Throughput optimization for multimedia applications over high speed networks

S. Zeadally G. Gheorghiu A.F.J. Levi
Department of Electrical Engineering
University of Southern California
University Park, Los Angeles, California 90089, USA
Phone: (213)740-1450
Fax: (213)740-9280
E-mail: zeadally@marco.usc.edu

Abstract

Digital video services, scientific visualization and other multimedia applications require delivery of high network throughput to end user applications. In this paper we identify some of the bottlenecks in the data path between high-speed networks and applications which are responsible for poor application performance. We then present solutions to overcome these bottlenecks at various levels namely: network, operating system, and user. Finally, we show the effectiveness of these solutions on the performance of our multimedia applications.

Keywords

Multimedia, networking, operating systems, performance, TCP/IP

1. INTRODUCTION

The past few years have seen development of applications with high bandwidth requirements such as medical image transfer, video conferencing, scientific process simulation, and visualization. Popular local area networks such as Ethernet and Token Ring (4 Mbits/s and 16 Mbits/s) are incapable of providing the bandwidth needed by these multimedia applications. New network technologies such as Fiber Distributed Data Interface (FDDI), Fast Ethernet (100BASE-T), and Asynchronous Transfer Mode (ATM) have emerged and are capable of providing high bandwidth to users' desktops. However, the challenge remains for operating system designers and application developers to deliver the bandwidth of these networks to end user applications. In order for applications to reap the benefits of high-speed networks, the entire path from network to application must be optimized. This involves removing bottlenecks introduced both at the operating system and application levels as depicted in Figure 1.

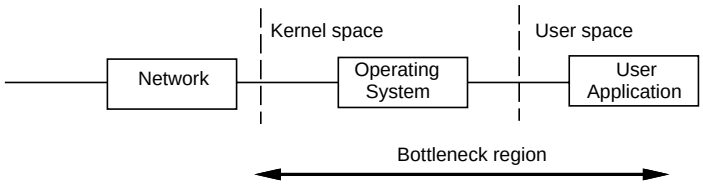

Figure 1 Illustration of bottlenecks in typical multimedia network-application data path for conventional end-systems considered in this study.

This paper describes our experiences delivering multimedia services over high-speed networks and our attempts to optimize network-application throughput. We deal with *real-life multimedia applications* as opposed to using *raw data*. The use of raw data in performance tests provides an upper bound on achievable performance that can be delivered by the underlying software and hardware. However, raw data does not identify other bottlenecks that are normally associated with actual applications such as the presentation of information (e.g. video display in a window) or the effect of running multiple applications concurrently. Some of the work done by other researchers in similar areas includes (Keller, 1993) (Patel, 1993) (Rowe, 1993). They have focused principally on optimizing only one area in the network-application data path (e.g. at application level). Our work differs from these efforts in that we apply optimizations wherever possible to the *entire* data path between network and applications as opposed to focusing on just one stage in the transfer whereby later stages introduce other overheads which degrade the overall application throughput. For instance, no matter how fast the network delivers data to the host, if the operating system itself cannot transfer the data to the application at a high enough sustained rate, then there will be a bottleneck introduced in the network-application data path.

The structure of this paper is organized as follows. In section 2, we discuss techniques that we applied to increase application-application throughput. Section 3 describes the experimental setup we have used for our multimedia applications. Section 4 presents an analysis of in-host data movement for our scientific graphics visualization application and digital video playback. Section 5 discusses the results obtained by implementing the various optimization techniques given in section 2. In section 6, we analyze the overall performance when multiple applications are running. Finally, section 7 makes some concluding remarks and presents future work.

2. IMPROVING APPLICATION-APPLICATION THROUGHPUT

To achieve high application-application throughput in a network environment, it is essential that high sustained throughput be delivered by the network, operating system, and application.

Recent advances in the performance of network physical layers have essentially solved the network bandwidth problem. As a result, it is now feasible to deliver large volumes of data at high rates with minimal loss. In this work, we have used a conventional Ethernet network, a higher speed FDDI network, and an experimental network called Jetstream (Watson, 94) (section 3) in order to study their impact on application performance. The basic physical layer characteristics of these networks include: Manchester code Ethernet signalling at 20 Mbits/s giving a maximum data rate of 10 Mbits/s, 4b/5b coding on FDDI signalling at 125 Mbits/s delivering a maximum data rate of 100 Mbits/s, and 16b/20b coding for Jetstream signalling at 1 Gbit/s giving a maximum data rate of 800 Mbits/s. These maximum data rates are reduced by media access control, network subsystem, application, and of course, limitations imposed by the host architecture.

The last few years have witnessed significant hardware improvements that have led to the development of powerful computers. Some of these improvements include: increased CPU performance, high bus bandwidth, large memories, and fast disk systems. However, there has been little change in the structure of conventional operating systems such as UNIX, consequently the availability of new hardware technologies has not been exploited to the fullest. This has made existing operating systems become a bottleneck in end systems. Well-known overheads include data copying,

network protocol processing, context switches, and interrupts (Kanakia, 1988) (Pasquale, 1992) (Ousterhout, 1990). Several techniques have been proposed and implemented to avoid these overheads (Druschel, 1993) (Dittia, 1995) (Jacobson, 1990) (Pasquale, 1994). In this work, we minimized data copying by using a UNIX kernel which supports 'single-copy' TCP/IP, a modified version of TCP/IP. Throughout this paper, we refer to 'two-copy' TCP/IP as the standard version that normally comes with UNIX operating systems. In this case, data transfer between network and application normally involves two copies: the first copy is between a network buffer and kernel buffer followed by a copy from the kernel buffer to a user application (for an incoming packet). The reverse takes place for an outgoing packet. However, in the case of a single-copy TCP/IP implementation we use for our experiments in this paper, there is only one data copy between network and application thereby eliminating the copy to kernel buffer. Moreover, the single-copy implementation of TCP also supports RFC 1323 window scaling (Jacobson, 1992), and is capable of calculating checksum during data movement.

It is not easy to come up with general techniques to increase throughput at the application level. The main reason is that different applications have different requirements and each is implemented in its own way. However, it is true that multimedia applications have a common element: they all present information (e.g. video display) to the end user. Most desktop applications running on UNIX platforms are built on standard X window systems to increase their ease of use, and offer a common look and feel to users. The X window system has become the de facto standard graphical user interface for UNIX systems. We argue that there is scope for improving application performance in the X environment in the area of data presentation. In this context, we note that without careful tuning, data display by the X server can degrade performance in the final delivery of information to the user. Our choice was to use the X shared memory extensions (Corbet, 1991) in order to speed up image display.

The usual way to display an image is to use the X11 library call *XPutImage()* on an application's data. The call to *XPutImage()* moves data from the application's buffer via Inter-Process Communication (IPC) to a private buffer of the X server (using UNIX domain sockets when the X client and the X server are on the same machine). The data is then moved by the X server to the frame buffer. With X shared memory extensions support, there is no data movement involved between the application and the X server. Instead the image data is placed into a memory segment that is shared between the application and the X server. In this case, a call to *XShmPutImage()* allows the X server to move data directly from the shared memory segment containing the application image data to the frame buffer.

3. EXPERIMENTAL ARRANGEMENT FOR MULTIMEDIA APPLICATIONS

The experiments described in this section have been carried out between two HP 9000 Series 700 workstations (99 MHz PA-RISC) which reside on the Jetstream network. Jetstream is a Gbit/s token-ring network which uses copper coaxial or fibre optic cable for the physical link. The network adapter for the HP 9000 Series 700 workstations is made up of two cards - one is called Afterburner which is equipped with 1 MByte of video random access memory used in a dual ported configuration. Afterburner is the host interface; the other card, Jetstream, is the link adapter. The shared memory present on the Afterburner board enables the support of *single-copy* implementations of network protocols such as TCP/IP and UDP/IP. Further details of Afterburner and Jetstream are given in (Watson, 1994) and (Dalton, 1993).

The hardware architecture of the host is illustrated in Figure 3. The Standard Graphics Connector (SGC) (DeBaets, 1992) is the system bus used on the HP 9000 Series 700 workstation and has a maximum data transfer rate of a Gbit/s. However, it is only possible to achieve a maximum of 400 Mbits/s transfer rate by the CPU between memory and input/output (I/O) space (e.g. graphics devices or network interface) (Frink, 1992). A principal feature of the hardware architecture is the memory and system bus controller chip which connects the CPU to memory and the I/O system components. The system bus controller chip communicates to the SGC bus via two system bus interface chips. The workstations used in our experiments have 128 MByte of main memory and two GByte of hard disk. The operating system used was HP-UX 9.01 with single-copy TCP/IP support.

Figure 2 Visualization application.

We have used two different applications in our experiments: one is a scientific visualization application and the other is digital video playback.

The visualization application uses a series of gray scale images (8 bit/pixel) of a volume rendered CAT-scan medical image of a child head as shown in Figure 2. The head can be rotated and viewed at different angles. In our experiment, the images are stored on a remote server and sent over the network to the client machine which displays the images. The image set consists of 20 image frames each of size 512x512 pixels (almost 2.1 Mbits per frame) and stored in pixmap format. This makes direct display by the X server possible without requiring any further manipulation. The user interface to the visualization application supports simple operations such as play, stop, and rewind.

The video application uses the PowerVideo700 hardware video codec (compression / decompression) from Parallax (Parallax, 1994) which supports motion-JPEG. Motion-JPEG applies JPEG (Joint Photographic Experts Group, a standardized image compression technique for still images) to individual frames of a video sequence. The video board is capable of handling high quality video at 30 frames per second in real-time during either recording or playback sessions. The PowerVideo700 is an overlay card which resides in one of the EISA slots of the EISA interface attached to the SGC bus as shown in Figure 2. We have used the MovieTool software from Parallax for recording and playing digital video stored as motion JPEG files.

For our experiments, the files used during video playback were stored on the hard disk of a remote machine. This disk was mounted on the host machine using NFS via the Jetstream interface (details are given in section 4). In a typical video playback session, the use of NFS enables the host machine to receive compressed video clips over the Jetstream network. The video board decompresses the incoming compressed video stream. The analog signals originating from the graphics card are digitized and the result is overlaid with the uncompressed video image. After the overlay is completed, the entire frame is converted back to analog and sent to the monitor (Figure 3).

In all measurement tests for both the video and scientific visualization applications, we use average playback frame rate as our quantitative metric to characterize application performance. Moreover, in all experiments, these applications were run as normal user processes, along with the usual system processes and daemons in the background.

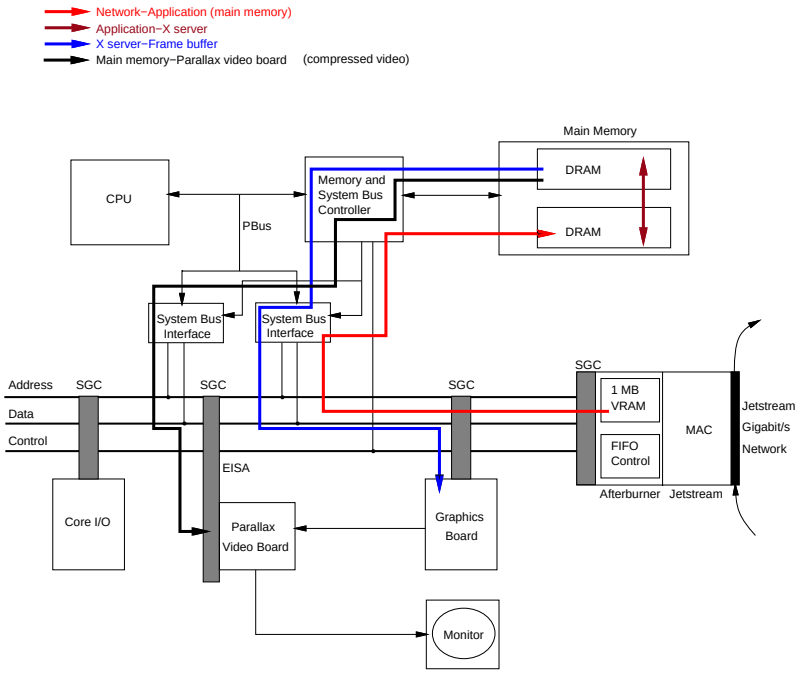

Figure 3 HP 9000 Series 700 architecture data paths for applications displaying network data and video playback. A network adapter connects the workstation to the Jetstream network. A Parallax video board performs video decompression and drives the monitor. Graphics data is passed from main memory to the graphics board and overlayed on the monitor by the Parallax video board.

4. ANALYSIS OF IN-HOST DATA MOVEMENT

In this section, we identify the data paths used when running the video and visualization applications based on the architecture presented in Figure 3.

To understand the impact of the underlying architecture on application performance, we measured the throughput at different stages when moving data from the network to the X window display for the visualization application. The major aim of performing such analysis is to identify areas where performance can be improved, and at the same time assess the suitability of the HP 9000 Series 700 workstation architecture in supporting high-speed network applications. Our observations are applicable to other similar networked multimedia applications (e.g. medical imaging, video conferencing).

Equation 1 summarizes the inverse of total throughput R for typical applications that read data from the network and use the X window system for display:

$$\frac{1}{R} = \frac{1}{Z_{Network-Application}} + \frac{1}{Z_{Application-Xserver}} + \frac{1}{Z_{Xserver-Framebuffer}} \quad \text{(Equation 1)},$$

where Z represents the throughput at different key stages of the data path from network to frame buffer as depicted in Figure 3. Furthermore, Equation 1 applies to conventional bus-based systems of the type used in our experiments.

To verify the correctness of the above equation, we ran the graphics visualization application via Jetstream and obtained a frame rate of 33 frames/second. This corresponds to a throughput of 69.3 Mbits/s (33 frames per second

multiplied by 2.1 Mbits per frame). We then measured the throughput values corresponding to the three stages of the data path of Equation 1 as follows:

- The application reads data from a socket into its buffer. The fact that we are using a single-copy TCP/IP stack allows direct data transfer from a network interface buffer to the application's buffer (avoiding the additional copy to a kernel buffer). The throughput obtained was 200 Mbits/s ($Z_{Network-Application}$).

- The application acting as an X client sends the data to the X server using inter-process communication. The rate of data movement is 240 Mbits/s ($Z_{Application-Xserver}$).

- The X server then moves the data to the frame buffer. The transfer rate obtained along this path is 230 Mbits/s ($Z_{Xserver-Framebuffer}$).

Using the measured throughput values, we calculated the total throughput R from Equation 1 and obtained 74 Mbits/s. This value is slightly higher than the observed Jetstream throughput of 69.3 Mbits/s. The difference of 4.7 Mbits/s is due to the fact that we did not take into account various overheads such as system calls, context switches, interrupts and memory allocation by the X server during data copying.

To understand the impact of running multiple network applications on the performance of the end system, we have chosen to simultaneously run both the video and the visualization applications over Jetstream. For this to be possible in the case of the video application, we used NFS over the Jetstream network interface as shown in Figure 4. This enables playback of digital video clips stored on a remote disk which has been mounted on the host machine (used as an NFS client). Our UNIX kernel supports NFS 2.0 which uses UDP.

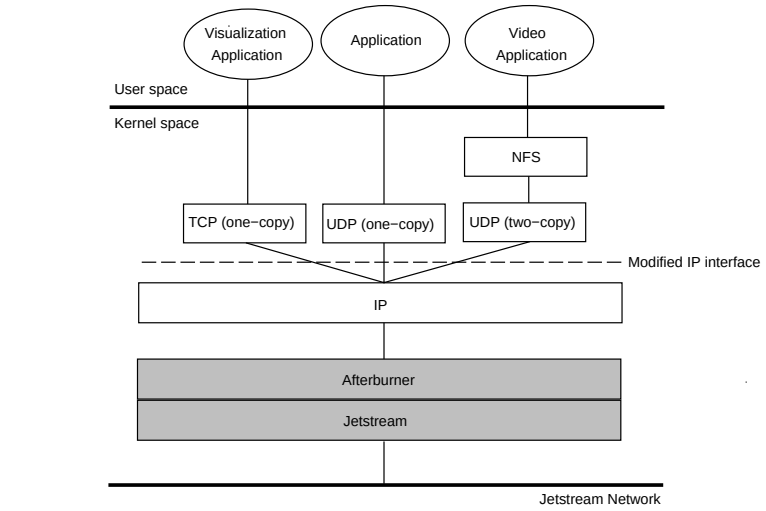

Figure 4 Protocol stacks with one-copy and two-copy support. IP layer has been modified to allow NFS support (which uses two-copy UDP/IP) to co-exist with applications using single-copy UDP/IP.

Although the UNIX kernel we have used does support both single-copy TCP/IP and single-copy UDP/IP stacks, it was not possible for NFS to use the single-copy UDP/IP stack. This is because NFS does not understand the buffer structures used in the single-copy implementation of UDP/IP. We did not consider it worth modifying NFS to allow it to support our single-copy protocol stacks. The justification was that we do not see significant throughput improvement with a version of NFS that allows single-copy protocols since the disk (at the NFS server) will still be

the bottleneck (although latency would be slightly better). For our experiments, it would have been sufficient to use a single-copy TCP/IP stack (for the visualization application) and a two-copy UDP/IP stack (for video application over NFS) by simply disabling single-copy support for UDP/IP in the UNIX kernel. However, the disadvantage of this approach is that it prevents other applications from using single-copy UDP/IP. Our solution was to modify the IP layer in order to distinguish packets destined for NFS which will use two-copy UDP/IP from all other incoming network packets which will use single-copy UDP/IP or single-copy TCP/IP (Figure 4).

5. NETWORK MEASUREMENTS AND RESULTS

Initial experiments were conducted using *raw data* to investigate how much of the available network bandwidth can actually be delivered to the application. Figure 5 presents the observed throughput using a two-copy TCP/IP stack over Ethernet, FDDI and Jetstream. The laboratory Ethernet has been used for Ethernet tests. In the case of FDDI, an EISA FDDI adapter designed for HP 9000/700 EISA systems was used to attach to an FDDI network. Performance measurements were made with a tool called *netperf* (Jones, 1993) which measures the transfer of data from a producer process (generating the data) to a consumer (receiving data) running on a remote machine. A socket buffer size of 32 KBytes (KB) has been used at both workstations. We have chosen this socket buffer size because it is close to the limit possible on standard UNIX systems (usually 48 KB) and we wanted to demonstrate the throughput achievable with an unmodified kernel. The maximum network data bandwidths for Ethernet, FDDI, and Jetstream are 10 Mbits/s, 100 Mbits/s, and 800 Mbits/s respectively. However, the maximum raw data throughput values obtained using two-copy TCP/IP in our experiments were around 9.6 Mbits/s, 74 Mbits/s and 90 Mbits/s for Ethernet, FDDI, and Jetstream respectively. These results confirm our initial assumption that the end system has become the bottleneck in a high-speed network environment.

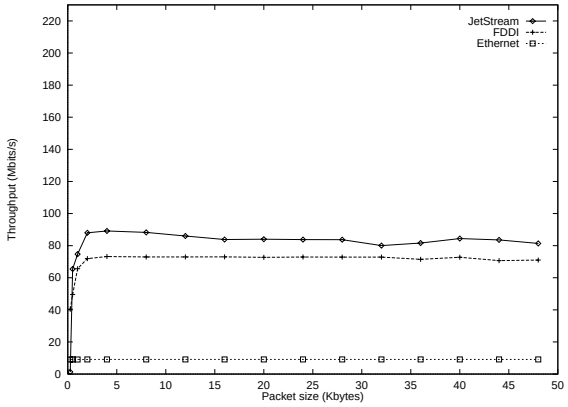

Figure 5: Measured raw data throughput over Ethernet, FDDI and Jetstream using two-copy TCP/IP and 32 KB socket buffer size.

Figure 6 shows raw data application throughput using single-copy TCP/IP over the Jetstream network. The use of a single-copy kernel increases throughput by almost 56% using 32 KB socket buffer size. Increasing the socket buffer size, if the kernel allows it, results in higher throughput values. For instance, the single-copy kernel enabled us to specify a socket buffer size of 256 KB. In this case the maximum throughput achieved was around 200 Mbits/s.

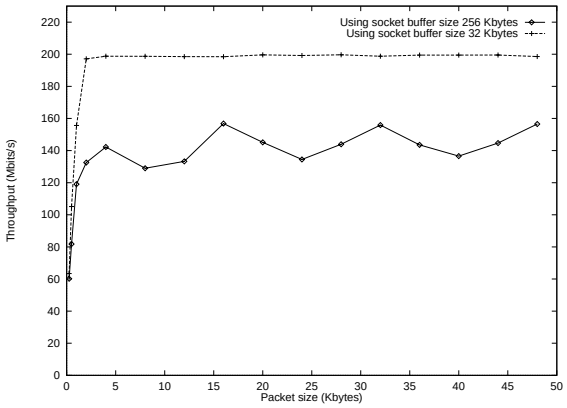

Figure 6 Measured raw data throughput over Jetstream using single-copy TCP/IP and socket buffer sizes of 32 KB and 256 KB.

Having established the achievable throughput possible using raw data, we then used the visualization application to measure the maximum throughput for a real application running over various networks. In contrast to raw data throughput results, we obtained frame rates and corresponding throughput values given in Table 1. Two important observations can be made: first, performance based on raw data is not enough to characterize application throughput; second, in the case of slow networks like Ethernet, software (e.g. operating system) on end systems is able to deliver most of the available network bandwidth to the application. However, as network speed increases, the discrepancy between network bandwidth and actual application throughput is increasing.

Table 1 Measured graphics visualization application frame rate and the equivalent throughput using two-copy TCP/IP and 32 KB socket buffer size over Ethernet, FDDI and Jetstream

Network type	*Frame rate (frames/s)*	*Throughput (Mbits/s)*
Ethernet	4	8.4
FDDI	11	23.1
Jetstream	22	46.2

We now discuss how the various optimizations mentioned in previous sections can be applied to the data path described by the three terms of Equation 1.

- *Network*: We use the Jetstream Gbit/s network capable of supporting high bandwidth applications.

- *Operating system*: A UNIX kernel that supports single-copy TCP/IP has been used. This allows direct data copy from the network to the application. As a result, data movement, considered to be the major bottleneck in current operating systems, is minimized. Moreover, network protocol overheads such as checksum calculations have also been significantly reduced. We have therefore optimized the first term of Equation 1 with $Z_{Network-Application}$ being 200 Mbits/s. This value is the maximum throughput obtained when using raw data as illustrated in Figure 6.

- *Application*: We have used the X shared memory extensions to eliminate data movement from the application to the X server. This optimizes the overall throughput by eliminating the second term $Z_{Application-Xserver}$ of Equation 1. It is also worth noting that applications should exploit the capability of using large socket buffer size

whenever the kernel allows it. Although this is done at application level, it influences the throughput between network and application at the operating system level.

Figure 7 summarizes the effects of the various optimization approaches on the performance of the visualization application. It is interesting to note from the graph that for socket buffer sizes up to 48 KB (the maximum allowable by standard UNIX kernels), the application performs better using two-copy TCP/IP and X shared memory extensions than using single-copy TCP/IP without X shared memory extensions.

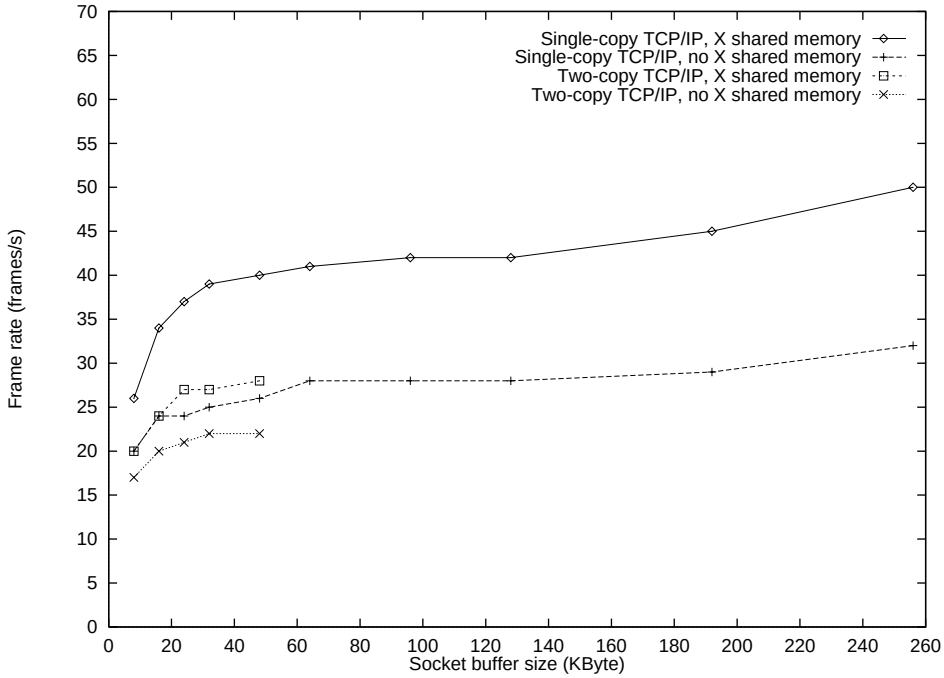

Figure 7 Measured effect of single-copy TCP/IP, X shared memory extensions, and socket buffer size on final throughput for the graphics visualization application.

The underlying architecture did not allow further optimization of the last term of Equation 1. This is because the system and memory bus controller shown in Figure 3 becomes the bottleneck when subjected to intensive data traffic to and from main memory and system bus. This is not a problem for one way traffic between main memory and a peripheral device or vice-versa. However, it becomes a limitation in the case of networked multimedia applications where data flows in continuously from network to main memory and back out from main memory to graphics display. A possible solution is to transfer data directly from a network device to the frame buffer over the system bus, a mechanism commonly referred as *kernel-level streaming* (Murphy, 1996) (Fall, 1993). Unfortunately, the SGC bus implementation in the machines used for our experiments does not allow the needed slave-slave bus transactions.

After applying all the above optimizations, Equation 1 becomes:

$$\frac{1}{R} = \frac{1}{Z_{Network-Application}} + \frac{1}{Z_{Xserver-Framebuffer}} \quad \text{(Equation 2).}$$

Calculating the value of R from Equation 2, using 200 Mbits/s for $Z_{Network-Application}$ and 230 Mbits/s for $Z_{Xserver-Framebuffer}$, we obtain an overall expected throughput of 107 Mbits/s. To verify the correctness of Equation 2, we measured the average frame rate for the visualization application after we implemented all the above optimizations. The value obtained was 50 frames/second which translates to a throughput of 105 Mbits/s. The

difference between the measured and the expected values is smaller (2 Mbits/s) than that obtained in section 4 (i.e. 4.7 Mbits/s). This is because the elimination of the second term of Equation 1 has also reduced overheads such as memory allocation by the X server.

6. OVERALL PERFORMANCE FOR MULTIPLE APPLICATIONS

To quantify the impact of running *both* video and visualization applications on overall performance, we used the following metrics: CPU usage, frame rate (visualization application), and percentage of frames dropped (video application). CPU usage was measured using *Glance* (Hewlett-Packard, 1992), a performance monitor tool that comes with standard HP-UX operating system. For the video application, the number of frames dropped was obtained from the diagnostics information generated by the device driver of the Parallax video board (Parallax, 1994). Table 2 summarizes the results obtained using the Jetstream network. All tests have been performed on a kernel that supports single-copy TCP/IP. For the visualization application, we used a socket buffer size of 256 KB. Digital video playback was via NFS at 30 frames/second, the size of each frame being 512x380 pixels and 24 bit color per pixel.

To better understand the degradation in performance when both applications are running concurrently, we first make some observations on their performance when executed on their own. When the video application is running by itself, there are no video frames dropped. For the visualization application, the frame rate without using X shared memory is 33 frames/second. However, when running both applications, there was a 39% drop of frames displayed with video and the frame rate observed for visualization decreased to 26 frames/second as shown in the Table 2. The degradation of video performance is due to the high percentage of CPU time (56%) spent in system mode. This is because the visualization application uses IPC to move data to the X server which involves multiple kernel-user interactions. The 34% of CPU cycles left for user mode are not sufficient for the demands of video, which requires 42% user-mode CPU time. On the other hand, 56% of CPU time spent in system mode is not sufficient for the needs of the visualization application which requires the CPU to spend 66% of its time in system mode. Thus, it is evident that the conflicting requirements of the two applications affect their overall performance.

Table 2 Measurement of impact of CPU utilization on video and graphics visualization applications performance.

	%CPU in user mode	*%CPU in system mode*	*Frame rate (frames/s)*	*% frames dropped*
VIDEO	42	48	--	0
GRAPHICS	27	66	33	--
VIDEO & GRAPHICS	34	56	26	39
GRAPHICS-SHMEM	40	54	50	--
VIDEO & GRAPHICS-SHMEM	41	50	35	10

VIDEO video playback application
GRAPHICS visualization application without X shared memory extensions
GRAPHICS-SHMEM visualization application with X shared memory extensions

From Table 2, we note that with X shared memory extension support, not only the visualization application has a higher frame rate on its own, but there is also an improvement in overall performance when both applications run. That is, only 10% of frames are dropped by the video application and the frame rate increased from 26 to 35 frames per second for the visualization application. The use of shared memory significantly reduces kernel-user interactions by eliminating data movement by IPC. As a result, less time is spent in system mode (50%) thereby increasing the availability of CPU for user mode (41%). This obviously benefits the video application. Also, the frame rate increase for the visualization application can be explained by the fact that with shared memory it requires 54% of CPU in system mode as opposed to 66% when not using X shared memory.

7. CONCLUSIONS AND FUTURE WORK

In this paper, we demonstrate that to achieve high application-application throughput in a high-speed network environment, we need to solve bottlenecks at *all* levels: network, operating system, and application. We have shown how using various optimization techniques, it is possible to increase network-application performance. These techniques include the use of a Gbit/s network, single-copy schemes (including improved protocol processing), and X shared memory extensions. Figure 8 summarizes the throughput optimizations for the visualization application. At each level of the data path between network and application, we apply the optimizations from the previous level. Thus, the final throughput of 105 Mbits/s for Jetstream is the result obtained after applying optimizations at all levels. Compare this to the results shown in Table I, where without any optimization, the throughput was 8.4 Mbits/s for Ethernet, 23.1 Mbits/s for FDDI, and 46.2 Mbits/s for Jetstream.

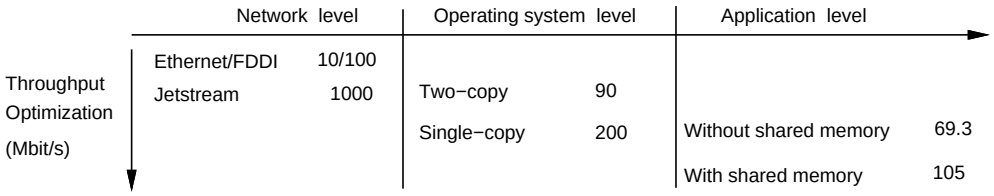

Figure 8 Summary of all applied optimizations. Each level includes the optimizations from the previous level.

As pointed out in section 4, the hardware architecture prevents the set up of a direct data path between network adapter and display for those network applications that require minimal or no data processing. Furthermore, as depicted in Figure 9, the system and memory bus controller interconnects CPU, memory, and the I/O subsystem, thereby typically becoming the bottleneck during concurrent access or transfer of data between these components. This can limit the performance of network multimedia applications which involve *simultaneous* data transfer from network to main memory, and from memory to display device.

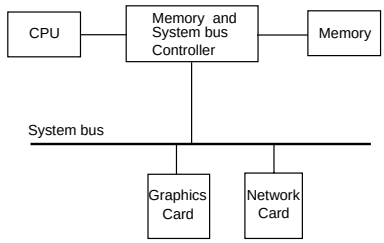

Figure 9 Simplified HP 9000 Series 700 architecture.

We are investigating new architectures which will better cope with the demands of multimedia applications in the context of high-speed networks. We anticipate that new switch-based bus architectures will allow greater flexibility in setting up different data paths between components of the system. Historically, switching logic and interconnect

components were expensive thereby limiting their use in systems. However, progress in silicon and packaging technology has changed the relative cost of interconnections and now it is possible to build general purpose computer systems based on switched bus architectures (Boxer, 1995). In this context, the data path used to derive Equation 1 no longer holds since in the case of these architectures, many paths can be used simultaneously to improve performance.

We believe that the next generation of networked multimedia applications will require more than just network displays: in addition, it should be possible to *manipulate* the multimedia data before storing, displaying or transmitting over the network. In this context, we are exploring a design space that will provide the user with the capability of setting up data paths between devices and also the flexibility of selecting portions of multimedia data in transit (e.g. from network card to display) and performing any manipulation required.

ACKNOWLEDGMENTS

The authors wish to thank the many employees of Hewlett Packard laboratories, Bristol, UK for their support and encouragement during the course of this project, in particular we are grateful to Aled Edwards for his valuable discussions on many aspects of this work. We thank Dr. Ulrich Neumann for his help in developing the visualization application. We also thank Kaleb Keithley of The X Consortium for his explanations on the X shared memory extensions. We are grateful to the anonymous reviewers for their comments on the paper. This research has been funded in part by the Integrated Media System Center, a National Science Foundation Engineering Research Center with additional support from the Annenberg Center for Communication at the University of Southern California, the California Trade and Commerce Agency, and the DARPA POLO consortium agreement MDA972-94-3-0038.

REFERENCES

Boxer, A. (1995) Where buses cannot go. *IEEE Spectrum*, 41-45.

Corbet, J. and Packard, K. (1991) The MIT Shared Memory Extension. *MIT Consortium.*

Dalton, C., Watson, G., Banks, D., Calamvokis, C., Edwards, A. and Lumley, J. (1993) Afterburner. *IEEE Network*, Vol. 7 No. 4, 36-43.

DeBaets, A. and Wheeler, K. (1992) Midrange PA-RISC Workstations with Price/Performance Leadership. *Hewlett-Packard Journal,* 6-11.

Dittia, Z., Cox Jr., J. and Parulkar, G. (1995) Design of the APIC: A High Performance ATM Host-Network Interface Chip, in *Proceedings of IEEE INFOCOM*.

Druschel, P. and Peterson, L. (1993) Fbufs: A High-Bandwidth Cross-Domain Transfer Facility, in *Proceedings of Fourteenth Symposium on Operating System Principles.*

Fall, K. and Pasquale, J. (1993) Exploiting In-Kernel Data Paths to Improve I/O Throughput and CPU Availability, in *Proceedings of Usenix Winter Technical Conference.*

Frink C., Hammond, R., Dykstal, J. and Soltis D. (1992) High-Performance Designs for the Low-Cost PA-RISC Desktop, *Hewlett-Packard Journal*, 55-63.

Hewlett-Packard (1992) HP Visual User Environment 3.0 User's Guide, Hewlett-Packard Company.

Jacobson V. (1990) Efficient Protocol Implementation, *ACM SIGCOMM Tutorial.*

Jacobson, V., Braden R. and Borman D. (1992) TCP Extensions for High Performance, *RFC 1323*.

Jones, R. (1993) Netperf: A Network Performance Benchmark, Revision 1.7, *Information Networks Division, Hewlett Packard*.

Kanakia, H. and Cheriton, D. (1988) The VMP Network Adapter Board (NAB): High-Performance Network Communications for Multiprocessors, in *Proceedings ACM SIGCOMM, Symposium on Communication Architectures and Protocols*, 175-187.

Keller, R., Effelsberg, W. and Lamparter, B. (1993) Performance Bottlenecks in Digital Movie Systems, in *Proceedings of the 4th International Workshop on Network and Operating System Support for Digital Audio and Video*, Lancaster House, Lancaster, UK, 163-174.

Murphy, B. J., Zeadally, S. and Adams, C. J. (1996) An Analysis of Process and Memory Models to Support High-Speed Networking in a UNIX Environment, in *Proceedings of Usenix Winter Technical Conference.*

Ousterhout, J. K. (1990) Why Aren't Operating Systems Getting Faster as Fast as Hardware ?, in *Proceedings of Usenix Summer Conference* , 247-256.

Parallax Hardware Guide (1994). XVideo700, MultiVideo700, and PowerVideo700, Parallax Graphics Inc, Santa Clara, CA.

Pasquale, J., Anderson, E. and Muller, P.K. (1994) Container Shipping - Operating System Support for I/O Intensive Applications, *IEEE Computer*, Vol. 27 No. 3, 84-93.

Pasquale, J., Polyzos, G., Anderson, E. and Kompella, V. (1992) A Digital Video-conferencing Experiment Using DECstation 5000 Workstation and an FDDI Network, *Internal Report*, Department of Computer Science and Engineering, University of California, San Diego, CA.

Patel, K., Smith, B. and Rowe, L. (1993) Performance of a Software MPEG Video Decoder, in *Proceedings of First ACM International Conference on Multimedia*, Los Angeles, CA, 75-82.

Rowe, L. and Smith, B. (1993) A Continuous Media Player, in *Proceedings of the 3rd International Workshop on Network and Operating System Support for Digital Audio and Video,* Lecture Notes in Computer Science, Springer Verlag, Berlin, 376-386.

Watson, G., Banks, D., Calamvokis, C., Dalton, C., Edwards, A. and Lumley J. (1994) AAL5 at a Gigabit for a Kilobuck, *Journal of High Speed Networks*, Vol. 3 No. 2, 127-145.

BIOGRAPHY

Sherali Zeadally is a researcher in the Electrical Engineering Department at the University of Southern California. He received the B.A. degree in Computer Science from University of Cambridge, England, in 1991, and the Ph.D. degree in Computer Science from University of Buckingham, England, in 1996. His current research interests include operating systems internals, distributed systems, high speed networks and multimedia.

Grig Gheorghiu is a Ph.D. candidate in the Computer Science Department at the University of Southern California. He received the B.S. degree in Computer Science from University of Bucharest, Romania, in 1993. His research interests include distributed systems and multimedia applications over high speed networks.

Anthony F. J. Levi is a Professor of Electrical Engineering at the University of Southern California. He received the B.S. degree from the University of Sussex, England in 1980 and the Ph.D. degree from University of Cambridge, England, in 1982. He is a Fellow of the Optical Society of America and a member of the American Physical Society. From January 1984 to mid-1993 Dr. Levi worked at AT&T Bell Laboratories, Murray Hill, New Jersey. In mid-1993 he left AT&T to take up a position as Professor of Electrical Engineering at the University of Southern California. Professor Levi's research interests include scaling of photonic devices to sub-micron dimensions, integration of electronic and photonic devices for high-speed network applications, and optimization of networks for high-sustained throughput applications. To date, he has published over 150 refereed journal papers and holds 9 U. S. patents in these and related research subjects.