17th ACM Symposium on Operating Systems Principles (SOSP'99)
Published as *Operating Systems Review*, **34**(5):32–47, December 1999

The interactive performance of SLIM: a stateless, thin-client architecture

Brian K. Schmidt*, Monica S. Lam*, J. Duane Northcutt†

*Computer Science Department, Stanford University
{bks, lam}@cs.stanford.edu

†Sun Microsystems Laboratories
duane.northcutt@sun.com

Abstract

Taking the concept of thin clients to the limit, this paper proposes that desktop machines should just be simple, stateless I/O devices (display, keyboard, mouse, etc.) that access a shared pool of computational resources over a dedicated interconnection fabric — much in the same way as a building's telephone services are accessed by a collection of handset devices. The stateless desktop design provides a useful mobility model in which users can transparently resume their work on any desktop console.

This paper examines the fundamental premise in this system design that modern, off-the-shelf interconnection technology can support the quality-of-service required by today's graphical and multimedia applications. We devised a methodology for analyzing the interactive performance of modern systems, and we characterized the I/O properties of common, real-life applications (e.g. Netscape, streaming video, and Quake) executing in thin-client environments. We have conducted a series of experiments on the Sun Ray™ 1 implementation of this new system architecture, and our results indicate that it provides an effective means of delivering computational services to a workgroup.

We have found that response times over a dedicated network are so low that interactive performance is indistinguishable from a dedicated workstation. A simple pixel encoding protocol requires only modest network resources (as little as a 1Mbps home connection) and is quite competitive with the X protocol. Tens of users running interactive applications can share a processor without any noticeable degradation, and many more can share the network. The simple protocol over a 100Mbps interconnection fabric can support streaming video and Quake at display rates and resolutions which provide a high-fidelity user experience.

SOSP-17 12/1999 Kiawah Island, SC
© 1999 ACM 1-58113-140-2/99/0012…$5.00

1 Introduction

Since the mid 1980's, the computing environments of many institutions have moved from large mainframe, time-sharing systems to distributed networks of desktop machines. This trend was motivated by the need to provide everyone with a bit-mapped display, and it was made possible by the widespread availability of high-performance workstations. However, the desktop computing model is not without its problems, many of which were raised by the original UNIX designers[14]:

> "Because each workstation has private data, each must be administered separately; maintenance is difficult to centralize. The machines are replaced every couple of years to take advantage of technological improvements, rendering the hardware obsolete often before it has been paid for. Most telling, a workstation is a large self-contained system, not specialised to any particular task, too slow and I/O-bound for fast compilation, too expensive to be used just to run a window system. For our purposes, primarily software development, it seemed that an approach based on distributed specialization rather than compromise could better address issues of cost-effectiveness, maintenance, performance, reliability, and security."

Many of the above issues still apply today, despite the fact that higher-performance personal computers are readily available at low cost. It has been well-established that system administration and maintenance costs dominate the total cost of ownership, especially for networks of PC's. In addition, there are many large computational tasks that require resources (multiprocessors and gigabytes of memory) that cannot be afforded on every desktop.

A great deal of research went into harvesting the unused cycles on engineers' desktop machines[1]. However, the return to time-sharing a centralized pool of hardware resources is the most effective and direct solution to this problem. Thin-client computing is the modern version of the old time-sharing approach, and there have been numerous projects that focus on thin-client strategies. For example, the Plan 9 project completely redesigned the computing system with an emphasis on supporting a seamless distributed model[14]. The terminal at the desktop in Plan 9 is still a general-purpose computer running a complete virtual memory operating system, but it is intended to run only programs with low resource demands.

More recently, thin-client-based system architectures have been developed, examples of which include X Terminals, Windows-Based Terminals, JavaStations, and other Network Computers. In these systems the desktop unit executes only the graphical interface of applications, which allows more sharing of computing and memory resources on the server at the expense of added network traffic in the form of display protocols, such as X[13], ICA[3], and RDP[10]. The protocols are highly optimized for specific software API's to reduce their bandwidth requirements. Thus, they typically require a nontrivial amount of processing resources and memory to maintain environment state and application data, such as font libraries and Java applets. Users are still subjected to some degree of system administration, such as network management and software upgrades.

Finally, the thinnest possible clients are dumb terminals which only know how to display raw pixels. The MaxStation from MaxSpeed Corporation is refreshed via 64Mbps dedicated connections, which can only support a resolution of 1024x768 with 8-bit pixels[9]. However, a standard 24-bit, 1280x1024 pixel display with a 76Hz refresh rate would require roughly 2.23Gbps of bandwidth. The VNC viewer, on the other hand, allows access to a user's desktop environment from any network connection by having the viewer periodically request the current state of the frame buffer[16]. While this design allows the system to scale to various network bandwidth levels, its interactive performance (even on a low-latency, high-bandwidth network) is noticeably inferior to that of a desktop workstation.

1.1 SLIM: a stateless thin-client architecture

The premise of this paper is that commodity networks are fast enough to use a low-level protocol to remotely serve graphical displays of common, GUI-based applications without any noticeable performance degradation. This leads us to take the notion of thin clients to the limit by removing all state and computation from the desktop and designing a low-level hardware- and software-independent protocol to connect all user-accessible devices to the system's computational resources over a low-cost commodity network. We refer to such thin-client architectures as SLIM (Stateless, Low-level Interface Machine) systems, and we illustrate their major components in Figure 1.

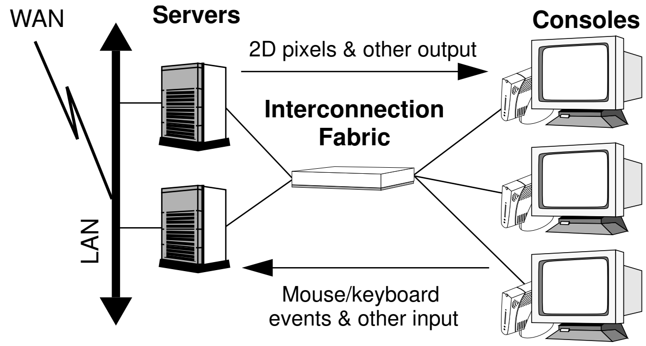

Figure 1 Major components of the SLIM architecture.

As an extreme design with the thinnest possible terminals, SLIM maximizes the advantages of thin-client architectures: resource sharing, centralized administration, and inexpensive desktop units to minimize cost per seat. In addition, the SLIM architecture has the following distinctive characteristics:

- *Statelessness*. Only transient, cached state (such as frame buffer content) is permitted in SLIM consoles; the servers maintain the true state at all times. Thus, the user is isolated from desktop failures and may move freely between terminals. In our implementation, users can simply present a smart identification card at any desktop, and the screen is returned to the exact state at which it was left.
- *Low-level interface.* The low-level SLIM protocol is designed to move raw I/O data between the servers and consoles. Thus, a SLIM console is merely an I/O multiplexor connected to a network. The protocol can be implemented by simply redirecting server I/O to SLIM consoles over the network at the device driver level. Thus, applications on the server require no modification. SLIM consoles are not tied to a particular display API (e.g. X, Win32, Java AWT), and so applications running on different system architectures with different API's can be easily integrated onto a single desktop.
- *A fixed-function appliance.* As it merely implements a fixed, low-level protocol, a SLIM console bears more similarity to a consumer-electronics device than a computer. There is no software to administer and upgrade; its simplicity makes it amenable to a low-cost, single-chip implementation; and finally, a SLIM console requires no fan and generates no unwanted noise.

1.2 Contributions of this paper

This paper presents the SLIM design and provides a quantitative analysis of the two questions critical to the success of the SLIM architecture: (1) the extent to which today's interconnect can support graphical displays of interactive programs, and (2) the resource sharing advantage of such a system.

This paper focuses on the workgroup environment where SLIM consoles are connected to the servers via a private, commodity network carrying only SLIM protocol traffic. In particular, our experiments are performed on a dedicated 100Mbps switched ethernet. SLIM consoles are intended as replacements for desktop machines, which may then be moved to the machine room as additional servers. Installations will still require their usual complement of file servers and large servers for computational-intensive tasks. We show that SLIM systems are indistinguishable from dedicated desktop units for a wide range of interactive, GUI-based applications and certain classes of multimedia programs. However, high-end, display-intensive applications, such as 3-D modelling, are outside the scope of the current implementation. Our results indicate that service to the home over broadband connections would have acceptable performance, but this approach would not work well in high-latency, low-bandwidth settings such as dial-up connections to the internet.

This work was done in cooperation with the research group at Sun Microsystems Laboratories that developed the recently announced Sun Ray™ 1 enterprise appliance product, which is an embodiment of the SLIM architecture. The prototype of the product was used by the developers (over 60 engineers, managers, marketing personnel, and support staff) as their only desktop computing device for the past year, and they have found their interactive experience to be indistinguishable from that of working on high-end, workstation-class machines. This paper attempts to quantify the interactive performance of the SLIM architecture scientifically, using the Sun Ray 1 prototype and product as a basis for the experiments.

While much research went into developing the methodology and benchmarks for analyzing a system's computational aspects (e.g. the SPEC benchmarks), little has been done in the way of evaluating the interactive performance of a system. Thus, part of this paper's contribution is a methodology for analyzing interactive systems. The findings of this paper are also useful for measuring the interactive performance of other systems.

Our methodology is to base the experiments on modern, highly-interactive applications such as Photoshop and Netscape, as well as streaming video and 3-D games. We measure these applications along dimensions that govern their interactive performance, such as input rates, display update rates, and bandwidth requirements. Since interactive jobs require actual users to provide input, we conducted a set of user studies to collect application profiles. As such experiments are expensive to run, we logged all the information related to their network traffic and resource utilization. In this way, we can investigate different aspects of the system by post-processing the data, rather than conducting more user studies. The data collection requires minimal resources, and thus does not perturb the results.

A difficult problem we had to address was how to measure the effect of sharing on interactive performance and to determine the level of sharing that can be supported by a system. Our solution is to utilize a yardstick application, i.e. one with well-defined characteristics. We quantify the effect of sharing on interactive performance by measuring the additional latency experienced by the yardstick application under different loading conditions.

Using this methodology, we evaluated the performance of the Sun Ray 1 implementation and compared it with the X protocol. Our results demonstrate that the SLIM protocol is capable of supporting common, GUI-based applications with no performance degradation. Surprisingly, the low-level SLIM protocol does not translate to a greater bandwidth demand when compared to X. This is because X was found to only optimize applications with low-end bandwidth requirements. A Sun Ray 1 console can display a 720x480 video clip at 20Hz and allows Quake to be played at a resolution of 480x360. The server, and not the bandwidth to the console, turns out to be the bottleneck for these applications. Finally, our experiments show a high potential for resource sharing. Depending on the application class, anywhere from 10 to 36 active users can share a 300MHz processor without any noticeable degradation of interactive performance. While the network is often regarded as the major performance bottleneck in thin-client systems, it can support more active users (by an order of magnitude) than the processor and memory.

Centralizing all computing resources is not without its disadvantages, and there are still many opportunities for further research to combine the best of centralized and distributed computing. Distributed computing offers users isolated performance guarantees, higher levels of security and privacy, user-level customization of the system, higher tolerance to server and network failures, and off-line computing. These topics are beyond the scope of this paper.

The rest of this paper is organized as follows. Section 2 provides an overview of the SLIM architecture. Section 3 describes our experimental methodology, and Sections 4–6 contain the experiments and their results. We begin with a stand-alone assessment of each architectural component, followed by a characterization of interactive applications, including input, display, and communication requirements. Then, we explore the effects of resource sharing on interactive performance. Section 7 discusses multimedia support, and Section 8 compares the SLIM approach with related systems. We summarize our findings and discuss future work in Section 9.

2 The SLIM architecture and implementation

In this section we describe the design and rationale for each of the components in the SLIM architecture: the interconnect fabric, the SLIM protocol, the consoles, and finally the servers. In addition, we present the details of the Sun Ray 1 implementation[12][18] on which our experiments are based.

2.1 Interconnection fabric

In the SLIM system, raw display updates are transmitted over the network to display devices. Thus, the SLIM interconnection fabric (IF) is the centerpiece of the architecture, and yet it is perhaps the simplest component. It is defined to be a private communication medium with dedicated connections between the desktop units and the servers. Such functionality can easily be provided by modern, off-the-shelf networking hardware. The Sun Ray 1 supports a 10/100 Base-T ethernet connection, and we used switched, full-duplex 100Mbps ethernet with Foundry FastIron Workgroup switches in our experiments.

The IF is defined in this manner so that the system can make response time guarantees and thereby provide high interactive performance, regardless of loading conditions. Although it is possible to share bandwidth on the IF, care must be taken to ensure that response time does not suffer as a result. In addition, there is no need to provide higher level (e.g., Layer 3 and above) services on the IF, nor the complex management typically provided on LAN's.

2.2 The SLIM protocol

The SLIM protocol takes advantage of the fact that the display tends to change in response to human input, which is quite slow. Thus, instead of refreshing the monitor across the IF, we achieve considerable bandwidth savings by refreshing the display from a local frame buffer and transmitting only pixel updates. Although the display is refreshed locally, it represents only soft state which may be overwritten at any time. The full, persistent contents of the frame buffer are maintained at the server.

The protocol consists of a small number of messages for communicating status between desktop and server, passing keyboard and mouse state, transporting audio data, and updating the display. The display commands are outlined in Table 1. They compress pixel data by taking advantage of the redundancy commonly found in the pixel values generated by modern applications. For example, the BITMAP command is useful for encoding text windows.

Command Type	Description
SET	Set literal pixel values of a rectangular region.
BITMAP	Expand a bitmap to fill a rectangular region with a (foreground) color where the bitmap contains 1's and another (background) color where the bitmap contains 0's.
FILL	Fill a rectangular region with one pixel value.
COPY	Copy a rectangular region of the frame buffer to another location.
CSCS	Color-space convert rectangular region from YUV to RGB with optional bilinear scaling.

Table 1 SLIM protocol display commands.

In contrast, most other remote display protocols (e.g. X[13] and ICA[3]) send high-level commands (e.g. "display a character with a given font, using a specific graphics context"), which require considerable amounts of state and computation on the desktop unit. By encoding raw pixel values, the SLIM protocol represents the lowest common denominator of all rendering API's. To take advantage of this protocol, applications can be ported by simply changing the device drivers in rendering libraries. For example, we have implemented a virtual device driver for the X-server, and all X applications can run unchanged.

Although the SLIM protocol is intended to be used primarily by low-level device drivers, applications may utilize a software library to transmit display operations in a domain-specific manner. For example, we have implemented a video playback utility and a 3-D game by converting each display frame to YUV format and using the CSCS command to transmit the data directly to the console. Although these applications could function as regular X clients, this approach is more efficient and produces higher quality results. The library itself contains packet sequencing commands, methods for obtaining geometry information and input events, as well as bandwidth and session management routines.

Another feature of the SLIM protocol is that it does not require reliable, sequenced packet delivery, whereas other remote display protocols (e.g. X[13], ICA[3], and VNC[16]) are typically built on top of a reliable transport mechanism. All SLIM protocol messages contain unique identifiers and can be replayed with no ill effects. Since the preferred IF implementation is a dedicated connection between console and server, errors and out-of-order packets are uncommon. Thus, systems using the SLIM protocol can be highly optimized with respect to error recovery. In the Sun Ray 1 implementation, SLIM protocol commands are transmitted via UDP/IP between the servers and consoles, and its application-specific error recovery scheme allows for more efficient recovery than packet replay and avoids "stop and wait" protocols.

2.3 SLIM consoles

A SLIM console is simply a dumb frame buffer. It receives display primitives, decodes them and hands off the pixels to the graphics controller. This allows us to make the desktop unit a cheap, interchangeable, fixed-function device. It is completely stateless and runs neither an operating system nor any applications, and the performance a user experiences is decoupled from the desktop hardware.

The Sun Ray 1 enterprise appliance includes four major hardware components: CPU, network interface, frame buffer, and peripheral I/O. It utilizes a low-cost, 100MHz microSPARC-IIep processor with 8MB of memory, of which only 2MB are used. The network interface is a standard 10/100Mbps ethernet controller, and the frame buffer controller is implemented with the ATI Rage 128 chip (with support for resolutions up to 1280x1024 at a 76Hz refresh rate and 24-bit pixels). A four-port USB hub is included for attaching peripherals, including keyboard and mouse. The firmware on the console simply coordinates the activity between the components, i.e. it moves data between the network and appropriate devices.

2.4 SLIM servers

In the SLIM architecture, all processing is performed on a set of server machines which may include a variety of architectures running any operating system with any application software. These servers neither replace nor remove the standard set of servers common today, e.g. file servers, print servers. They merely consolidate and reduce the computing resources that were previously on desktops.

We only need to add to these servers a small set of system services: daemons for authentication, session management, and remote device management. The authentication manager is responsible for verifying the identity of desktop users; the session manager redirects the I/O for a user's session to the appropriate console; and the remote device manager handles peripherals attached to the system via a SLIM console.

3 Evaluation methodology

In this section we present our evaluation methodology and compare it with some related techniques.

3.1 Our approach

Our performance analysis of the system consists of four steps: (1) establish the basic performance of each component (the IF, server, and consoles) of the SLIM system using stand-alone tests, (2) characterize GUI-based applications by their communication requirements, evaluate how well the SLIM architecture and protocol support such applications, and compare the result with that of the X protocol, (3) evaluate the opportunity for sharing offered by the SLIM architecture by measuring its interactive performance under shared load and also by obtaining load profiles of actual installations of the system, and (4) measure the limit of SLIM by evaluating its performance on multimedia applications.

Category	Application
Image Processing	Adobe Photoshop 3.0
Web Browsing	Netscape Communicator 4.02
Word Processing	Adobe Frame Maker 5.5
PIM	Personal Information Management tools (e.g. e-mail, calendar, report forms)
Streaming Video	MPEG-II decoder and live video player
3-D Games	Quake from id Software

Table 2 Benchmark applications.

To ensure that our results would have the widest applicability, we have chosen a set of commonly-used, GUI-based applications, as listed in Table 2. The set covers a wide range of resource and display demands. We instrumented the SLIM protocol driver used by these applications so as to measure the I/O properties of their human interface. Measurements were taken by having a group of 50 people separately run the applications for at least ten minutes on the Sun Ray 1 prototypes with a very lightly loaded server. The resulting profiles are thus indicative of real-world use, and they reflect performance for individual users in isolation. Depending on the nature of the experiments and machine availability, we employed a variety of test configurations during our study, and we summarize them in Table 3.

To characterize interactive performance under shared load, we use an indirect method. We create a highly interactive application with a fixed and regular resource requirement, and instrument it to measure the response time its user experiences. We run this application with different system loads and use its measured latency to gauge the system's responsiveness. To simulate multiple active users we use load generators to "play back" the resource profiles that were recorded during the user studies. This approach has the advantage of providing a consistent metric across systems and applications.

Experiment	Desktop Unit	Server				
		Model	Processor(s)	RAM	Swap	OS
IF response time	Sun Ray 1 prototype	Ultra 2 workstation	1 296MHz UltraSPARC-II	512MB	1GB	Solaris 2.6
x11perf	Sun Ray 1	Enterprise E4500	8 336MHz UltraSPARC-II	6GB	13GB	Solaris 2.7
User studies	Sun Ray 1 prototype	Ultra 2 workstation	2 296MHz UltraSPARC-II	512MB	1GB	Solaris 2.6
Processor sharing	Simulated	Enterprise E4500	10 296MHz UltraSPARC-II	4GB	4.5GB	Solaris 2.7
IF sharing	Same as server	Ultra 2 workstation	2 296MHz UltraSPARC-II	512MB	1GB	Solaris 2.7
Multimedia tests	Sun Ray 1	Enterprise E4500	8 336MHz UltraSPARC-II	6GB	13GB	Solaris 2.7

Table 3 Hardware configurations for empirical studies. In all cases the interconnection fabric was 100Mbps ethernet with Foundry FastIron Workgroup switches. All machine hardware is manufactured by Sun Microsystems, Inc.

3.2 Related techniques

Endo *et al.* have also attempted to measure the interactive performance of a general system[7]. They argue for the use of interactive event latency (i.e. response time) metrics to evaluate system performance, which is similar to our own measurement goals. They instrumented Windows NT versions 3.51 and 4.0 as well as Windows 95 and recorded processing latency for each input event for a set of stand-alone applications. The main difficulty they experienced is that it is impossible to know exactly how much processing time is needed to service an event without instrumenting each application. So, they had to rely on heuristics, which prevented them from studying more than one simultaneously active application. Our indirect benchmark technique addresses this problem. In addition, their approach is aimed at quantifying the interactive performance of stand-alone desktop systems, such as a PC, whereas we are interested in evaluating remote display systems which include a network component.

Another related evaluation technique is capacity planning to determine the number of users a server can support[4][19][21]. Such studies are useful for making resource allocation decisions, but they do not adequately assess interactive performance. The major problem with this technique is that it tends to take a long-term, coarse-grained view of system activity, thereby losing information on interactive performance. In addition, user activity is modelled in one of three ways, all of which have their drawbacks. First, canned scripts or macros are frequently used, but they have no interactive delays and thus merely measure throughput. Second, recorded scripts with interactive delays are used to emulate users, but timing dependencies between the client (e.g. application) and server (e.g. the X-server) make such emulations only valid when the system is in underload. Finally, queueing theory is used to model users based on average resource profiles. While this approach has the advantage of being able to leverage well-known mathematics techniques to make precise predictions, it cannot model the system in overload. In all cases, our indirect benchmark approach provides a better assessment.

4 Performance of SLIM components

We begin our analysis of the SLIM system with a stand-alone characterization of each major component in the Sun Ray 1 implementation. The results are summarized in Table 4 and Table 5.

Benchmark	Result
Response time over a 100Mbps switched IF	550 µs
x11perf / Xmark93	3.834
x11perf / Xmark93 — no display data sent on IF	7.505

Table 4 Stand-alone benchmarks for the Sun Ray 1.

4.1 Response time over the IF

Response time is a major concern for any remote display architecture. Other approaches, such as X terminals and WinTerms, process input events locally on the desktop. A SLIM system, however, must transmit all input events to a server and then wait for it to send back the results. Humans begin to notice delays when latency enters the 50–150ms range[17]. To provide a high-quality interactive experience, it is crucial that response time remain within these bounds.

Typical local area networks are shared and carry a variety of traffic. Depending on the current load, latency can be highly variable, and it is difficult to make response time guarantees in such an environment. We ensure that latency remains within the prescribed limits by using a switched, private network for the interconnection fabric. In the SLIM architecture each desktop unit has a direct connection to the servers, and only SLIM protocol traffic is carried over it. Since there is no interference or other outside effects, the added latency is exactly the round-trip delay over the interconnect.

To demonstrate the effectiveness of the SLIM approach, we wrote a simple server application which accepts keystrokes from a SLIM console and responds by sending characters to the console. We measured the total elapsed time from the instant a keystroke is generated at the SLIM console to the point at which rendering is complete and the pixels are guaranteed to be on the display.

With the test setup shown in Table 3, the elapsed time was found to be 550µs. In contrast, if we type the characters in an Emacs editor, as opposed to our simple application, the delay is found to be 3.83ms. Clearly, in such an environment the communication medium is a negligible source of latency, and the end result is that the response time which users experience is effectively dependent on the processing time on the server. Users are, therefore, unable to distinguish between the response times observed from a remote SLIM console and a monitor connected directly to the server itself.

It is interesting to note that on a stand-alone workstation, the service time is much higher at 30ms[11]. This is most likely due to buffering of input in the keyboard device driver so that block data transfers can be used. Such an optimization is not required for network I/O since it can be memory-mapped directly into a user's address space. Thus, the SLIM implementation can actually be more responsive than a dedicated workstation.

4.2 Server graphics performance

A key factor in the display performance of any thin-client system is how well the server can generate the protocol. In the Sun Ray 1 implementation of the SLIM architecture, the X-server is responsible for translating between the X and SLIM protocols and transmitting display update commands over the IF. To analyze the graphics performance of a SLIM server, we ran the SPEC GPC x11perf benchmark.

The x11perf benchmark is older and no longer used, but it provides a useful indication of performance. We ran the benchmark on the system shown in Table 3, and we used the Xmark93 script to generate a single figure of merit from the results. The X-server achieved a rating of 3.834. Although the X-server must interpret the commands and transmit display data to the console, its Xmark value is quite comparable to values for X terminals, which receive commands and interpret them locally. For example, the NEC HMX reported a value of 4.20 to SPEC. If we eliminate the actual transmission of SLIM protocol commands, the Xmark performance rating of the X-server improves to 7.505, indicating that network operations represent a significant performance factor for this benchmark.

4.3 Protocol processing on the desktop

The performance-limiting factor on a SLIM console is the highest sustained rate at which it can process protocol commands. To determine this limit, we created a server application which transmits sequences of protocol commands to a Sun Ray 1 console up to the point where the terminal cannot process the transmitted commands and begins to drop them. Once these sustained rates were determined for various command types and sizes, we calculated the protocol processing cost in terms of a constant overhead per command as well as an incremental cost per pixel. The results are presented in Table 5.

Protocol Command	Startup Cost	Cost per Pixel
SET	5000 ns	270 ns
BITMAP	11080 ns	22 ns
FILL	5000 ns	2 ns
COPY	5000 ns	10 ns
CSCS (16 bits/pixel)	24000 ns	205 ns
CSCS (12 bits/pixel)	24000 ns	193 ns
CSCS (8 bits/pixel)	24000 ns	178 ns
CSCS (5 bits/pixel)	24000 ns	150 ns

Table 5 Sun Ray 1 protocol processing costs.

The SET command has a fairly high incremental cost because pixels must be expanded from packed 3-byte format to 4-byte quantities suitable for the frame buffer. On the other hand, the FILL and COPY commands are very inexpensive. The BITMAP command has a high start-up cost in order to set the state of the graphics card and must be amortized over a large number of pixels to be of the most benefit. The color space convert and scale (CSCS) command not only has a high start-up cost in order to configure the graphics controller, but it also has a fairly high cost per pixel. Still, it is more efficient than the SET command when fairly large numbers of pixels are sent, which is usually the case for applications that utilize image or video data. Also, it provides a significant reduction in bandwidth, making it worthwhile despite the high processing overhead.

5 Characterization of interactive applications

The SLIM architecture is based partly on the observation that the display is typically updated in response to user activity. We expect humans to have a relatively slow rate of interaction, and so the display should be quiescent much of the time, thus requiring modest resources to provide good performance. To verify this observation, we analyzed the interaction and display patterns of modern applications during active use and demonstrate that SLIM can meet their demands. Most of the results in this section apply to any architecture, not just SLIM.

To characterize I/O requirements of interactive programs, we use the GUI-based benchmark applications listed in Table 2, i.e. Photoshop, Netscape, Frame Maker, and PIM. As mentioned in Section 3.1, we employ user trials to gather data during real-world use of the applications. The data were collected on two identical systems, as listed in Table 3. Each server had a 1Gbps uplink from its IF and ten terminals attached to it. The servers were completely underloaded at all times, and users had the impression that they were working on a dedicated workstation. Thus, the gathered traces are indicative of stand-alone operation.

To obtain these results, we instrumented the SLIM protocol driver in our implementation of the X-server. All X and SLIM protocol events were recorded and timestamped. The logging overhead is not measurably significant and therefore does not perturb the results. These logs enable us to determine input and display characteristics as well as network traffic requirements.

This section is organized as follows. We begin by analyzing the rates of human interaction, followed by a characterization of the sizes of display updates induced by that interaction. Next, we assess the compression efficiency of the SLIM protocol as well as its ability to scale down to lower bandwidths. We continue with an analysis of the protocol processing costs on both the console and server, and we conclude with a comparison of the overall bandwidth requirements of the benchmark applications under the SLIM and X protocols.

5.1 Human input rates

Based on the data gathered in the user studies, we first analyzed typical rates of human interaction. We defined input events to be keystrokes and mouse clicks. All input events are transmitted to the server for processing, and mouse motion events do cause some display updates for these applications (e.g. rubberbanding). However, such motion events are demarcated by a button press and release, which serve to toggle the drawing mode. In these cases the motion-induced display updates are more naturally regarded as a single update caused by the initial button press. For each of the GUI-based benchmark application sessions, we calculated the frequency of input events, and Figure 2 presents the results.

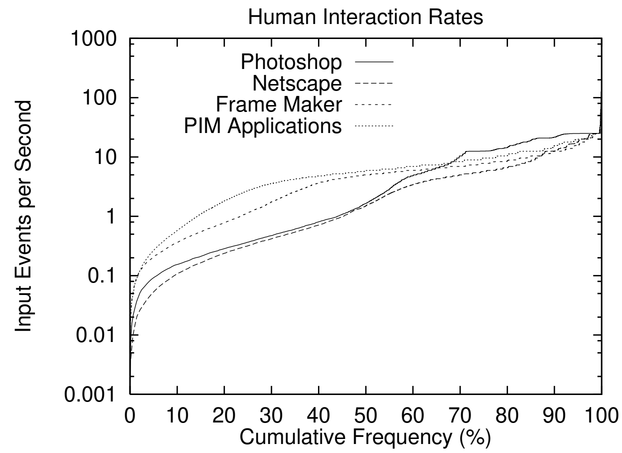

Figure 2 Cumulative distributions of user input event frequency. Input events are defined as keystrokes and mouse clicks. Histogram bucket size is 0.005 events/sec.

The most important thing to note in this graph is that human input rates are typically much lower than monitor refresh rates (which are at least 60Hz). In particular, we see that for each application, less than 1% of input events occur with frequency greater than 28Hz. This is important because it indicates an application-independent "upper bound" on the rate at which humans generate input. Also, roughly 70% of all events occur at low frequencies, i.e. less than 10Hz (or more than 100ms between events). Despite the diversity of these applications, the interaction patterns are quite similar. Even so, Netscape and Photoshop tend to be much less interactive than Frame Maker or PIM, as indicated by their substantially larger percentage of events occurring at least one second apart.

5.2 Pixel update rates

Next, we consider the pixel changes induced by human input. Correlating input events to display updates can only be done by instrumenting all applications. However, source code is frequently unavailable, and adding correct instrumentation to a complex program, such as Netscape, is prone to errors. Therefore, we use the following heuristic to estimate the correlation between input events and display updates: all pixel changes that occur between two input events are considered to be induced by the first event. Although this is not true in all cases, it provides a close enough approximation for the applications in this experiment over the course of the ten-minute user sessions.

To obtain these values, we sum the number of pixels affected by SLIM display primitives recorded between the events, and we present the results in Figure 3.

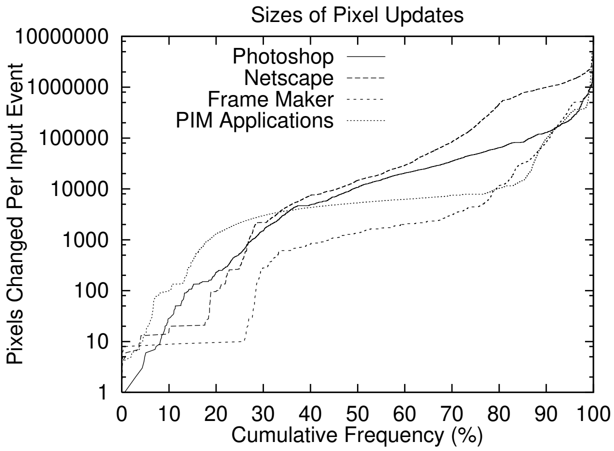

Figure 3 Cumulative distributions of pixels changed per user input event. Input events are defined as keystrokes and mouse clicks. Histogram bucket size is 1 pixel.

This graph depicts the cumulative distributions of pixels altered per event. All displays used in these experiments had a resolution of 1280 by 1024 pixels (i.e. 1.25Mpixels). The important thing to note is that display updates typically affect only a small fraction of the full display area. For example, nearly 50% of all input events for any application cause less than 10Kpixels to be modified. Further, only 20% of Frame Maker or PIM events affect more than 10Kpixels, and only 30% of Netscape or Photoshop events affect more than 50Kpixels. Also, note that Netscape is more demanding than Photoshop, but as we will later see, its compressed bandwidth requirement is much lower.

This result, coupled with the rates of human interaction, indicate that (for this class of applications) the contents of the display change only slowly and moderately over time. This is important because it has a significant effect on the SLIM architecture, namely that even as display requirements increase over time, human input rates (and therefore required update rates) are unlikely to change. Thus, it is unlikely that SLIM consoles would need to be upgraded for users of these types of applications.

5.3 Compressed pixel update rates

To assess the effectiveness of the SLIM pixel encoding techniques, we analyze the compression factor afforded by each of the SLIM display commands. In addition, we examine the sizes and network transmission delays of display updates once they have been compressed with the SLIM protocol.

In Figure 4 we present the data reduction afforded by each of the SLIM protocol display commands for the GUI-based benchmark applications. The important observation to make is that the protocol provides a factor of 2 compression for Photoshop and a factor of 10 or more for all other applications. The FILL command is extremely

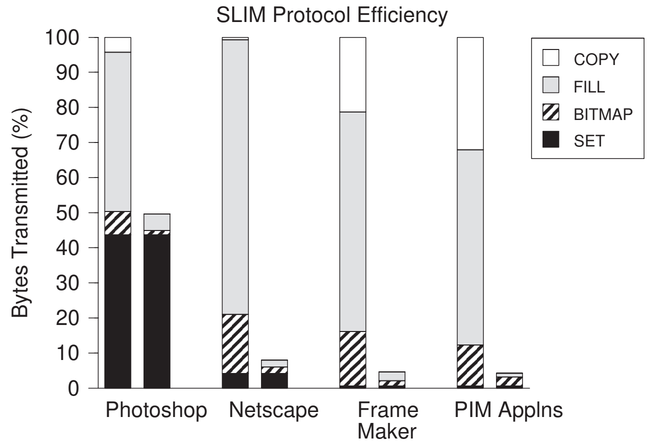

Figure 4 Efficiency of SLIM protocol display commands. The left-hand bar in each set depicts uncompressed data, and the right-hand bar depicts use of the SLIM protocol. CSCS is not used in these benchmark applications.

effective, reducing bandwidth by 40%–75% across the applications. The BITMAP and COPY commands are utilized to different extents, but they also provide substantial compression. PIM and Frame Maker enjoy particularly large savings because they employ a great deal of bicolor text and scrolling. The SET command represents pixel data which cannot be compressed via the SLIM protocol, but only Photoshop has a substantial portion of such commands. Thus, we can see that the protocol has done a good job of compressing the pixel data where possible.

In Figure 5 we present the cumulative distributions of the amount of SLIM protocol data transmitted per user input event for the GUI-based benchmark applications. From this graph we can see that (over a 100Mbps interconnection fabric) the transmission delays will be quite small. For example, even a large update of 50KB incurs only 3.8ms of transmission delay. More specifically, only 25% of Photoshop and Netscape events require more than 10KB to encode the display update, and only 5% require more than 50KB. Frame Maker and PIM have even lower requirements; only 17% of events require more than 1KB for display updates, and only 2% require more than 10KB.

5.4 Scalability of the SLIM protocol

Although the SLIM protocol can comfortably accommodate the GUI-based benchmark applications with a low-latency, 100Mbps interconnection fabric, it is useful to consider how well it would operate over lower bandwidth connections. To obtain a sense of how interactive performance would be affected, we modified the X-server to restrict the bandwidth it could utilize, and then we ran our normal window sessions in the constrained setting.

At 10Mbps users could not distinguish any difference from operating at 100Mbps. To simulate a high-speed home connection, such as a cable modem or DSL, we

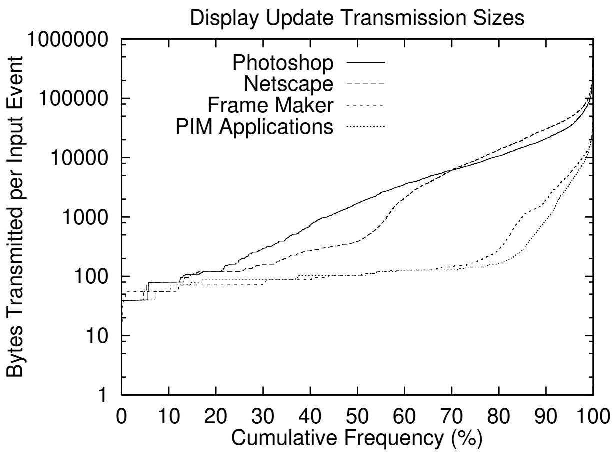

Figure 5 Cumulative distributions of SLIM protocol data transmitted per user input event. Input events are defined as keystrokes and mouse clicks, and the histogram bucket size is 1 byte.

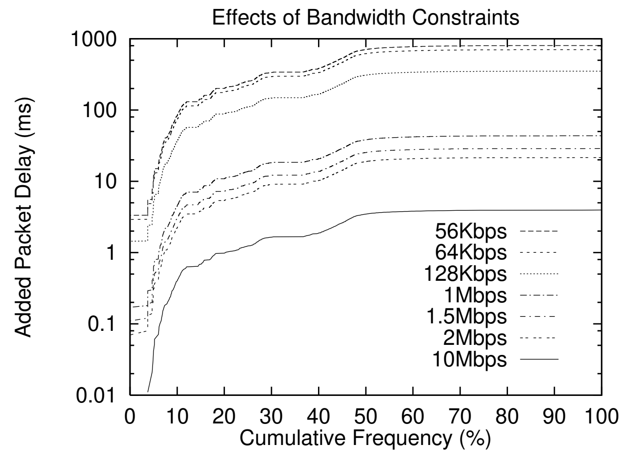

Figure 6 Cumulative distributions of added packet delays for Netscape traces of SLIM protocol commands captured at 100Mbps and retransmitted on simulated networks with lower bandwidths. Bandwidth is averaged over 50ms intervals, and the histogram bucket size is 0.01ms.

tested the system with 1–2Mbps bandwidth limits. Performance was quite good, with only occasional hiccups when large regions had to be displayed. Finally, to simulate low-speed home connections such as ISDN or telephone modems, we tested the system with 56–128Kbps bandwidth limits. We found performance to be extremely poor. It is possible to accomplish tasks such as reading e-mail or editing text, but the experience is painful. Of course, the SLIM protocol was not designed for such low-bandwidth connections, and optimizations like header compression and batching of command packets could have a dramatic effect.

To quantify this experience, we used the protocol logs from the resource profiles mentioned above and simulated transmitting the packets over lower bandwidth connections. We chose Netscape as a representative example and recorded the packet transmission delays in excess of the delays experienced at 100Mbps. Figure 6 depicts the cumulative distributions for a variety of bandwidth levels.

At 10Mbps the added packet delays are always less than 5ms, which is well below the 50–150ms threshold of human tolerance. The added packet delays at 1–2Mbps approach 50ms, entering the realm where humans would notice but consider acceptable. However, at 56–128Kbps we see a sharp increase in packet delays beyond 100ms, indicating that response time would be unacceptably high.

5.5 SLIM protocol processing costs

A key goal of SLIM is to minimize the resources on the desktop. However, reading protocol commands from the network and decoding them for display incurs processing overhead. The CPU is mostly idle until a display command arrives, followed by a burst of activity which entirely consumes the processor. To analyze this cost, we monitored the service time on the Sun Ray 1 prototype console for the SLIM protocol commands sent as part of each display update during the user studies described above. In Figure 7 we report the cumulative distributions of these display latencies.

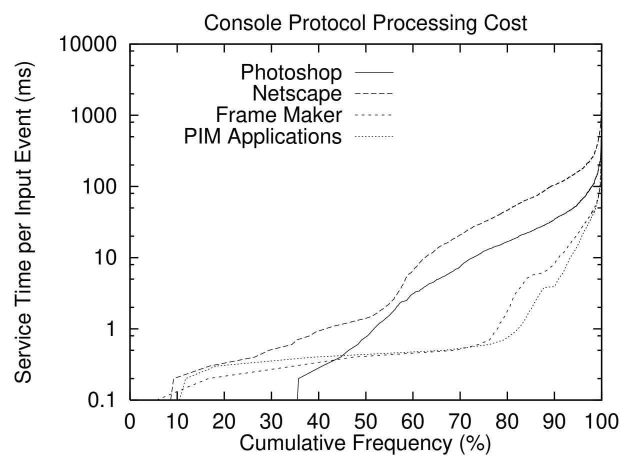

Figure 7 Cumulative distributions of display update service times on the Sun Ray 1 prototype console. Histogram bucket size is 0.1ms.

The important thing to note in this graph is that response time is almost always below the threshold of perception, i.e. in 80% of all cases service time is below 50ms. A small fraction of service times exceed 100ms and may be noticeable, but note from Figure 3 that there are correspondingly large display updates, for which human tolerance is typically higher.

The total service time for a display update is the combination of these console service times and the network transmission delays discussed in Section 5.3. As we can see, the network adds little extra cost, and the total service time is quite short. Thus, we conclude that the simple, low-performance microSPARC-IIep is more than adequate to meet the demands of these applications. In fact, an even lighter weight implementation could possibly be used, and a combined processor and memory implementation of the SLIM console could provide exceptional performance at an extremely low cost point.

Finally, although we have focussed on the cost of decoding protocol messages on the desktop, the server also incurs overhead due to encoding pixel values and generating protocol messages. However, this overhead is extremely low, accounting for a mere 1.7% of the X-server's total execution time when servicing the benchmark applications. Thus, we conclude that the added cost of protocol processing to send the pixel updates is marginal compared to the savings in bandwidth it affords.

5.6 Average bandwidth requirements

To summarize the results from this section and put them in perspective in relation to other thin-client architectures, we calculated the network bandwidth requirements for our set of benchmark applications under three protocols. In Figure 8 we present the average network bandwidth for each application using the X protocol, the SLIM protocol, and a simple protocol in which all the changed pixels are transmitted (labelled "Raw Pixels" in the chart).

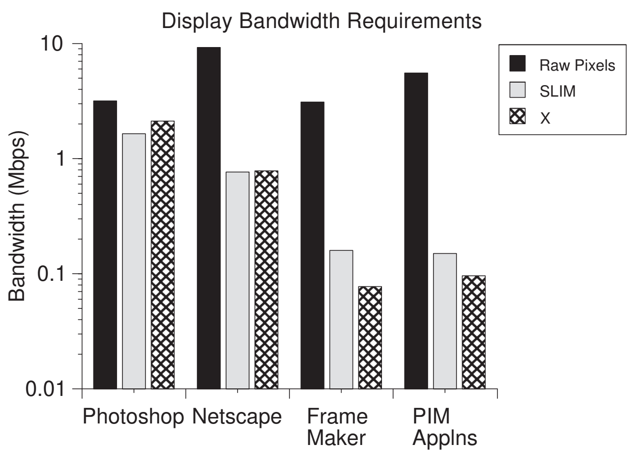

Figure 8 Average bandwidth consumed by the interactive benchmark applications under the X, SLIM, and raw pixel update protocols.

There are two important points to observe in this graph. First, it is quite interesting to note that the X and SLIM protocols have similar bandwidth requirements. X performs slightly better on the Frame Maker and PIM applications which were, in fact, the classes of programs for which X was optimized. However, this is insignificant because the bandwidth requirements of these programs are so low. On the other hand, Photoshop and Netscape represent a new class of applications in which image display is the common operation, and they require an order of magnitude more bandwidth. In these cases, SLIM outperforms X, and the absolute size of the bandwidth differential is substantially larger than for the other applications. We thus conclude that the SLIM protocol is at least competitive with X in terms of bandwidth requirements while providing a much simpler, lower-level interface which requires substantially less computing resources.

The second item to note is that the overall bandwidth requirements are quite small. We have already seen that network-induced delays do not adversely affect interactive performance, and this result demonstrates that network occupancy is also low. This implies that there is substantial opportunity to share the interconnection fabric which we discuss in the following section.

6 Interactive performance for multiple users

The key advantage of SLIM and other thin-client architectures is the savings provided by sharing computational resources. In this section we explore how interactive performance changes when there is interference from other simultaneously active users.

Many engineering workgroups have servers configured to support the requirements of their demanding computational activities, and we will continue to have these compute servers for executing long-running jobs in a SLIM system. Here we are interested mainly in the sharing of machines for running those applications typically executed on our desktops.

6.1 Sharing the server processor

We want to determine how many active users a server can support while still providing good interactive performance. As discussed in Section 3, this is difficult to determine and is a subjective assessment that may be different for each individual. Because human perception is relatively slow, users can tolerate fairly large response times. Thus, a processor can be oversubscribed while still providing good interactive performance. To account for this type of scenario, we must model the system in overload. However, large-scale user studies are impractical, and script-based techniques will fail due to timing constraints that cannot be met.

Our approach is to use a load generator to simulate active users. In addition to the traffic logs mentioned above, we supply the load generator with detailed per-process resource usage information, including CPU and memory utilization. These resource profiles were collected during the user studies with a tool that samples the number of CPU cycles consumed and physical memory occupied by each process at five-second intervals. This tool is similar to that presented in [20], and its overhead per analyzed process was measured and found to be insignificant. The load generator reads a resource usage profile and mimics its consumption of CPU, memory, network, disk and other resources over time. It does not replay the recorded X commands, SLIM commands, or other high-level operations. It merely utilizes the same quantity of resources in each time interval as the original application did. In this way, we can produce the correct level of background load even when the resources are oversubscribed.

While the simulated user loads are running, we use an application with well-known properties and requirements as a yardstick to gauge interactive performance. This application repeatedly consumes 30ms of dedicated CPU time to simulate event processing, followed by 150ms of "think time." We defined the application in this way so that it would be more demanding than any other interactive application. In particular, it requires nearly 17% of the server, which is greater than the average CPU requirement for Photoshop (14%), Netscape (13%), Frame Maker (8%), and PIM (3%). Also, the interrupt rate is equivalent to a fast typist and is well beyond the sustained rates for any of our benchmarks. When the system becomes overutilized, the amount of real time needed to process the simulated event will exceed 30ms. We measure the value of this added delay as we increase the number of simulated active users.

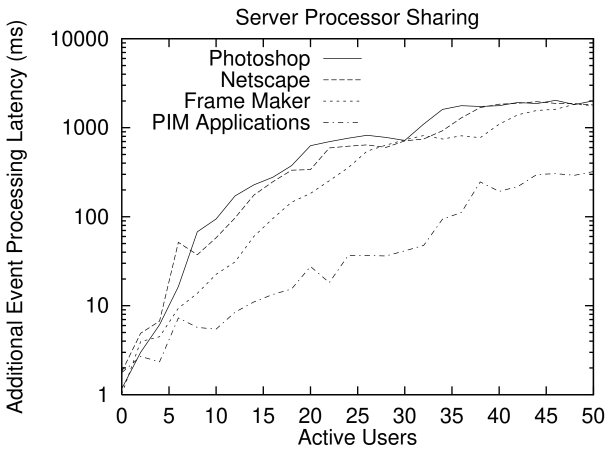

Figure 9 Average latency added to 30ms processing time for events occurring at 150ms intervals as the number of active users increases (1 active CPU). User processor loads are generated with a trace-driven simulator which "plays back" previously recorded resource usage profiles.

The experimental setup is listed in Table 3, and the server had a single processor enabled. We modelled both CPU and memory loads for the active users, and the results are presented in Figure 9. To put them in perspective, we ran the GUI-based, benchmark applications ourselves under the experimental conditions and found that when the added delay on the yardstick application reached around 100ms, interactive performance was noticeably poor. As we can see from the average CPU requirements listed above, the processor on the server is significantly oversubscribed at this point. This is important because it demonstrates that a system can provide good performance for interactive applications even when the processor is fully utilized. In particular, we could tolerate up to about 10-12 simultaneously active Photoshop users, 12–14 Netscape users, 16–18 Frame Maker users, or 34–36 PIM users on the server (in addition to the yardstick application). Of course, other people will have different tolerance limits and application mixes, but this experiment helps quantify how well a system will perform in a multi-user environment.

Another issue we wanted to investigate was how well the system would scale as processors were added to the server. As the number of CPU's increases, contention for shared caches and memory bus bandwidth also increases, potentially reducing the scalability of the system. So, we chose Netscape as a representative application and used the same setup described above. We varied the number of active CPU's in the system from 1 to 8, with a corresponding increase in the number of active users. Figure 10 presents the results, which we normalized by reporting the number of active users per processor.

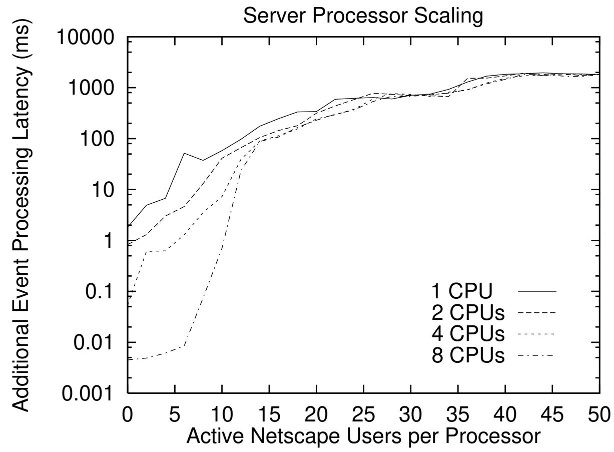

Figure 10 Average latency added to 30ms processing time for events occurring at 150ms intervals as the number of active Netscape users increases (1–8 active CPU's). User processor loads are generated with a trace-driven simulator which "plays back" previously recorded resource usage profiles.

We can see from the graph that the system scales quite well with no obvious contention effects. In addition, note that when the system has a relatively smaller number of active users per CPU, configurations with more processors outperform those with less. The reason is that a multiprocessor system is better able to find a free CPU when one is required.

6.2 Sharing the interconnection fabric

Although the preferred embodiment of the SLIM interconnection fabric is a dedicated private network, significant reductions in cost are possible by sharing bandwidth on the IF. To analyze the cost of sharing the SLIM interconnection fabric, we again used the load generator discussed above to play back the network portion of the resource profiles in order to simulate active users on the IF. While the background traffic is being generated, we used an application with well-known properties as a yardstick to gauge interactive performance. This application simulates a highly interactive user with sizeable display updates by repeatedly sending a 64B command packet to the server followed by a 1200B response and then 150ms of "think time." We measure the average round-trip packet delay as the number of active users is increased.

The experimental configuration is listed in Table 3. We used three workstations: one to act as a SLIM console, one to serve as a sink for background SLIM traffic, and one to act as a server. The machine which played the role of an active console executed the application described above and recorded round-trip packet delays for the network traffic. The machine acting as a server used the load generator to send background SLIM traffic to the sink host and responded to incoming packets from the simulated console. All workstations were connected to a network switch. In this way, the link to the server was shared by both the measured and background traffic, making it the point of contention in the system. Figure 11 presents the results for our benchmark applications.

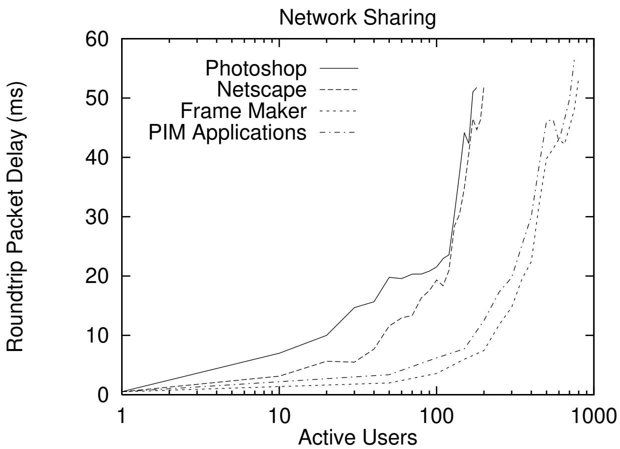

Figure 11 Network round-trip latency for 64B upstream packet followed by 1200B downstream packet. User traffic loads are generated with a trace-driven simulator which "plays back" previously recorded resource usage profiles.

To put these results in perspective, we again ran the benchmark applications for ourselves under the experimental conditions. We found the system to be quite usable until packet delays for the test application hit about 30ms, at which point response time suffered greatly and packet loss became a problem. Thus, we could tolerate up to about 130–140 simultaneously active Photoshop or Netscape users, or about 400–450 Frame Maker or PIM users on a shared network. This is interesting because it demonstrates that the network is a less critical resource than the processor, memory, or swap space on the server.

6.3 Case studies

Since the Sun Ray 1 is a commercial product, we have had the opportunity to monitor its use in real-world settings. At two installations we used standard resource monitoring tools (e.g. ps, netstat, vmstat) to collect performance data continuously over the course of several days. Snapshots of aggregate CPU load, aggregate network bandwidth usage, and number of active users were taken every 10 seconds throughout the day. For each site, Figure 12 presents the day-long profile which has the highest average processor load and the highest number of active users. We report the maximum value recorded in each five-minute period.

The first graph presents a load profile from a university lab which was previously supported by a collection of X terminals. Students work on programming assignments and other normal computing tasks, and major applications include MatLab, StarOffice, Netscape, e-mail and compilers. The server is a Sun Enterprise E250 with two 400MHz UltraSPARC-II CPU's, 2GB of RAM, a 1Gbps link to the IF, 13GB of swap space, and the Solaris 2.7 operating system. There are 50 terminals attached to the server, and at the busiest part of the day many are in use (Total Users in the graph). However, far fewer users are actively running jobs on the system (Active Users in the graph). Both processors reach full utilization during peak periods. However, aggregate network load is below 5Mbps, making the 1Gbps uplink massive overkill.

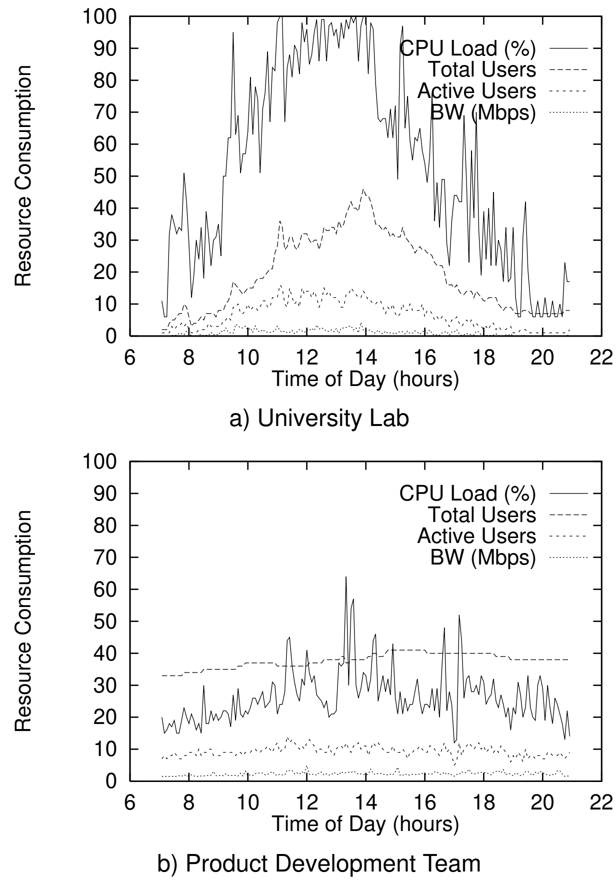

Figure 12 Day-long plot of aggregate CPU utilization, network bandwidth, and active users for real-world installations of the Sun Ray 1 system.

The second graph presents a load profile from our computer product development team. Engineers and other staff use CAD tools for hardware design, text editors and compilers for software development, as well as Netscape, StarOffice, Citrix MetaFrame, e-mail, calendar, etc. for office productivity. The server is a Sun Enterprise E4500 with eight 336MHz UltraSPARC-II CPU's, 6GB of RAM, a 1Gbps link to the IF, 13GB of swap space., and the Solaris 2.7 operating system. Across two buildings, there are over 100 terminals attached to the server, and many are in constant use (with a smaller fraction in active use). Users in this group leave their sessions active and frequently utilize the mobility feature via the smart card. Again, aggregate network load is below 5Mbps. The server processors are never fully occupied, and there is certainly sufficient headroom to scale down the server. A major benefit of this system is the reduction of administration cost from the more than 100 workstations previously used by this group to the single, shared server.

7 Multimedia applications

Real-time applications, such as video or 3D-rendered games, represent an important class of applications that place high demands on system resources in order to provide desired performance. As such, these kinds of applications have traditionally been viewed as totally unsuitable for remote display systems; their bandwidth and processing requirements are expected to be prohibitive. In addition, these kinds of applications differ from the ones studied previously because they have updates that occur at high rates, affect many pixels, and are not tied to human interaction. Thus, they represent the worst-case scenario for a SLIM system.

Multimedia applications typically render directly to the frame buffer in order to maximize performance. To achieve the same effect in a SLIM system, the application must utilize the SLIM protocol directly. This means that there is a porting cost and implementation overhead, and as we will see below, the added overhead can be quite high. However, it is fair to point out that reducing the resolution of the media streams and scaling them locally on the SLIM console reduces this extra cost to a manageable level, thereby enabling applications to achieve display rates well beyond human tolerance limits with little or no noticeable degradation.

Also, because SLIM desktop units have so little resources, we must ensure that multimedia applications do not starve other applications for protocol processing service on the console. Therefore, we implemented a simple network bandwidth allocation mechanism on the Sun Ray 1 console. Under this scheme applications (e.g. the X-server or Quake) executing on possibly different servers make bandwidth requests to the display console based on their past needs. These requests are transparent to the application programmer, as they are made by the X-server on behalf of X applications and by the video library (see Section 2.2) on behalf of multimedia programs. The console sorts the requests in ascending order and grants them one at a time until a request exceeds the available bandwidth, at which point all remaining requests are granted a fair share of the unallocated bandwidth. In this way, high-demand multimedia applications can run while other traditional applications still receive good interactive service from the console.

To analyze SLIM's suitability for these applications, we experimented with three multimedia applications: an MPEG-II player, an NTSC video player, and Quake, an interactive 3-D game from id Software. Using the software library mentioned in Section 2.2, we enhanced these applications with the ability to display directly on Sun Ray 1 consoles via the SLIM protocol. The applications transmit synchronized audio, and they have similar real-time quality of service demands. More specifically, humans require a frame rate of at least 24Hz to achieve smooth display and proper motion rendition. Quake has the additional requirement of timely user interaction.

Although Quake is a 3-D game, it does not use standard 3-D interfaces, such as OpenGL. Thus, we use it more to assess interactive game performance than to analyze rendering performance for 3-D graphics. Since Sun Ray 1 consoles have no hardware acceleration for 3-D operations, 3-D graphics performance is directly related to the speed of the server processor anyway.

Using the test configuration listed in Table 3, we demonstrate that despite substantial resource requirements, even high-demand applications are supported by the SLIM architecture. In particular, the bandwidth and processing capabilities of the consoles are more than sufficient to meet the demands of these applications, and it turns out that server performance is the primary bottleneck. Of course, it is important to note that in all these cases, a 10Mbps interconnection fabric would not be able to provide adequate performance.

7.1 MPEG-II player

Support was added to Sun's ShowMeTV video player to utilize the SLIM protocol, and we use it for stored video playback. In this experiment, we selected an MPEG-II clip with a resolution of 720x480 and used the CSCS command with 6 bits/pixel compression. We extract frames from the MPEG-II codec at the point where they are in YUV format, and the SLIM video library is then used to transmit them to the desktop.

This application nearly consumes an entire CPU. Disk I/O and video decompression on the server are the performance-limiting factors. The displayed frame rate was fairly uniform at 20Hz (roughly 40Mbps), which is quite good, given the resources available on the desktop. Still, full frame rate (30Hz for this clip) can be achieved by sending every other line and scaling at the desktop. Degradation is not noticeable for the most part, and the bandwidth is reduced by half.

7.2 Live NTSC video

To test live video, we used a custom application which obtains JPEG-compressed NTSC fields from a video capture card, decompresses them up to the point where YUV data are available, and transmits them to the desktop via the CSCS command. Since NTSC is interlaced, we can capture only 640x240 fields and scale up to full size (640x480).

The decompression operation fully consumes the processor, and this application is not multi-threaded. Thus, the server CPU is the performance-limiting factor, and the displayed frame rate ranges from 16Hz to 20Hz (roughly 19–23Mbps), depending on characteristics of the video. To create a situation in which protocol processing on the console is the bottleneck, we simulate 4-way application-level parallelism by simultaneously executing four half-size (320x240) players. In this case, we observe a display frame rate of 25–28Hz (59–66Mbps). Thus, if we were to multi-thread this application, it would display full-size video which is quite watchable (at the cost of four heavily-utilized CPU's).

7.3 Quake

We use Quake as a representative example of 3D-rendered games. Running it under X produces poor results, and the key to increasing performance would be to use the YUV color space directly in the application. However, we only had access to the code which puts pixels on the display, and that would be a costly porting operation anyway. So, the next best option is to add a translation layer which converts frames to a format suitable for use by the SLIM CSCS protocol command.

When the game engine renders a frame, it produces 8-bit, indexed-color pixels. Our translation calculates a YUV lookup table based on the RGB colormap. To display a frame, we convert the 8-bit pixel values to 5-bit YUV data via table lookup and color component subsampling. Then, the frame is transmitted to a Sun Ray 1 console.

When we run Quake at a resolution of 640x480, the display rate ranges from 18Hz to 21Hz (roughly 22–26Mbp), depending on scene complexity. Although this is good performance, we found the interactive experience to be somewhat lacking. However, if we run Quake at 3/4 resolution (480x360), display rate improves to 28–34Hz (roughly 20–24Mbps), which we found to be playable despite occasional hiccups. The performance-limiting factor in these cases was the CPU, due almost exclusively to the high cost of translation. For example, at the 640x480 resolution it took roughly 30ms/frame to do the YUV conversion and 13ms/frame to transmit the data, which means that the upper bound on display rate was only about 23Hz.

To further improve performance, we could parallelize the application. Because we had limited access to source code, we simulated the effect of parallelizing an instance of the game at full resolution (640x480) by running four instances at 320x240 resolution, thereby creating a situation in which the limiting factor was the protocol processing performance of the desktop unit. With this setup we achieved frame rates ranging from 37Hz to 40Hz (roughly 46–50Mbps), which we found to be smooth and responsive with no hiccups or other noticeable effects.

8 Related work

There are a wide range of lightweight computing devices that have been developed, including network computers, JavaStations, the Plan 9 Gnot, and network PC's. These machines all differ from the SLIM desktop unit in a significant manner. Although they are low-resource implementations, they are full-fledged computers executing an operating system and windowing software. In this section, we compare SLIM to other thin-client systems and discuss how they differ.

8.1 X window system

Because we use X-based applications in our studies, we are able to make the most direct comparison with the X window system. The X protocol[13] was designed to be fairly high-level, with the goal of minimizing bandwidth by expending computing resources locally. SLIM, on the other hand, is designed to minimize local processing, at the cost of a potentially higher bandwidth overhead. Our results in Figure 8, however, show that both approaches are highly competitive. That is, while the SLIM protocol is much simpler and can be serviced by a low-cost processor, it requires roughly the same bandwidth as X for most applications. For higher-bandwidth, image-based applications, SLIM has a significant advantage over X.

In addition, X terminals are not well-suited to running multimedia applications. Under the X protocol, each frame would have to be transmitted using an `XPutImage` command with no compression possible, i.e. a full 24 bits must be transmitted for each pixel. The situation is different for SLIM. In the worst case, it is the same as X, but typically, some form of compression is possible. If the SLIM CSCS command is not being used, there is still opportunity for finding redundancy among pixels which can reduce bandwidth somewhat. Using the CSCS command, at most 16 bits are required per pixel (a 33% bandwidth reduction) and compression up to 5 bits per pixel (an 88% bandwidth reduction) is possible by altering the color-space conversion parameters. If video is scaled to a higher resolution, the server must perform the operation prior to sending the image to an X terminal, whereas the Sun Ray 1 console can do it locally at substantial savings in bandwidth. For example, transmitting a half-size video stream to a SLIM console and scaling it to full-size would require roughly 18Mbps of bandwidth. However, over 105Mbps of bandwidth would be needed under X. Thus, SLIM and X are extremely competitive at low bandwidth, while SLIM provides substantial savings on higher bandwidth operations, where its impact is much more significant on overall performance.

8.2 Windows-based terminals

Windows-based terminals (WinTerms) employ the Citrix ICA protocol[3] or the Microsoft RDP protocol[5][10] to communicate with a server running Windows NT, Terminal Server Edition. These protocols are quite similar in nature to X but are tied to the Windows GUI API. On the other hand, the low-level SLIM protocol has no such bias and can be used by a system with any rendering API. Another difference between them and the SLIM protocol is that they are highly optimized for low-bandwidth connections. This is accomplished via a variety of techniques, including Windows object-specific compression strategies (e.g. run-length coding of bitmaps), caching of Windows objects and state at the terminal, and maintaining backing store at the terminal. Because the resources included in the terminals directly determine the performance and bandwidth savings possible, these types of systems can have expensive terminals which constantly require upgrades to improve performance. In addition, multimedia capabilities are limited, but because the protocols are extendable, extra support could be added.

8.3 VNC

Like SLIM, the Virtual Network Computing[15][16] (VNC) architecture from Olivetti Research uses a protocol which is independent of any operating system, windowing system, and application. The protocol commands are similar to SLIM, but VNC is designed to access the user's desktop environment from any network connection through a variety of viewers, including a web browser over the internet. The Sun Ray 1 implementation, however, is limited to operating in a workgroup setting with a direct connection to the server.

The key difference between the two approaches, though, is the manner in which the display is updated. With the Sun Ray 1, updates are transmitted from the server to the consoles as they occur in response to application activity. VNC, on the other hand, uses a client-demand approach. Depending on available bandwidth, the VNC viewer periodically requests the current state of the frame buffer. The server responds by transmitting all the pixels that have changed since the last request. This helps the system scale to various bandwidth levels, but has the drawback of larger demands on the server in the form of either maintaining complex state or calculating a large delta between frame buffer states. In either case, our experience with the system is that even in low-latency, high-bandwidth environments, VNC is fairly sluggish. In addition, VNC includes no support for multimedia applications but leaves open the possibility of including specialized compression techniques in the future.

8.4 Other remote-display approaches

The MaxStation[9] from MaxSpeed Corporation is a terminal-based system which uses peripheral cards installed in a PC server machine to transmit video directly to attached consoles. Monitors are refreshed over a 64Mbps connection, but the bandwidth limit restricts the maximum resolution that can be supported to 1024x768 8-bit pixels. The use of expansion cards to create dedicated connections is much less flexible and more difficult to maintain than a commodity switched network, like the Sun Ray 1 uses for the IF.

The Desk Area Network[2][8] (DAN) is a multimedia workstation architecture which uses a high-speed ATM network as the internal interconnect. The frame buffer is a network-attached device which reads and displays pixel tiles. The frame buffer is similar to a Sun Ray 1 desktop unit, but the DAN architecture was designed as a dedicated, stand-alone workstation with sufficient internal bandwidth to transmit high-volume multimedia data (such as video streams) between system components.

9 Conclusions and future work

The SLIM system is a new thin-client architecture. The desktop unit is a stateless, low-cost, fixed-function device that is not much more intelligent than a frame buffer. It requires no system administration and no software or hardware updates. Different applications and operating systems can be ported to display on a SLIM console by simply adding a virtual device driver for the SLIM protocol to their rendering API's. In addition, the communication requirements are modest and can be supported by commodity interconnection technology.

This paper makes three important contributions. First, we developed an evaluation methodology to characterize interactive performance of modern systems. Second, we characterized the I/O properties of real-world, interactive applications executing in a thin-client environment. Finally, via a set of experiments on an actual implementation of the SLIM system, we have shown that it is feasible to create an implementation which provides excellent quality of service. More specifically, we demonstrated the following:

- Using a dedicated network ensures that added round-trip latency (less than 550µs) between the desktop and the remote server is so low that it cannot be distinguished from sitting directly at the console local to the server.
- By only transmitting encoded pixel updates, network traffic requirements of common, interactive programs are well within the capabilities of modern 100Mbps technology. In fact, 10Mbps is more than adequate, and even 1Mbps provides reasonable performance.
- The low-level SLIM protocol is surprisingly effective, requiring a factor of 2–10 less bandwidth to encode display updates than sending the raw pixel data. Further, its bandwidth demands are not much different from the X protocol's, i.e. the simplicity of the protocol did not result in the expected increase in bandwidth cost.
- Yardstick applications quantified multi-user interactive performance and showed that substantial amounts of sharing are possible, e.g. the yardstick application and 10 to 36 other active users can share a processor with no noticeable degradation. Network sharing is an order of magnitude larger than processor sharing.
- The consoles are more than adequate to support even high-demand multimedia applications, and server performance turns out to be the main bottleneck. High-resolution streaming video and Quake can be played at rates which offer a high fidelity user experience.

Still, many research topics remain to be addressed before we can realize the full potential of an ultra-thin architecture like SLIM. Further research is necessary to provide interactive performance guarantees in a shared environment. As we move to a model where there are a large number of users sharing a large number of servers, we may need to rethink the design of the operating system to create a truly scalable, secure, heterogeneous system that fully exploits the mobility and resource sharing ability in the architecture.

Acknowledgments

The Sun Ray 1 implementation would not have been possible without the hard work of the prototype development team: Jim Hanko, Jerry Wall, Alan Ruberg, Lawrence Butcher, and Marc Schneider. We would also like to thank the members of the Sun Ray 1 product team. The experiments would not have been possible without a great deal of technical assistance from and helpful discussions with Jim Hanko and Jerry Wall. Thanks are also due to the many people who participated in our user studies. We would like to thank Sunil Marangoly, Kent Peacock, Kevin Pon, Felicia Stanger and our customers for their help with system resource monitoring. Finally, we would like to thank Sun Microsystems Inc. for its financial support of this study.

References

[1] T. Anderson, D. Culler, and D. Patterson, "A Case for Networks of Workstations: NOW," *IEEE Micro*, 15(1), February 1995, pp. 54–64.

[2] P. Barham, M. Hayter, D. MacAuley, and I. Pratt, "Devices on the Desk Area Network," *IEEE Journal on Selected Areas in Communications*, 13(4), May 1995, pp. 722–32.

[3] Boca Research, Inc., "Citrix ICA Technology Brief," *Technical White Paper*, Boca Raton, FL, 1999.

[4] Compaq Computer Corporation, "Performance and Sizing of Compaq Servers with Microsoft Windows NT Server 4.0, Terminal Server Edition," *Technology Brief*, Houston, TX, June 1998.

[5] Databeam Corporation, "A Primer on the T.120 Series Standard," *Technical White Paper*, Lexington, KY, May 1997.

[6] A. Dearle, "Toward Ubiquitous Environments for Mobile Users," *IEEE Internet Computing*, January-February 1998, pp. 22–32.

[7] Y. Endo, Z. Wang, J. Chen, and M. Seltzer, "Using Latency to Evaluate Interactive System Performance," *Symposium on Operating System Design and Implementation*, October 1996, pp. 185–199.

[8] M. Hayter and D. McAuley, "The Desk Area Network," *Operating Systems Review*, 25(4), October 1991, pp. 14–21.

[9] Maxspeed Corporation, "Ultra-Thin Client — Client/Server Architecture: MaxStations under Multiuser Windows 95," *Technical White Paper,* Palo Alto, CA, 1996.

[10] Microsoft Corporation, "Comparing MS Windows NT Server 4.0, Terminal Server Edition, and UNIX Application Deployment Solutions," *Technical White Paper*, Redmond, WA, 1999.

[11] J. Nieh and M. Lam, "The Design, Implementation and Evaluation of SMART: A Scheduler for Multimedia Applications," *Operating Systems Review*, 31(5), December 1997, pp. 184–97.

[12] J. D. Northcutt, J. Hanko, G. Wall, A. Ruberg, "Towards a Virtual Display Architecture," *Project Technical Report SMLI 98-0184*, Sun Microsystems Laboratories, 1998.

[13] A. Nye (Ed.), *X Protocol Reference Manual*, O'Reilly & Associates, Inc., 1992.

[14] R. Pike, D. Presotto, K. Thompson, and H. Trickey, "Plan 9 from Bell Labs," *Proceedings of the Summer 1990 UKUUG Conference*, July 1990, pp. 1–9.

[15] T. Richardson, F. Bennet, G. Mapp, and A. Hopper, "Teleporting in an X Window System Environment," *IEEE Personal Communications*, No. 3, 1994, pp. 6–12.

[16] T. Richardson, Q. Stafford-Fraser, K. Wood, and A. Hopper, "Virtual Network Computing," *IEEE Internet Computing*, January–February 1998, pp. 33–38.

[17] B. Shneiderman, *Designing the User Interface: Strategies for Effective Human-Computer Interaction, 2nd edition*, Addison-Wesley, Reading, MA, 1992.

[18] M. Schneider and L. Butcher, "NewT Human Interface Device Mark II Terminal Hardware Specification," *Project Technical Report SMLI 98-0200*, Sun Microsystems Laboratories, Palo Alto, CA, 1998.

[19] Sun Microsystems Inc., "Java Server Sizing Guide," *Technical White Paper*, Palo Alto, CA, October 1997.

[20] A. Tamches and B. Miller, "Fine-Grain Dynamic Instrumentation of Commodity Operating System Kernels," *Symposium on Operating Systems Design and Implementation*, February 1999, pp. 117–30.

[21] Unisys, "Sizing and Performance Analysis of Microsoft Windows NT Server 4.0, Terminal Server Edition, on Unisys Aquanta Servers," *Technical White Paper*, Blue Bell, PA, August 1998.