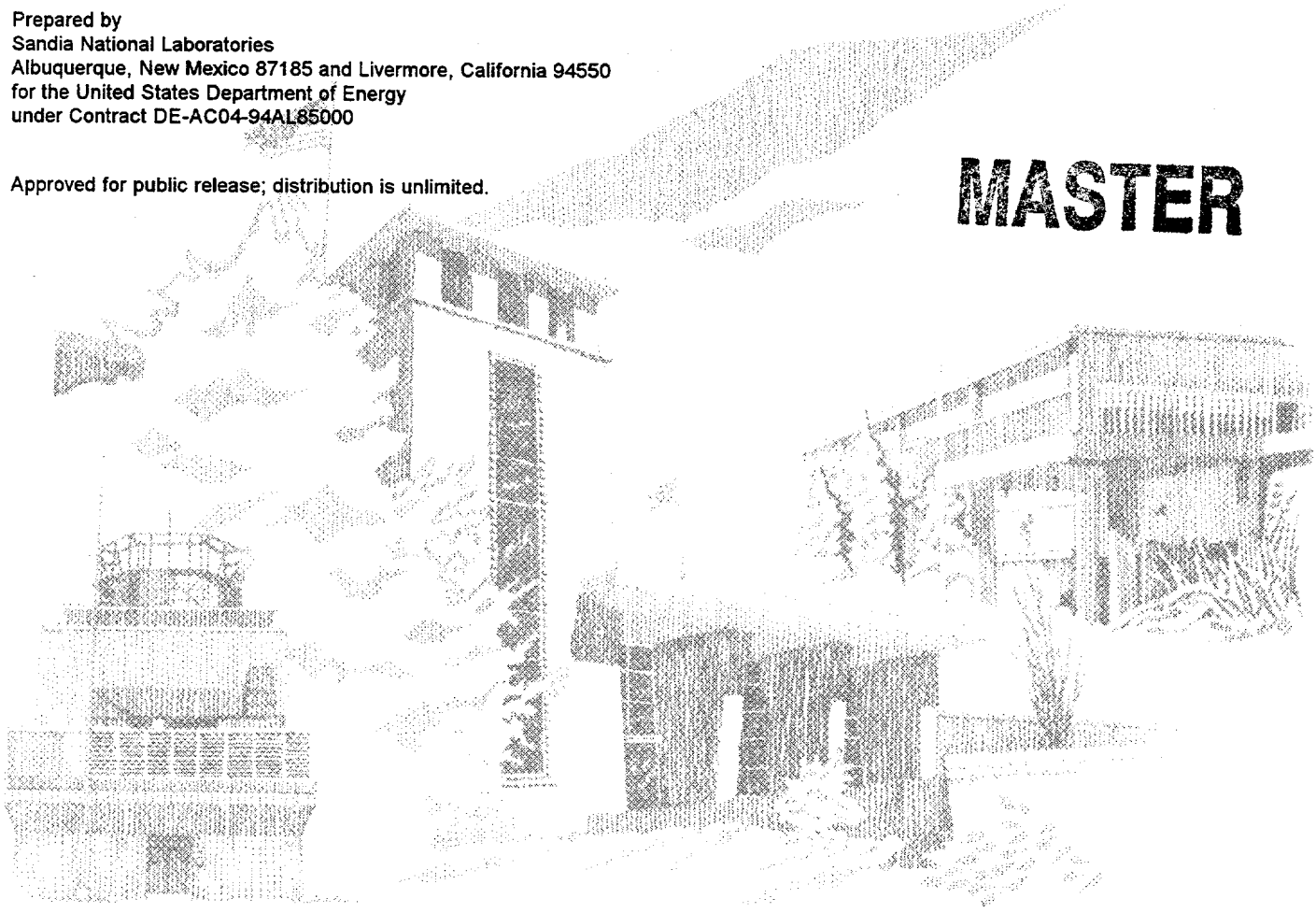
SANDIA REPORT

SAND94–2594 • UC–705
Unlimited Release
Printed January 1997

Performance Modeling of Network Data Services

Rena A. Haynes, Lyndon G. Pierson

Prepared by
Sandia National Laboratories
Albuquerque, New Mexico 87185 and Livermore, California 94550
for the United States Department of Energy
under Contract DE-AC04-94AL85000



Issued by Sandia National Laboratories, operated for the United States Department of Energy by Sandia Corporation.

Available to DOE and DOE contractors from
Office of Scientific and Technical Information
P.O. Box 62
Oak Ridge, TN 37831

Prices available from (615) 576-8401, FTS 626-8401

Available to the public from
National Technical Information Service
U.S. Department of Commerce
5285 Port Royal Rd
Springfield, VA 22161

NTIS price codes
Printed copy: A03
Microfiche copy: A01

SAND94-2594
Unlimited Release
Printed January 1997

Distribution
Category UC-705

Performance Modeling of Network Data Services

Rena A. Haynes
Lyndon G. Pierson

Scientific Computing Systems
Sandia National Laboratories
Albuquerque, NM 87185

Abstract

Networks at major computational organizations are becoming increasingly complex. The introduction of large massively parallel computers and supercomputers with gigabyte memories are requiring greater and greater bandwidth for network data transfers to widely dispersed clients. For networks to provide adequate data transfer services to high performance computers and remote users connected to them, the networking components must be optimized from a combination of internal and external performance criteria. This paper describes research done at Sandia National Laboratories to model network data services and to visualize the flow of data from source to sink when using the data services.

Acknowledgment

The authors with to express special thanks to Jethro H. Greene for his work in developing software to describe and process packet trains in an object-oriented design. His enthusiasm for software development and object-oriented thinking kept the authors on the right track.

Table of Contents

Introduction

Networks at major computational organizations are becoming increasingly complex. This complexity is being driven by the introduction of large massively parallel computers connected to widely dispersed client systems. For example, at Sandia National Laboratories an 1840 node Intel Paragon massively parallel computing system is available to computational scientists throughout the United States.

The computational capacity of large compute servers allows scientists to process very large calculations which in turn generate large amounts of data. Data sets generated on the Sandia Intel Paragon average from 200 Megabytes to 18 Gigabytes. Data generated on these systems are typically off-loaded to remote file or mass storage servers so that new calculations can be run to produce new data. Off-loaded data may be further processed by visualization servers to create animation sequences that are again saved to file or mass storage servers where they will be retrieved for display on various remote clients.

The computational environment described in the preceding paragraph requires high performance data transfer services. For networks to provide high performance data transfers, both hardware and software components must be optimized from a combination of internal and external performance criteria. In order to determine these criteria, the major data transfer services at Sandia were analyzed using a network modeling methodology called the packet train model.

The packet train model, which describes network traffic as packet streams between pairs of nodes on the network, was developed during a study of traffic on a local area network at MIT [5]. Since its development, the packet train model has been used to analyze network traffic to identify locality of reference in networks [5], to analyze network delays caused by congestion and limited buffers [6], and to characterize network traffic [7]. The study described in this paper used the packet train model in a more generalized object oriented fashion to describe and analyze components involved in network data transfers to and from mass storage systems. In addition, this paper describes prototype simulation and visualization software that was developed for this research.

The following sections describe the packet train model in more detail, the object-oriented packet train model specification developed for this research, and the prototype simulation and visualization software developed for this research. Summary and conclusions from this research are also given.

Packet Train Model

Local area network (LAN) traffic studies have typically used a statistical model to describe packet arrival. The primary characteristics of these models are that packets are independent and the arrival of packets is based on a probability density function. In measuring traffic patterns in a

LAN at MIT, Jain and Routhier [5] found that the network packet arrival process can best be described in the context of a train model.

In the packet train model, a train consists of a number of packets traveling in both directions between two network nodes. A virtual train track is equivalent to a network node pair. The interval between packets on a given train is specified by the maximum allowed intercar gap (MAIG), which is a system parameter based primarily on network hardware and software between the two nodes. If no packets are seen between a pair of network nodes for the MAIG time, the current train has ended and the next packet for the virtual track is the first car, or locomotive, of the next train. The inter-train time is the time between the last car, or caboose, of the previous train and the locomotive of the next train. Inter-train time is largely dependent on types of network applications and their frequency of use.

While the traffic train model has been used primarily for characterizing and analyzing physical networks, the same methodology can be used to characterize and analyze distributed software components. The packet train model for network packet arrival is especially appropriate for modeling data movement to and from mass storage systems. Client interfaces to mass storage systems are used primarily for bulk file transfers. As such, traffic between a mass storage system and its client is characterized by bursts. The probability that the next packet is part of the mass storage system/client train is largely dependent on the average file size and the current length of the train. A characterization of the usage of the Network Storage Service at Sandia [4] found that even a stateless protocol like NFS exhibits the bursty nature of file transfers.

Another reason that the packet train model is important for mass storage data movement is that the architectures of mass storage systems are being distributed to allow scaling for performance and capacity. In general, name space operations, such as file creation and file status listing, are performed by mass storage system components that are separate from components that move data. While the number of components involved in moving a block of data between the client and server storage is minimized, many client/server components may be involved in moving separate blocks of data in parallel. Each component of a mass storage system can exist on an individual network node. The following section describes how the packet train model can be used to model data transfers to and from a distributed mass storage system.

A Packet Train Model of a Distributed Mass Storage System

Architectures for mass storage systems are described in [3] and [11]. The generic description in [3] and the implementation description in [11] both contain components that exist in all mass storage systems. These components are a name server, a bitfile server, some number of storage servers, a physical volume library, some number of physical volume repositories, and some number of movers. Each of these components is involved in some way when a file is transferred between a client and the mass storage system. Each component typically interacts with a metadata component that manages metadata required for the component's functionality. In some

systems the metadata management component is internal to each server, while other systems provide a separate component that requires network traffic. For this study, communication to the metadata management component and its processing are considered part of the server processing. The functionality and communication characteristics of each component are discussed in more detail in the following paragraphs.

The name server component of a mass storage system is responsible for mapping the name of a file to an object identifier in the mass storage system. For some systems, like the HPSS architecture described in [11], the name server is also the component that authorizes access to file objects. In providing these functions for a file transfer, the name server component primarily communicates with the client application at the start of a file transfer.

The bitfile server component of a mass storage system provides a file abstraction for objects in the mass storage system that may reside in many physical storage locations. The bitfile server component allows access to mass storage files based on starting file offset and length. To provide this functionality, the bitfile server communicates with the client application for each write or read request. The bitfile server then translates the client application request which contains starting file offset and length information to corresponding request(s) containing storage segment identifier, offset, and length, for one or more storage servers. In some mass storage system architectures, data must exist on the top level of the storage hierarchy before any can be transferred to the client application. In this case, the bitfile server would start a caching operation that would involve communication between multiple storage servers and movers to transfer the data from current storage to top level storage. Because caching is not the normal path for data transfers to/from mass storage systems, it was not considered in this model of mass storage network data traffic. After all data for an I/O request have been transferred, the bitfile server receives a reply message that indicates the success or failure of the data transfer.

A storage server component of a mass storage system translates storage segments into locations on physical volumes of storage. A storage server typically deals with only one type of storage, e.g., disk or tape. The storage server also schedules mounting and dismounting of removable media in the mass storage system. When an I/O request is received from the bitfile server, the storage server translates the storage segment information to physical volume location information. If the physical storage volume required to complete the I/O request is a removable volume, the storage server schedules all mounts required for the I/O with the physical volume library component. No data movement is started until all physical volumes involved in the transfer are ready to transfer. The storage server communicates with mover components to initiate the read or write. Once all data for a request are transferred, the storage server receives status messages from all movers involved in the request. This information is converted to a reply to the bitfile server.

The physical volume library component of a mass storage system manages all physical volumes in the system. This component receives requests from storage server component to mount or dismount physical volumes. Mount/dismount requests from storage servers are converted to

mount/dismount requests to the physical volume repository that contains the volume specified. The time required for mounting the physical volumes required for an I/O request is the greatest part of the startup time for an I/O request. In general, a mount is required only on the first request for I/O to a file on the mass storage systems.

Physical volume repositories are the components that communicate with robotic devices to mount or unmount physical volumes. These components receive a request from the physical volume library component and communicate with a robot, which may be electronic or human, to cause the physical volume to be mounted on a device for reading/writing. As stated in the previous paragraph, the time required to mount the physical volume is the largest part of the startup time for an I/O request, but should only impact the first request. Consequently, physical volume repository component communication and processing plays a minor role in network data traffic in mass storage systems.

The mass storage system components that play the largest role in network data traffic to and from mass storage systems are the mover components. A mass storage system mover is responsible for transferring data from a source device to a sink device. A device can be a standard I/O device with geometry, such as tape or disk, or a device without geometry, such as network or memory. Mover components are involved in data transfer startup traffic because they communicate with the physical volume library component to verify that a particular physical volume is mounted and ready on an I/O device. They also communicate with the storage server component to receive read/write requests and with other mover components when data are migrated or cached between storage levels in the mass storage hierarchy. The primary mover communication of interest to this study, however, is the communication with client applications that occurs as a result of a client read/write request.

The previous paragraphs have described the major internal components of a mass storage system. Another component, external to the mass storage system, is the client application. Mass storage systems typically support standard network-based interfaces as well as a programming interface. The implementation of the client application will affect the network traffic involved in a data transfer to/from a mass storage system. For example, the most widely available network interfaces for file transfer are the File Transfer Protocol (FTP) [9] and the Network File System (NFS) [10] services. Both of these applications involve a client and a server process. In most mass storage systems, file transfers using these protocols cause data to be transferred from the mover component to the FTP or NFS server component which forwards the data to the FTP or NFS client. A mass storage system like HPSS, however, allows data to be sent directly from the mover component to the client component.

The analysis, described in the preceding paragraphs, of the communication and processing required in transferring files to and from mass storage systems indicates that the network traffic can be broken into four phases. These phases are the initial startup, the input/output request startup, the data movement, and the reply phase.

The initial startup phase determines the mass storage data to be transferred to the client and readies software and hardware to deliver the data. The initial startup phase involves communication between the client and the name server, the client and the bitfile server, the bitfile server and the storage server, the storage server and the physical volume library, the physical volume library and the physical volume repository, and the physical volume library and the mover. In packet train terminology, each of these component pairs can be considered a virtual track. Except for traffic involving the physical volume library, the "trains" that travel these tracks are short with small MAIGs. Because the communication involving the physical volume library includes latencies involved in mounting removable storage media, these "trains" will have large MAIGs relative to the others in this phase.

The input/output request phase communicates requests for input/output from the client to a mover that has the storage device ready to receive or transmit the data. This phase involves communication between the client and the bitfile server, the bitfile server and the storage server, and the storage server and the mover. The trains that follow these tracks are short with small MAIGs.

The data movement phase of network data traffic for mass storage file transfers is where the bulk of the traffic occurs. This phase consists only of communication between the client and mover in architectures where an intermediate agent like FTP or NFS is not needed. If a server agent is used, this phase consists of communication between the client and the server agent as well as communication between the server agent and the mover component. Trains that follow these tracks are significantly longer than trains in the other phases. The actual train size will depend on file and input/output request block sizes. The MAIG value will be dependent on network and input/output device latencies.

The reply phase of network data traffic for mass storage file transfers consists of the communication to relay the status of the file transfer to the various components involved in the transfer. This phase involves communications between the mover and the storage server, the storage server and the bitfile server, and the bitfile server and the client. If an intermediate agent, like FTP, is used it also includes communication between the intermediate agent and the client. Like communication in the initial and the input/output request phases, the trains that follow these tracks are short with small MAIGs.

This packet train analysis of network data traffic to/from mass storage systems indicates that file transfers typically involve a series of short trains to setup and request the data transfer, followed by a series of long trains to move the data, and another series of short trains to return status information. It suggests that performance improvements for large file transfers should focus on reducing latencies in networks and devices and maximizing request block sizes. Performance improvements for small file transfers must include latencies in setup and input/output request processing.

Object-Oriented Packet Train Model Specification

The previous section described how the packet train model of network data traffic can be used to analyze a complex distributed software architecture. Verification of this analysis would require a simulation tool in which requests can be input and tracked throughout the system, with the results compared with actual system operation. As a start toward developing such a tool, the specifications for two major objects in the packet train model were developed in this research.

The two primary objects in the packet train model of network data traffic are the packet train and the network nodes. The packet train object represents the data flowing on a virtual train track between two network node objects. The network node objects are the components that operate on packet trains. The following paragraphs describe the packet train and network node specifications in more detail.

The packet train specification that was developed contains fields that can be used to determine train length, amount of time on the virtual track that runs from the source node to the destination node, the virtual tracks used, and counters to hold cumulative attributes for the packet train as it travels from the source node to the destination node. The length of a packet train is determined from the number of cars in the train. The amount of time spent on the virtual track connecting the previous node to the current node is given by the MAIG value. Virtual tracks are identified by network node pairs. The previous virtual track is defined by the previous and current network nodes. The next virtual track is defined by the current and next network nodes. The total virtual track is specified by the source and destination nodes. The attribute counters defined for the packet train specification are cumulative delay, cumulative node count, and total error exposure.

Packet train object operations include routing, dividing, merging, display, creation, and termination. A routing operation switches a packet train from one virtual track to another. The dividing operation splits a packet train into two or more packet trains. The merging operation combines two packet trains. The display operation outputs the packet train length, MAIG, number of cars, in-route time, and counter information. Packet train creation occurs at a source node and packet train termination occurs at the destination node.

The network node object is the processing component in the packet train data traffic model. A network node object may represent a hardware or software component. It may be expanded into multiple layers represented by multiple nodes or collapsed into a single layer represented by a single node. Fields in the network node object identify the node, the node layer, a list of adjacent nodes, and node characteristics. Node characteristics include input data bit rate, output data bit rate, maximum packet processing rate, maximum packet size and maximum train size. The values of the node characteristics determine the effect of node operations on a packet train. The network node object also contains queue structures that are used to receive and send packet trains and counters for each queue to maintain cumulative traffic information.

The operations defined for the network node object are construction, destruction, display, and process packet train. Network node construction occurs when a new node object is defined. The destruction operation removes a network node. The display operation outputs the current status of the network node queues and counters. The packet-train process function removes a packet train from an input queue, processes it based on node attributes, and routes the train(s) to the next virtual track.

Prototype Simulation and Visualization of the Object-Oriented Packet Train Model

As a preliminary step towards using the packet-train model in a simulation of network data traffic, a prototype simulation and visualization system was developed. As shown in the previous section, the packet train model easily lends itself to an object-oriented approach. The specification described above was rapidly translated into C++ software that was used to model some primitive network data traffic and network processing components. The simulation generated packet trains at a source network node. The source node processed the created trains based on its node characteristics and its processing operation, which caused the packet train to be routed to the next network node. The next node processed the packet train based on its node characteristics and sent the train to the next node. The packet train was processed in this manner until it reached the destination node.

A prototype visualization tool was developed in addition to the simulation prototype. This visualization tool made use of an emerging software package that provides a component approach to graphical interfaces. This software, Tk and Tcl, was developed by John Ousterhout [8]. It allows an application to be build from a set of reusable components. This object-oriented approach allowed a rapid prototype for visualizing the simulation.

The prototype visualization tool was started with a Tcl script that created displays for the network objects in the simulation. The visualization tool accepted input from a pipe file that was filled by the prototype simulation. Values from the pipe file were used to update the displays. As packet trains were received and processed in the simulation, the displays were updated to show the amount of traffic received and sent from each network node.

Summary and Conclusions

Improving the performance of network data services is becoming increasingly important as the need for a high performance networking infrastructure is recognized. Two of the major data services used in networks today are the File Transfer Protocol (FTP) and the Network File System (NFS). FTP and NFS services are used primarily in local area networks and campus area networks. The major systems used as source/sink for FTP and NFS data movement at Sandia are

mass storage systems. A study undertaken to determine the usage characteristics of these systems indicated that both FTP and NFS traffic between these systems and their clients is bursty.

A network modeling methodology called the packet train model has been used to analyze and characterize network traffic on physical networks. This research extended the model to include the application software involved in file data movement to and from mass storage systems. Our analysis showed that file transfers typically involve a series of short trains to setup and request the data transfer, followed by a series of long trains to move the data, and another series of short trains to return status information. It suggests that performance improvements for large file transfers should focus on reducing latencies in networks and devices and maximizing request block sizes. Performance improvements for small file transfers must include latencies in setup and input/output request processing.

An object-oriented design of two major objects in the packet train model was defined, and prototype software was developed to simulate and visualize the flow of data from source to sink. Communication bottlenecks were visually apparent when communication failures were modeled. Our analysis indicates that the packet train model is well suited for simulating network data services.

References

[1] Comer, Douglas. Internetworking with TCP/IP: Principles, Protocols, and Architecture. Prentice-Hall, Inc., Englewood Cliffs, N.J., 1988.

[2] Cargill, Tom. C++ Programming Style. Addison-Wesley Publishing Company, Reading, MA., 1992.

[3] Garrison, R., et al. (eds.), Reference Model for Open Storage Systems Interconnections: Mass Storage Reference Model Version 5, IEEE Storage System Standards Working Group (P1244), September 1994.

[4] Haynes, Rena A. "Network Storage Service Usage Characteristics", Digest of Papers, Twelfth IEEE Symposium on Mass Storage Systems, (April, 1993), 241-247.

[5] Jain, Raj, and Shawn A. Routhier. "Packet Trains--Measurements and a New Model for Computer Network Traffic", IEEE Journal on Selected Areas in Communications, Vol. SAC-4, No. 6, (September, 1986), 986-995.

[6] Joudeh, Ihab A., and Eric B. Hall. "Delay Analysis for Packet Trains over Computer Communication Networks", National Telecommunications Conference, Cat. No. 92CH3120-3, Washington, D.C., (May, 1992), 13/7-14.

[7] Heimlich, Steven A. "Traffic Characterization of the NFSNET National Backbone", USENIX Conference Proceedings, Washington, D.C., (January, 1990), 207-227.

[8] Ousterhout, John K. Tcl and the Tk Toolkit. Addison-Wesley Publishing Company, Reading, MA. , 1994.

[9] Postel, J. and J. Reynolds. File transfer protocol (FTP). Request for Comments 959, Network Information Center, October 1985.

[10] Sandburg, R., D. Goldberg, S. Kleiman, D. Walch, and B. Lyon. "Design and Implementation of the Sun Network Filesystem", USENIX Conference Proceedings, Berkeley, CA.,(June, 1985), 119-130.

[11] Teaff, D., Coyne, R., and R. Watson. "The Architecture of the High Performance Storage System (HPSS)", Fourth NASA GSFC Conference on Mass Storage Systems and Technologies, College Park, MD, March 28-30, 1995.

DISTRIBUTION:

2 MS	0806	L. G. Pierson, 4616
1	0806	T. J. Pratt, 4616
1	0661	W. D. Swartz, 4818
1	0807	R. M. Cahoon,4918
1	0807	J. P. Noe, 4918
5	0807	R. A. Haynes,4918
1	0188	C. E. Meyers, 4523
1	9018	Central Technical Files, 8523-2
5	0899	Technical Library, 4414
2	0619	Review & Approval Desk, 12630 For DOE/OSTI