continues.

In a distributed system, the directories and target object that are referred to by a pathname may be stored at sites other than the site that requests the target object. There are two approaches to pathname expansion in a distributed environment. In the first approach, called *transparent pathname expansion*, each directory is brought across the network from the storage site and searched at the using site (the site that requests the pathname expansion). This approach has the advantage that distributed operation is identical to single site operation as long as there is a transparent mechanism for reading remote pages. However, a substantial amount of network traffic may be generated during pathname expansion as each directory component is opened, read, and closed.

In the second approach called *remote pathname expansion*, the pathname is expanded at each site that stores a pathname component. When the using site finds a pathname component that is stored remotely, it packages the remainder of the pathname into a network request message and sends it to the site that stores that component. The storage site services the request

by continuing pathname expansion, opening and searching the directories locally. If all of the remaining components are stored at the storage site, then the result of the pathname expansion is returned to the using site. If the storage site finds that a component is stored remotely it sends the remainder of the pathname to the next storage site and pathname expansion continues there.

This approach reduces network traffic for pathnames that contain many directories all stored at a single remote site. However, if the pathname contains directories that are stored at several sites, network traffic is not reduced. More importantly, if there is a significant level of user program access directly to directory pages, substantial directory page traffic results in addition to remote pathname expansion messages.

Measurements of the UCLA Locus network indicate that approximately 20% of all commands issued require user program access to directories. Common tasks such as listing the contents of a directory, copying or removing all entries in a directory, expanding wildcard arguments, and accessing parent directories, all use information found in directory pages. With remote pathname expansion, directory pages remain at the storage site, so this approach does not effectively reduce message traffic in many important cases.

The Locus distributed operating system supports transparent pathname expansion. References to a remotely stored directory that is currently open at a using site do not generate network traffic if the directory pages are found in the local buffer cache. However, when a remotely stored directory is closed, all of the associated file pages are flushed from the buffer cache to insure that only one version of the directory exists in the system. Thus, the

next open of a remote directory involves rereading all the directory pages to be searched.

An example of the network traffic generated to expand a pathname (*/th/foo/file1*) under Locus where the directories and target file reside at a site different than the using site is shown in figure 4.1. The *th* directory is opened and read across the network. The directory is searched for an entry that contains the string name *foo* and when such an entry is found, the *th* directory is closed. Next, the *foo* directory is opened and read across the network and the string name *file1* is searched for. When a match is found, the *foo* directory is closed and the target data file (*file 1*) is opened.

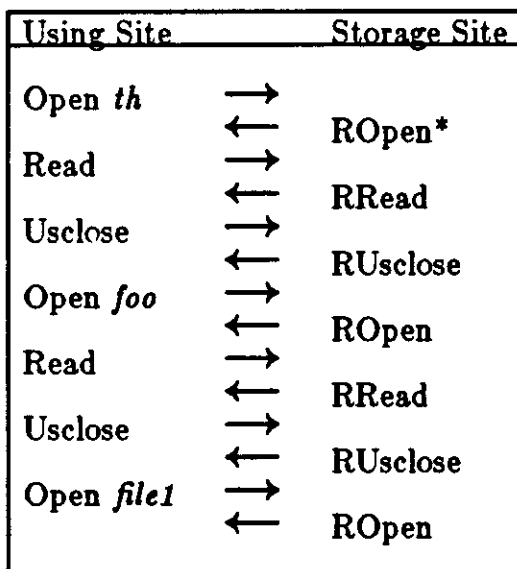


Figure 4.1: Message Traffic for Pathname Expansion of */th/foo/file1*

Fortunately, the activity of most users is usually confined to a small, slowly changing subset of the entire name hierarchy. Furthermore, most

*Responses are indicated as Rmessage-type, ie. the Open Response is shown as ROpen. The explicit ACK messages sent for each Locus message (except for Read and RRead) are not shown.

directories have a high read to modify ratio. This behavior implies that the number of messages due to pathname expansion can be reduced by caching directory pages and related file descriptors at each site even after the directory has been closed. If significant directory reference locality exists, then current references to directory pages are likely to be found in the directory cache and will not generate network traffic. Total network traffic will be decreased and more importantly, the elapsed time to reference remotely stored objects will be reduced.

4.4 Issues in Distributed Cache Design

In order to properly evaluate the potential effectiveness of a cache, it is important first to understand the necessary design goals and issues. Then measurements and simulations can be used to determine specific parameters. The goals in the design of a hardware or software cache [Smi 82] are:

1. Maximize the probability of finding a reference in the cache (hit ratio)
2. Minimize the time to access the information that is in the cache (access time)
3. Minimize the delay due to a miss
4. Minimize the frequency and overhead of invalidating a cache entry.

A careful selection of cache parameters, including cache size and replacement algorithm, is necessary to achieve these goals. A distributed cache presents the additional problem of guaranteeing multicache consistency. If caches at several sites store a copy of the same information (e.g. a directory page) and one of the sites modifies the information without notifying the

others, an inconsistent state results with possible disastrous consequences.

The hit ratio for the cache is improved by increasing the cache size, given a significant amount of directory locality. However, the memory requirements for the system and the time to access a given cache element also increase as the cache size increases. Furthermore, some directory references are unique (never rereferenced), so increasing the cache size beyond a certain point will not improve the hit ratio. In a distributed cache, the overhead of maintaining multicache consistency increases as cache size increases because a larger cache increases the probability that a site is caching a page that must be invalidated.

There are several approaches to maintaining consistency for a multisite directory cache. In the simplest approach, directory pages are flushed from the cache whenever the directory is closed. A request to open for modification is blocked until all sites have closed the requested directory. This policy insures multisite cache consistency since a site that has a directory open for modification knows that no other valid copies of the directory page(s) exist at other sites. However, the cache is only useful during the period that a directory is open.

In another approach, directory pages remain in the cache after the directory has been closed. Whenever a storage site receives a request for modification for a directory page, it broadcasts a message to all other sites identifying the directory page. Each site examines its cache for the page and invalidates it if present. Although this scheme may work well for a few sites, as the number of sites increases, the network traffic generated becomes prohibitive.

A better solution is for the storage site to record the sites that have requested a given directory page. When the storage site receives a request to modify the directory page, only those sites in the list are notified to flush the page from their caches. The overhead for cache invalidation is acceptable if there is a low rate of directory modification to directories that are shared among sites and if only a small number of sites share a directory when that directory is modified.

When the directory cache is full and a cache miss occurs, some existing cache entry must be replaced with the target reference. A good replacement algorithm is necessary to achieve a satisfactory hit ratio. Choices include a global LRU algorithm, LRU per process or user, or first in first out. The delay due to removing the selected entry from the cache and bringing in the target reference must also be minimized.

4.5 Directory Reference Measurements

A trace-driven simulation was used to investigate directory reference locality and evaluate different design choices for the distributed directory cache. Directory reference strings were collected on each site of the 15 site UCLA Vax Locus network for 10 hour periods for 6 days. The network commonly supports more than 150 active users during the five hour busy period of each day. There are various organizations administrating subsets of the machines; as a result, certain communities of users regularly cross many machine boundaries, while other users' activity is primarily local.

An event trace was recorded each time the operating system referenced a directory component during pathname expansion. Each event contained the

name of the directory reference, its file descriptor, an indication of whether the directory was stored at the site that issued the reference or at some remote site, an indication of whether the directory reference was found in the existing buffer cache of the using site, the type of reference (for read or for modification), a networkwide timestamp, plus other information. References to directories in which the directory was the target file (e.g. directory listing commands) were not recorded. Approximately 2 to 3 million entries were collected per day.

The reference strings for each site were combined and sorted by time.* Cache simulations were run on the combined, multisite reference string so the caches on all sites were simulated simultaneously. The extent of directory sharing between sites and the effect of multisite cache invalidation could thus be evaluated.

4.5.1 Measurement Results

The data which was collected showed differences in certain measurements among sites, but for any given site, the measurements were quite stable.

For a given site, the percent of references for remotely stored directories, the percent of references for modification, and the percent of remote references not found locally, showed little variance during the 6 days of measurements. For all sites, the percent of directory references for modification also varied little, with an average of 2.5% and a standard deviation of 0.5. The variance among sites for the local versus remote values was more

*Locus contains an intersite time synchronization facility that keeps the clocks on all sites within a few milliseconds of one another.

pronounced because different sites support different mixes of local and remote service. Although local directory references were dominant on all sites, the percent of references for remotely stored directories varied from 2.3% to 14.6%. The percent of remote references not found at the using site varied from 20.2% to 83.2%. On sites where remote traffic was mostly due to the activity of background processes that kept directories open, this percentage was low, but on sites where remote traffic supported normal interactive computing, the value was much higher.

Directory reference traces collected for 10 hours for a single example site are shown in figure 4.2. As expected, most of the references are local (89%). However, a significant portion (71%) of the remotely stored directory pages must be brought across the network during pathname expansion. The remaining references to remotely stored directories are references to directories that are currently open locally and are therefore found in the buffer cache ("incore") of the using site. The goal of a directory cache is to increase the number of remotely stored directory pages that are found incore at the using site. The number of directory references that are requests for modification is quite low (1.3%) but single site measurements are not enough to determine the amount of directory sharing when modification is requested. The results from the trace-driven simulation using the combined data from all sites is needed to determine the overhead of multicache invalidation.

	All references	Local references	Remote references
# of references	354491	317199	37292
% of references	100.0	89.48	10.52
% of references for mod	1.27	1.07	2.99
% of references not incore	17.41	11.12	70.87

Figure 4.2: Results of Directory Reference Event Collection

4.5.2 Locality in Directory Referencing

The property of "locality of reference" has been observed in program execution, file access, as well as database access. We are interested in the degree to which locality is also exhibited by operating system functions and in name to object translation in particular. If locality exists, caching recently used information is likely to reduce the overhead of operating system management.

A measure of the locality present in a reference string is given by Rodriguez-Rosell [Rod 76] and used by Kearns [Kea 83] to demonstrate locality in database reference strings:

$$L(t,r) = w(t,r)/r$$

where $L(t,r)$ is interpreted as the instantaneous locality measure at time t . This measure can be applied to directory reference strings by setting:

t = time as measured in # of directory references

r = window size in # of directory references

$w(t,r)$ = working set size at time t for window r

= the # of distinct directories referenced among the r most recently referenced directories in the reference string

Averaging $L(t,r)$ over the number of references gives the average locality $\overline{L(r)}$ for a given window size. A reference string that exhibits little rereferencing

and thus has poor locality will have a value of $\overline{L(\tau)}$ near 1 over a wide range of window sizes. If there is substantial rereferencing in a reference string, then as the window size increases we expect $\overline{L(\tau)}$ to decrease rapidly as references are likely to be found in the current working set so the average working set size does not increase.

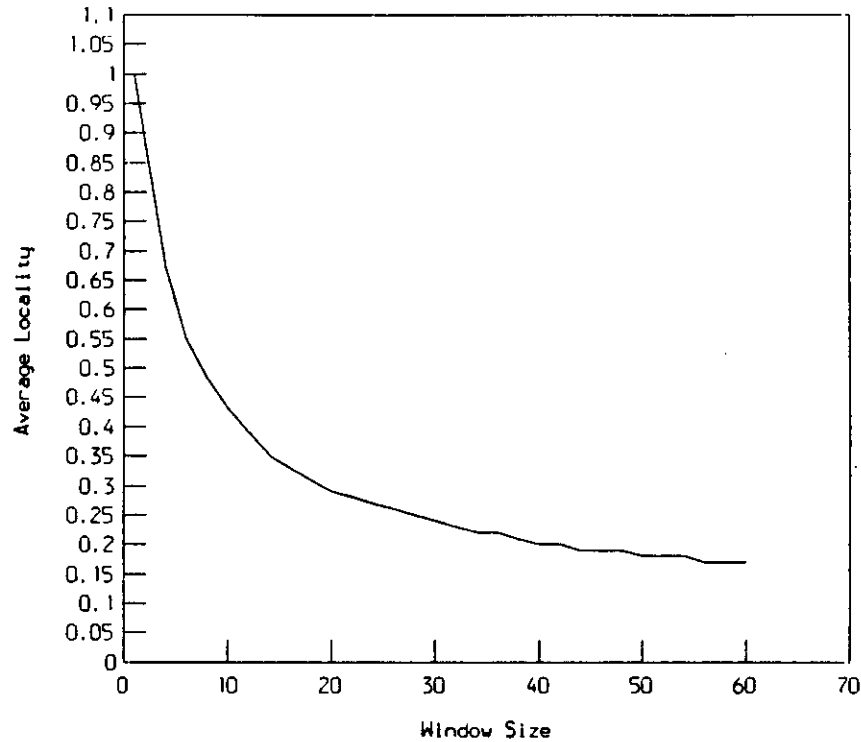


Figure 4.3: Average locality $\overline{L(\tau)}$ as a function of window size τ for directory references

A plot of the average locality $\overline{L(\tau)}$ versus window size for the example single site directory reference string is shown in figure 4.3. The sharp decrease in $\overline{L(\tau)}$ implies a significant level of locality in directory referencing and suggests that a directory cache of reasonable size would be very beneficial. At a window size of about 40, the curve flattens out, indicating that some small number of directories are never rereferenced.

4.5.3 Trace-driven Simulation Results

A trace-driven simulation was used to evaluate directory cache size, replacement algorithms, and the overhead of maintaining multicache consistency. A plot of the miss ratio versus cache size for a directory cache using a global LRU replacement algorithm is shown in figure 4.4. The miss ratio is the probability of not finding a remotely stored directory during pathname expansion at the using site. The cache size is given in number of directory pages with the assumption that each directory is stored on a single page. The cache size would be slightly larger to support the small number of directories stored on more than one page. A cache size of just 15 directory pages for the Locus site considered in figure 4.2 reduces the miss ratio from about 71% without a directory cache to just 11%. Almost 95% of the remotely stored directory references will be found at the using site with a cache of 40 pages. Therefore, in most cases, the elapsed time to access remote objects will be greatly reduced since 95% of all accesses to remote objects will not include the overhead of pathname expansion. The hit ratio for all sites with a cache of 40 pages varies from about 87% to 96%. Increasing the cache size further does not appreciably improve the miss ratio as some small number of directory pages are never rereferenced.

Instead of using a single, global cache at each site, a cache could be dedicated to each user. Cache entries would be replaced as each user cache filled up. Our measurements show that the number of active user ids per site varied from a few to over twenty. Users frequently execute processes remotely, especially on server sites, thus increasing the number of user ids per site. Results from the trace-driven simulation show that although a cache per

user (with a per user LRU replacement algorithm) uses a smaller cache size per user to achieve the miss ratio of the global cache, the total number of pages dedicated to the directory cache is substantially greater.

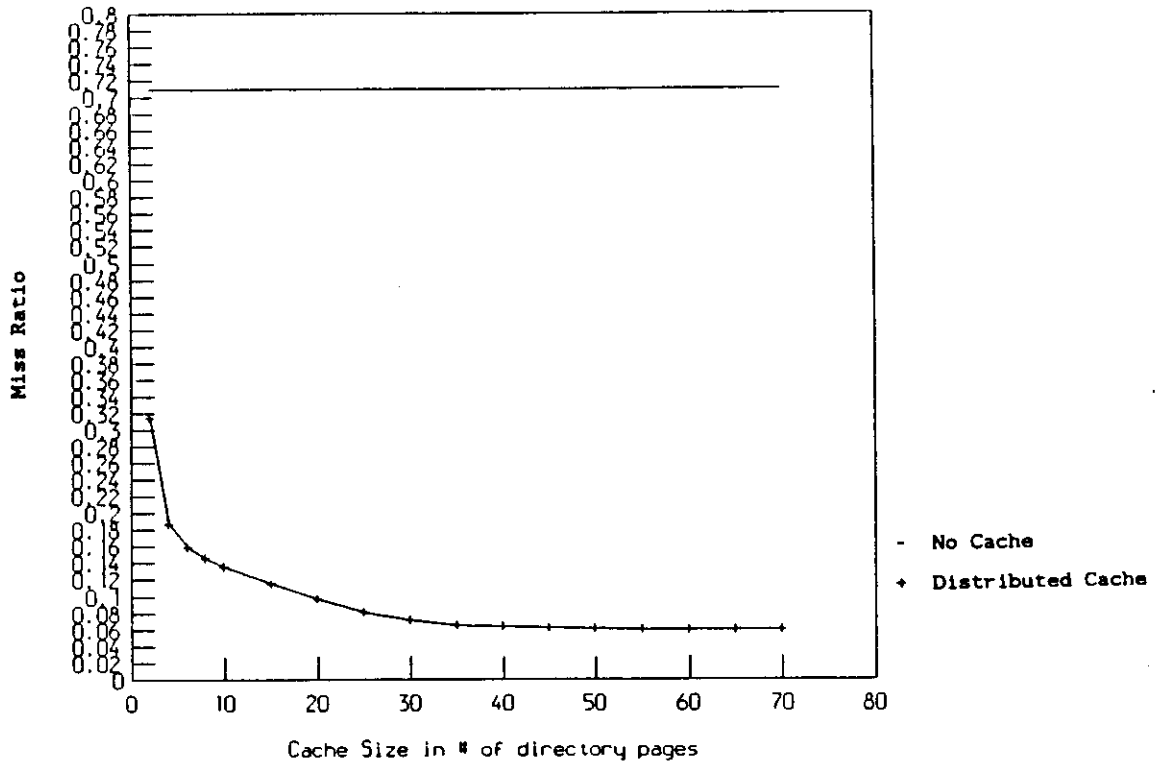


Figure 4.4: Miss ratio versus window size for a directory cache

When a request for modification is received by the storage site of a directory, the storage site must send a cache invalidation message to each site that currently has the page in its cache. Both the number of directory references that require cache invalidation messages and the number of sites that must receive cache invalidation messages must be very low for the distributed cache to be effective.

Data from the directory reference traces for a single day indicate that over a 10 hour period there were 2,474,407 total directory references issued by all sites. Results from the trace-driven simulation show that even if each site

provides a cache of 60 directory pages, only 1265 or 0.051% of the references require cache invalidation. The distribution of the number of sites that need to be sent cache invalidation messages for each reference that requires cache invalidation is shown in figure 4.5 for a cache of 60 pages. Only 93 references or .0038% of the total number of references require more than a single cache invalidation message. Thus, the overhead of maintaining multicache consistency is quite low and should not be costly even as the number of sites in the system grows substantially.

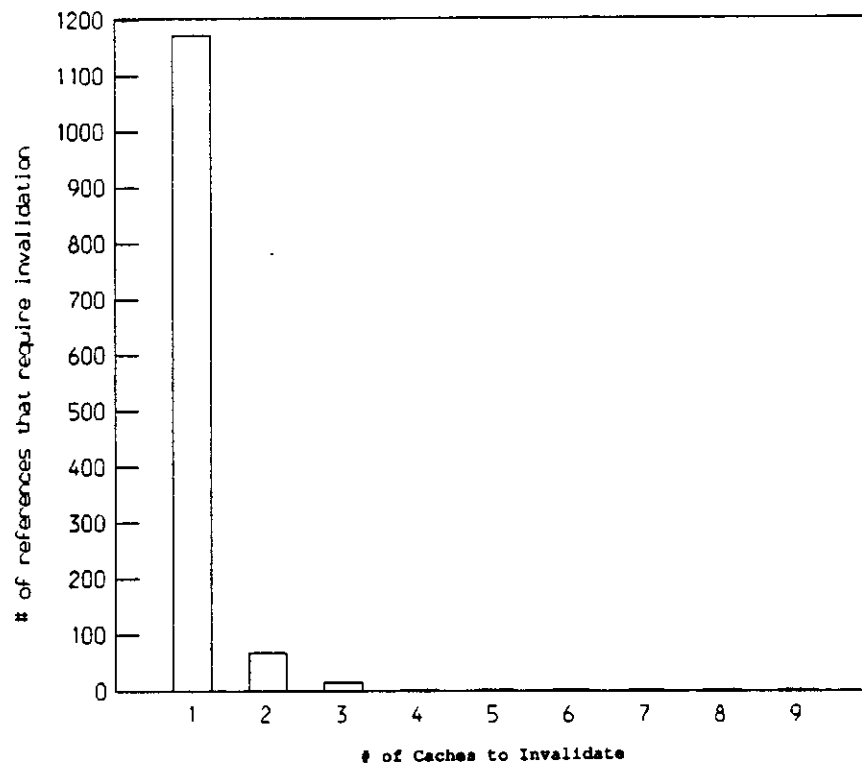


Figure 4.5: Distribution of # of sites that require cache invalidation messages

A plot of the miss ratio versus cache size for a single site cache (without cache invalidation) and the miss ratio versus cache size for a distributed cache where cache invalidation is generated by the simulation, is shown in figure 4.6. The slightly higher miss ratio of the cache invalidation plot is

mostly due to counting all references for modification as misses (since they must be serviced by the storage site) rather than from the removal of invalidated entries from the cache.

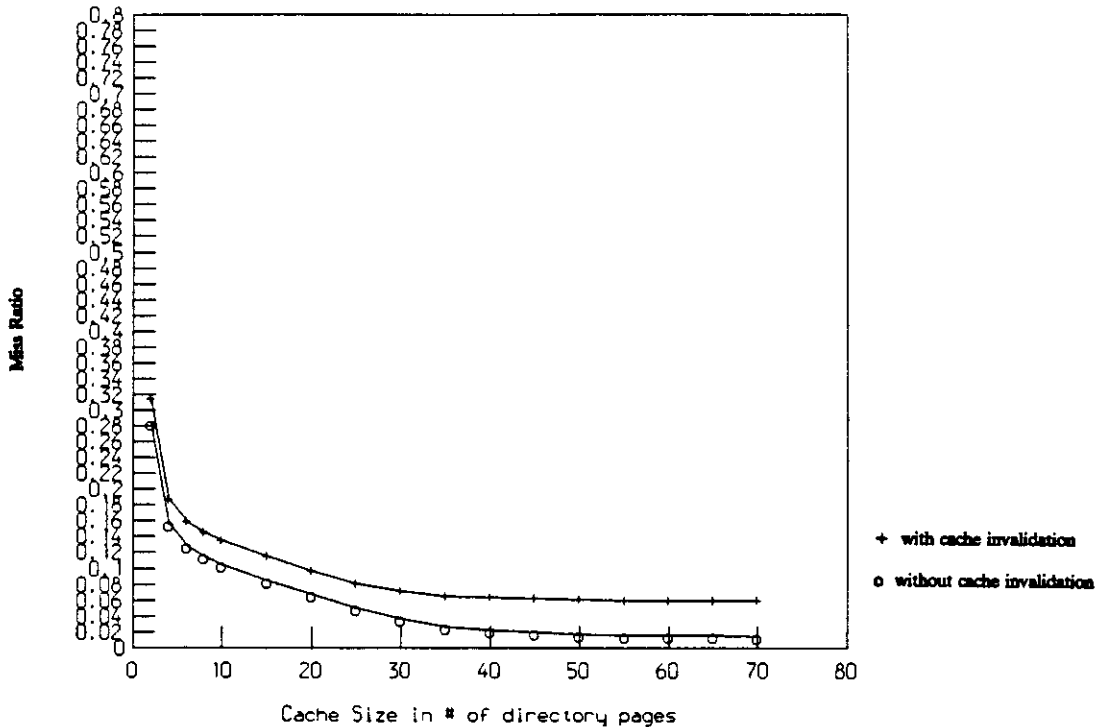


Figure 4.6: Plot of miss ratio with and without cache invalidation

4.6 Operation of the Distributed Directory Cache

Pathname expansion with a directory cache works as follows. When a using site searches a remotely stored directory during pathname expansion and the directory pages are not found in the directory cache at the using site, a "no open read" (NOR) request message is sent to the site that stores the directory. The storage site returns the file descriptor (inode) and the first directory page to the using site, which enters the items into the local directory cache. The storage site adds the using site to a list of all sites that

currently have that file open for NOR. Additional directory pages are read by the using site as usual and stored in the cache. Remotely stored directory pages are removed from the using site directory cache on an LRU basis.

When a cache miss forces a directory page to be removed from the cache, the storage site must remove the using site from the NOR list for that directory. To reduce the delay due to a cache miss, an explicit "removed from cache" message of an NOR directory is not sent to the storage site, but is instead piggybacked on the next open message to that storage site. A table of the last few NOR "removed from cache" messages are kept for each storage site to be sent with the next open message. If an overflow of this table occurs before the next open message is sent, a "removed from cache" message is dropped. Given the low rate of cache invalidation, the overhead of sending an invalidation request to a site that doesn't actually have the directory in its cache is minimal.

All opens for modification are sent to the site that stores the directory. When a request for modification is received for a directory which has a non-null NOR list, the storage site sends a message to each site in the list to invalidate the file descriptor and pages associated with that directory. After acknowledgement is received from all sites, the storage site opens the directory for modification. After the directory has been closed for modification, it can be reopened NOR by other using sites.

4.7 Summary

The primary goal of the distributed name service cache discussed in this chapter is the reduction in system response time to users' remote requests

by reducing the number of times a using site communicates with a remote site. Name service in the distributed system which was studied exhibited a high level of locality, with an exceptionally low level of conflict between modifications to directories on one site and their concurrent use elsewhere in the network. Since name management is a critical function to be supported by an internet operating system, a name service cache can have a substantial positive effect in overcoming the performance limitations of long haul links.

CHAPTER 5

A Protocol for an Internet Operating System

5.1 Introduction

A new approach to protocol design, semantics based protocol design, is proposed as a methodology to help extend transparency beyond local area networks. New, higher semantic level message primitives for the most frequent user commands are described and their impact on reducing network traffic is explored. The role of modifications to the lower layers in the protocol hierarchy to support the new message primitives and further reduce traffic is also discussed.

5.2 Methodology

The first step in semantics based protocol design is to investigate the behavior of the client. One must determine the characteristics of the applications to be supported by answering the following questions: How much data will be typically transmitted? Do frequent applications dominate client behavior? What are the response time requirements for the applications? How much demand will be placed on the network channel?

Next, message primitives to support the client's behavior must be defined. System calls provided by a single site operating system may or may not be appropriate as message primitives. During the execution of a system call, other system routines are often called internally by the kernel. If each

system routine corresponds to a message primitive, many messages may be transmitted for each system call that operates on remotely stored data. Instead, the semantic level of the message primitive can be raised so system routines are called from the storage site rather than the using site to reduce traffic.

Frequently, system calls act as a group to provide an operating system service. For instance, in Unix, to create a process running new code, the *fork* system call is executed to create a process running the same program as the caller, followed by an *exec* call which replaces the code and data of the running process with a new program and data image. In this case, a single message primitive could be defined which combines the semantics of the *fork* and *exec* system call to create a process which runs a new program on a remote site.

The choice of message primitives is one of the most important factors in the overall performance of a protocol. Lantz et. al. [Lan 84] report performance improvements of up to 3000% when the message primitives of their Network Graphics Protocol are defined so values are returned after an entire sequence of graphic operations instead of being returned after each operation. Similar performance gains have been measured in the Internet Locus testbed and are discussed in chapter 6.

Raising the semantic level of message primitives must be balanced by three constraints. First, it is often desirable to move data to the using site to take advantage of local buffer caching. However, if an operation is moved entirely to the storage site, care must be taken to return relevant data to the using site so caching can be used to reduce the need for network traffic during

subsequent operations. Next, it is best for the user process that waits for the completion of a system call to wait at the using site rather than the storage site to avoid the overhead of building and scheduling a user process for remote service. Third, the performance gain of building higher level message primitives must be balanced against the amount of new kernel code that must be written.

The final step in semantics based protocol design is to tune the lower protocol layers to efficiently support the message primitives. The characteristics of the underlying network should be exploited for efficient service. If a LAN based protocol is to be extended to an internet environment, existing patterns of message traffic can be measured to determine what message types dominate traffic and to provide information for deciding what optimizations are possible.

5.3 Client Behavior

Given the rather poor communication characteristics of the lower levels in the distributed system hierarchy it is reasonable to ask whether network transparency is feasible in an internet environment. Surprisingly, a close examination of the required characteristics of typical interactive computing reveals that there is (in principle) substantial opportunity.

The actual amount of data which must be intrinsically transferred to support the execution of interactive commands for a majority of cases is surprisingly small. A study at Stanford University of the Unix operating system found nearly 50% of all files to be 1K bytes or less in size, and about 75% of the files to be 4K bytes or less when weighted by frequency of use.

[Laz 84]

Measurements of user commands were gathered from the collection of Locus systems at UCLA to determine if certain applications dominate client behavior. Users added a statement to their *.logout* files to save the history of commands that were typed during an interactive session. A sample size of 10,092 command lines were collected.

The most frequently executed interactive commands are shown in figure 5.1. Note that the most frequent 10 commands represent the majority of what users actually do. The complete measurements from UCLA show that 10% of the commands out of an available repertoire of over 330 commands account for more than 80% of total command usage.* If these frequent commands can be supported effectively, than major strides toward achieving performance transparency over the Internet will have been taken. Although commands that use file arguments may require transporting data across an Internet link when the data is stored remotely, the statistics from the Stanford study suggest that the actual amount of data that must be moved is usually small. Thus, the intrinsically limited bandwidth may not be so insurmountable an impediment to providing normal interactive response time across an Internet link.

*A study at Bell Laboratories reports a similar result [Han 84].

Command	Function	Frequency of Occurrence
ls	list contents of directory	11.9%
vi	editor	8.7%
cd	change to new directory	8.5%
more	examine a file	7.5%
rm	remove a file	3.3%
dirs	list directory stack	2.6%
jobs	identify background jobs	2.6%
fg	bring a job to the foreground	2.2%
make	compile a C source program	2.2%
grep	search for a string	2.1%

Figure 5.1: The 10 Most Frequently Executed Unix Commands

Furthermore, the several tenths of a second delay across an Internet link is relatively small compared to the usual acceptable elapsed time of a few seconds for interactive commands. That time in principle permits transmission of the needed small amount of data without substantially increasing the user perceived delay, even over modest bandwidth connections.

5.4 Message Primitives for Internet Locus

The LAN-Locus protocols are used as a base to build a protocol for an internet operating system. Emphasis is placed on efficiently supporting those applications that dominate client behavior. The message primitives used by Internet Locus to support the most frequent user commands that access remote data are discussed in this section. The *cp* command (the 12th most frequent command) which is often used to copy data between sites is included in this section while the *dirs*, *jobs*, and *fg* commands are omitted because they rarely access remote data.

5.4.1 List a Directory

The most frequently used Locus command is *ls*, which lists the contents of a directory. About 65% of the *ls* commands that are executed include the *-l* option which provides information for each entry such as owner, size, and protection. Under Locus, when the *ls -l* command is issued and the target directory is located on a remote site, the directory page is brought to the using site after pathname expansion and a Stat Request is sent to the storage site for each directory entry. The message traffic for the *ls -l* command of a directory that contains 54 entries is shown in figure 5.2

Instead of sending a Status request message for each directory entry and processing the Response at the using site, message traffic can be reduced by moving most of the processing to the storage site. A new message primitive, *dirstat*, has been defined for Internet Locus which reads a directory page and returns the status for the number of directory entries that fit in a network message. Under Internet Locus, message traffic for the *ls -l* command for the directory shown in figure 5.2 is reduced from 63 request/response pairs to just 2 request/response pairs.

The *ls* command without options requires the target directory to be opened, read, and closed. When directory pages are cached at the using site as described in chapter 4, the *ls* command without options rarely causes any network traffic.

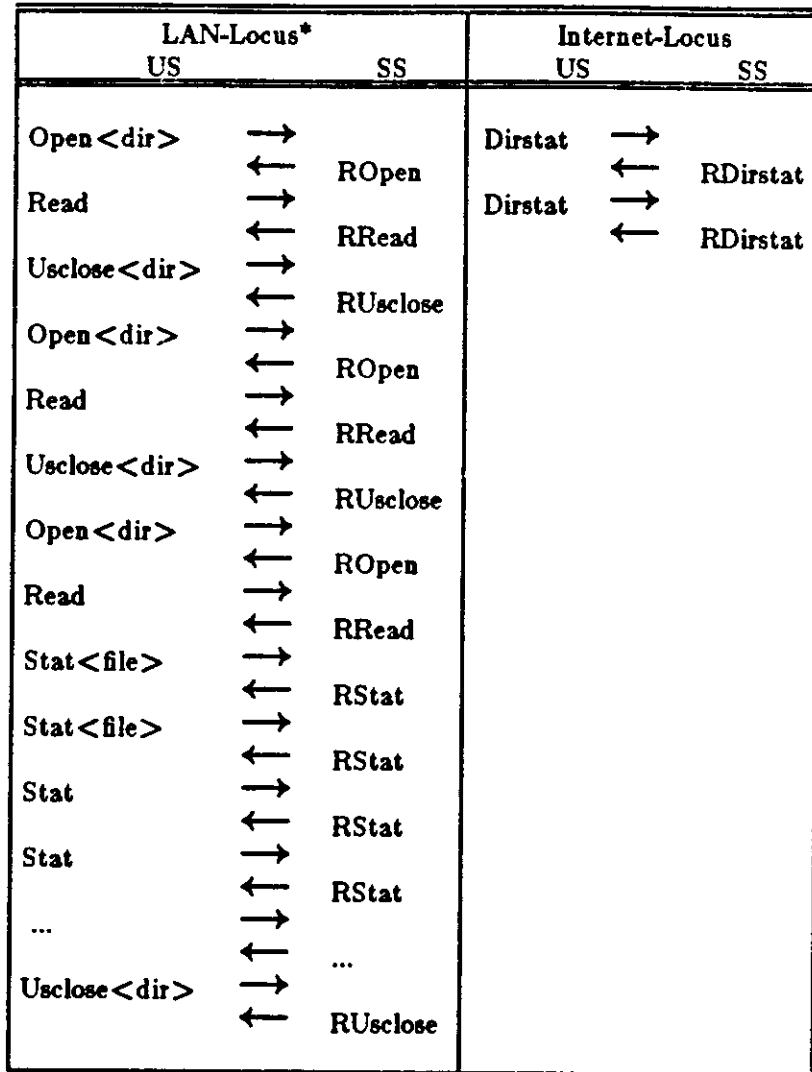


Figure 5.2: Message Traffic for the List Directory Command
ls -l

5.4.2 Examine a File

The *more* command is used to examine a text file, one screenful at a time, on a terminal. A comparison of the message traffic generated from the

*Responses are indicated as Rmessage-type, ie. the Open Response is shown as ROpen. The explicit ACK messages sent for each LAN-Locus message (except for Read and RRead) are not shown in figures 5.2 - 5.8. Only 4 of the 54 Status Request/Response message pairs that are transmitted are shown.

more command under Locus and Internet Locus is shown in figure 5.3. Under Locus, each pathname component in the string name of the target file is opened, read and closed as part of pathname to object translation. A system call to determine the status of the target file is issued, resulting in a Status request message and response. The target file is opened and a response to the Open request message is returned. Finally, a read system call is issued and the first page of the target file is returned to the user site from the storage site.

Under Internet Locus, when the desired directory pages are in the buffer cache at the using site, no messages are transmitted for pathname to object translation. In the *more* command, as in many other commands and programs, the Read system call follows the Open (for Read) system call. A new message primitive has been defined that combines the semantics of these two system calls. The new message means: "Open this file for read access and if the open succeeds, return the first page of the file". Since many files contain a single page of data, this message primitive is very effective in reducing delay.

The Stat and Open system calls are also often used together by commands and programs. The Stat call returns the attributes of a file. These attributes are checked against some criteria (e.g. is this file a directory?) and the Open call is issued depending on the outcome. Instead of transmitting separate Stat and Open requests for a remote file, the Stat and Open calls can be combined to form a single message primitive. A single StatOpen request is sent which means: "Perform a Stat on the file and if the attributes of the file match the desired criteria, open the file. In the Internet Locus version of the

more command, the effect of the StatOpen primitive is simulated by opening the file prior to the Stat system call so the Stat is executed at the using site.

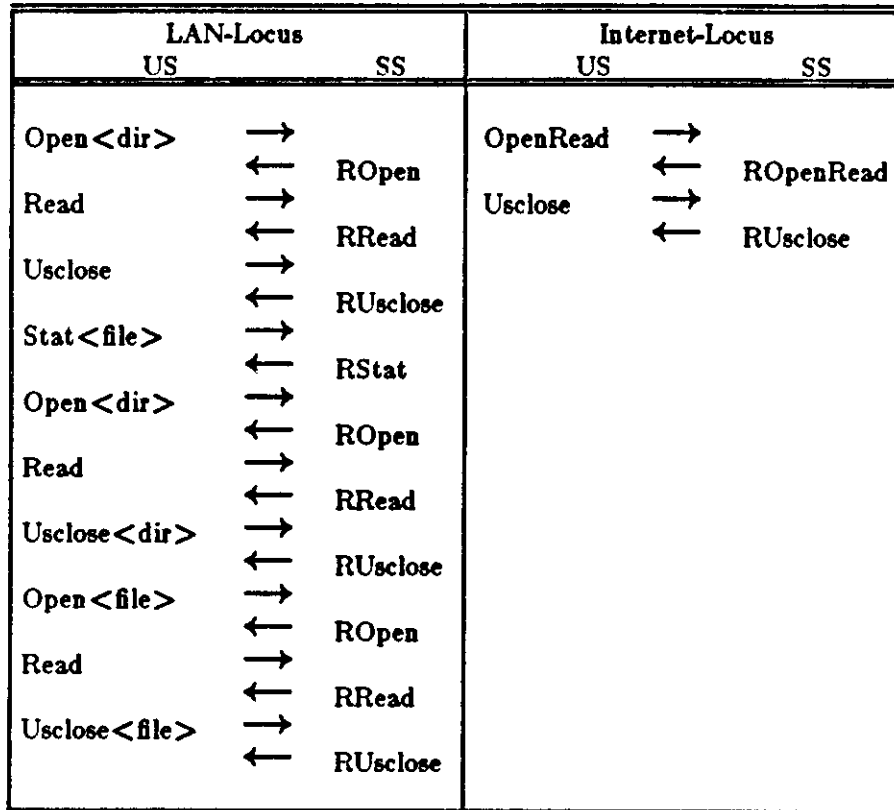


Figure 5.3: Message Traffic for the Examine File Command *more file1K*

5.4.3 Change Working Directory

The *cd* command changes a user's current working directory. In LAN-Locus, the new directory is opened and closed during the *cd* command. A reference to a remote file in the current working directory causes the directory to be reopened and the page(s) brought to the using site as part of path-name expansion. In Internet Locus, the current working directory is kept open. The page(s) of the current working directory in the using site cache are

rarely flushed so pathname expansion for a file within the current working directory occurs without network traffic. Likewise, the *ls* command of the current working directory does not generate any traffic.

The reduction in message traffic during the execution of the *cd* command under Internet Locus compared to LAN-Locus is not substantial. However, since the directory pages of the current working directory are kept at the using site under Internet Locus, the reduction in message traffic due to pathname expansion for commands that reference files in the current working directory is often dramatic.

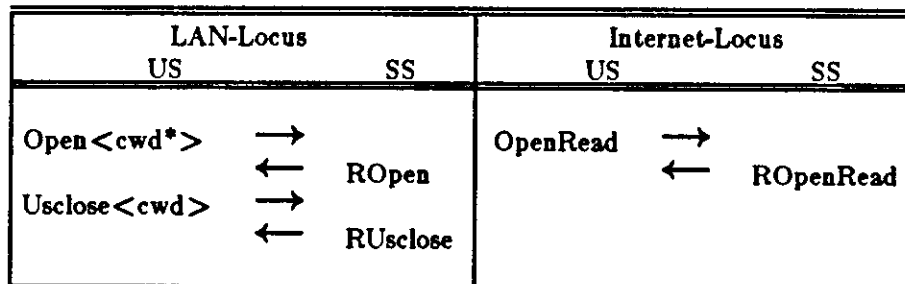


Figure 5.4: Message Traffic for the Change Working Directory Command
cd /th/abs

5.4.4 Edit a File

Vi is a display oriented text editor. When *Vi* is invoked, the file to be edited is sent from the storage site to the using site (see figure 5.5) and copied into a temporary area managed by the editor. Updates are performed on the temporary copy at the using site.

*cwd = current working directory

When the *Vi* write command is issued, the new version of the file is transmitted to the storage site. A shadow page mechanism is used at the storage site to insure that either all of the changes to the file are saved or none are saved. Each page of the updated version of the file is written into a new physical page on disk at the storage site. However, the disk inode contains pointers to the pages of the old version of the file while the incore copy of the inode contains pointers to the newly allocated pages. When the using site closes the file, a commit request is sent to the storage site. The storage site services the commit by moving the incore inode information to the disk inode thus permanently saving the new version of the file.

As suggested in chapter 3, a new message primitive can be defined to combine the semantics of a commit followed by a close which reduces the number of messages needed to save a new version of a file. In addition, the StatOpen message can be used to remove the Stat message exchange. As a result, a file can be saved on a remote site with just 2 message exchanges plus a write message per page, compared to 10 message exchanges plus a write per page under LAN-Locus.

LAN-Locus			Internet-Locus		
US		SS	US		SS
Open <dir>	→		OpenRead	→	
	←	ROpen		←	ROpenRead
Read	→		Usclose	→	
	←	RRead		←	RUsclose
Usclose	→				
	←	RUsclose			
Open <file>	→				
	←	ROpen			
Read <file>	→				
	←	RRead			
Open <dir>	→				
	←	ROpen			
Read	→				
	←	RRead			
Usclose <dir>	→				
	←	RUsclose			
Usclose <file>	→				
	←	RUsclose			
Open <dir>	→		StatOpen	→	
	←	ROpen		←	RStatOpen
Read	→		Write	→	
	←	RRead	USCommit	→	
Usclose <dir>	→			←	RUSCommit
	←	RUsclose			
Stat <file>	→				
	←	ROpen			
Open <dir>	→				
	←	ROpen			
Read	→				
	←	RRead			
Usclose <dir>	→				
	←	RUsclose			
Open <file>	→				
	←	ROpen			
Write <file>	→				
USCommit	→				
	←	RUSCommit			
Usclose <file>	→				
	←	RUsclose			

Figure 5.5: Message Traffic for the VI Command
vi /ic/file1K; write file

For a large file, the delay due to the write messages overshadows the delay due to the StatOpen and file Commit__close messages. Therefore, an efficient data stream protocol, as described in section 5.5.4, is necessary for editing large files across a slow internet link.

5.4.5 Remove a File

The network traffic for the LAN-Locus implementation of the remove file command, *rm* is shown in figure 5.6. A Stat message is first sent to the storage site to determine if the file exists or is a directory. Another Stat message is sent to determine if the user has permission to remove the file. The parent directory of the file and the target file are opened for modification and the directory page(s) are sent to the using site. The file is removed by setting the inode value of its directory entry to zero. The updated directory entry is sent in a 1024 byte Write message and the directory is committed and closed. The link count of the file is decremented, its version number is incremented, and the file is committed so all sites that store a copy of the file see the updated (deleted) version. When all storage sites have seen the delete, the inode can be reallocated.

Much of the processing for the remove operation can be moved to the storage site to greatly reduce the number of messages transmitted during the *rm* command. The two Stats and file Open can be combined into a single StatOpen message that causes the target file to be opened for modification if the file exists and is not a directory. The parent directory can be opened and read in a single message exchange. With variable length buffers (see section 5.5.2), only the single updated directory entry needs to be sent in the Write message. The operation of committing and closing the parent directory and

target file can also be combined into a single message exchange.

LAN-Locus*		Internet-Locus	
US	SS	US	SS
Stat<file>	→	StatOpen	→
	←		← RStatOpen
Stat<file>	→	OpenRead	→
	←		← ROpenRead
Open<dir>	→	Write	→
	←	USCommit	→
Read	→		← RUSCommit
	←		
Open<file>	→		
	←		
Write<dir>	→		
Commit<dir>	→		
	←		
Usclose<dir>	→		
	←		
Commit<file>	→		
	←		
Commit<file>	→		
	←		
Usclose<file>	→		
	←		

Figure 5.6: Message Traffic for the Remove Command
rm file1K

5.4.6 Copy a File to an Existing File

The network traffic generated under LAN-Locus to copy a 1K byte file from one site to an existing file on another site is shown in figure 5.7. Pathname expansion contributes significantly to the total network traffic generated by the simple copy command running under LAN-Locus as each directory component is opened, read, and closed across the network.

*Messages due to pathname expansion are not shown

By caching directory pages, combining the separate Close and Commit messages, and generating status requests after the file is opened, the number of messages in addition to the Write message drops from 14 request/response pairs in LAN Locus to 2 request/response pairs in Internet Locus. It is especially important in the Unix context to be able to copy small files efficiently because, as mentioned earlier, small files dominate normal file usage.

LAN-Locus			Internet-Locus		
US		SS	US		SS
Open <dir>	→		Open	→	
	←	ROpen		←	ROpen
Read	→		Write	→	
	←	RRead	Uscommit	→	
Usclose	→			←	RUscommit
	←	RUsclose			
Stat <file>	→				
	←	RStat			
Open <dir>	→				
	←	ROpen			
Read	→				
	←	RRead			
Usclose	→				
	←	RUsclose			
Stat <file>	→				
	←	RStat			
Open <dir>	→				
	←	ROpen			
Read	→				
	←	RRead			
Usclose <dir>	→				
	←	RUsclose			
Open <file>	→				
	←	ROpen			
Write <file>	→				
Uscommit <file>	→				
	←	RUscommit			
Usclose <file>	→				
	←	RUsclose			

Figure 5.7: Message Traffic for the Copy File Command
cp /th/file1K file1K

5.4.7 Copy a File to a New File

When the *cp* command is executed and the destination file does not exist, a new file must be created at the storage site. Under LAN-Locus 8 requests and responses plus a Write message are used to create a file. The parent directory of the new file is opened and read and then opened again for modification. An open message is sent to the storage site with the inode set to zero to request the creation of a new file. An inode is allocated at the storage site and returned to the using site in the response to open. A directory entry for the new file is sent to the storage site in a 1024 byte message. The directory is committed and closed to make the newly created file permanent. Finally, pages from the source file are sent to the storage site and copied into the new file.

There are many opportunities to reduce the number of messages to create a new file. The parent directory can be opened initially for modification and the directory page(s) returned to the using site in the response to open to eliminate an open request and response and a read request and response. A new message primitive, Create can be defined to request the creation of a new file. The semantics of Create are: "Open a new file and if the open succeeds, write, commit, and close the parent directory and commit the inode for the new file." With these optimizations, the number of messages to copy a 1K byte file into a new file is reduced from 10 request/response pairs plus 2 writes to 3 request/response pairs and 1 write message.

LAN-Locus*		Internet-Locus	
US	SS	US	SS
Open <dir>	→	OpenRead	→
	←		←
			ROpenRead
Read	→	Create	→
	←		←
			RCreate
Open <dir>	→	Write	→
	←		←
Read	→	Uscommit	→
	←		←
			Ruscommit
Open <new file>	→		
	←		
Write <dir>	→		
Uscommit <dir>	→		
	←		
Usclose <dir>	→		
	←		
Uscommit <new file>	→		
	←		
Write <new file>	→		
Uscommit <new file>	→		
	←		
Usclose <new file>	→		
	←		

Figure 5.8: Message Traffic for the Copy File Command
cp /th/file1K newfile1K

5.4.8 Remote Execution: Make and Grep

The *make* command is used to execute a program if the target files for the program have been modified or do not exist. *Make* is most often used to compile a related group of source files. *Grep* is used to search a set of files for lines that match a specified pattern. Both commands often access large amounts of data.

In a network transparent distributed system, one has the choice of moving the data to the execution site or moving the processing to the data

*Messages due to pathname expansion are not shown

storage site. This freedom has tremendous potential for reducing delay due to data movement across a long haul link.

Since *make* and *grep* are commonly executed commands, their load modules will often exist at the site that stores the data to be accessed. Therefore, if the site of execution for these commands moves from the initial using site to the storage site, only a new process image has to be created at the storage site, without the transfer of the load module for the command. However, enough data from the user's environment must also be transferred so the execution of the process at the storage site is semantically equivalent to execution of the process at the site the command was initially issued from. Items such as current open file descriptors, IPC identifiers, and search paths must be correctly interpreted by the relocated process.

Total Size of Source File(s)	Data to Process	Process to Data
< 1K bytes	80	42
10K bytes	107	42
22K bytes	116	42
44K bytes	163	42

Figure 5.9: Total # of Messages Transferred for *make -f Makefile >& make.log*

In the example shown in figure 5.9, the makefile, source files and log file are all stored on a remote storage site. By moving the site of compilation to the storage site, instead of moving the source files to the site that initially issues the *make* command, the number of messages is significantly reduced.

In Locus, a user can select the execution site for subsequent commands by issuing the *setopts* system call. However, for Internet Locus, the policy for moving a process to the storage site to improve performance may depend on the amount of data to be accessed and the distance between the using site and storage site. A mechanism whereby the system dynamically determines the appropriate execution site for a command is a subject of ongoing research.

5.5 Protocol Support for Message Primitives

By raising the semantic level of message primitives, the number of network messages transmitted to service frequent user commands can be greatly reduced. By moving the process to the data, the number of messages required for certain data intensive commands can also be significantly reduced. However, the communication cost of supporting network transparency across long haul links may still be too great unless the lower protocol layers are tuned sufficiently for the higher delay, lower bandwidth environment.

Measurements of network traffic for an existing LAN-Locus network were collected to determine the relative frequency of different message types. The results were used to identify potential optimizations to improve performance when Locus is extended to the internet environment. A discussion of the network measurements is given in Appendix B. The most frequently transmitted message types are shown in figure 5.10.

5.5.1 Piggyback Acknowledgements

In LAN-Locus, the Acknowledge (ACK) message contributes more to overall network traffic than any other message type. When any Locus message is received (except Read and Response to Read), the receiver transmits a

148 byte Acknowledge message to the sender. The explicit ACK message uses network bandwidth, contributes to delay, and wastes valuable processor resources as the receiving CPU must field the network interrupt and copy the ACK into a buffer and the sending CPU must format and transmit the ACK.

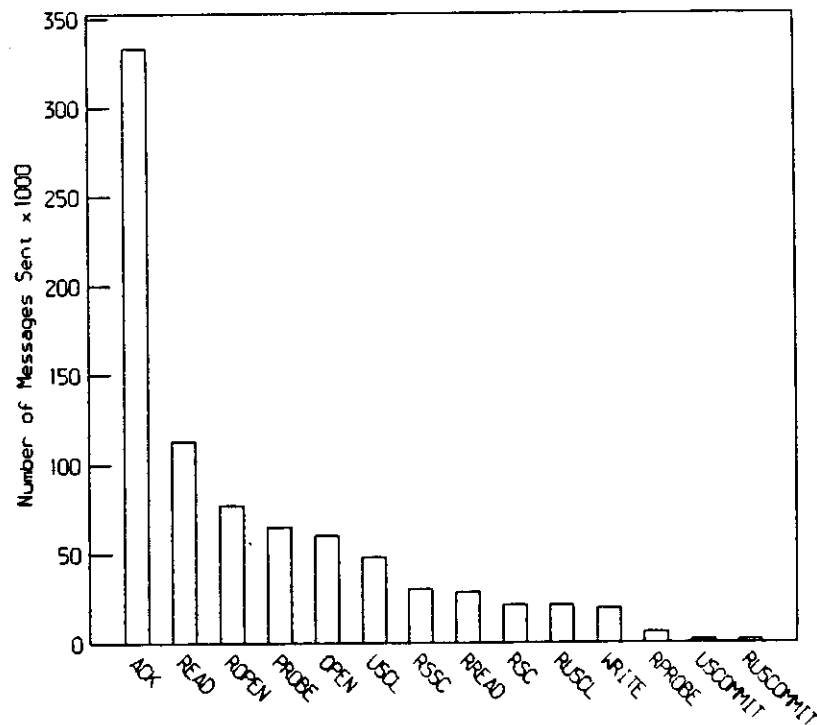


Figure 5.10: Distribution of Message Types

Two optimizations are implemented in Internet Locus. First, the ACK message is reduced to just 12 bytes so only the essential fields are transmitted. This decreases transmission delay over slow communication links. Second, ACKs are piggybacked in the header of outgoing messages. If there is no outgoing traffic within a timeout period, an explicit ACK is sent. Since most Locus traffic is characterized by request-response sequences, a response

message with a piggybacked ACK in its header acts as an acknowledgement for the request.

Response messages by themselves are not sufficient to provide acknowledgments in all cases. Occasionally, a request cannot be serviced quickly at the storage site so the response is delayed. The storage site must return a separate ACK message without waiting for the request to complete so network resources such as sequence numbers and timeout buffers can be released by the using site.

5.5.2 Variable Length Messages

A simple but important protocol modification concerns message sizes. The transport layer of LAN Locus supports two fixed message sizes. Requests are sent as small control messages and Responses consist of either a control message or a data buffer plus a control message. Supporting only two fixed message lengths simplifies the implementation of the transport layer and is adequate in the low delay, LAN environment. However, the Arpanet handles single packet messages of less than 1008 bits much more efficiently than multipacket messages. Instead of supporting just two fixed message lengths, Internet Locus supports variable length control messages and variable length buffers to reduce delay across long haul links. Only the specific fields necessary for each control message are transmitted.

5.5.3 Data Compression

Another strategy to reduce message traffic is data compression. Data compression techniques have been studied for many years but their application in computer networks has been limited. The encoding scheme developed

by Huffman [Gal 78], based on the idea of assigning short codes for frequent symbols and longer codes for rare symbols, will typically compress a text file by about 38% and a binary file by 19% [Uni 81]. Unfortunately, the savings achieved by transmitting a smaller message are usually offset by the expense of calculating the relative frequencies of the message symbols, encoding the message, and prepending the decoding tree to the message.

There is the potential, however, for a fixed encoding data compression algorithm to be useful in reducing message traffic for a distributed operating system, especially when a 1024 byte packet (the typical Vax UNIX page size) can be reduced to less than 1008 bytes so it can be transmitted in a single Arpanet message. In the fixed encoding scheme, symbol probabilities in transmitted messages are collected over a period of time. A fixed Huffman encoding is calculated from the symbol probabilities and the code is distributed to all sites in the network. All sites use the same fixed code so no decoding tree is prepended to messages. The processing cost of encoding messages using a fixed code can be made quite small, especially since each message is checksummed anyway.

Extensions to this data compression scheme include combining the checksum and encoding routines; recalculating the symbol probabilities after a period of time so a slowly adaptive encoding is used; and using different fixed codes for different file types such as text files, binary files, and source code. A detailed evaluation of the costs and benefits of data compression for Internet Locus remains for future work.

5.5.4 Data Stream Protocol

The LAN-Locus protocol is not well suited for efficient transfer of large amounts of data across long haul network links. Only a single message can be in transit at a time to a given site. Under LAN-Locus, the network channel is blocked until the acknowledgement for the previous message has been received. This *stop and wait* protocol performs satisfactorily in the low delay, high bandwidth LAN environment but must be replaced by a data stream protocol if sufficient throughput across long haul links is to be achieved.

In Internet Locus, a data stream protocol based on a sliding window of outstanding messages has been implemented. Several messages, up to the current window size, can be in transit to a given site at a time. Acknowledgments are cumulative and signify that all messages up to and including the message with the acknowledged sequence number were correctly received.

A *go-back-N* policy is implemented to handle communication errors. If a site receives an out of sequence or damaged message it discards all subsequent messages from the sender until a correct copy of the expected message is received. When the sender fails to receive an ACK for an outstanding message within a timeout period, the lost message plus all subsequent unacknowledged messages are retransmitted. A more complex *selective reject* policy, which forces the receiver to handle out of sequence messages, is not necessary due to the relatively low error rate for the internet topologies under investigation in this dissertation.

Although data stream parameters such as window size and acknowledgement timeouts are setup manually in the current testbed environ-

ment, in a real Internet Locus system these parameters could be established dynamically as part of the topology change algorithm. The elapsed time to receive the response to Probe message during topology change could be used to configure the window size and timeouts for different sites.

The data stream protocol allows files to be written efficiently over long haul links. In LAN-Locus, when the using site issues a Write message, the site must wait for an Acknowledgement message from the receiver before the next Write message can be transmitted. In Internet-Locus, the using site can send Write messages continuously, as long as the number of outstanding messages is less than the maximum window size.

The data stream protocol, however is not sufficient to achieve good performance when reading large files across slow links. The LAN-Locus Read protocol must also be adapted to the higher delay environment.

Under LAN-Locus, sequential read access to pages of a file stored at a remote site works as follows. The using site first issues a Read Request message. Upon receiving the Read Request, the storage site retrieves the desired page from disk (unless it is in the buffer cache) and initiates a *one page readahead* to retrieve the next sequential page from disk. The first page is returned to the using site who issues the next Read Request. If the communication link between sites is a LAN, this request message often reaches the storage site just after the readahead has completed so the storage site can fulfill the request without expensive disk access. Another readahead is initiated so pages are retrieved at a rate close to the maximum available disk rate.

In the long haul environment, the network penalty is far greater than disk access. Therefore readahead should occur across the network, not just from the disk to main memory at the storage site. A *network readahead* scheme works as follows. The using site issues a normal Read Request with a flag set to indicate sequential read access and sets up enough input buffers to receive as many pages as the window size allows. When the storage site detects sequential read access, it retrieves pages sequentially from disk, sending each page in a Read Response message, subject to window flow control. The data stream protocol insures reliable transmission of Read Responses.

When the using site issues a read system call during sequential read access, if the Read Response has already reached the using site, the read is completed immediately without any additional network traffic. If the desired page is not in an input buffer, the read call blocks until the page arrives. *Network readahead* effectively eliminates intermediate Read Request messages and provides data transfer at the maximum throughput allowed by the window size.

5.5.5 Summary

Given the poor communication characteristics of long haul links, the extension of transparency to the internet environment appears to be a formidable task. Fortunately, a close examination of typical interactive computing reveals several favorable characteristics:

- Files are typically small
- A small set of commands dominate total command usage

- An acceptable elapsed time of a few seconds for command completion is much greater than the time to transmit several messages across several packet switching nodes in a long haul network

In this chapter we have demonstrated that by raising the semantic level of message primitives, the number of network messages transmitted to service frequent user commands is greatly reduced. By moving the process to the data, the number of messages required for certain data intensive commands is also significantly reduced. Network overhead is further decreased if acknowledgments are piggybacked on outgoing messages, variable length headers and buffers are transmitted, and a protocol that supports multiple outstanding messages is used.

CHAPTER 6

The Performance of an Internet Operating System

6.1 Introduction

Solutions to problems in many areas of computer science succeed or fail depending on their performance. In this chapter, the performance of Internet Locus is discussed. Performance goals are stated, the Internet Locus testbed is described, and measurement results are presented. The contributions of the strategies discussed in chapters 4 and 5 are evaluated and their importance for systems other than Locus is suggested.

6.2 Performance Evaluation of Distributed Operating Systems

Previous attempts at building network transparent systems across long haul links have been unsuccessful largely due to poor performance. If the protocols described in chapter 4 and 5 are to be presented as a solution to internet transparency, their performance in a real internet environment must be satisfactory.

We define the performance goals for an internetwork operating system as follows:

1. The elapsed time for the completion of interactive commands when data is stored across a long haul link or local area network should be nearly equal to the elapsed time when data is stored at the using site

for small amounts of data.

2. The elapsed time to transfer large amounts of data across a long haul link or local area network using native operating system commands should be less than or equal to the elapsed time when using a specialized file transfer protocol.

The first goal states that the response time for interactive commands in a distributed system should be generally independent of resource location. If this goal is not met, there is little justification for considering a collection of sites connected by internet links as a single, integrated system. Of course, for large amounts of data, the transfer time across a typical long haul link will not match the transfer time from a local disk due to the limited bandwidth of the link. The second goal states that the user should be able to invoke the normal, single site data transfer commands to transfer data across any level in the distributed system hierarchy and achieve performance that is at least as good as the performance of specialized file transfer protocols. If this goal is not reached, it will be difficult to argue that native operating system commands should be used instead of non-transparent file transfer commands.

The performance of a distributed system is often evaluated at a much finer grain than the elapsed time of interactive commands. A common performance measure is the time to read or write a single page of data from a remote site [Gol 82]. Although this performance measure has merit, its use as a means of evaluating performance as perceived by users is inappropriate. Under LAN-LOCUS, for example, the elapsed time to read a single page of data across an Ethernet is approximately 15 ms. which is close to the average time to read a page of data from disk. When a user examines a one page file,

however, 18 or more messages may be sent between sites in addition to the read request and response for the file. These additional messages contribute significantly to latency and may in fact dwarf the contribution from the single read request and response. We therefore have selected the elapsed time of common commands as the measure for performance evaluation of Internet Locus.

6.3 The Internet Locus Testbed

A testbed was built at UCLA to explore the feasibility of network transparency in an environment of LAN "subnets" connected by long haul networks. The testbed was designed so the characteristics of the link between the subnets could be easily adjusted. Two Locus subnets, each internally connected by an Ethernet, communicate across an Internet link. Messages from one Locus subnet are reflected off an echo host and sent to the other Locus subnet. The echo host is easily changed to alter the "distance" between the two Locus subnets.

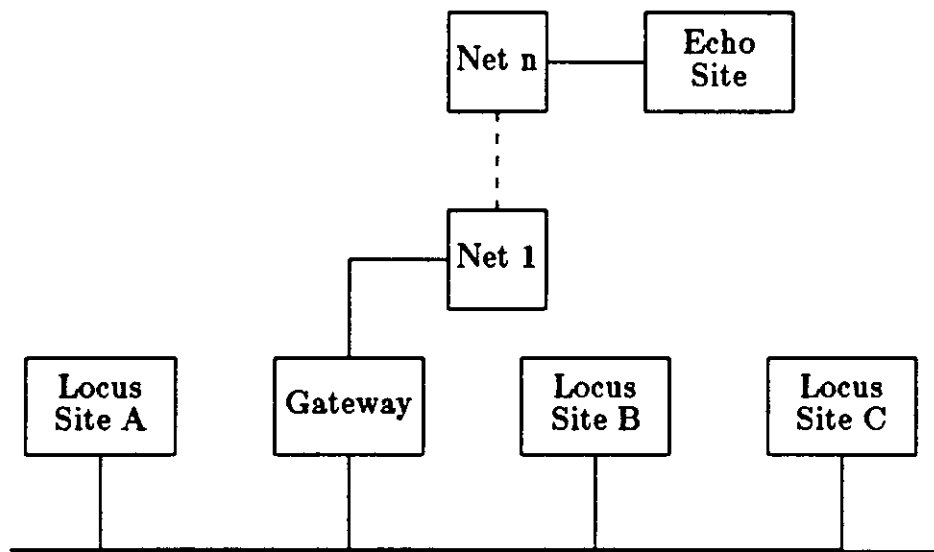


Figure 6.1: Internet Locus Testbed

The Internet Locus testbed is shown in figure 6.1. The Locus sites and Gateway site are VAX 11/750s attached by Interlan interfaces to a 10Mbps Ethernet cable. The Gateway site has an additional 1822 interface to an Arpanet IMP [Bol 78]. Each of the Locus sites runs the same Internet Locus software. The Gateway site runs specialized software to route packets through the testbed. The format of an Internet Locus packet is shown in figure 6.2. A packet that includes a full buffer is 1212 bytes in length (not including local net header) which is larger than the maximum message size supported by the Arpanet. Large packets must be fragmented at the Gateway for transmission over the Arpanet and reassembled for transmission over the Ethernet.

Packets travel through the Internet Locus testbed as follows. A packet, originating from a Locus site, is first encapsulated in an ICMP [Dar 81] echo packet. The packet is sent from the Locus originating site over the Ethernet cable to the Internet Locus gateway site. The packet is forwarded by the gateway site to the Internet Protocol (IP) [Dar 81] destination site who swaps the IP source and destination addresses and returns the packet to the gateway site. The gateway then forwards the packet to the Locus destination site. The IP destination address is configured by the user, so any Internet site that supports IP and ICMP is a possible candidate to serve as an Internet Locus echo site. By this strategy, Locus is routinely run on sites that communicate via local area networks, long haul networks, and satellite networks merely by changing the identity of the echo host.

The Internet Locus testbed introduces two sources of overhead that would not be present in a real Internet Locus system. First, the eight byte

ICMP header, which is used solely to bounce packets off echo hosts, would be omitted in a real system. Second, the echo hosts run standard TCP/IP software that invokes a user process to handle each Internet Locus packet. The scheduling and running of a user process at the echo site for each packet increases the communication cost of packet transfer in the testbed. In contrast, the Locus sites in the testbed handle the IP and ICMP packet headers at interrupt level so the user process overhead introduced at the echo site would not be present in a real Internet Locus system. We therefore expect that the performance that can be achieved in a real Internet Locus system will be slightly better than the performance measured in the Internet Locus testbed.

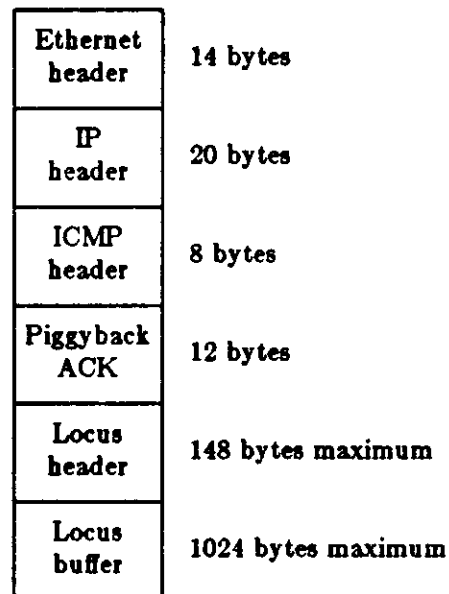


Figure 6.2: Internet Locus Packet Format

6.4 Performance of Interactive Commands in an Internet Environment

One of the important goals of Internet Locus is to provide interactive users with response similar to that perceived in a LAN or single site environment. To determine how well Internet Locus meets this goal, frequently executed Unix commands were run from a site (the using site or US) at UCLA with the target data stored at a remote site (the storage site or SS) across an Internet link. By adjusting the echo site in the Internet Locus testbed, tests were run for storage sites effectively located at UCLA, California Institute of Technology (CIT), Rand, and the University of Delaware (U-Del).

Site	# of IMP hops	Locus Header	Locus Header plus buffer
CIT	1	34	198
RAND	3	102	594
U-DEL	5	170	990

Figure 6.3: Estimated Delay (in milliseconds) to Transmit Messages from UCLA to Selected Internet Sites

The estimated delay to send an Internet Locus message from a site at UCLA to each of the Internet destination sites is shown in figure 6.3. Assuming negligible delay due to Ethernet propagation and gateway processing, the delay D , is given by:

$$D = (I + (M/C)) * H$$

where I = IMP processing delay
 C = Channel bandwidth
 M = # of bits in message
 H = # of IMP hops from source to destination

For the Arpanet, with 50 Kbps links, and 2ms. IMP processing delay [Hav 82],

$$D = (2\text{ms.} + (M/50\text{Kbps})) * H$$

where $M / 50\text{Kbps} = 32\text{ms.}$ for a Locus header (148 + 54 bytes)
 $M / 50\text{Kbps} = 196\text{ms.}$ for a Locus header plus buffer

The delay calculations show that the elapsed time to transmit several request and response messages between UCLA and any of the selected Internet destination sites is within the normal response time expected for interactive commands.

In figures 6.4 through 6.7, two columns of measured performance data are given. The first column shows the results of running LAN Locus across the Internet by encapsulating LAN Locus protocol messages in IP headers. The second column contains the results of repeating the tests using the new Internet Locus protocols. The times reported have been adjusted to reflect a one way trip to the echo site, thus simulating the case where one Locus subnet is located at UCLA and the other is located at the echo site.

6.4.1 List a Directory

The most frequently used Unix command is *ls*, which lists the contents of a directory. When the *-l* option is used, additional information for each entry is listed, such as owner, size, and protection.

Using Site	Storage Site	LAN-Locus (sec.)	Internet-Locus (sec.)
UCLA-A	UCLA-B	3.0	2.0
UCLA	CIT	15.0	2.0
UCLA	RAND	27.0	2.0
UCLA	U-DEL	43.0	3.0

Figure 6.4: Elapsed Time for the List Directory Command
ls -l

6.4.2 Examine a File

The *more* command is used to examine a text file, one screenful at a time, on a terminal.

Using Site	Storage Site	LAN-Locus (sec.)	Internet-Locus (sec.)
UCLA-A	UCLA-B	< 1.0	< 1.0
UCLA	CIT	3.0	1.0
UCLA	RAND	5.0	1.0
UCLA	U-DEL	9.0	2.0

Figure 6.5: Elapsed Time for the Examine File Command
`more file1K`

6.4.3 Edit a File

The elapsed time to edit a 1K byte file is shown below. The time includes reading the file from the storage site and writing it back.

Using Site	Storage Site	LAN-Locus (sec.)	Internet-Locus (sec.)
UCLA-A	UCLA-B	2.0	2.0
UCLA	CIT	6.0	2.0
UCLA	RAND	10.0	3.0
UCLA	U-DEL	20.0	5.0

Figure 6.6: Elapsed Time for the Edit File Command
`vi file1K; write file1K`

6.4.4 Copy a File

The elapsed time to copy a 1K and 2K byte file from one site to another across an Internet link is shown below.

cp /th/file1K file1K			
Using Site	Storage Site	LAN-Locus (sec.)	Internet-Locus (sec.)
UCLA-A	UCLA-B	< 1.0	< 1.0
UCLA	CIT	4.0	1.0
UCLA	RAND	7.0	2.0
UCLA	U-DEL	18.0	3.0

cp /th/file2K file2K			
Using Site	Storage Site	LAN-Locus (sec.)	Internet-Locus (sec.)
UCLA-A	UCLA-B	< 1.0	< 1.0
UCLA	CIT	4.5	1.0
UCLA	RAND	7.0	2.0
UCLA	U-DEL	19.0	3.5

Figure 6.7: Elapsed time for the Copy File Command

6.4.5 Discussion

These measurements indicate that, in the absence of network congestion, typical interactive commands that access small amounts of data can be serviced in an Internet environment in a manner that users are likely to find entirely satisfactory. The first performance goal of Internet Locus is largely met, even for topologies where the using site and storage site are separated by several switching nodes and located 3000 miles apart.

It is also clear that a number of users can be interactively served in a concurrent manner across a single Internet link of the kind considered here. This is because the time required to transmit the needed data for typical interactive commands, even across the Internet, is relatively small compared to the expected response time of the command. For example, *ls -l* may

require approximately three seconds of elapsed time for a 1K byte directory. The Internet is occupied for little more than a quarter of a second for the request and resulting data transfer. Hence even though queuing effects may cause some interaction among pending Internet requests, little user perceived delay is likely.

6.5 Performance of Commands Requiring Large Data Transfers

In this section we examine the performance of transferring large amounts of data using the traditional layered protocol approach, represented by FTP, compared to a kernel to kernel implementation of a distributed operating system, represented by the *dd* command* running under Internet Locus. The performance of moving a process to the data, which is not available using FTP, is also reported.

There are two ways to evaluate elapsed times measured in the Internet Locus testbed. The elapsed times can be adjusted to simulate a system where the storage site is located at the echo site or the echo site can be considered as part of the system and the elapsed times used without adjustment. In section 6.4 the elapsed times for command completion are divided in half to simulate the case where the storage site is located at the echo site. However, if a data stream protocol is used and the maximum number of outstanding messages is greater than 1, it is no longer correct to divide the elapsed time in half. The sender does not wait for each message to travel from the receiver to the echo site back to the sender before transmitting the next message and instead transmits messages continuously as long as the number of outstanding

*The *dd* command is often used to transfer large amounts of data in the Unix environment.

messages is less than the maximum window size.

It is therefore difficult to simulate a system where the storage site is located at the echo site when the window size is greater than 1 using elapsed time measurements from the Internet Locus testbed. As a result, the measurements of the data stream protocol are analyzed as if the echo site was part of the system and Internet Locus throughput is calculated using the total elapsed time for command completion, without any adjustment for the time that packets travel to and from the echo site.

To accurately compare FTP throughput with Internet Locus throughput, an FTP implementation for the testbed was developed in which each FTP packet is sent via the echo site just as each Internet Locus packet is. An ICMP header, added to each FTP packet, requests the echo site to swap the source and destination IP addresses and return the packet to the Internet Locus gateway which forwards the packet to the destination.

6.5.1 FTP and dd Performance Across a LAN

FTP and *dd* performance were measured in the production LAN-Locus environment at UCLA. The unmodified version of FTP was used since there was no echo site between the source and destination.

The average throughput for transferring large files across a 10Mbps Ethernet using the *dd* command was approximately 300Kbps. The average throughput using FTP was approximately 23Kbps. The extremely low throughput of FTP is mostly due to the overhead of the TCP/IP implementation which is used by FTP to transfer data. As described in Appendix A, the TCP/IP module incurs a large overhead from process context switching and

data copying for each message transferred.

6.5.2 FTP and dd Performance Across the Internet

The throughput of the testbed version of FTP versus the throughput of the *dd* command under Internet Locus for large files is shown in figure 6.8. In each case, a file is transferred from an initial site at UCLA to a storage site at UCLA through an echo site located at CIT, RAND, or U-DEL. The throughput shown is the actual user data throughput so the number of bits used in the calculation does not include the local network, IP, TCP, or Locus headers.

For all file sizes and echo sites, Internet Locus outperforms FTP. This performance difference is due mainly to the kernel implementation of Internet Locus versus the user process implementation of FTP. The throughput of FTP improves when the delay and occasional loss of packets at the echo site are eliminated by sending packets directly to the destination rather than via the echo site. For instance, the average throughput of FTP to RAND is approximately 23 Kbps compared to approximately 9 Kbps as measured in the testbed when RAND is used as an echo site. Similar performance improvements for the *dd* command are expected for a real Internet Locus system.

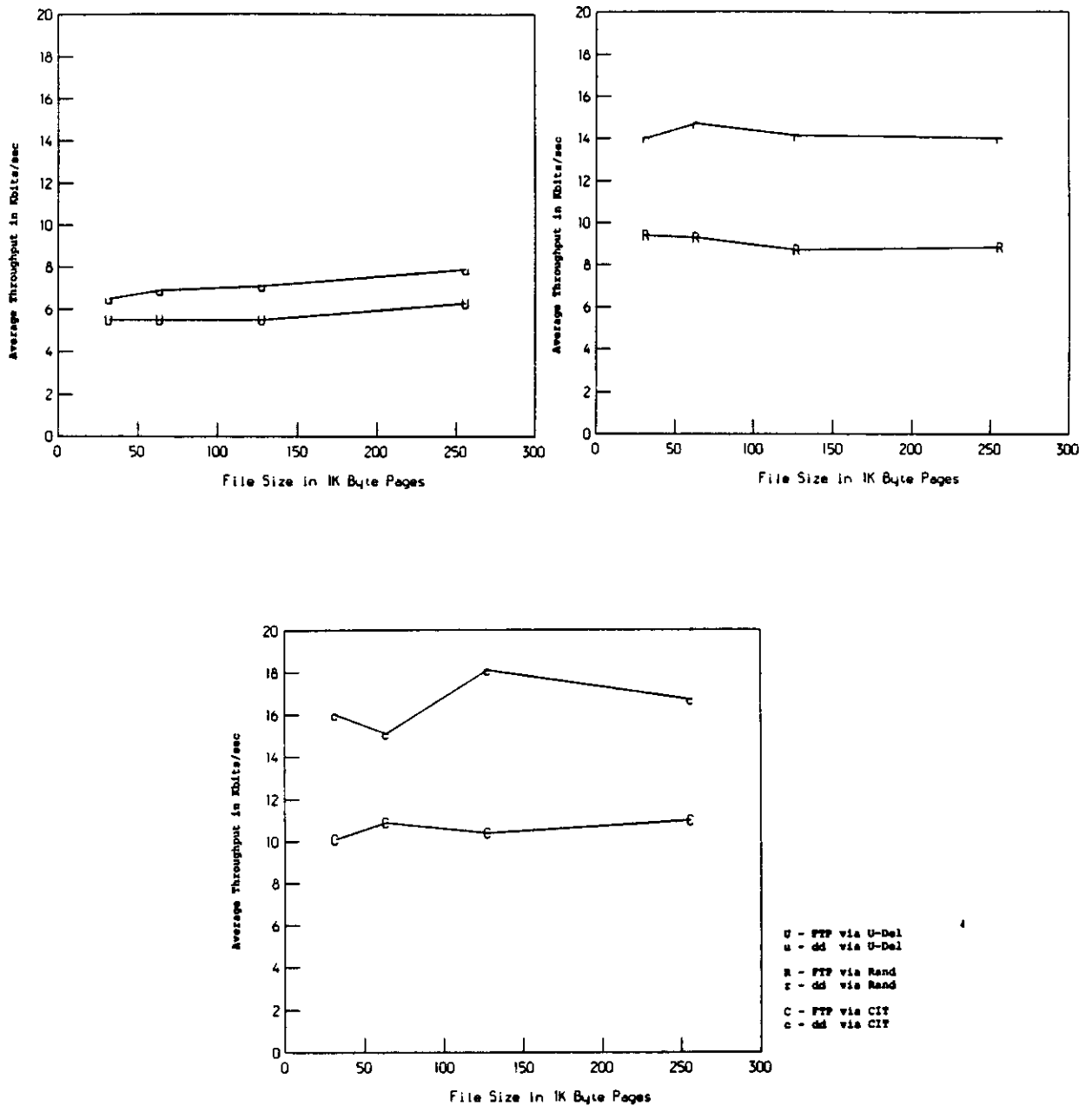


Figure 6.8: Throughput of FTP versus the *dd* Command

6.5.3 The Effect of Window Size on Throughput

The effect of window size on the average throughput of the *dd* command is shown in figure 6.9 for an echo site at RAND. Without the data stream protocol (window size = 1), the throughput of FTP exceeds the throughput of the *dd* command. However, for a window size of 2 or greater, the *dd* command outperforms FTP. With a window size of 6, the average

throughput of *dd* is approximately 14 Kbps compared to only 9 Kbps for FTP.

The Internet Locus gateway cannot handle packet bursts when the number of outstanding messages is greater than 7 or 8 due to the limited supply of buffers in the gateway. By inserting a small delay between consecutive packets, the window size can be increased but the additional delay negates any throughput improvement due to the larger window size.

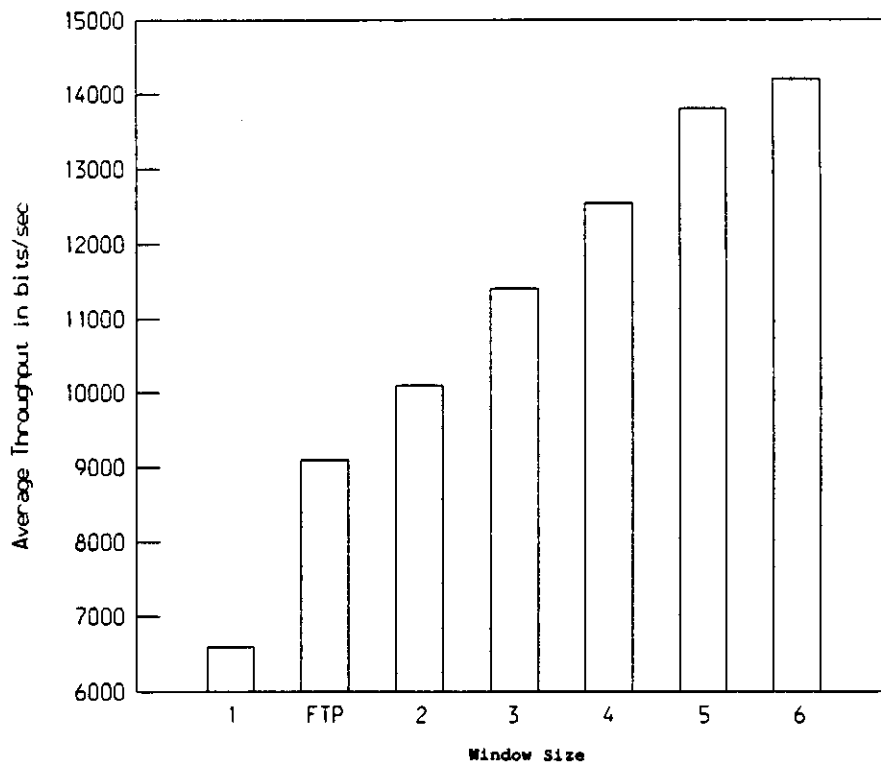


Figure 6.9: Effect of Window Size on Throughput

6.5.4 Discussion

The performance measurements from the Internet Locus testbed indicate that the throughput for data transfer across long haul links using native operating system commands meets the performance goal of being equal to or

exceeding the throughput of data transfer using FTP. Since sites in an internet operating system can store state information about the topology of the system, accurate window sizes and timeouts are easier to determine than in a system of autonomous hosts where data stream parameters must be fixed for all topologies or be determined every time a connection is established.

The general flow control problem for a system of high speed LANs connected by slower speed long haul links is difficult to solve. In end-to-end window schemes, the flow of messages is regulated by the round-trip delay from source to destination but the flow of messages across the high speed link from the sender to the gateway is uncontrolled. Since a large window size may cause packet buffers in the gateway to overflow, the window size in an end-to-end scheme is usually too small to achieve high throughput over a high delay, internet path. An approach to achieving high throughput without using a window of outstanding messages is described by Clark [Cla 85]. In this *rate control* scheme, the sender transmits a number of packets back-to-back (a "burst") but pauses between bursts. If the maximum burst size and the minimum inter-burst period are determined properly, high throughput may be achieved without overflowing buffers in gateways or intermediate packet switches. The Internet Locus testbed is an excellent facility for studying this *rate control* scheme and other flow control algorithms for an internet environment.

6.5.5 Compile a Program

The elapsed time to compile (*make*) a 10K byte source file is shown in figure 6.10. The *make* command was issued from a using site (US) at UCLA. The source file, makefile, and log file were located at a remote storage site.

When the compilation was executed at the US, the files were brought across the internet link. When the compilation was executed at the storage site (SS), only a new process image had to be created at the SS because the compiler load module itself was available at the SS. The times reported in the *Compile at US* column are due to the processing time for the compilation plus the time to transfer the data between the SS and US. The times reported in the *Compile at SS* column are due to the processing time for the compilation plus the time to move the site of execution from the US to the SS.

By moving the site of compilation to the storage site, instead of moving the data to the site that initially issued the make command, elapsed time for the compilation is significantly reduced. In fact, the elapsed time for moving the process to the data, even when the internet path includes a satellite hop, indicates that it is reasonable to compile a program from a site at UCLA with source files stored in Hawaii or Korea.

Using Site	Storage Site	Compile at US (sec.)	Compile at SS (sec.)
UCLA-A	UCLA-B	59	56
UCLA	CIT	144	57
UCLA	RAND	226	58
UCLA	U-DEL	307	62
UCLA	HAWAII	1046	105
UCLA	KOREA	1538	126

Figure 6.10: Elapsed Time for the Compile Command
`make -f Makefile >& make.log`

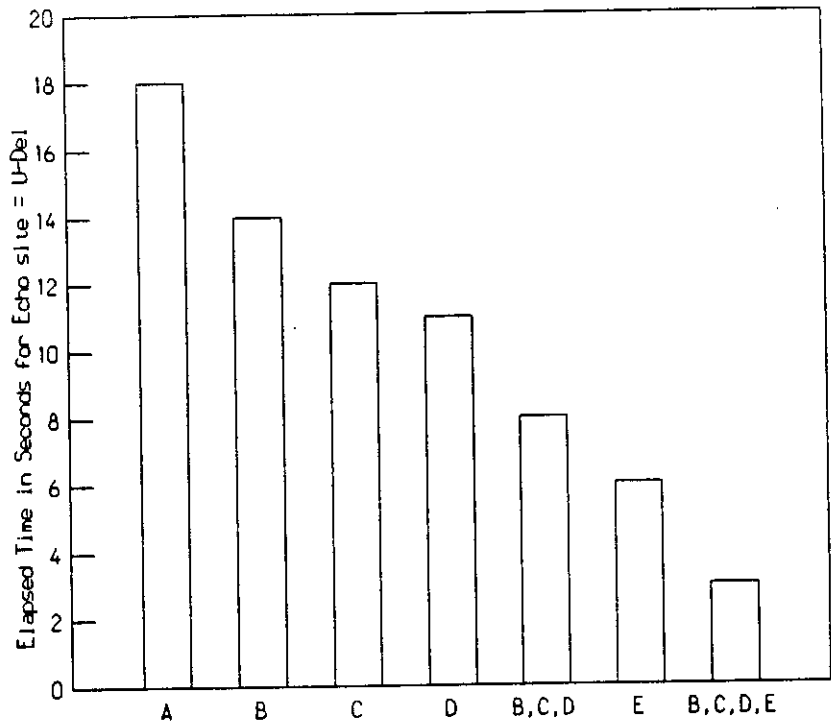
6.6 Evaluation of Internet Locus Strategies

The following strategies were used to achieve the Internet Locus performance results reported in sections 6.4 and 6.5:

- Reduce pathname expansion overhead
- Use new message primitives
- Use variable length packets to match underlying network characteristics
- Piggyback acknowledgements

No single strategy emerged as sufficient to reach the Internet Locus performance goals. Rather, the aggregate of all strategies was often necessary to achieve satisfactory performance. It is difficult to evaluate the absolute importance of each of the strategies. A strategy that is important in reducing the latency for one command may play a much smaller role in reducing the latency for some other command. Some strategies become more important as the distance between the using site and storage site increases. Other strategies may reduce communication overhead in general but contribute little to reducing the elapsed time for single commands.

Figure 6.11 illustrates the importance of each of the strategies and the aggregate effect of all of the strategies for the copy command. In this example, a 1K byte file stored at UCLA is copied to a site effectively located at U-Del.



- A - Lan Locus
- B - Piggyback Acks
- C - Variable length headers
- D - New Message Primitives
- E - Reduce Pathname Expansion overhead

Figure 6.11: Performance Effect of Internet Locus Strategies For the Copy Command

Piggybacking acknowledgments contributes least to reducing latency during the execution of the copy command in this example. The message traffic for the copy command consists mostly of request/response pairs. After a request is sent, the user process at the using site blocks until the response is received. The elapsed time for command completion is not greatly decreased when separate ACK messages are eliminated since the using site must wait for the response message before command execution can proceed. The importance of piggybacking ACKs is mainly to reduce general communication overhead.

High level message primitives and variable length packet headers contribute approximately the same towards reducing latency for the copy command. The new message primitives eliminate 15 of the 29 total non-ACK messages sent under LAN-Locus (see figure 5.7 for the copy command message traffic) and the short headers reduce the number of bits transmitted.

Reducing the number of messages due to pathname expansion contributes most to reducing the elapsed time for the example shown in figure 6.11, and exceeds the total contribution from using new message primitives, short headers, and piggyback ACKs combined. In this example, the parent directory of the target file is opened, read, and closed three times during the execution of the copy command under LAN-Locus. If the pathname had more than a single directory component, the number of messages due to pathname expansion would be even greater. Although reducing the number of messages due to pathname expansion plays a significant role in reducing the elapsed time for the copy command, all four strategies are needed to reduce the elapsed time to a satisfactory level.

The effect of the Internet Locus strategies for the *ls -l* command of a 54 entry directory located at Rand is shown in figure 6.12. Unlike the copy command, where reducing pathname expansion messages contributes most to reducing elapsed time, in this example the single most effective Internet Locus strategy is the use of new message primitives.

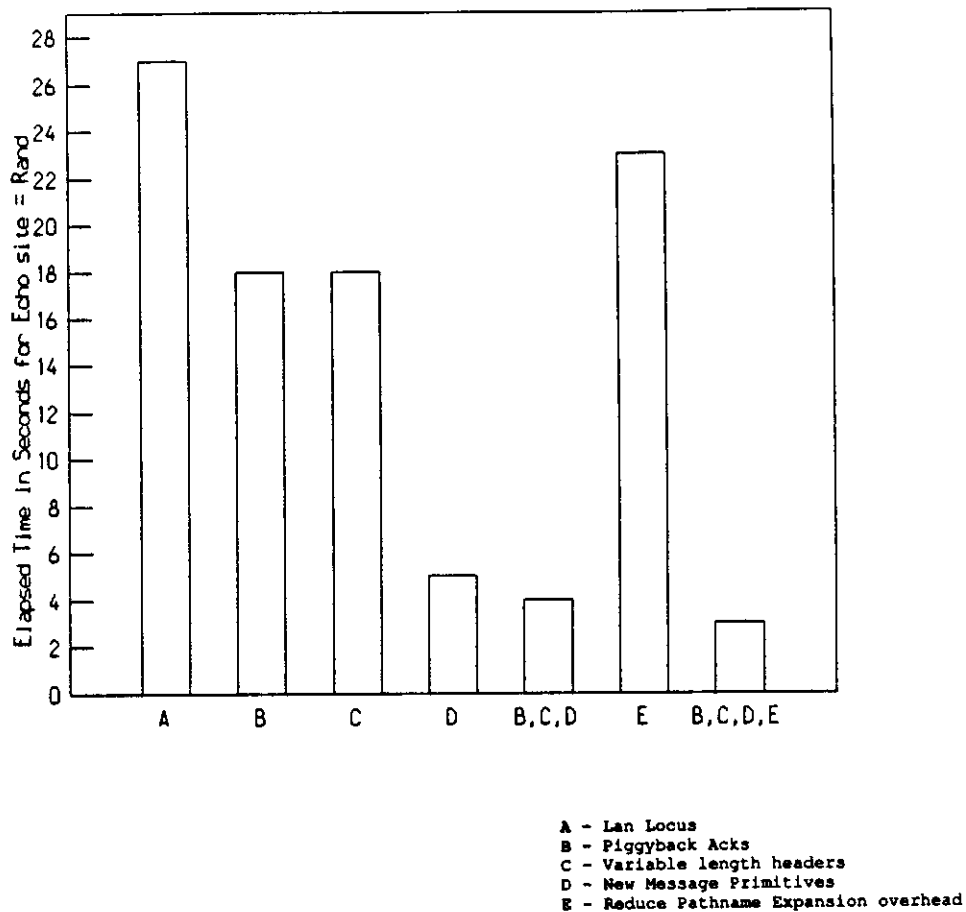


Figure 6.12: Performance Effect of Internet Locus Strategies For the Directory Listing Command

The importance of variable length headers is illustrated in figure 6.13. The graph displays the round trip delay for sending messages of variable size from UCLA to CIT, RAND, and UDEL. The increase in delay when messages cross the Arpanet single packet limit of 126 bytes is apparent. For instance, the round trip delay from UCLA to UDEL jumps from 300ms for a 98 byte message to 636ms for a 128 byte message. It is especially important to avoid this more than 200% increase in communication costs when the distance between the using site and storage site is large.

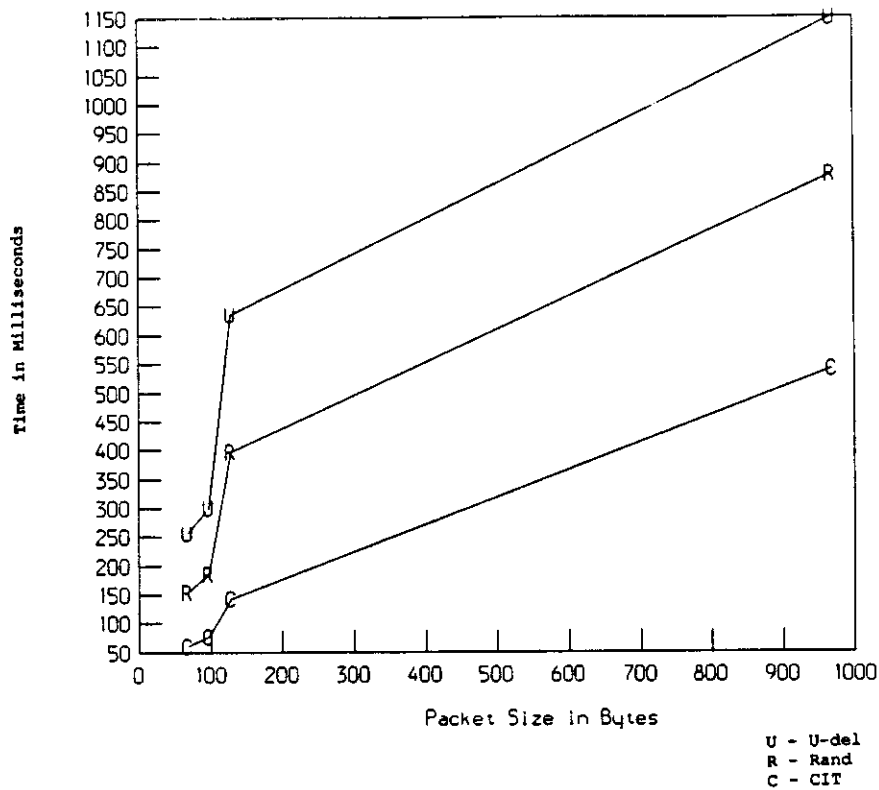


Figure 6.13: Round Trip Delay

6.7 Use of Internet Locus Strategies in Other Environments

The lessons learned during the development of Internet Locus are not limited strictly to Locus and may be beneficial to a variety of systems. One of the most important lessons is that efficient mapping between names and objects can play a significant role in improving the performance of a system. Whether the system is a distributed operating system or a distributed database management system, if references to remote objects exhibit sufficient locality, name to object caches can be effective in reducing latency. Systems that use name caches include the R* distributed database management system [Lin 80], and the Grapevine internet mail system.

Another important lesson is that the level of message primitives directly impacts performance. If more information can be transmitted in messages without loss of functionality, remote performance improves. Experience reported by the developers of the Network Graphics Protocol, as well as experience with Internet Locus confirm this lesson. A third lesson is that network characteristics, which are frequently assumed to be of no concern to applications above a given protocol layer, cannot always be ignored. A slight change in packet size may have a large impact on performance.

Although the Internet Locus strategies are most important for communication across long haul networks, they may also be beneficial in local area networks. For a lightly loaded LAN, little gain is expected because of the very low delay to transfer packets between sites. However, if there is significant contention on the network, the Internet Locus strategies can be useful in reducing the total amount of traffic, reducing CPU overhead spent on processing packets, and reducing the time to access remote data.

CHAPTER 7

Conclusions

7.1 Summary of Goals and Accomplishments

Network transparency has been viewed as a concept of similar significance to virtual memory [Den 83]. It had previously been successfully applied only to local area network based distributed systems. In this dissertation we have shown that transparency can be effectively applied to an internet environment that contains LANs and long haul networks.

An Internet operating system protocol that provides remote service for frequently executed user and application tasks with only a small amount of network overhead was designed and implemented. A distributed name cache was shown to further reduce network traffic. Moving the execution site of a process to the data storage site was also used to improve elapsed times and was especially effective when large amounts of data were accessed.

The two performance goals for an internet operating system have been largely met. The elapsed time for the completion of typical interactive commands, when data is stored across long haul links, has been shown to be nearly equal to the elapsed time when data is stored at the using site. The elapsed time to transfer large amounts of data across long haul links using native operating system commands has been shown to be superior to the elapsed time when a specialized file transfer protocol is used.

The focus on homogeneous operating systems and limited internet topologies is both a strength and weakness of this research. By limiting all sites to run the same operating system, we were able to use kernel to kernel communication instead of building software on top of existing operating systems. This design choice played an instrumental role in the performance of Internet Locus, but limits the types of systems that can participate in the internet operating system. Other approaches, such as the bridge architecture described in [Loc 83] are available to provide remote file access and update and remote execution across heterogeneous system interfaces.

By focusing on limited internet topologies, where the maximum separation between using site and storage site was approximately 5 IMP hops, we were able to keep the communication costs low enough so the performance goals could be met. In other internet configurations, where the time to transfer a block of data exceeds 1 or 2 seconds, remote access may be too expensive for full network transparency.

7.2 Future Work

The work described in this dissertation provides a foundation for much additional research. The Internet Locus testbed is an ideal environment to study data compression, file replication, and file caching for an internet operating system. A data compression scheme, as described in section 5.5.3, should be implemented and its effect on reducing message delay should be evaluated. The use of file replication to reduce remote access times should be studied. The overhead of maintaining replicated files across long haul links should be measured and the advantages and disadvantages of file replication compared to file caching should be explored.

Another area that merits further study is the interconnection of heterogeneous systems. If kernel code can be modified in different operating systems, can network transparency be achieved with satisfactory performance between heterogeneous systems?

The impact of internet transparency on distributed applications such as a distributed database management system should also be investigated. Since Internet Locus provides many of the necessary functions for database management operation such as transaction support, name-to-location mapping, and synchronization control, these functions do not have to be reimplemented by the database application. However, many problems remain, such as determining the optimal placement of database relations, selecting the best strategy for query processing and evaluating the effectiveness of a name mapping cache for a database system run on clusters of workstations connected by long haul networks.

Other extensions to the Internet Locus work include a study of message primitives for LANs connected by high bandwidth, high delay satellite links and the use of an internet operating system to enhance supercomputer access across long haul links.

Finally, a full scale implementation of Internet Locus at two or more geographically separated sites will allow the ideas of this dissertation to be evaluated in a more realistic setting than a testbed and may help uncover new distributed applications that can take advantage of internet transparency.

7.3 Contribution to Computer Science

Long haul computer networks have yet to fulfill the promise of easy access to geographically distributed resources. Users of these networks have been painfully aware of the boundaries between machines. Earlier research on network transparent operating systems has shown that increased reliability, improved performance, greater data sharing, and easy access to distributed resources can be achieved over a local area network. This dissertation extends the principle of network transparency to an internetwork environment consisting of local area networks connected by long haul networks. The successful demonstration of internetwork transparency marks the true beginning of the age of distributed processing.

APPENDIX A

The Implementation of Internet Locus

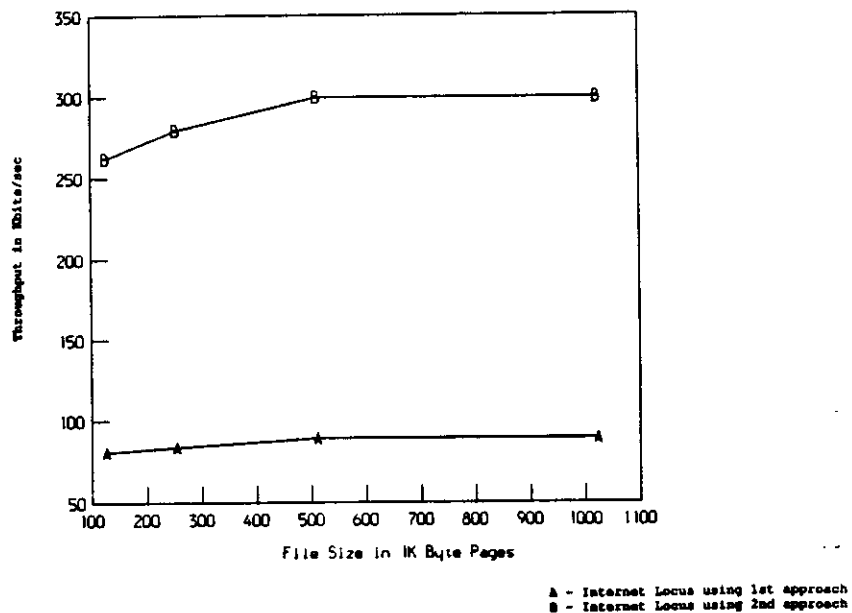
Since the Locus operating system supports both TCP/IP and the Locus protocol, there are two possible approaches for building Internet Locus. In the first approach, the existing IP implementation is used. When a Locus packet is ready to be transmitted over the local area network, the IP *write* routine is called (instead of the normal device driver routine) with the Locus packet as data. The packet is copied into user space and a user process which executes the IP module is scheduled. The IP module handles addressing, message queueing, fragmentation, and ICMP processing. Eventually, the local area network device driver is called and the Locus packet, encapsulated in an IP and ICMP header, is transmitted. The major benefit of this approach is that very little new IP code has to be written. This approach was used in the first Internet Locus implementation.

The alternative approach for Internet Locus is to reimplement IP at the kernel level and bypass the existing IP module. When a Locus packet is ready to be transmitted, an IP and ICMP header are added to the packet with the header fields correctly filled in. A significant amount of IP addressing and fragmentation code and ICMP code has to be written. The benefit of this approach is that extra copying of data and the overhead of user process scheduling is avoided.

A comparison of the throughput of LAN-Locus and Internet Locus using each of these approaches is shown in the figure below. The *dd* command, which was used to copy data between sites connected by a 10Mbps Ethernet, was executed for file sizes of 128k bytes to 1M bytes.*

The difference in performance between the approaches is dramatic. While the performance of Internet Locus using the second approach matches the performance of LAN Locus, Internet Locus using the first approach is significantly inferior. These results reenforce the claim that protocols to support operating system functions should be placed at the kernel level rather than the user process level. After the first implementation of Internet Locus was shown to be unquestionably inferior to LAN-Locus across the Ethernet, Internet Locus was completely reimplemented using the second approach.

THROUGHPUT OF THE DD COMMAND FOR INTERNET LOCUS



*Throughput is calculated from the elapsed time of the *dd* command and is not the aggregate throughput possible through the network channel. The Internet-Locus implementations do not include any of the enhancements discussed in chapters 4 and 5.

APPENDIX B

Locus Network Measurements

Ethernet statistics and Locus protocol statistics were gathered on the Locus Alpha network at UCLA for several days during 1983. The Alpha network, used by students and faculty engaged in distributed system research, consisted of four Vax 11/750s connected by a 10Mbps Ethernet cable. The Interlan Ethernet interface collects the following statistics: the number of packets transmitted and received, the number of packets in the receive fifo, the number of collisions, the number of packets with an alignment error, the number of collision fragments received, and the number of packets received with a CRC error. A measurement package was developed and added to the Locus network driver routines to collect the Ethernet statistics and also to record the number and protocol type of each network message transmitted and received by the site.

The Locus network measurements were collected on a single site from Saturday, May 28, 1983 through Tuesday, May 31. The following table shows the total number of messages transmitted and received by the site over the 4 day measurement period. The average number of messages transmitted and received was 1,356,921 per day. Not surprisingly, message traffic on the days that school was not in session was less than the message traffic on Tuesday which was the only school day measured.

Total Message Traffic	
Day	# of Messages
Saturday	1205669
Sunday	1125943
Monday	1436089
Tuesday	1659982

DISTRIBUTION OF MESSAGE TYPES FOR TUESDAY, MAY 31, 1983	
Message Type*	Number of Msgs Transmitted
Acknowledgment	333217
Read	113566
Open Response	76954
Probe	65326
Open	59894
USClose	47968
SSClose Response	29775
Read Response	28079
Remote Service Call	21279
USClose Response	21260
Write	18894
Probe Response	5520
USCommit	1629
USCommit Response	1629
Device Open	227
BESS	227
RSC Response	62
SSClose	55
Device Open Response	55
BESS Response	55
Newtop	32
Sndind	28
Gettop	22
Channel Open	18
New Partition	14
Site Down	12
Pmount	11
VV Prop	3
RReqlbnlst	2
Reqlbnlst	2
Clfss	2
VV Update	1
Site Down Response	1

* A description of each message type can be found in [Wal 83a]

The distribution of message types transmitted shows the large number of acknowledgements compared to other types of messages. A method to reduce the number of ACKS is described in chapter 5. The number of reads is greater than the number of writes as expected. Reads and opens are especially frequent in Unix because each directory in a pathname must be opened and searched.

SELECTED MESSAGES SENT PER HOUR FOR TUESDAY, MAY 31 1983							
Time	Open	ROpen	Read	RRead	Write	USClose	USCommit
12:30am	1841	1115	1591	223	2	1543	0
1:30	1841	1170	1585	239	0	1543	0
2:30	2669	1184	1798	332	6	1755	2
3:30	1953	1011	1775	181	2	1654	4
4:30	1894	1140	12079	317	3	1594	5
5:30	1841	1624	17266	1548	0	1548	0
6:30	1841	1572	1585	1309	0	1543	0
7:30	1840	4209	1666	1002	4	1527	2
8:30	1715	1457	1516	372	5	1413	5
9:30	2673	2651	2743	1277	116	2287	146
10:30	2653	4865	2482	1533	65	2180	66
11:30	4779	4318	5945	2549	449	3565	417
12:30pm	3766	6622	6940	2033	681	3171	331
1:30	2752	2365	3932	1327	1338	1875	17
2:30	2155	2988	2881	1280	841	1811	12
3:30	2816	2411	3921	969	355	2394	81
4:30	2594	4524	3972	2124	257	2235	75
5:30	5325	4970	6642	1971	779	3544	261
6:30	3529	2114	8012	551	1042	2895	193
7:30	1964	17264	4436	4552	2019	1641	10
8:30	1841	2171	4291	920	1804	1543	0
9:30	1901	1772	4078	572	1514	1591	3
10:30	1830	1711	3916	750	1286	1534	0
11:30	1881	1719	8514	846	6331	1582	4

The above table shows the distribution of selected message types transmitted per hour. Since most traffic consists of a request followed by a response, the number of messages received can also be estimated from the

table. For instance, from 6:30pm to 7:30pm 17,264 Open Responses were transmitted and 17,264 Open Requests were received.

The Open message is used for both the Open system call and the Stat system call which returns file descriptor information. Since the Stat call does not have a corresponding Close call, the number of Stats transmitted can be found by subtracting the number of Using Site Closes (USClose) sent from the number of Opens sent. For instance, at 12:30am out of 1841 Opens transmitted, 298 were actually Stat system calls, not Opens.

The message collision statistics confirm that the Ethernet is lightly loaded. The number of transmissions that result in a collision is less than 1% even at peak traffic hours. Furthermore, there were only 9 duplicate messages and 11 duplicate acknowledgements recorded over the entire 4 day period. Duplicates occur when a message is retransmitted because a timer has expired and both the original message and the retransmission are received at the destination site. There were no CRC errors or frame alignment errors recorded.

MESSAGE COLLISIONS OVER 24 HOURS - TUESDAY, MAY 31, 1983

Time	# of Msgs Tx	# of Collisions
12:30am	19282	4
1:30	19472	5
2:30	21976	4
3:30	19442	4
4:30	30103	65
5:30	37516	15
6:30	22132	5
7:30	26641	8
8:30	18564	9
9:30	29467	8
10:30	37475	24
11:30	46302	13
12:30pm	47869	12
1:30	35161	8
2:30	31889	9
3:30	32076	11
4:30	39038	10
5:30	51612	12
6:30	39392	4
7:30	76878	21
8:30	33332	11
9:30	30802	9
10:30	33008	21
11:30	48905	16

References

- [Bar 81] Bartlett, J., "A NonStop Kernel", Proceedings of the Eighth Symposium on Operating System Principles, Pacific Grove, California, December, 1981.
- [Bir 82] Birrell, A. Levin, R., Needham, R. Schroeder, M., "Grapevine: An Exercise in Distributed Computing", CACM, Vol. 25, No. 4, April 1982.
- [Bir 84] Birrell, A., Nelson, B., "Implementing Remote Procedure Calls", ACM Transactions on Computer Systems, Feb. 1984
- [Bog 80] Boggs, D., Shoch, J., Taft, E., and Metcalfe, R. "PUP: An Internetwork Architecture," IEEE Transactions on Communications, com-28:4, pp. 612-624 April 1980.
- [Bol 78] "Specifications for the Interconnection of a Host and an IMP", Report no. 1822, Bolt Beranek and Newman Inc, Cambridge, MA., 1978.
- [Bro 82] Brownbridge, D., L. Marshall, B. Randell, "The Newcastle Connection", Software- Practice and Experience, Vol. 12, pp. 1147-1162, 1982.
- [Cer 74] Cerf, V. and Kahn, R., "A Protocol for Packet Network Intercommunication", IEEE Transactions on Communication", vol. COM-22, pp. 637-648, May 1974.
- [Che 83] Cheriton, D., Local Networking and Internetworking in the V System, Proceedings of the 8th Data Communications Symposium, North Falmouth, MA., pp. 9-16, Oct. 1983.
- [Cla 85] Clark, D., "NETBLK: A Bulk Data Transfer Protocol", Request for Comments (in progress), MIT, 1985.
- [Dar 81] "DARPA Internet Protocol and Internet Control Message Protocol Specification", RFC 791, 792, USC/Information Sciences Institute, Los Angeles, CA., Sept. 1981.
- [Den 72] Denning, P., "On modeling program behavior" in Proc. Spring Joint Computer Conference, vol. 40, AFIPS Press, 1972, pp. 937-944.
- [Den 83] Denning, P., private communication, 1983.

- [Gal 78] Gallager, R., "Variations on a Theme by Huffman", IEEE Transactions on Information Theory, Vol. IT-24, No. 6, Nov. 1978.
- [Gold 83] Goldberg, A., Lavenberg, S., Popek, G., "A Validated Distributed System Performance Model", Performance 83 - The 9th International Symposium on Computer Performance Modeling, Measurement, and Evaluation, College Park, MD., May 1983.
- [Gur 85] Gurwitz, R., "Object Oriented Interprocess Communication in the Internet", BBN Labs, note in progress, 1985.
- [Han 84] Hanson, S., Kraut, R., and Farber, J., "Interface Design and Multivariate Analysis of Unix Command Use", ACM Transactions on Office Information Systems, Vol. 2, No. 1, January, 1984.
- [Hav 82] Haverty, J., "The C/30 Packet Switch", in the Internet Protocol Transition Workbook, J. Postel, ed., USC/Information Sciences Institute, Los Angeles, CA., March 1982.
- [Hol 81] Holler, E., "The National Software Works", Distributed Systems - Architecture and Implementation, Lecture Notes in Computer Science 105, Springer-Verlag, 1981.
- [Kar 84] Karels, M., private communication, 1984.
- [Kea 83] Kearns, DeFazio, "Locality of Reference in Hierarchical Database Systems", IEEE Trans. on Software Eng., March 1983.
- [Lan 84] Lantz, K., Nowicki, W., Theimer, M., "Factors Affecting Performance of Distributed Applications", SIGCOMM 84, Computer Communication Review, Volume 14, Number 2, 1984.
- [Laz 84] Lazowska, E., Zahorjan, J., Cheriton, D., Zwaenepoel, W., "File Access Performance of Diskless Workstations", Stanford University Technical Report 84-06-01, June 1984.
- [Lea 82] Leach, P., Stumpf, J., Hamilton, J., and Levin, P., "UIDS as Internal Names in a Distributed File System" ACM Sigact-Sigops Symposium on Principles of Distributed Computing, Ottawa, Canada, August, 1982.
- [Lin 80] Lindsay, B., "Object Naming and Catalog Management for a Distributed Database Manager" IBM Research Report RJ2914 (36689) IBM Research Lab, San Jose, Calif., August, 1980.
- [Lin 84] Lindsay, B., Haas, L., Mohan, C., Wilms, F., Yost, R., "Computation and Communication in R*: A Distributed Database Manager", ACM Trans. on Computer Systems, Vol. 2. No. 1, Feb. 1984.

- [Lis 82] Liskov, B., "On Linguistic Support for Distributed Programs", IEEE Trans. on Software Eng., May 1982.
- [Loc 83] The Locus Distributed System Architecture, Edition 2.5, October 1983, Locus Computing Corp., Santa Monica, CA., to be published by MIT press.
- [Mad 76] Madison, A., Batson, A., "Characteristics of Program Localities", CACM, Vol. 19, No. 5, May 1976.
- [Maj 84] Majumdar, S., "Locality and File Referencing Behavior: Principles and Applications", M.S. thesis, University of Saskatchewan, August 1984.
- [Pop 81] Popek, G., Walker, B. Chow, J., Edwards, D., Kline, C., Rudsin, G., and Thiel, G., "LOCUS: A Network Transparent, High Reliability Distributed System", Proceedings of the 8th SOSP, Pacific Grove, CA., December, 1981.
- [Pos 80] Postel, J., "Internetwork Protocol Approaches," IEEE Transactions on Communications, com-28:4, pp. 604-611, April 1980.
- [Ras 81] Rashid, R., and Robertson, G., "Accent: A Communication Oriented Network Operating System Kernel", Proceedings of the 8th SOSP, Pacific Grove, CA., pp. 64-75, December, 1981.
- [Rei 85] Reiher, P., and Popek, G., "Locus Naming in a Large Scale Environment", submitted to SOSP 85.
- [Ric 83] Richardson, M., Needham, R., "The TRIPOS Filing Machine, a front end to a File Server", Proceedings of the 9th ACM Symposium on Operating System Principles, 1983.
- [Rod 76] Rodriguez-Rosell, J., "Empirical Data Reference Behavior in Data Base Systems", IEEE Computer, November 1976, pp. 9-13.
- [Smi 82] Smith, A., "Cache Memories", Computing Surveys, Vol. 14, No. 3, September 1982.
- [Sto 80] Stone, H., (ed.) "Introduction to Computer Architecture", Science Research Associates, Chicago, IL., p. 210, 1980.
- [Tho 73] Thomas, R., "A Resource Sharing Executive for the Arpanet", Proc. National Computer Conference 42:155-163, AFIPS, June 1973.
- [Uni 81] Unix Programmer's Manual, 7th Edition, Dept. of Electrical Engineering and Computer Science, University of California, Berkeley, California, June, 1981

- [Wal 83a] Walker, B., "Issues of Network Transparency and File Replication in the Distributed Filesystem Component of LOCUS", Ph.D. Dissertation, UCLA Computer Science, 1983.
- [Wal 83b] Walker, B., Popek, G., English, B., Kline, C., Thiel, G., "The LOCUS Distributed Operating System", Proceedings of the 9th ACM Symposium on Operating System Principles, Oct., 1983.

