Performance Analysis in High-Speed Wide Area IP over ATM Networks: Top-to-Bottom End-to-End Monitoring

Brian Tierney,
William Johnston,
Jason Lee, and Gary Hoo

Imaging and Distributed Computing Group
Ernest Orlando Lawrence Berkeley National Laboratory[1]
University of California
Berkeley, CA, 94720

Abstract

We describe an approach to the analysis of the performance of distributed applications in high-speed wide-area networks. The approach is designed to identify all of the issues that impact performance, and isolate the causes due to the related hardware and software components. We also describe the use of a distributed parallel data server as a network load generator that can be used in conjunction with this approach to probe various aspects of high-speed distributed systems from top-to-bottom of the protocol stack and from end-to-end in the network. To demonstrate the utility of this approach we present the analysis of a TCP over ATM problem that was uncovered while developing this methodology. This work was done in conjunction with the ARPA-funded MAGIC gigabit testbed.

Contents

1. The work described in this paper is supported by ARPA, Computer Systems Technology Office (http://ftp.arpa.mil/ResearchAreas.html) and the U. S. Dept. of Energy, Office of Energy Research, Office of Computational and Technology Research, Mathematical, Information, and Computational Sciences Division (http://www.er.doe.gov/production/octr/mics), under contract DE-AC03-76SF00098 with the University of California. Authors: tierney@george.lbl.gov, wejohnston@lbl.gov, Lawrence Berkeley National Laboratory, mail stop: B50B-2239, Berkeley, CA, 94720, ph: 510-486-5014, fax: 510-486-6363, http://www-itg.lbl.gov). This is report no. LBL-38246.

1.0 Introduction

The MAGIC testbed is a large-scale, high-speed, ATM network: It is a heterogeneous collection of ATM switches and computing platforms, several different implementations of IP over ATM, a collection of "middleware" (distributed services), etc., all of which must cooperate in order to make a complex application operate at high speed. As developers of high-speed network-based distributed services, we often observe unexpectedly low network throughput and/or high latency. The reason for the poor performance is frequently not obvious. The bottlenecks can be (and have been) in any of the components: the applications, the operating systems, the device drivers, the network adapters on either the sending or receiving host (or both), the network switches and routers, and so on. It is difficult to track down performance problem because of the complex interaction between the many distributed system components, and the fact that problems in one place may be most apparent somewhere else.

Network performance tools such at "ttcp" and "netperf" are commonly used to determine the throughput between hosts on the network. While these are useful tools to start with, we have observed many cases where ttcp performance is reasonable, but real application performance is still poor. Real distributed applications are complex, bursty, and have more than one connection in and/or out of a given host at one time; tools like ttcp do not adequately simulate these conditions.

We have developed a methodology and tools for monitoring, under realistic operating conditions, the behavior of all the elements of the application-to-application communications path in order to determine exactly what is happening within this complex system. We have instrumented our applications to do timestamping and logging at every critical point. We have also modified some of the standard Unix network and operating system monitoring tools to log "interesting" events using a common log format. This monitoring functionality is designed to facilitate performance tuning and network performance research. This allows us to measure network performance in a manner that is a much better "real-world" test than ttcp. It also allows us to measure throughput and latency characteristics of our distributed application code.

The goal of this work is to produce predictable, high-speed components that can be used as building blocks for high-performance applications, rather than having to "tune" the applications top-to-bottom as is all too common today. In this paper we describe the techniques for monitoring and analysis, and some experimental results from using this method. We also describe the architecture and performance of a prototype application and a distributed - parallel data server, called the DPSS (Distributed Parallel Storage System, formerly known as the Image Server System, or ISS), that is used to drive many of the experiments.

2.0 Monitoring Mechanisms

We have instrumented several applications and servers, and have generated tools to monitor the system at many levels. As messages enter and leave all parts of the user-level system, timestamps are taken and logged using a common logging format. Several of these instrumented applications and tools are described below.

2.1 MAGIC and the Distributed Parallel Storage System (DPSS)

We have designed and implemented a wide area network-based, distributed-parallel storage system as part of an ARPA-funded collaboration known as the MAGIC gigabit testbed [13], and as part of DOE's high-speed distributed computing program. This technology has been quite successful in several environments. The DPSS provides an economical, high-performance, widely distributed, and highly scalable architecture for caching large amounts of data that can potentially be used by many different users. Our

current implementation is optimized for providing access to large, image-like, read-mostly data sets. In the MAGIC testbed, the DPSS is distributed across several sites separated by more than 1000 Km of high speed network that uses IP over ATM as the network protocol, and is used to store very high resolution images of several geographic areas. “TerraVision” is a terrain visualization application that uses the DPSS to let a user explore / navigate a “real” landscape represented in 3D by using ortho-corrected, one meter per pixel images and digital elevation models (see [10]). TerraVision requests from the DPSS, in real time, the sub-images (“tiles”) needed to provide a view of a landscape for an autonomously “moving” user. Typical use requires aggregated data streams as high as 100 to 200 Mbits/sec. Even the current prototype DPSS is easily able to supply these data rates using several disk servers distributed across the network.

The combination of the distributed nature of the DPSS the high data rates required by TerraVision and various load simulators makes this a good system with which to test a high-speed network in a much more realistic manner than ttcp-like tools allow.

2.1.1 DPSS System Architecture

The DPSS is an implementation of a distributed-parallel data storage architecture. It is essentially a logical “block” server that is distributed across a wide area network, and is used to supply data to applications located anywhere in the network. Figure 1 illustrates the architecture. There is no particular organization

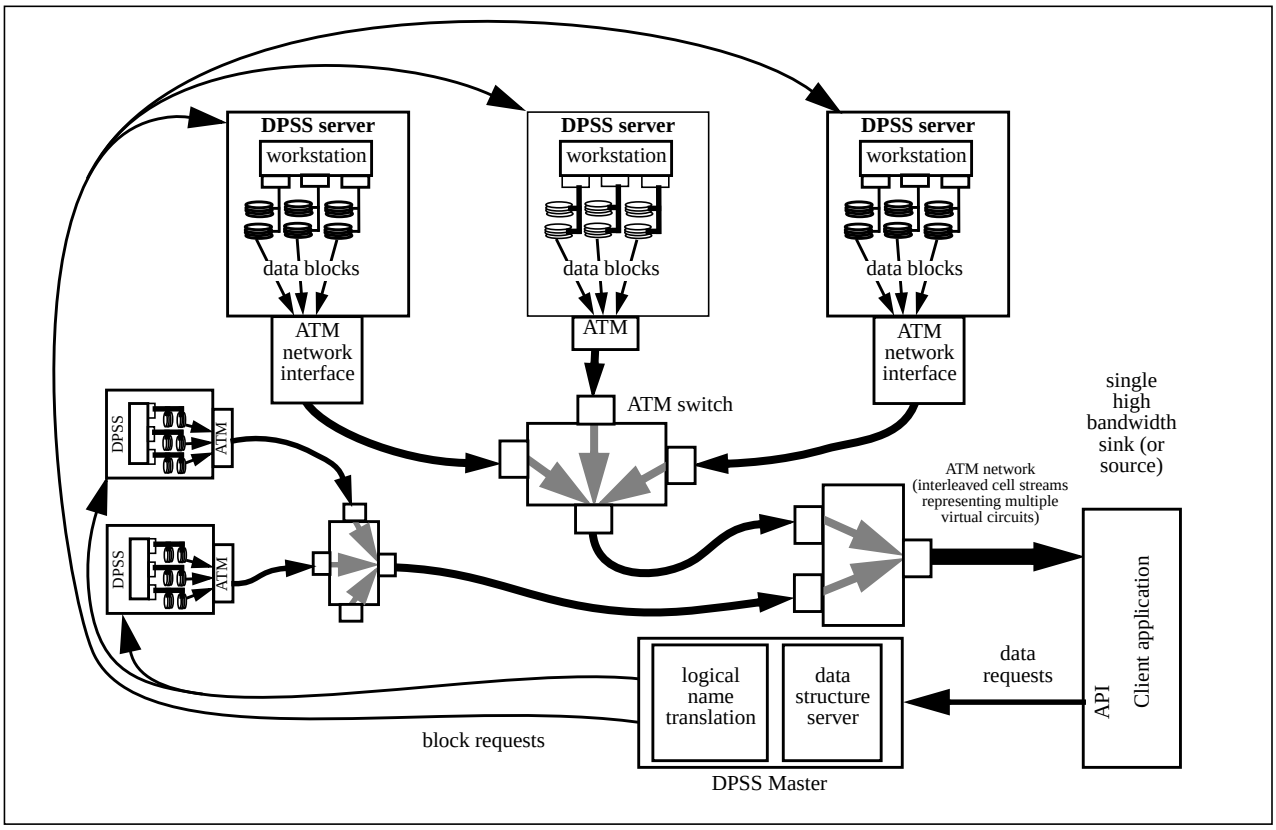

Figure 1 Distributed-Parallel Server System Architecture

assumed for the data blocks; the DPSS inherently provides random access. However, disk layout strategies that maximize parallelism are clearly desirable. The data organization is determined by the application as a function of data structures and access patterns, and is implemented during a data load process. When data structures and access patterns are well understood, specific placement algorithms can be designed to optimize data placement for maximum parallelism, as is the case with TerraVision (e.g. see

[4]). In other cases blocks can be scattered randomly across the disks and servers, a strategy that can work surprisingly well. The usual goal of the data organization is that data is declustered (dispersed in such a way that as many system elements as possible can operate simultaneously to satisfy a given request) across both disks and servers. This strategy allows a large collection of disks to seek in parallel, and all servers operate in parallel to send the requested data to the application, enabling the DPSS to perform as a high-speed data server.

At the present state of development and experience, the DPSS that we describe here is used primarily as a large, fast, wide area network distributed “cache”. Reliability with respect to data corruption is provided only by the usual OS and disk mechanisms, and data delivery reliability of the overall system is a function of user-level strategies of data replication and/or re-request and retransmission.

2.1.2 Client Use of the DPSS

The client-side (application) use of the DPSS is provided through a library-based API that handles initialization (for example, an “open” of a data set requires discovering all of the disk servers with which the application will have to communicate) and the basic block request / receive interface. It is the responsibility of the client (or, more typically, its agent - a data object structure server) to maintain information about higher-level organization of the data blocks; to maintain sufficient local buffering so that “smooth playout” requirements may be met locally; and to run predictor algorithms that will pre-request blocks so that application response time requirements can be met. The prediction algorithm enables pipelining of the operation of the disk servers with the goal of overcoming the inherent latency of the disks. (See [17] and [19]). None of this has to be explicitly visible to the user-level application, but some agent in the client environment must deal with these issues because the DPSS always operates on a best-effort basis: if it did not deliver a requested block in the expected time or order, it was because it was not possible to do so. In fact, a typical mode of operation is that pending block requests are flushed from the disk server read queues when the next set of requests arrive from the application. The application then routinely re-requests some fraction of the data. This deliberate “overloading” of the disk servers ensures that they will be kept busy looking for relevant blocks on disk and caching them in server memory. This behavior is one aspect of the pipelining strategy on the servers.

2.1.3 DPSS Implementation

In our prototype implementations, the typical DPSS consists of several (four - five) Unix workstations (e.g. Sun SPARCStation, DEC Alpha, SGI Indigo, etc.), each with several (four - six) fast-SCSI disks on multiple (two - three) SCSI host adapters. Each workstation is also equipped with an ATM network interface. A DPSS configuration such as this can deliver an aggregated data stream to an application at about 400 Mbits/s (50 Mbytes/s), using these relatively low-cost, “off the shelf” components, by exploiting the parallelism provided by approximately five disk servers, twenty disks, ten SCSI host adapters, and five network interfaces.

The software implementation is based on Unix interprocess communication mechanisms and a POSIX threads programming paradigm to manage resources on the disk servers (see [17] and [21]). The three primary operating systems (Sun’s Solaris, DEC’s OSF, and (soon) SGI’s IRIX) all have slightly different implementations of threads, but they are close enough that maintaining a single source is not too difficult.

The implementation supports a number of transport strategies, including TCP/IP, RTP/IP [15] and UDP/IP. RTP and UDP do not guarantee reliable data delivery and never retransmit. Lost data are handled at the application level. This approach is appropriate when data has an age determined value: data not received by a certain time is no longer useful, and therefore should not be retransmitted. This is the case for certain visualization scenarios. (This paper, however, focuses on TCP performance issues.)

Other papers describing the DPSS are [20], which focus on the major implementation issues, [17], which focuses on the architecture and approach, as well as optimization strategies, and [21], which focuses on DPSS applications and DPSS performance issues.

2.1.4 DPSS Timing Facility

A request for a data block takes the following path through the DPSS (see Figure 2). A request (a list of blocks) goes from the application to the name server, where the logical block names are translated to physical addresses (server: disk: disk offset), then the individual requests are forwarded to the appropriate disk servers. At the disk servers, the data is read from disk into local cache, and then sent to the application (which has connections to all the relevant servers). Timestamps are gathered before and after each major function, such as name translation, disk read, and network send. All timestamps are then logged by the DPSS servers so that the precise timing of each step is noted for each data block. The timestamps are also sent, with the data block, to the requesting application, where logging can be performed using the DPSS client library.

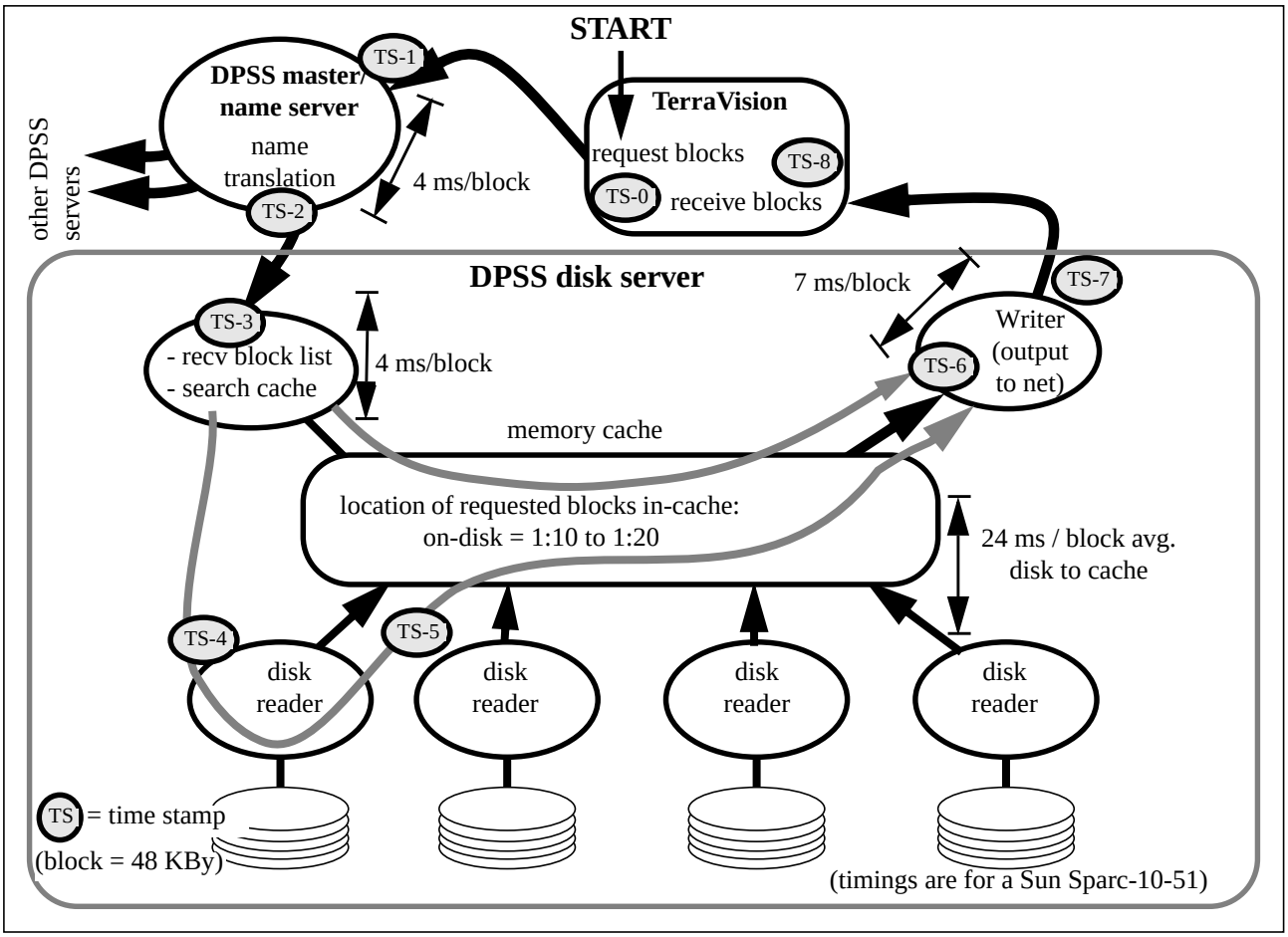

Figure 2 DPSS Performance Characterization Points and Optimal Average Timings

Timestamp consistency is provided by GPS-based NTP (described below), which allows us to make precise throughput and latency measurements throughout the DPSS system and underlying network. Instead of trying to analyze the aggregate delay between sending a request and receiving the associated data block (tile), we can pinpoint delays to within narrowly-specified steps in the data path.

2.1.5 Prototype Clients: TerraVision and tv_sim

TerraVision uses the DPSS client library's logging facilities to log events associated with an application session. It also uses the same log format to monitor a data block's progress from the network into the graphics pipeline. We have developed a simulator program, *tv_sim*, that can (among other things) generate data requests and receive data blocks from the DPSS in a manner similar to TerraVision's. Using this program we can generate synthetic request patterns, or repeatedly use actual TerraVision session tile request traces, and attempt to verify and analyze performance bottlenecks in the DPSS, the application, or in the network in a controlled environment. TerraVision is a complex software suite running on complex hardware, and patterns of requested data are complex. *tv_sim* can emulate the TerraVision data request patterns through the trace-driven operation facility, but is a "null" application that can be run at much higher overall request rates than real applications.

tv_sim consists of separate sender and receiver processes. The sender process periodically makes data requests of the DPSS, while the receiver reads the data and maintains a running summary of throughput. The receiver discards the data so as to concentrate on reading data as fast as the DPSS disk server can provide it (there is a receiver process per DPSS disk server). When *tv_sim* has completed, the sender and each receiver record a history of the data blocks through all components of the system.

The *tv_sim* data request sending rate, in terms of block lists per second and blocks per list, can be set by the user, as can the saving of history logs in the DPSS standard format. The sender can also use trace / playback files of TerraVision sessions instead of generating its own lists of block requests, as mentioned above. Additionally, the user can specify the use of multiple data sets, overall running time, and other runtime characteristics. *tv_sim* and the DPSS thus can be configured to impose almost arbitrary load patterns on a network and to record the results.

2.2 OS and Network Layer Monitoring

To complement the monitoring at the application level and in the DPSS, we also monitor various operating system and network conditions. We currently collect and log the following types of information:

- TCP retransmits
- CPU usage (user and system)
- CPU interrupts
- AAL 5 information
- ATM switch buffer overflows
- ATM hosts adapter buffer overflow

3.0 Monitoring Tools

3.1 Common logging format

To easily process the several gigabytes of log files which can be generated from this type of logging, all events are logged using a common format:

keyword; hostname; seconds; nano-sec; data; data; data;.....;

The logging format is a semi-colon separated list of fields. The "keyword" is a unique identifier describing what is being logged. By convention, the first part of the keyword is a reference to the program that is doing the logging (e.g.: DPSS_SERV_IN, VMSTAT_SYS_CALLS, NETSTAT_RETRANSSEGS, TV_REQ_TILE). Each log line contains both the hostname of the system on which the event occurred and a timestamp. The

timestamp is modeled after the format returned from the Unix “gettimeofday” call. However, since some systems return seconds and microseconds while others return seconds and nanoseconds, we have chosen to use nanoseconds for all logging.

The end of every logging record can contain any number of “data” elements. These can be used to store any information about the logged event that may later prove useful. For example, the NETSTAT_RETRANSSEGS event records one data element, the number of TCP retransmits since the previous event; and the DPSS_START_WRITE event records the logical block name, the data set ID, a “user session” ID, and an internal DPSS block counter. The log records are “associated” by virtue of being collected and carried in the data block request message as it works its way through the system, and the request message is attached to the returned data block.

3.2 Log File Analysis Tools

Tools to analyze log files include perl scripts[2] to extract information from log files and write data files in a format suitable for using gnuplot[3] to graph the results. These tools were used to generate the graphs in Figure 5 - Figure 8. Gnuplot includes various device drivers, and when using the driver for FrameMaker MIF files, gnuplot groups the graphics primitives that result from plotting data from one file, and at the next level down, each associated set of line segments, into graphic objects and sub-objects. Therefore, each of the log file elements, such as block histories, flushed block histories, TCP retransmits, etc., are organized as objects, and the individual block life-lines are kept as sub-objects within these larger objects. The FrameMaker graphics tool can manipulate these objects independently, and this proved invaluable in isolating and marking significant events. This sort of interactive active data analysis is very useful, and enabled the type of analysis illustrated in Figure 9.

3.3 Use of NTP

To be able to perform meaningful analysis of a network-based system using timestamps, the clocks of all systems involved must be synchronized. For example, the DPSS name server, DPSS disk server, and application are typically on different physical hosts scattered over the network. All MAGIC hosts run the ‘xntpd’ program [12], which synchronizes the clocks of each host both to time servers and to each other. The MAGIC backbone segments are used to distribute NTP data, allowing us to synchronize the clocks of all hosts to within about 250 microseconds of each other. The location of the NTP servers in the MAGIC network are shown in Figure 7 (below).

3.4 OS and Network Analysis Tools

Access to operating system source code for Sun’s Solaris 2.4 allowed us to modify the existing tools netstat and vmstat to do logging. Netstat displays the contents of various network-related data structures, while vmstat reports statistics of, among other things, virtual memory, disk, and CPU activity. Both programs were modified to present only a relevant subset of their information in the common logging format. In addition, netstat was modified to poll and report continuously (it normally provides only a snapshot of current activity). Currently we poll at 100 ms intervals, and since the kernel events are not timestamped, the data obtained this way represents all events in this interval, a circumstance that is represented by bars at the TCP retransmit points in Figure 5 - Figure 9.)

2. For more information see: http://www.metronet.com/perlinfo/perl5.html

3. For more information see: http://www.cs.dartmouth.edu/gnuplot_info.html

Also, most network devices (e.g., switches and host interfaces) allow for SNMP queries, which give information on several aspects of the network device. In the case of ATM switches, for example, we have used this capability to monitor cell loss and hardware buffer overflows.

4.0 Analysis and Results

This section contains a snapshot of the state of our performance measurements during late 1995. As will be illustrated, there are several aspects of the overall system that affect performance. One of the aspects that is changing quickly is workstation ATM interfaces, and the numbers quoted here are primarily intended to indicate trends rather than analysis of specific products. For example, over the past two years the throughput of a Fore Systems SBA-200 interface card operating in a Sun SS-20 has gone from 55Mbits/second to 105 Mbits/seconds, due to upgrades in vendor-supplied software (both OS and driver upgrades). It is therefore likely that specific numbers like these will have changed by the time this paper is published.

The following sections describe performance results and analysis based on our monitoring and logging methodology as applied to the DPSS, TerraVision, and related simulators operating together in ATM LANs and the MAGIC WAN. Throughput and latency results of many software and hardware components are given, and some of the resulting performance issues are discussed.

4.1 Network Tuning

Experiments using our monitoring approach have been run in several network environments, including a local FDDI ring, a local ATM network, the BAGNet metropolitan area ATM network [1], and the MAGIC network [2]. With the faster networks such as ATM OC-3, system default values for several network parameters do not provide expected performance. The MTU size, TCP window size, and TCP write buffer size are several of the parameters that need to be tuned to optimize performance in an ATM network. These issues have been investigated by our colleagues at the University of Kansas and Minnesota Supercomputer Center, and are described in detail in [3], [6] and [7]. The pacing of ATM cells (bandwidth limiting) out of fast workstations such as the DEC 3000/600 is also very important, and is described in detail in [6].

The cited TCP performance work indicates that for maximum throughput on an ATM LAN, applications should use 64 Kbyte TCP windows, use 9180 byte MTU, and send data in write buffers of 64 Kbytes. For maximum throughput on an ATM WAN (with round trip latency of 12 ms), applications should use 128 Kbyte TCP windows, use 9180 byte MTU, and send data in write buffers of 64 Kbytes. Note that setting the TCP window too large as well as too small will cause the throughput to drop.

4.2 DPSS Performance

4.2.1 LAN Experiment Environment

We first describe experiments done on an ATM LAN. Table 1 shows the disk server platform configurations that were used for these experiments:

Table 1 Platform Characteristics for the LAN, DPSS Server Configuration

System	Host	OS version	ATM card	ATM software	# disks
sender 1	Sun SS10-42	Solaris 2.4, patch level 27	Fore 140 Mb TAXI	2.3	4
sender 2	Sun SS20-62	Solaris 2.4, patch level 27	Efficient OC-3	3.35	6
sender 3	DEC 3000/600	OSF/1 V3.0	OTTO OC-3	2.2	4
receiver	SGI Challenge L	IRIX 5.3	Fore 140 Mb TAXI	3.0	-
ATM switch	ASX-200	3.0	rev A modules	3.0	-

4.2.2 Throughput Measurements

A DPSS disk server's network throughput is measured by comparing the timestamps taken just before the network "send" for the current block and previous block. The application receiver's network throughput is measured by comparing the timestamps taken upon completing a read of a block from the network with the previous block. Accurate measurement of peak or burst throughput is difficult because the timestamps are taken in user process space, not kernel space. The operating system processes network traffic and buffers user-level data independent of the application as long as it has sufficient buffer space and processing time. Thus when an application performs a read from the network it cannot know whether the data is coming from the network at that moment, or from the operating system's local buffers. The elimination of protocol processing and buffer-copying time from the time that the application believes it is spending in a read can lead to unrealistically high burst throughput results (e.g. higher than the physical medium can achieve).

To avoid this problem, we define burst throughput to be the throughput for five consecutive data blocks. In the throughput numbers in this paper, a block is 48 KB. This is a large enough amount of data to greatly reduce the effect of OS buffering and processing on the throughput times.

We have measured burst receive throughputs from 44 Mbits/sec (Sun SS10-41) to 134 Mbits/sec (DEC Alpha 3000/600) for a single DPSS disk server using TCP transport.

Some sample throughput results are given in Table 2, where average throughput is measured over several thousand blocks. UDP speeds in this table *are from the point of view of the sender, not the receiver.* The sending host may appear to send much more UDP data than the application receives, because some of it never makes it to the receiver (or through the OS to the application). For the Sun systems cited in the table, the primary throughput limitation is the memory bandwidth. These systems require three copies of the data to get from disk to network (one for disk to memory and two for memory to network). Newer versions of the Sun OS uses single copy TCP implementations, which increases the disk server through-

Table 2 Maximum Server to Application Throughput (per server)

DPSS Server Configuration	Burst (Mbits/sec)		Average (Mbits/sec)	
	TCP	UDP	TCP	UDP
Sun SS10-41; fddi	44	43	26	27
Sun SS10-42; fddi	62	56	37	40
Sun SS20-62; fddi	48	75	38	56
Sun SS20-62; atm	68	73	47	50
DEC Alpha; atm	134	-	61	90

put by 25%. There are typically enough disks operating in parallel that, while they contribute significantly to the latency, they do not represent a throughput bottleneck. See [18].

4.2.3 Latency Measurements

The timestamping facility allows for the measurement of the latency of each component of the DPSS. We have found that several of the latencies are relatively fixed regardless of the DPSS load. For example, the latency for the name translation is always around two milliseconds, and the time to read a block from disk is 10 to 30 milliseconds, with an average of 23 milliseconds. The send time from the DPSS name server to a DPSS disk server is six milliseconds, and the send time (time to copy a block from user space on the disk server to user space on the receiving host, not including network delay) is 23 milliseconds.

Other latency values vary greatly depending on DPSS load. These are the time spent waiting for the disk ("read queue") and the time waiting for the network interface ("send queue"). As expected, the average read queue latency decreases as more disks are added to the DPSS. Both read queue and send queue latency also decrease on systems with faster memory copy speed because the timestamping includes various memory-to-memory copies (all of which are in kernel space - the DPSS implementation does no user level copying). Send queue latency is mainly dependent on the speed of the receiving host. (TCP writes block and therefore synchronize, modulo kernel buffer sizes, with the reading of data on the receiver.)

We have measured minimum total DPSS system latency (defined to be the case when the read queue and send queue latencies are zero) to be between 50 and 100 ms. The maximum latency is the request interval plus the time to send the block across the network. (This is because the DPSS discards, or flushes, any outstanding requests whenever a new request arrives. For example, the TerraVision application sends a list of block requests every 200 ms. If a requested data block doesn't arrive in 200 ms, TerraVision determines if it still needs the data, and if so, re-requests it.) The maximum latency in this case is about 220 ms (200 ms between requests to the DPSS plus the time to send a single block). Therefore on a fully loaded DPSS, the average total latency is approximately the average of the minimum and maximum latency, or 160 ms. This number has been verified experimentally (see Figure 3). This latency, then, is the "length" of the data pipeline, and establishes how much prediction and buffering must be done by the client in order to maintain good interactive response.

4.3 End-to-End Performance Experiments

Experiments have been performed to examine the detailed interaction between a DPSS, whose disk servers are distributed over both ATM LANs and a wide-area ATM network, and the TerraVision application. Our initial monitoring experiments have focused on two issues important to high-performance, highly distributed applications such as the TerraVision <=> DPSS combination.

The first area of interest is the general interaction of the DPSS as a high performance distributed application in an ATM WAN.

The second area of interest is that of striping data streams across multiple network interfaces on the application host. In particular, we wanted to analyze how well the Onyx received data from multiple ATM streams, and determine how well the Fore Systems ASX-200 ATM switch and the Onyx handled the multiplexing and demultiplexing of multiple ATM circuits (TCP streams) on a single circuit.

By way of background (since all of the experiments reported here are based on block request traces from the TerraVision application), the TerraVision strategy is always to request data that it needs, whether or not the data has been requested previously. All that the application "knows" is what data it currently needs: The application does not care that the data might have been previously requested, nor why it might not have arrived. The overall philosophy is that the DPSS always does its best to deliver requested data, and if it did not do so, then it could not. This point of view maps very well onto an unreliable datagram

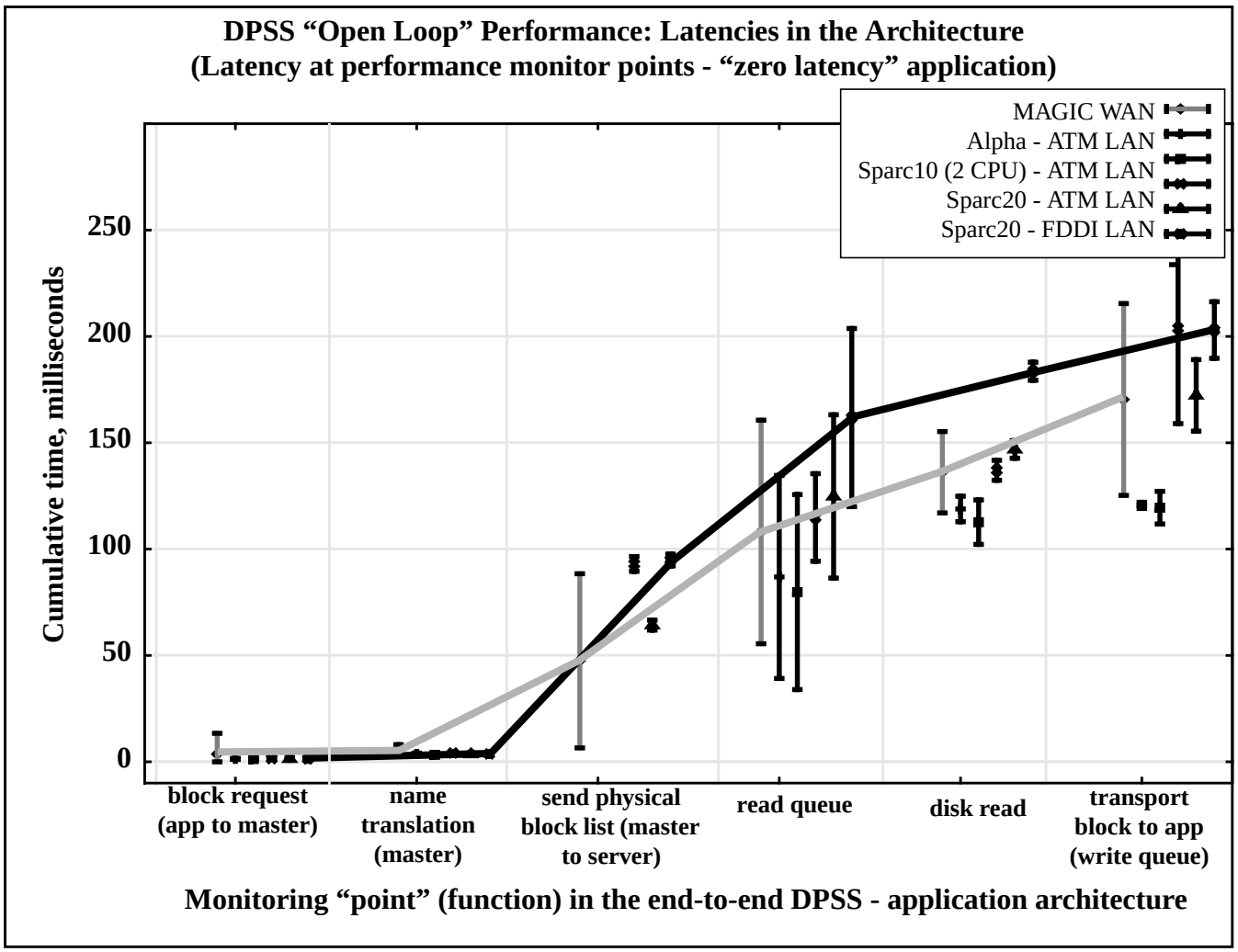

Figure 3 Experimental Determination of Overall System Latency

(The data points for each of the several experiments - characterized in the legend in the upper right corner by the platform type of the disk server, except for the “MAGIC WAN” which is an average over several servers - are displaced horizontally within the “monitoring point” labels to make the graph legible - this horizontal shift has no other significance. The “error bars” indicate variations in the results over several experiment runs. The solid line picks out one experiment in a LAN environment, and the gray line picks out the MAGIC WAN experiment. The cumulative time is a measured quantity based on timestamp information rather than a cumulative sum.)

protocol like RTP: the application can manage the retransmission problem in a way that is optimal to what the application is currently doing. In TerraVision, for example, the rendered view of the landscape may have gone beyond the point in the scene that the previously requested data represented, in which case there is no point in re-requesting (or, in the case of TCP, retransmitting) that data as it is no longer of any value.

The DPSS disk server semantics are that any time a new list of data block requests come in, all pending requests are discarded (on a per-user and per-data set basis). For the highest performance, and for the DPSS to operate at maximum efficiency, the application needs to predict ahead to ensure it requests more data than the DPSS can supply. Even if the DPSS cannot send all the requested data to the application, it is possible that the data was at least read from disk into the DPSS memory cache, where it will remain available for faster retrieval (for a short time). This approach ensures that the data pipeline stays full, and that disk server resources are never idle.

The traffic rates into and out of the name translation server (“master”) are very low, and while latencies in this service would have a significant impact on overall performance, this has not been an observable problem. (This is indicated in the figures showing block trace data where the difference between *app_send* and *server_in* monitor points represent the total latency to get the data requests out of the application, through the network, translated, and received by the server(s).) The fact that in our experiments most requests appear to be flushed at the *end_read* point (which actually indicates that the block is in the send queue) rather than at *start_read* point (in the read queue) indicates that the network and/or application host is more of a bottleneck than the disks in the test configurations. If the disks were the bottleneck, then there would be more lines ending at *start_read*, because that is where the requests would be stalled when the next request arrives, and therefore where the “life-line” is terminated by a flush. Note that in all the performance trace graphs that there are examples of the requested data being found in the server cache. (The nearly vertical life-lines on the left of the request groups show near zero read queue residency times.)

Having made the case that this sort of application is probably best optimized by relegating the retransmission decisions to the application, we nevertheless focus on the analysis of TCP transport, and all of the detailed analysis in this paper is done for performance experiments that use TCP. The reason for this is two fold: One, we wish to investigate the operation of TCP in the ATM network environment, and within a reasonable set of operational parameters, TCP (as will be shown) works well for this application. The second reason is that in order for RTP (or UDP) to provide a realistic (i.e., mixed application) high performance transport, in addition to the application making the decision to retransmit data, most of the TCP congestion control and response algorithms (e.g., slow start, congestion avoidance, and fast recovery) would also have to be implemented at the user level, and we have not yet done this.

4.3.1 LAN Experiments

The environment for the LAN experiment results shown below is described in section 4.2.1 above.

Figure 5 - Figure 6 illustrate top-to-bottom, end-to-end experiment results in a LAN environment. Each color or line style in the graphs indicates data from a different DPSS disk server, and different colors are also used for “flushed” data requests. (The reason that block requests may be flushed is described below.) The graphs used to illustrate the results of the experiments plot “real time” on the horizontal axis, and the monitoring points (see Figure 2) where the timestamps are recorded. As noted earlier, most of the timestamps represent critical points in the data request-response process from application to distributed storage system and back.

TerraVision sends a list of data requests every 200 ms, as shown by the nearly vertical lines starting at the *app_send* monitor points. (Each line - a “life-line” - represents the history of a data block as it moves through the end-to-end path.) The initial single life-lines fan out at the *server_in* monitor point as the request lists are resolved into requests for individual data blocks. Each block request is first represented individually in the read queue. When two life-lines cross in the area between *start_read* and *end_read*, this indicates that a read from one disk was faster that a read from another disk. (This phenomenon is clearly illustrated for the server represented by the crossing solid lines in Figure 4 at (A).) This faster read might be from disks with faster seek and read times (which is not the case in the experiment represented in Figure 4, as all participating systems used identical disks) or it might be due to two blocks being adjacent on disk so that no seek is required. (The disk layout algorithm [4] for the tiled image data used by TerraVision places data so that certain kinds of requests will find data blocks adjacent on disk.)

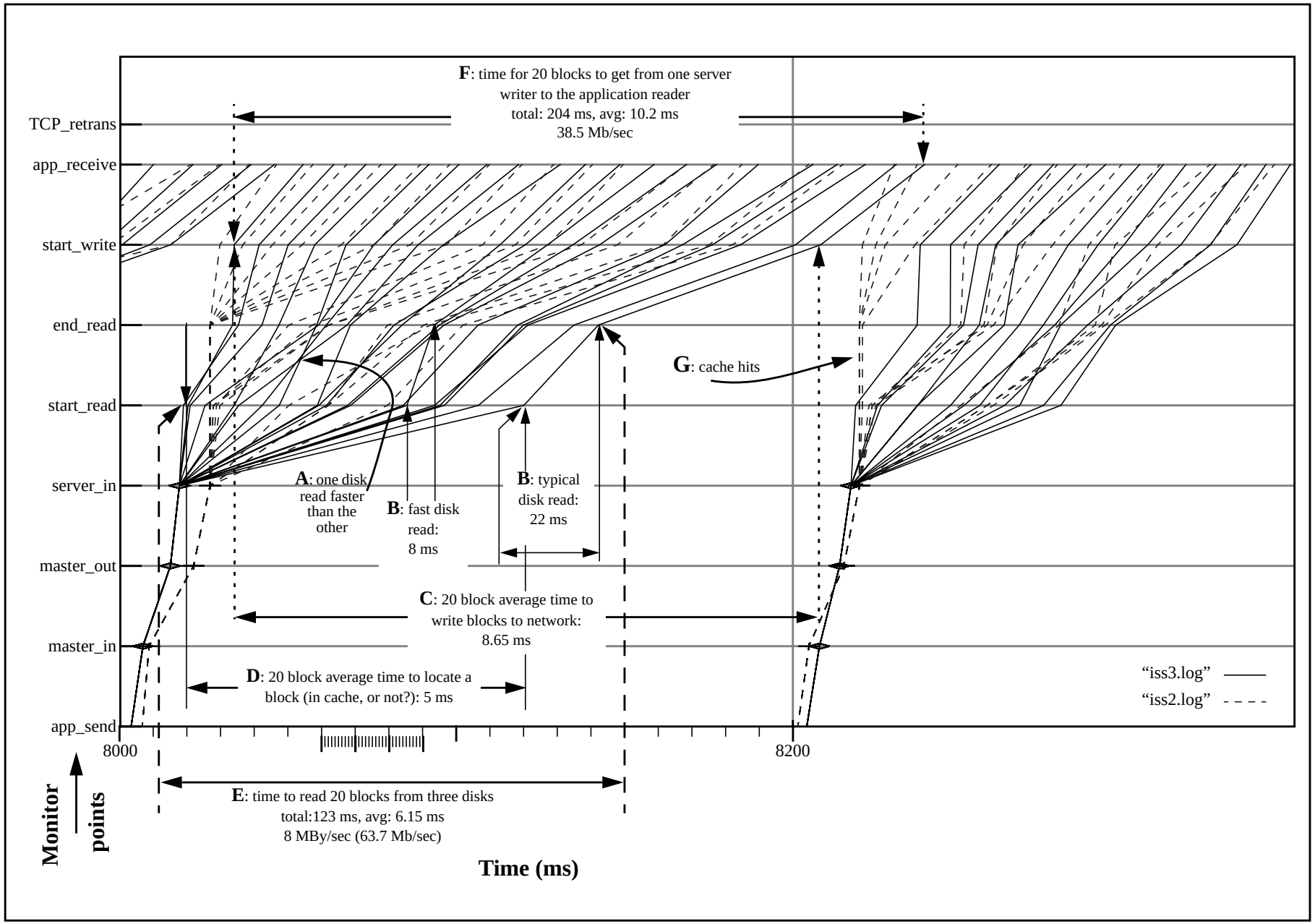

Figure 4 Detail From a Two Server, LAN, Experiment

Using these life-line graphs it is also possible to get fairly detailed information on individual operations within the disk servers. Such detailed performance analysis is illustrated in Figure 4, for example, and shows us:

- at “B” two different characteristic disk reads (one with an 8 ms read time and one with a 22 ms read time);
- at “C” the time to cache a block and enter it into the network write queue is about 8.6 ms;
- at “D” the time to parse the incoming request list and see if the block is in the memory cache is about 5 ms;
- at “E” the overall (four disks operating in parallel) server data read rate is about 8 MB/sec;
- at “F” the actual throughput for this server while dealing with real data requests is about 39 Mb/s (this throughput is receiver limited, and the “unconstrained” throughput is about three times that number - as given in Table 2);
- at “G”, there are two cache hits (blocks found in memory) as a result from previously requested, but not sent, data being requested.

Most of these numbers agree quite closely with the values given in Figure 2, which were estimated from individual software and hardware timings long before the DPSS actually operated as a system.

Considering the single-server LAN experiment illustrated in Figure 5, notice that many life-lines terminate at *end_read*, and that a few also end at *start_read*. Any individual data request that is not satisfied by the disk server before the next request list arrives is flushed (discarded) from all the queues, but the data is retained in the memory cache. For example, in Figure 5 the life-lines that started at 10,400 ms that were terminated at (B) did so because the TCP write delay (of unknown cause) at (C) “trapped” those blocks in the TCP write buffer. Block requests that were in the DPSS write queue when the next request list arrived (at 10,600 ms) are flushed from the queue. However, some of these blocks were re-requested in the 10,600 ms list, and these re-requests are satisfied very quickly because the data is in the disk server memory cache. This is seen in the nearly vertical life-lines to the right of (A).

Figure 6 illustrates “correct” operation of multiple servers. This LAN-based two-server experiment shows the interaction of life-lines for blocks from different servers, and a case where the independent servers are behaving almost “perfectly”: There are very regular block delivery patterns that alternate almost one-for-one between servers.

4.3.2 WAN Experiment Environment

End-to-end performance experiments and monitoring scenarios use data block request traces from the *TerraVision* application and *tv_sim*. The traces for the application running in the MAGIC WAN environment were obtained with TerraVision running on an SGI Onyx with eight 150 MHz MIPS R4400 processors, 256 MB of main memory (4-way interleaved), two RealityEngineII graphics processors, and a single Fore Systems 100 Mb/s TAXI ATM interface. (This configuration is the minimum required to get good interactive visualization of 3D landscape.)

Experiments were run on the MAGIC ATM testbed, using the hardware shown in Table 3 and the configuration illustrated in Figure 7.

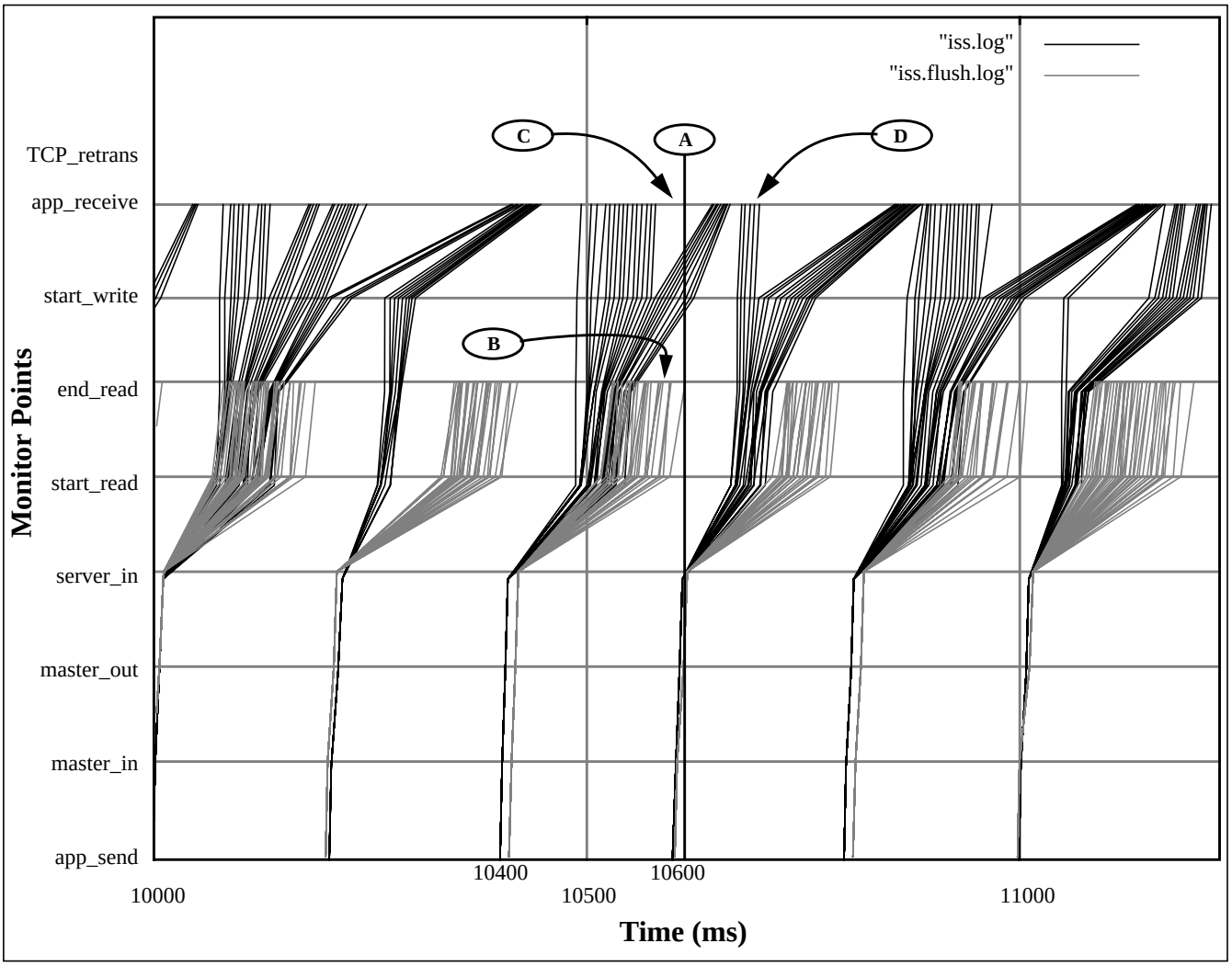

Figure 5 One Server Test (ATM LAN, one SS-20 as server, tv_sim on DEC 3000/600)

Table 3 Platform Characteristics for the WAN, DPSS Server Configuration

System	Host	OS version	ATM card	ATM software	# disks
sender 1 (EDC)	Sun SS10	Solaris 2.4, patch level 27	Fore 100 Mb TAXI	2.3	4
sender 2 (US West)	Sun SS10	Solaris 2.4, patch level 27	Fore 100 Mb TAXI	2.3	2
sender 1 (EDC)	Sun SS10	Solaris 2.4, patch level 27	Fore 100 Mb TAXI	2.3	4
receiver (TIOC)	SGI ONYX	IRIX 5.3	Fore 100 Mb TAXI	2.3	-
ATM switch at TIOC, EDC and MSC	ASX-200	3.0	rev A modules	3.0	-
ATM switch at KU	DEC AN-2				

A five-minute TerraVision session trace of data block requests was captured, and then using this list of block requests, *tv_sim* was used to repeatedly request and receive those blocks. Experiments were run using DPSS configurations of one disk server, two disk servers, and three disk servers configuration. (The number of disk servers is independent of the application data request strategy and transparent to the application, except for establishing the data transfer connections). Log files were collected in the various

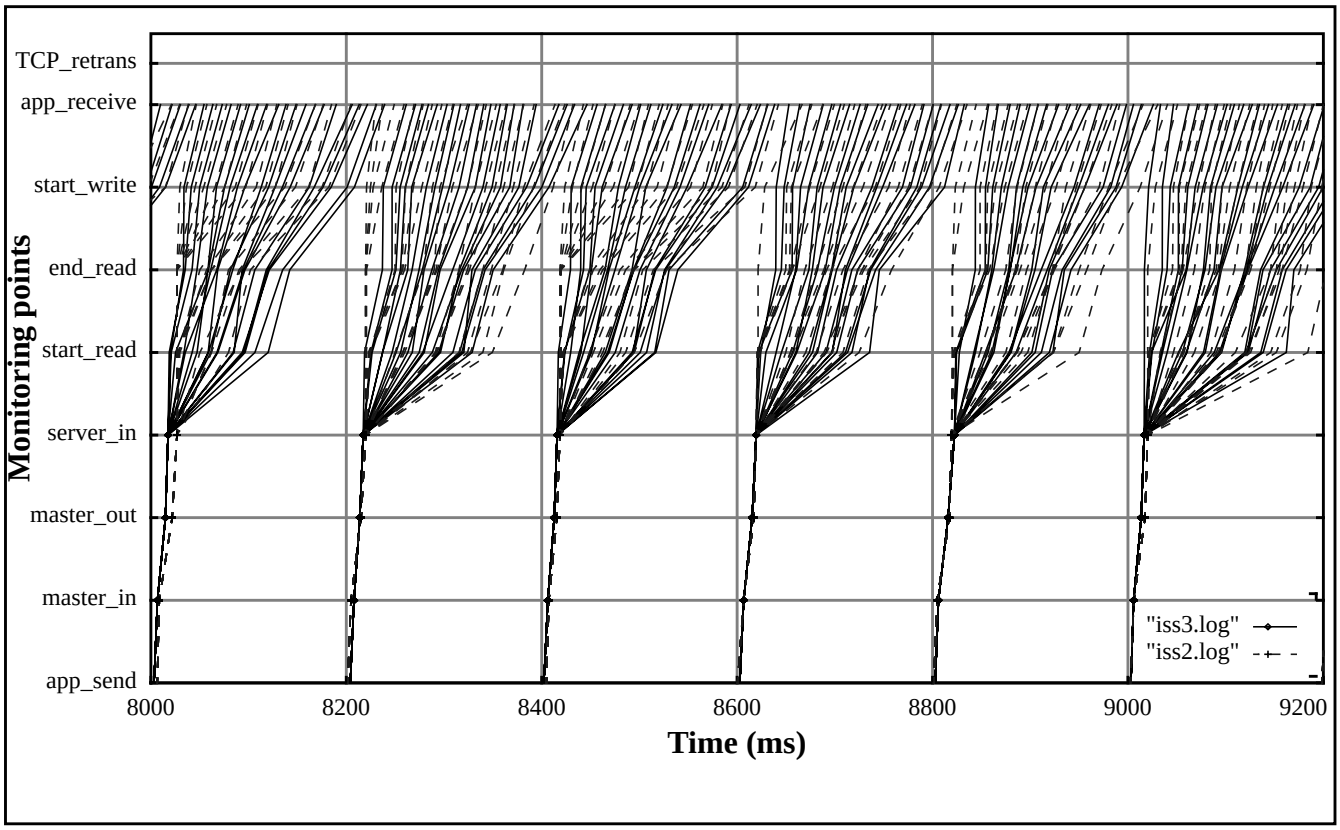

Figure 6 Two Server Test (ATM LAN, two SS-20s as servers, tv_sim on DEC 3000/600)

distributed components for satisfied and unsatisfied block requests, TCP retransmission information, CPU usage, and ATM cell loss in the host adapters and ATM switches.

4.3.3 WAN Experiments

Of particular interest is the experiment for the three-server configuration operating in the MAGIC WAN testbed (Figure 8). The graphs for the LAN experiments (Figure 5 - Figure 6) show mostly expected behavior: smooth operation, no unexpected latencies, no TCP retransmissions, and so on. However, in the WAN graph one sees TCP retransmissions and some extraordinarily long delays (up to 5500 ms).

First, let us analyze what the performance monitoring shows directly. In Figure 9 the various key features of the experiment are indicated. If we look at the long-delayed block life-lines (emphasized in this figure) we see the characteristic behavior of a data block getting into the write queue (*start_write* monitor point) and then incurring some very long delays getting to the application. These long delays are almost always accompanied by one or more TCP retransmit events. (The elongated blocks at the top of the figure indicate the interval during which the retransmit took place, and all of the graphical elements are shaded - black, gray, light gray - to distinguish the analysis and events separately for each of the three servers.) The start of the long delay transmissions are identified as A_b, B_b, or C_b (servers A, B, and C, in a blocked state). The reason that the server is blocked as a whole (actually just one application is blocked since each application has its own TCP connection to the disk server) is that once a block is written to the TCP socket, the user level flushes have no effect, and TCP will re-send the block until transmission is successful, even though the data is likely no longer needed and is holding up newer data. The server unblocks when the a retransmission is successful, letting the next write proceed. These unblock “events” are labeled A_u, B_u, and C_u.

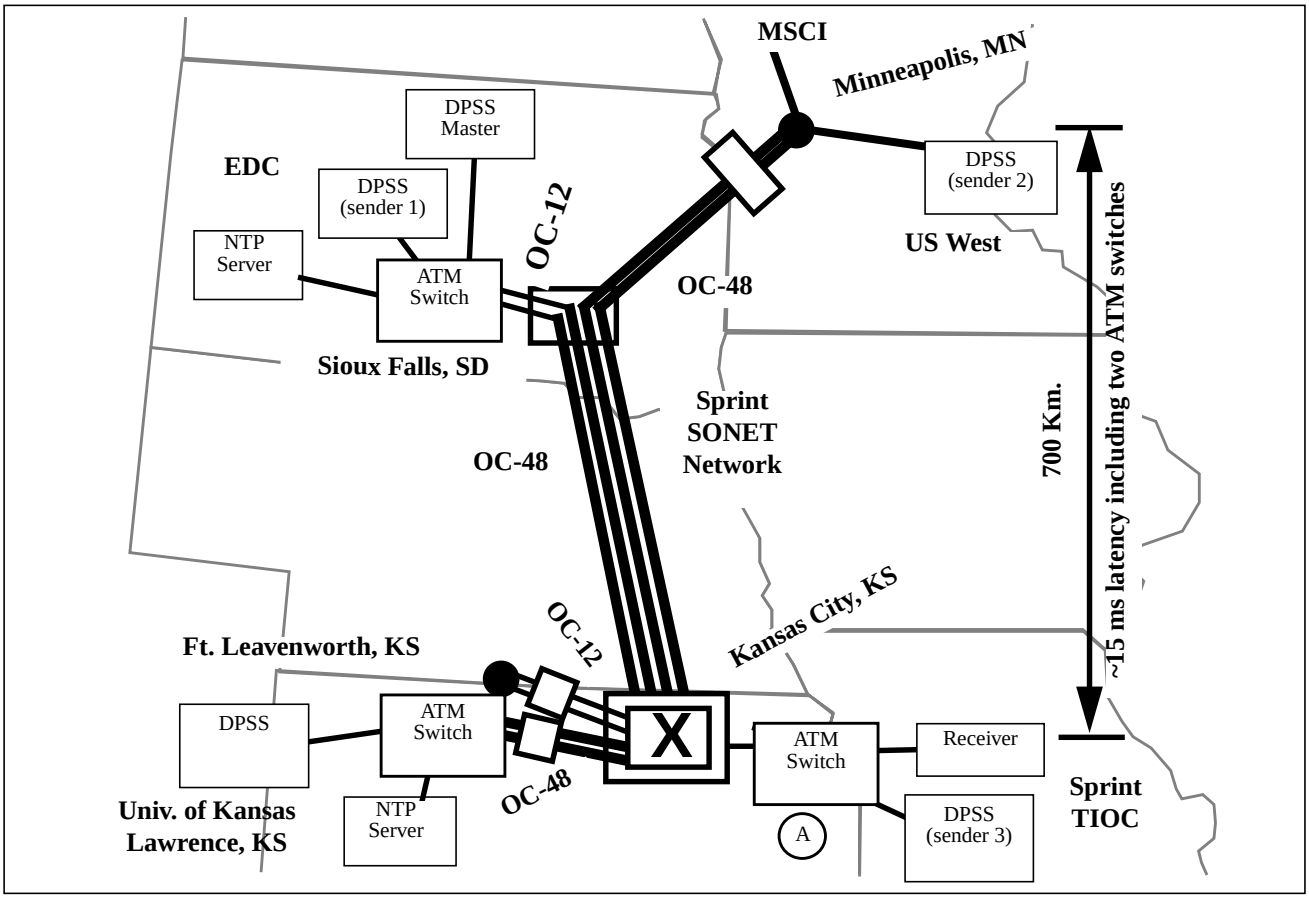

Figure 7 The MAGIC Network: Test Configuration

The impact of this behavior on the server as a supplier of data is shown at the bottom of Figure 10, where the horizontal bars indicate normal server operation by gray regions, and the blocked state by white regions. These server operation profiles show almost unbelievably poor performance: With three servers operating (EDC, USW, TIOC) we obtained throughput of 0.240 Mbits/sec, 3.8 Mb/s, and 4.4 Mb/s respectively, to deliver 106 blocks in 5 seconds out of about 900 requested blocks. This is in a network with minimum link speeds of 100 Mb/s and servers each with a verified data delivery capability of at least 45 Mb/s.

Beyond the poor performance, the operation profile clearly shows several places where there seems to be synchronization among the long-delay intervals. This synchronization probably indicates the cell-interleaved TCP streams problem reported by Romanow and Floyd [14]. If so, an implementation of the Romanow and Floyd, Early Packet Discard (EPD) algorithm in the switches might help. (At the time of the experiments reported here, EPD was not implemented in the MAGIC ATM switches, but should be by the Spring of 1996, and we will redo the experiments to see if EPD helps.). .

What we believe to be happening in this experiment is that TCP's normal ability to accommodate congestion is being defeated by an unreasonable network configuration. The reasoning is as follows.

The final ATM switch (Ⓐ in Figure 7) is where the three server streams come together, and this switch has a per port output buffer of only about 1500 bytes. The network MTU (minimum transmission unit) is 9180 Bytes (as is typical for ATM networks). The TCP congestion window cannot get smaller than the MTU, and therefore TCP's throttle-back strategy is pretty well defeated: On average, every retransmit fails, even at TCP's "lowest throughput" setting, because this smallest unit of data is still too large for the

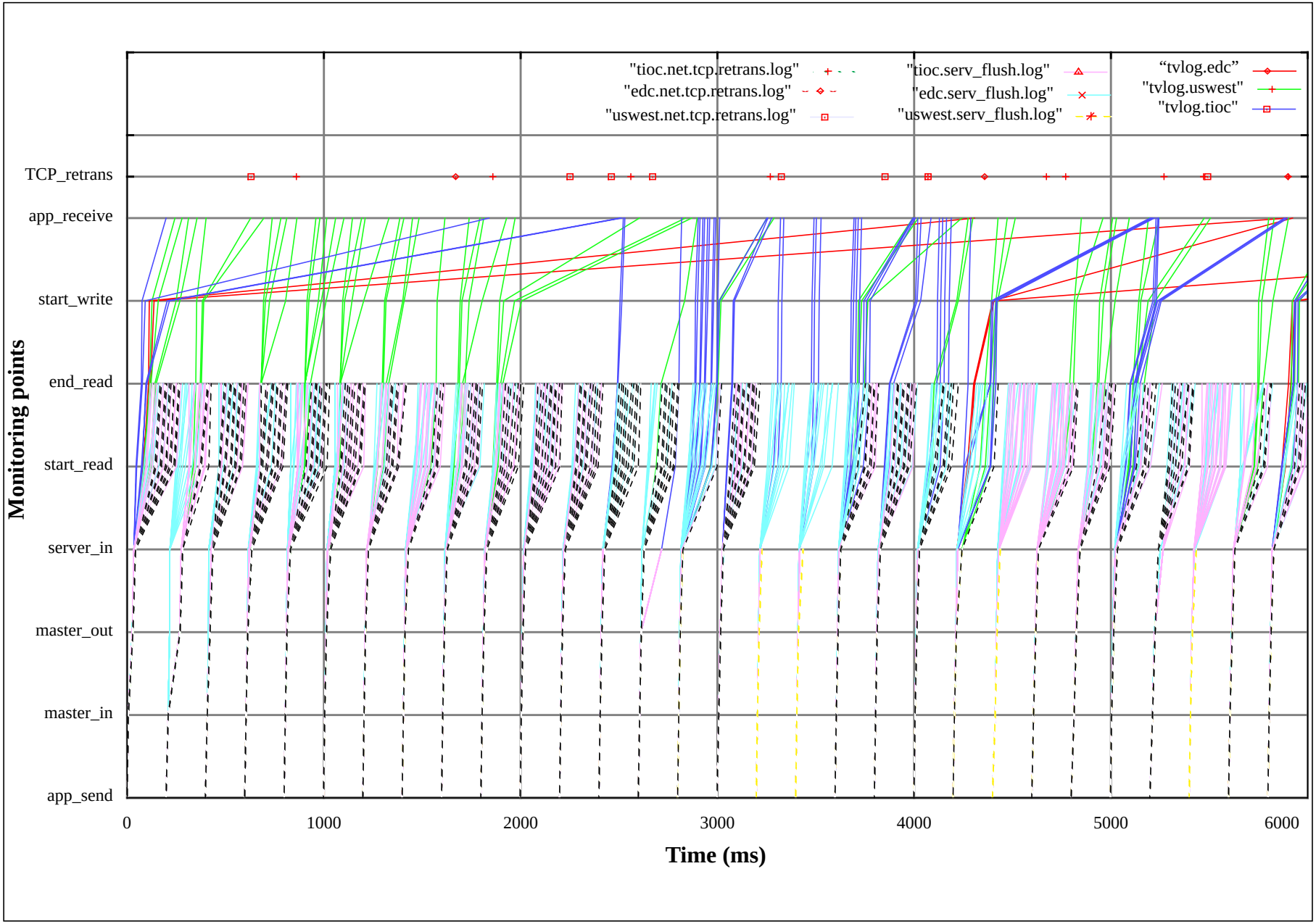

Figure 8 Three Server Test (MAGIC ATM WAN, three SS-10s as servers, tv_sim on SGI Onyx)

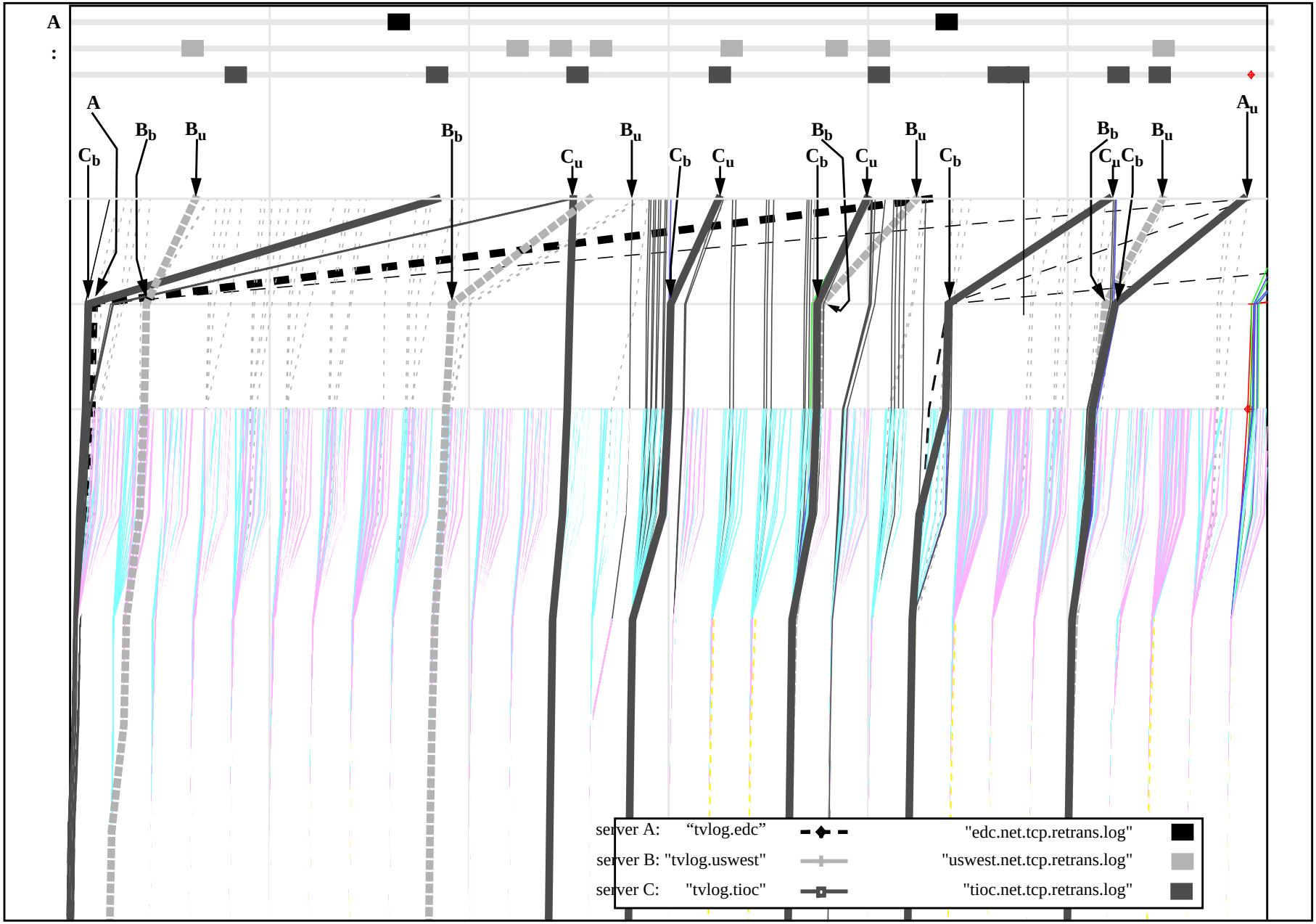

Figure 9 Detailed Analysis of a Three Server, ATM WAN Experiment

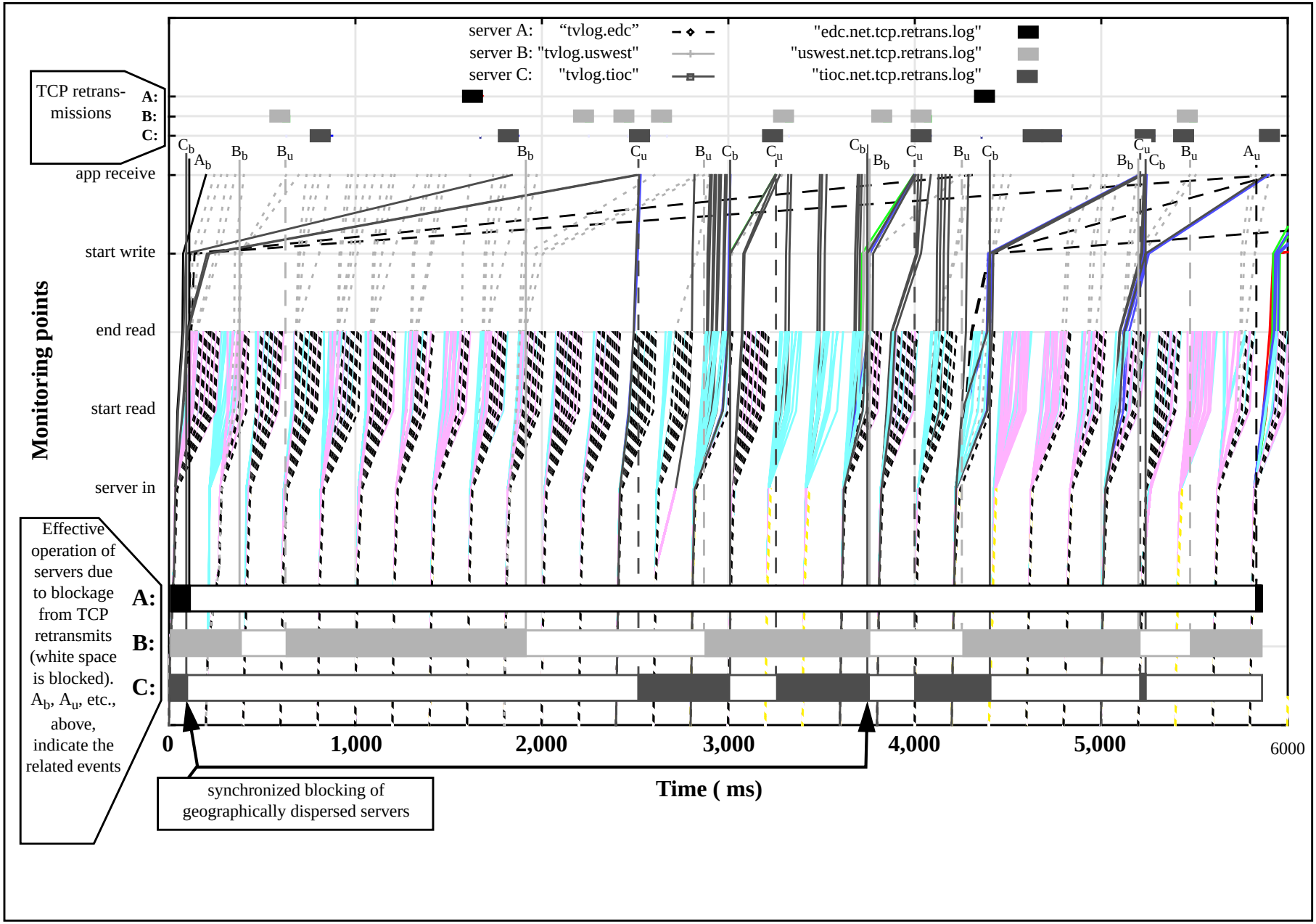

Figure 10 Analysis of a Three Server, ATM WAN Experiment

network buffers.) So, the situation is that three sets of 9 KBy IP packets are converging on a link with 10% that amount of buffering available, resulting in most of the packets (roughly 65%) being destroyed by cell loss at the switch output port.

4.3.4 Acceptable Solutions and Unacceptable Solutions

TCP's normal ability to adapt to congestion should be able to make better use of this network, even with the mis-configured switch. The problem is that TCP is being prevented from providing an effective response to this congestion because it cannot reduce the congestion window to a small enough size. (We understand that the Internet Research Task Force, End-to-End transport group is looking at this issue.)

The whole point of TCP is that it should optimize the use of available bandwidth, ideally distributing it uniformly across all data flows, but at least making best possible use of available bandwidth. In general it does a reasonable job of this, otherwise the Internet as we know it would not work.

TCP's adaptation mechanisms normally accommodate mapping a collection of high offered bandwidth streams (e.g. from the DPSS servers) into a low bandwidth path. The problem revealed in our experiments is that the TCP adaptation mechanisms make certain assumptions about the network that are not true for the MAGIC network as configured during our experiments.

The important assumption for the issue at hand is that the amount of output port buffering needs to be of the order of Number_of_streams × TCP_min_retransmission_size. In other words, there needs to be at least enough buffering to make the lowest-throughput TCP retry efforts successful.

This is the assumption that is violated in the current MAGIC configuration environment in which our prototype production experiment performed so badly. The minimum TCP retransmission size is the IP/ATM MTU size (512 bytes is a typical Internet value). If the output ports of a switch or router are not large enough to accommodate (at least several) minimum retransmission units, then it is clear that retransmissions, even at the lowest throughput that the TCP adaptation mechanism [8] can operate at, will fail, and TCP will therefore fail to provide optimal, or even reasonable, use of link capacity. (Recall that we obtained less than 10% of the link capacity in the experiment, with the rest of the capacity being "wasted" (unusable).) With the current switch configuration, the TCP congestion window size should probably be of the order of 256 Bytes in order to accommodate three streams.

Apart from re-engineering the network - which may or may not be possible in general - or fixing TCP (which we will do in MAGIC and rerun the experiments) what else can be done?

ATM Congestion Control:

There are various ATM cell-level congestion control schemes being developed by the ATM Forum Traffic Management Working Group that should help alleviate this problem. The ATM Forum Available Bit Rate (ABR) service specification includes a mechanism for switches and destinations hosts to send congestion and flow control information back to the source host congestion is detected. The first version of this should be available in UNI 4.0, which vendors will probably start shipping by the end of this year.

Small MTU Size:

We can reduce the network MTU to a very small value (e.g. 256 bytes). This should demonstrate the ability of TCP to deal with this type of congestion, and we expect that this approach will ameliorate the situation described above. (These experiments will be run in the near future, and we will post the results to http://www-itg.lbl.gov/DPSS/Experiments.) Experiments done at the University of Kansas, Lawrence [7] show that both large MTU size (illustrating the problem noted above) and small MTU size (preventing high performance operation of the servers) lead to low throughput. Figure 11 (data from the

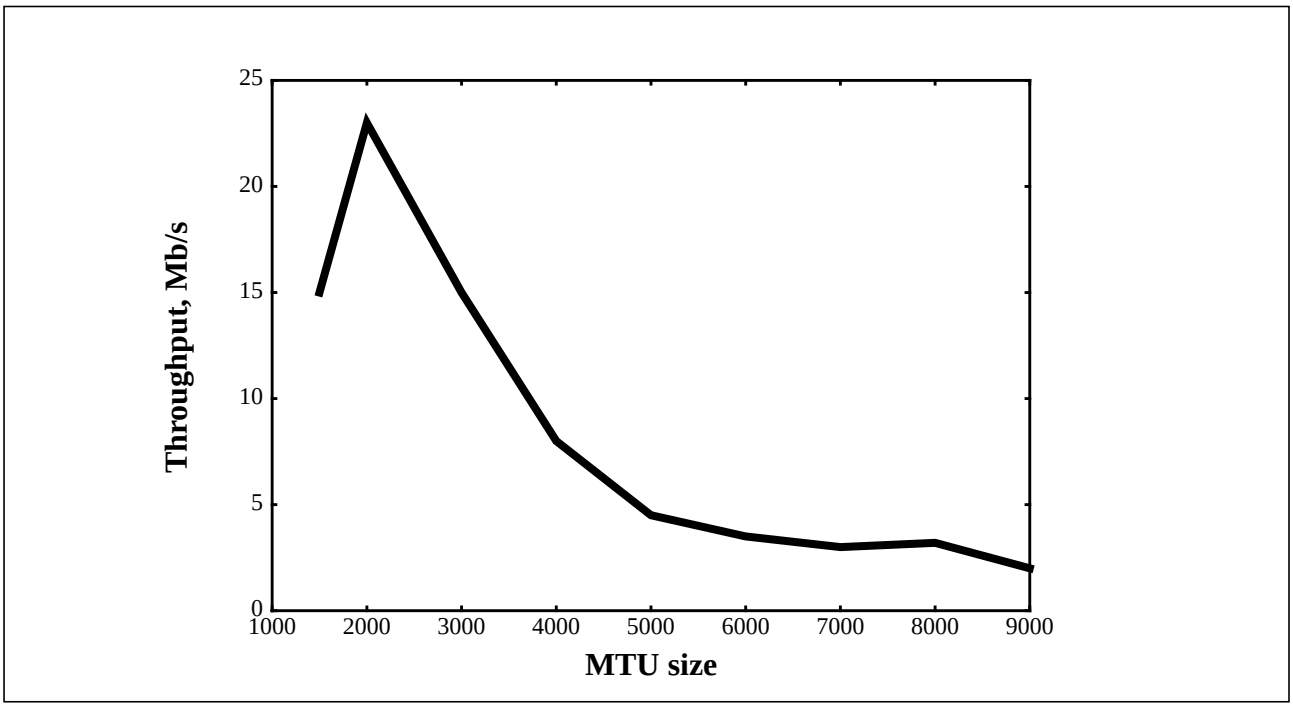

Figure 11 Single Stream Throughput vs. MTU Size for TCP Using Large Windows in an ATM WAN

KU experiments done in the MAGIC testbed) indicates the relationship between TCP throughput with varying MTU size for the kind of large windows (196KBy) used in a high bandwidth network. This approach, however, is clearly not an acceptable general solution because it prevents high performance operation of distributed applications everywhere in the network.

Cell Pacing:

We can also cell pace the disk servers to 1/N of the “broken link” bandwidth. Cell pacing is bandwidth limiting at the level of ATM cells. Different manufacturers implement this in different ways, but one common way is to set the link to CBR (continuous bit rate) at some reduced bandwidth (e.g. 30 Mbits/sec). This approach assures that cells arrive at the switches spread out over time, rather than in high speed bunches, as would be the case when bandwidth limiting is done at the IP level. (In that case the average cell output might be 30 Mb/s, but each IP packet would likely come in 155 Mb/s bursts.) The cell pacing experiment has been done: we have cell-paced all of the DPSS servers at 1/3 the final link capacity so that we know that the total offered load does not exceed the output link capacity of the final switch. Not surprisingly, this approach corrects all of the observed anomalies. Under these conditions, all three servers provided roughly equal throughput of 30 Mb/s, and the delivery of blocks was “well behaved”. (See Figure 12.)

However, from our point of view, this again is not an acceptable solution. It solves the problem by reducing everything to the lowest common denominator. It also assumes that you will know a priori how many streams will be coming a given link. As a “lowest common denominator” approach, this does not allow individual servers to use available bandwidth when, e.g., other servers are not transmitting because the data happens not to be evenly distributed; when a server or network link fails; etc.

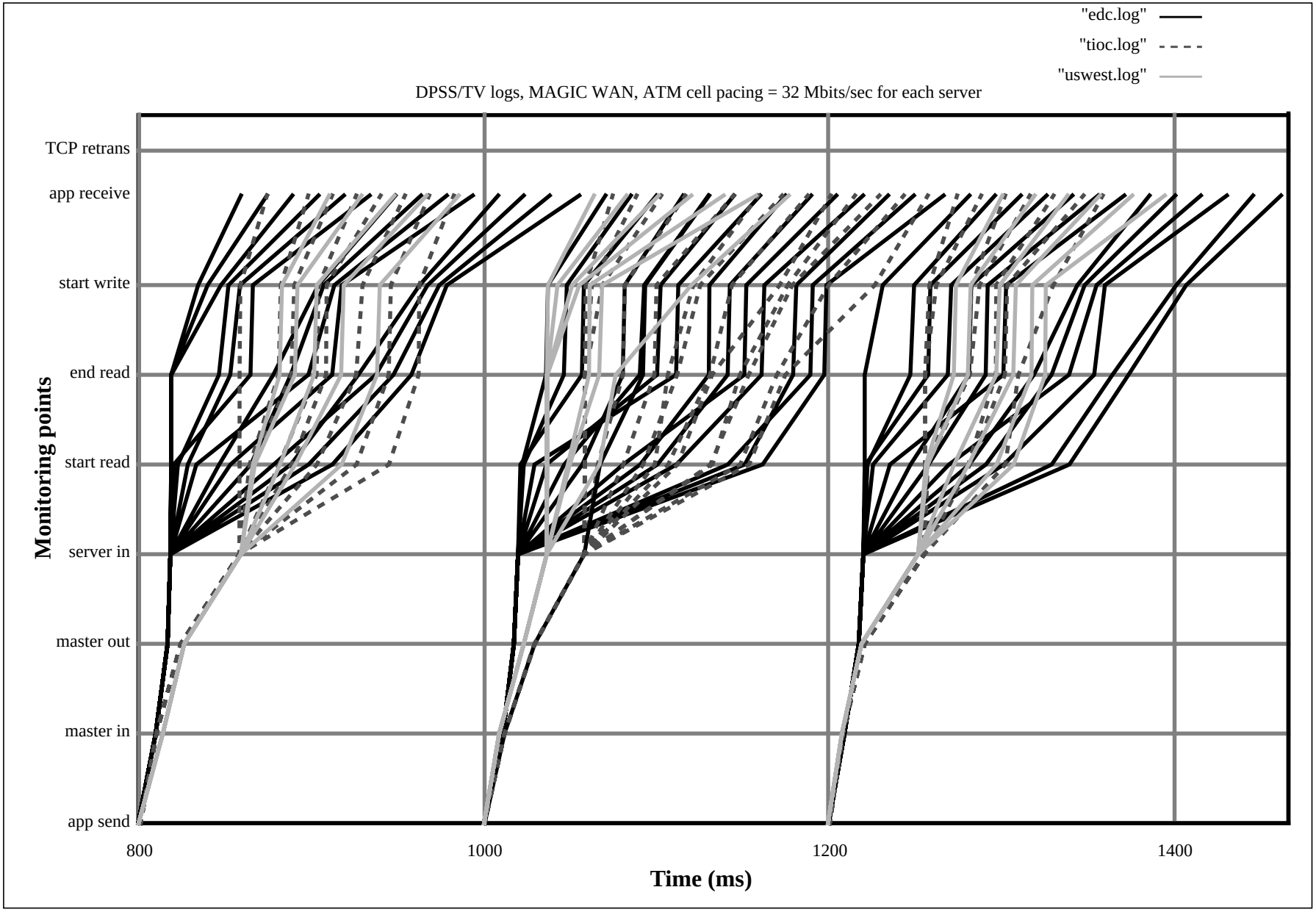

Figure 12 The Three Server ATM WAN Experiment with Cell Pacing

Re-engineering the Network:

The ATM LAN experiments reported above were done using a Fore Systems ATM switch with rev. C network modules. These modules (set of four output ports and line drivers) have large output port buffers, and this appears to solve the congestion problem that we have been describing for most situations. These new switch output modules have 624 KBytes of buffering, so assuming minimum TCP segment sizes of 9180 Bytes (the ATM MTU), the switch module should be able to support up to 69 simultaneous maximally throttled TCP connections.

In the case of the MAGIC testbed, we can and are, re-engineering the network. In this case, the small buffer switch in the final link will be upgraded to provide much larger output port buffering. Once this is done, we will run a range of experiments, including varying the switch output port buffer size, turning "early packet discard" off and on, etc. to verify that the various changes produce the expected results. Again, these experiments will be posted to the URL indicated above.

5.0 Computer Platform-Based Performance Issues

The second main issue that we have investigated in our quest for high performance distributed applications is the limits on getting data into and out of the computer systems attached to the networks. There are several issues in this regard.

The first issue is that the memory bandwidth of most workstations is relatively low compared to the processor speed, and even some I/O devices. Since DMA devices do not typically write directly into cache memory, I/O performance is limited by the transfer speed of the bulk memory sub-system. Early recognition of this fact lead us to design the DPSS server code as a collection of independently operating threads that only manipulate pointers - they never copy data in memory. This means that, to first order, the performance of a network disk system like the DPSS is dominated by the number of copies through memory needed to get data from disk to network. This is typically three: one for disk to memory and two for memory to network. We have run a simple memory bandwidth test that is designed to defeat the cache and measure bulk memory throughput, and for almost any platform we can then say with reasonable accuracy that the platform performance as a network disk server is 1/3 of the memory bandwidth.

Since we do not expect memory bandwidth (unlike processor speed) to improve rapidly, the key to better distributed system performance is in getting around the memory bandwidth limitations.

The easiest way to gain performance, then, is to eliminate copies. In the several years that we have worked on network disk systems, we have seen TCP implementations go from two copies to one copy in some production systems, and zero copy implementations discussed in the research literature. (See, for example, [5].)

Apart from memory bandwidth being a general limitation, ATM workstation interfaces and associated protocol stacks need to be highly optimized. This software did not start out that way, but progress is being made. As shown in Figure 13 for the one stream case, we have seen a 2 - 2.5 times increase in the throughput of workstation ATM interfaces in the past two years.

The second approach to gaining performance is to make use of parallelism in the platform hardware.

In addition to application, server, device driver, and OS development, the implementation of the DPSS involved some fairly careful evaluation and integration of hardware sub-systems. Our experience indicates that one seldom gets more than on the order of half the "raw" device performance once the

device is integrated into the system, and that “independent” sub-systems are independent only in (sometimes small) degrees.

However, in important and mature sub-systems like memory and disk, the “interference” between sub-systems is usually well managed in both a parallel access (e.g. memory) and parallel operations (e.g. disk) mode. This situation is illustrated by the memory sub-system performance that we have determined for several workstations (in terms of memory - not cache - throughput). (See Figure 14.) For all multiple CPU systems that we looked at, the performance of parallel CPU access to memory increases nearly linearly in the unloaded regime, and levels off and degrades very little in the saturated regime. For example, for the SGI we know that the memory throughput is just over 100 MBytes/sec, and that rate is achieved and sustained with four CPUs and greater than five processes accessing memory “simultaneously”. (Note that our measure of “throughput” is a copy - a read and a write - so is one-half of the value that manufacturers typically quote.)

However, our attempts to get similar parallelism by striping data streams across ATM network interfaces, especially for reading from the network, has not been nearly so successful. There has been some progress, and Figure 13 shows results for striping data across multiple ATM interfaces. For ATM network interfaces and device drivers, as the 1994 -> 1996 change illustrated in Figure 13 indicates, we are seeing improvements in “sub-system” performance, but the more complex issue of gracefully sharing platform hardware and system resources in saturated, parallel network device operation environments still requires considerable work.

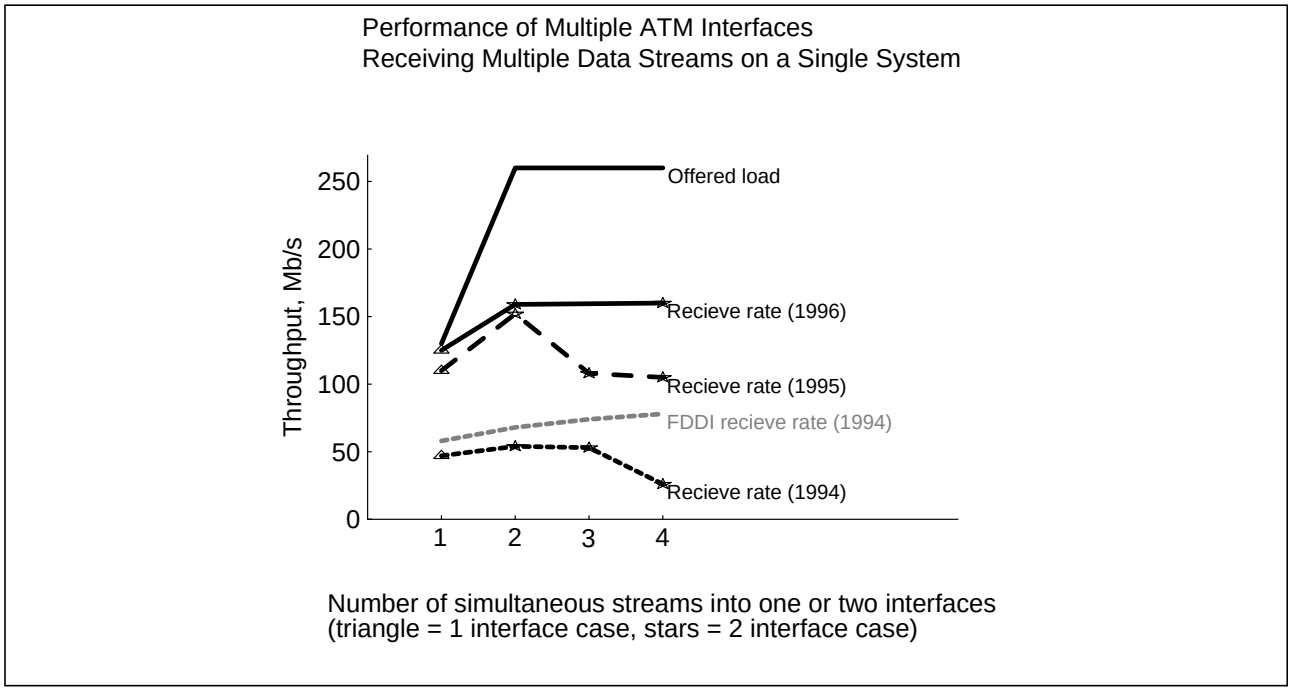

Figure 13 Striping Across ATM Interfaces: Multiple Input Data Streams on Multiple Network Adapters

(Multiple senders supplied data to a single system with two interfaces. The senders were always dissimilar systems - i.e., the sender and receiver were never from the same manufacturer.)

Figure 13 shows the limited gains obtained from sending multiple, simultaneous TCP streams to an SGI Onyx using two ATM network interfaces. (Note that this phenomenon is not unique to SGI and exists for most of the other platforms that we have tested, but the Onyx is the only platform capable of running the TerraVision application, so we have examined it in some detail.) In early tests the aggregate throughput

actually decreased for more than three simultaneous streams, and all indications were that either the OS or the ATM interface was dropping large numbers of cells. In recent tests, the overall throughput has increased substantially, but we still do not see even near linear speed ups when striping over interfaces. At the moment it is not clear why this is happening, and we are currently working with engineers from several workstation and ATM interface manufacturers to resolve this problem.

.

.

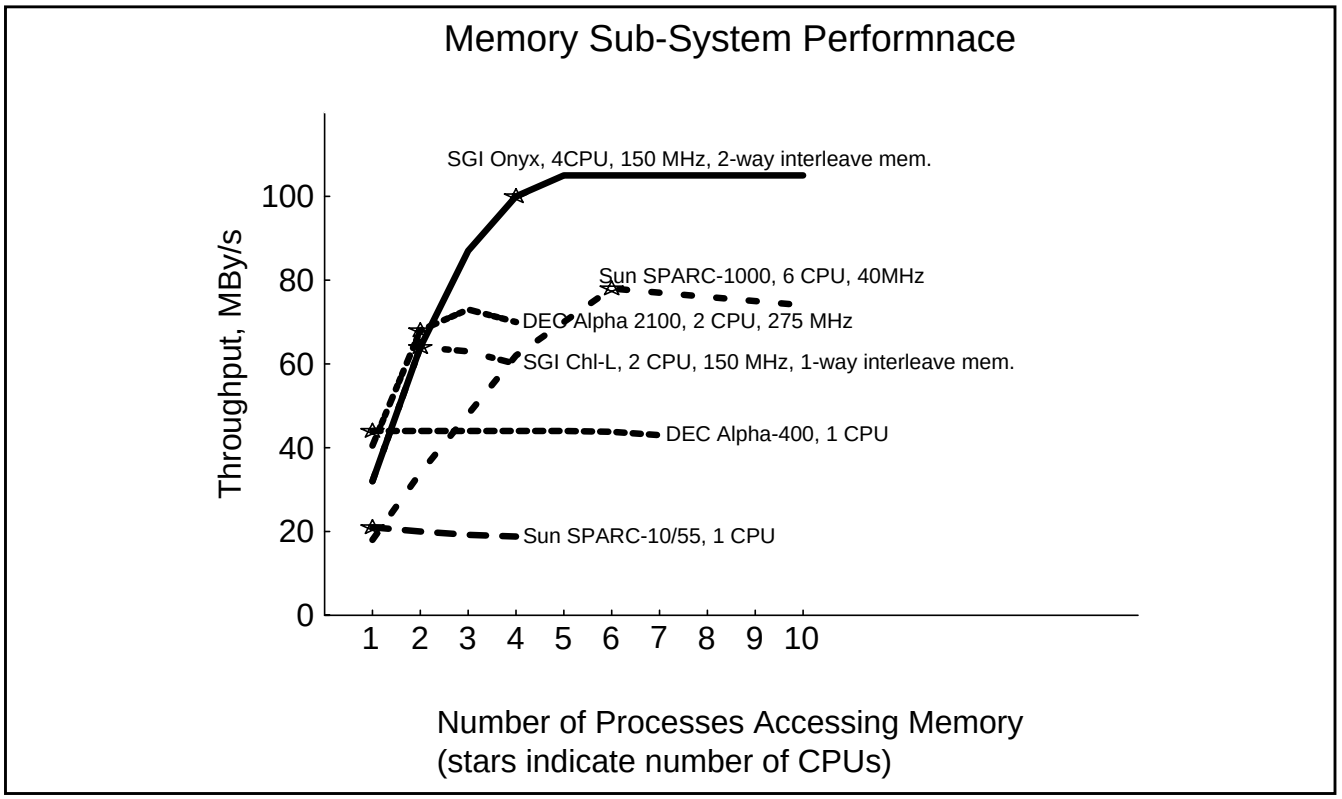

Figure 14 “Bulk” Memory Copy Performance (uncached data, all pages in physical memory)

6.0 Conclusions

In order to achieve high end-to-end performance in widely distributed applications, a great deal of analysis and tuning is needed. In the MAGIC testbed we are evolving a methodology that includes network-wide precision time sources and extensive instrumentation for time, latency, and throughput at all levels of the network, operating system, and applications. We monitor a large collection of parameters simultaneously (from the ATM level all the way up through disk performance on the storage servers and the application’s use of the delivered data) in order to identify and correct performance bottlenecks. This top-to-bottom, end-to-end approach is proving to be a very useful mechanism for analyzing the performance of distributed applications in high-speed wide-area networks. The approach is letting us identify any and all of the issues that affect performance, and to help determine which hardware or software components are the bottleneck. In one experiment described in this paper, we were able to use this mechanism operating in the MAGIC WAN ATM network to demonstrate that some very poor performance was due to a large number of TCP retransmissions, which in turn were due to the interaction of network parameters, TCP, and undersized output port buffers in one critically located ATM switch. As the bottlenecks are

identified and resolved, we will hopefully make progress toward the goal of composable, high performance modules that do not have to be tuned as an integrated system.

7.0 Future Work

We are refining the tools and the measurement techniques that capture and log events, and several of the other MAGIC consortium members are doing the same. (For example, a number of the "events" currently collected are the results of watching system variables for some interval, and then using the interval mid-point as the time stamp, when we should be getting the actual event timestamp.) We hope to be able to use the log files from the DPSS client library as "playback" files for 'netspec'[9], which is a distributed network performance measurement tool that is being designed and developed at the Telecommunications and Informations Sciences Laboratory, University of Kansas. Netspec supports multiple connections per session, and it will support multiple protocols. This will allow us to easily recreate many different traffic scenarios.

Apart from the immediate need for performance in MAGIC, the larger question that we hope to address by this methodology is whether high-performance use of networks, computing platforms, middleware, and applications has to be treated as a "system" problem (that is, all components considered and optimized together) or whether, as we find and correct problems, we will end up with an environment in which widely distributed, high-performance applications can be build by composing "stock" components, both hardware and software.

This work is ongoing, and progress reports will be published at http://www-itg.lbl.gov/DPSS.

8.0 References

[1] "BAGNet: A Metropolitan Area ATM Network" (http://www-itg.lbl.gov/BAGNet.html)

[2] "The MAGIC Gigabit Network" (http://www.magic.net/)

[3] Cavanaugh, John, Timothy Salo, "Internetworking with ATM WANs", (available as http://www.msci.magic.net/docs/magic/ip-atm.ps)

[4] Chen, L. T. and D. Rotem, "Declustering Objects for Visualization", Proc. of the 19th VLDB (Very Large Database) Conference, 1993.

[5] Chu, H-k. J., "Zero-Copy TCP in Solaris", proceedings of USENIX 1996 Annual Technical Conference, January 22-26, 1996, San Diego, CA (http://www.usenix.org/publications/library/proceedings/sd96/)

[6] Evans, Joseph B., Victor S. Frost, Gary J. Minden, "TCP and ATM in Wide Area Networks", CNRI Gigabit Network Workshop '94. (http://www.magic.net/tcp/overview.html)

[7] Ewy, B. J., J.B. Evans, G.J. Minden, and V. S. Frost, "TCP/ATM Experiences in the MAGIC Testbed", Fourth IEEE Symposium of High Performance Distributed Computing, August 1995, pp. 87-93.

[8] Jacobson, V., R. Braden, and D. Borman, "TCP Extensions for High Performance," Internet Engineering Task Force, Request for Comments (RFC) 1323, May, 1992. (Available from http://ds.internic.net/ds/dspg1intdoc.html.)

[9] Jonkman, Roelof J.T., "An Overview of NetSpec", Telecommunications & Information Sciences Laboratory, University of Kansas. (http://www.tisl.ukans.edu/Projects/AAI/products/netspec/)

[10] Lau, S, and Y. Leclerc, “TerraVision: a Terrain Visualization System,”, Technical Note 540, SRI International, Menlo Park, CA, Mar. 1994. Also see: http://www.ai.sri.com/~magic/terravision.html

[11] Mathis, M., J. Mahdavi, S. Floyd, A. Romanow, “TCP Selective Acknowledgment Options”, Internet Draft available as: ftp://ds.internic.net/internet-drafts/draft-ietf-tcplw-sack-00.txt

[12] Mills, D., “Simple Network Time Protocol (SNTP)”, RFC 1361, University of Delaware, August 1992.

[13] Richer, I. and B. B. Fuller, “An Overview of the MAGIC Project,” M93B0000173, The MITRE Corp., Bedford, MA, 1 Dec. 1993. (Available from http://www.magic.net/MAGIC_Summary.ps.)

[14] Romanow, A., and Floyd, S., “Dynamics of TCP Traffic over ATM Networks.” IEEE JSAC, V. 13 N. 4, May 1995, p. 633-641. (An earlier version appeared in SIGCOMM ‘94, August 1994, pp. 79-88.) See http://ftp.ee.lbl.gov/floyd/epd.html

[15] Schulzrinne, H., S. Casner, R. Frederick and V. Jacobson “RTP: A Transport Protocol for Real-Time Applications”, An Internet Request for Comments (RFC), January 1996. Available from: ftp://ds.internic.net/rfc/rfc1889.txt

[16] Stevens, R. W., *TCP/IP Illustrated*, Volume 1 The Protocols, Addison-Wesley Professional Computing Series, 1994.

[17] Tierney, B., W. Johnston, L. T. Chen, H. Herzog, G. Hoo, G. Jin, J. Lee and D. Rotem, “System Issues in Implementing High Speed Distributed Parallel Storage Systems”, Proceedings of USENIX High Speed Networking Symposium, August 1994, (Available from http://www-itg.lbl.gov/DPSS/papers.html.)

[18] Tierney, B., W. Johnston, H. Herzog, G. Hoo, G. Jin, and J. Lee, “Distributed Parallel Data Storage Systems: A Scalable Approach to High Speed Image Servers”, Proceedings of ACM Multimedia ‘94, Oct. 1994, LBL-35408. (Available as http://www-itg.lbl.gov/DPSS/papers.html.)

[19] The most current (and evolving) description of the DPSS technology is in the report LBL-36002 at http://www-itg.lbl.gov/DPSS/papers.html.

[20] Tierney, B., Johnston, W., Herzog, H., Hoo, G., Jin, G., and Lee, J., “System Issues in Implementing High Speed Distributed Parallel Storage Systems”, Proceedings of the USENIX Symposium on High Speed Networking, Aug. 1994, LBL-35775. Also see http://www-itg.lbl.gov/DPSS/papers.html.)

[21] Tierney, B., Johnston, W., Chen, L.T., Herzog, H., Hoo, G., Jin, G., Lee, J., “Using High Speed Networks to Enable Distributed Parallel Image Server Systems”, Proceedings of Supercomputing ‘94, Nov. 1994, LBL-35437. Available from http://www-itg.lbl.gov/DPSS/papers.html.)