

Proceedings of 2000 USENIX Annual Technical Conference

San Diego, California, USA, June 18–23, 2000

OPERATING SYSTEM SUPPORT FOR MULTI-USER, REMOTE, GRAPHICAL INTERACTION

Alexander Ya-li Wong and Margo Seltzer



© 2000 by The USENIX Association

All Rights Reserved

For more information about the USENIX Association:

Phone: 1 510 528 8649

FAX: 1 510 548 5738

Email: office@usenix.org

WWW: <http://www.usenix.org>

Rights to individual papers remain with the author or the author's employer.

Permission is granted for noncommercial reproduction of the work for educational or research purposes.

This copyright notice must be included in the reproduced paper. USENIX acknowledges all trademarks herein.

Operating System Support for Multi-User, Remote, Graphical Interaction

Alexander Ya-li Wong, Margo Seltzer

Harvard University, Division of Engineering and Applied Sciences

aywong@aywong.com, margo@eecs.harvard.edu

Abstract

The emergence of thin client computing and multi-user, remote, graphical interaction revives a range of operating system research issues long dormant, and introduces new directions as well. This paper investigates the effect of operating system design and implementation on the performance of thin client service and interactive applications. We contend that the key performance metric for this type of system and its applications is user-perceived latency and we give a structured approach for investigating operating system design with this criterion in mind. In particular, we apply our approach to a quantitative comparison and analysis of Windows NT, Terminal Server Edition (TSE), and Linux with the X Windows System, two popular implementations of thin client service.

We find that the processor and memory scheduling algorithms in both operating systems are not tuned for thin client service. Under heavy CPU and memory load, we observed user-perceived latencies up to 100 times beyond the threshold of human perception. Even in the idle state, these systems induce unnecessary latency. TSE performs particularly poorly despite scheduler modifications to improve interactive responsiveness. We also show that TSE's network protocol outperforms X by up to six times, and also makes use of a bitmap cache which is essential for handling dynamic elements of modern user interfaces and can reduce network load in these cases by up to 2000%.

1 Introduction

Continuing improvements in networking speed, cost, standardization, and ubiquity have enabled a literal “explosion” of the traditional computer system architecture. Network links are now replacing an increasing number of internal system busses, allowing the processor, memory, disk, and display subsystems to be spatially extruded throughout a network. Well-known examples of extruded subsystems include distributed shared memory and network file systems [17, 12]. While these are established areas of research, much less of the literature addresses systems that enable extrusion of the display and input

subsystems that interface with human users.

Such functionality is commonly referred to as *thin client* computing. Driven by IT concerns over cost and manageability, the thin client trend has triggered renewed interest in X Windows-like schemes and the introduction of thin client service into major commercial operating systems. This trend will accelerate as consumer products such as personal digital assistants, cellular phones, pagers, and hand-held e-mail devices evolve and converge into a yet another class of thin client terminals, these additionally being wireless, mobile, and ubiquitous.

We explore the question of how, in the current revival of interactive timesharing, underlying operating system design impacts thin client service, and how current metrics are inadequate for capturing the relevant impacts. The contributions of this paper are an approach for analyzing the performance and scalability of thin client server operating systems, a quantitative comparison and analysis of Windows NT, Terminal Server Edition and Linux with X Windows, two popular implementations, and a discussion of related work informed by our approach.

The balance of this paper is organized as follows. Section 2 gives background on the X Windows System and Windows NT, Terminal Server Edition. Section 3 describes our approach. Sections 4, 5, and 6 discuss the processor, memory, and network in turn as resources within our framework. In Section 7, we discuss related work on thin client performance, and we conclude in Section 8.

2 TSE and Linux/X Windows

For the remainder of this paper, we will use conventional client/server terminology although it is backwards from the X Windows use. In particular, we will refer to the machine in front of the user as the client (although in X Windows the X server runs on this machine) and the machine on which the application runs the server (although X Windows clients, such as *xterm*, run on this machine).

TSE and X Windows share similar architectures. Applications running on the server request display updates and receive inputs from OS-provided abstractions. In X

Windows, this is the Xlib GUI library, and in TSE, the Win32 GUI library. The code underlying these libraries transparently routes display requests and input events either to local hardware or remote client devices over the network. Xlib interaction is entirely user-level, while TSE display requests pass through the kernel.

Multi-user capabilities in TSE are provided by recent modifications to the NT kernel, including virtualization of kernel objects across user sessions, per-session mappings of kernel address space, and code sharing across sessions. Multi-user functionality under X Windows is provided by the underlying Unix implementation. In this paper, we evaluate the performance of X Windows as it runs on top of the 2.0.36 Linux kernel.

The network protocols used to carry display and input information are X for X Windows and the Remote Display Protocol (RDP) for TSE. Both are high-level protocols encoding graphics primitive operations like bitblts and text, polygon, and line drawing.

RDP is implemented in TSE as a display device driver, and so appears to user-level applications as would any other display device, such as a graphics card. RDP also employs compression and client-side bitmap and glyph (text) caching to reduce network traffic. Varying levels of encryption can be applied to the protocol stream over the wire. TSE users run the TSE client software to connect to a TSE server. The client is a 200KB executable which is available for DOS, all Windows platforms, and Unix platforms via third-party add-ons. The RDP specification is not published by Microsoft, making some analyses more difficult. Although it is not within the scope of this paper, the reverse-engineering of RDP is part of our ongoing work.

X Windows is implemented on the server-side as a GUI library called Xlib that is linked into applications designed for the X environment. Such applications communicate over the wire directly and require no special support from the underlying operating system other than a TCP/IP stack. Client machines run so-called “X servers,” which accept connections from X-enabled applications and display their interfaces. X server software is available on most major platforms including Windows, Unixes, and Macintosh.

We also include in our comparisons LBX, which is a protocol extension to X and is implemented as a proxy that lives on both ends of an X Windows connection. It takes normal X traffic and applies various compression techniques to reduce the bandwidth usage of X applications [9]. RDP, X, and LBX all ran over TCP/IP in our experimentation.

3 Our Approach

3.1 The Motivation

Traditional performance metrics in the systems domain do not apply to operating systems whose primary function is thin client service. The output of benchmark suites like Winstone, *lmbench*, and TPC are not particularly enlightening. Ultimately, those interested in deploying interface services need to know the maximum number of concurrent users their servers can support given some hardware configuration and what impact on users yields this maximum value. Each characteristic of thin client service places unique demands on what an appropriate benchmark must measure:

3.1.1 Interactive

Unlike HTTP or database servers for which throughput is critical and response time often reduces to a question of throughput, the primary service in which we are interested is interactive login. Endo et al., argued persuasively that for desktop operating systems like Windows, latency, not throughput, is a key performance criterion [7]. We contend that for thin client servers, latency is not only a key criterion, it is paramount. Any useful metric must yield information on whether the system satisfies the latency demands of users.

3.1.2 Multi-User

Benchmarks designed for single-user operating systems are not appropriate because a single user multi-tasking is not equivalent to multiple users uni-tasking or multi-tasking. On single-user systems, although asynchronous background tasks may consume system resources, system load is still typically limited by the rate at which the human user interacts with the foreground application. Furthermore, on a multi-user system there can be many foreground applications (one for each user), so latency demands must be met by more than just a single process. Furthermore, schemes to improve foreground application performance by favoring them over background processes are thwarted when most, if not all, processes are foreground.

3.1.3 Graphical

In a graphical environment, the user’s primary interaction is visual. Therefore, benchmarks need to consider the delicate tolerances of human perception, particularly with respect to latency.

3.1.4 Remote Access

On single-user systems like Windows 98 and NT Workstation, user interface richness and sophistication consume and are constrained by locally available video subsystem bandwidth. In remote-access environments like TSE and X Windows, the video subsystem at the server is irrelevant and the GUI is instead constrained by network bandwidth, the efficiency of the network protocol, and the video hardware at the client.

3.2 The Key Role of Latency

As discussed above, latency, not throughput, is the key performance criterion for thin client service. A user interacting with an operating system performs a set of operations by sending input and waiting for the system to respond. These are the operations on which the user is sensitive to latency. Previous work has found that tolerable levels of latency vary with the nature of the operation. For example, latency tolerances for continuous operations are lower than for discrete operations, and humans are generally irritated by latencies 100ms or greater [4, 13, 20]. Jitter, or an inconsistent level of latency, is also considered harmful.

The quality of a system can therefore degrade in three ways with respect to latency. First, for a given operation, latency can rise above perceptible levels, and performance suffers as latency continues to increase for that operation. Second, performance suffers as the number of operations that induce perceptible latency increases. Finally, performance suffers when perceptible latency continually changes and is unpredictable. Ideally, a “good” system would meet users’ latency demands for each operation performed and would do so consistently.

In an interface service environment, latency depends on three categories of factors:

3.2.1 Hardware resources

Relevant hardware resources include the processor, memory, disk, and network. Hardware inevitably introduces latency, and slow hardware contributes more. Latency can also be caused by resource scarcity. For example, while free memory remains, data access latency is bounded by the speed of the memory hierarchy level into which the active data set fits. But when physical memory is exhausted and paging to disk begins, average data access latency increases dramatically.

3.2.2 Operating system structure

Operating system design and implementation also influence user-perceived latency. Even bleeding-edge hard-

ware can be sabotaged if it is exposed to users through a poorly designed operating system abstraction. Long input handling code paths, inefficient context switches, bad scheduling decisions, and poor management of resource contention can all contribute to increased latency.

3.2.3 User behavior

User behavior indirectly affects latency through hardware resource limitations. Two classes of users running different application mixes will consume resources at different per-user rates. As concurrent use increases, the class of users with greater per-user resource demands will approach saturation conditions and potential increases in latency more quickly.

3.3 Applying our Approach

In the next three sections, we use these principles to guide a quantitative comparison of TSE and Linux/X Windows. Our analysis cuts along the axis of hardware resources, as we consider the processor, memory and the network in turn. For each resource, we consider the impact of user behavior and how the exercise of various applications generates load. Finally, we consider how load translates into user-perceptible latency, and how that translation is influenced by design characteristics of the operating system. Using this analysis, we highlight shortcomings in current operating system design with respect to thin client service, and suggest new potential directions for operating systems optimization. Some issues, such as network graphics and protocol efficiency, are unique to remote interactive access, while others, such as process and memory scheduling, have not been visited in the literature in quite some time but are critical in establishing a high-performance, thin client computing environment.

3.4 Experimental Testbed

Our testbed consisted of a server, a client, and a network listening host connected by a Bay Networks NetGear EN104TP 10Mbps Ethernet hub. The server was a 333MHz Intel Celeron system with the Intel 440EX AGPset, 96MB SDRAM, a 4GB IDE IBM DCAA-34330 hard disk, and a Bay Networks NetGear FA-310 Ethernet adapter. The client was an Intel Pentium II-400 with the Intel 440BX AGPset, 128MB SDRAM, an 11GB IDE Maxtor 91152D8 hard disk, and a 3Com 3C905B Ethernet adapter. The listening host was an Intel Pentium-233 with the Intel 440TX PCIset, 96MB EDO RAM, a 2GB IDE IBM DTNA-22160 hard disk, and a 3Com 3C589C Ethernet adapter. The server ran, alternately, the Linux 2.0.36 kernel and build 419 of Windows NT, Terminal

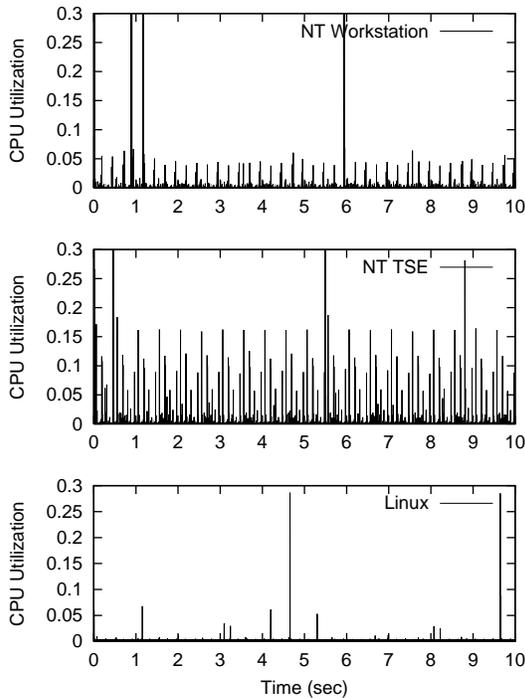


Figure 1: A comparison of the idle-state processor activity over a 10 second period in NT Workstation, TSE, and Linux.

Server Edition. The client system ran Windows 98 as its operating system and established remote access to the server with build 419 of the TSE client and White Pine’s eXodus X-Windows server version 5.6.4. The listening host ran the 2.0.36 Linux kernel. All network adapters were configured for 10Mbps half-duplex operation.

4 Processor

In this section, we examine the latency characteristics of operating system processor abstractions in the context of thin client service. We consider how user behavior generates processor load and then compare each operating system’s ability to minimize the user-perceived latency induced by varying levels of load.

4.1 From Behavior to Load

The addition of thin client support to an operating system has a measurable effect on both its requirements and characteristics with respect to latency.

When a system becomes multi-user (as NT did when Microsoft introduced TSE), it bears the additional burden of meeting human latency requirements for multiple concurrent foreground applications versus just one for single-user operation.

Support for multiple concurrent users produces additional system activity and potential latency increases for user-level applications. Multi-user support typically includes at least one daemon process to listen for and handle incoming session connections and additional per-user kernel state and ownership information. Remote-access support contributes yet more latency. Interface operations previously handled by just the graphics subsystem must now also pass through the network subsystem.

4.1.1 Compulsory Load

These latency contributors, those that are inherent in the operating system, are particularly important because they are behavior-independent. Because multi-user and remote-access support are core functionality required for thin client service, all users, regardless of the applications they use, are subject to at least the minimum latency induced by these components. Any calculation of user-perceived latency must start by measuring this baseline load, which we call “compulsory load.”

Methodology

Endo et al. introduced a novel methodology for measuring user-perceived latency they describe as “measuring lost time” [7]. Using a combination of the Pentium Performance Counters and system idle loop instrumentation, they are able to determine when, and for how long the CPU is busy handling user input events. This yields a method for measuring user-perceived latency with a precision not previously achieved.

Endo et al. employed this technique to generate a set of idle system profiles for three versions of Windows: 95, NT 3.51, and NT 4.0. We use this data as a baseline and use identical methodology to measure the idle system profiles of TSE and Linux for comparison. We discuss the significance of these idle-state profiles as it bears on latency in section 4.2.1.

From Windows NT to TSE

As shown in Figure 1, TSE exhibits greater overall idle-state CPU activity than NT does, demonstrating the increase in compulsory latency caused by the changes made to transform the NT kernel into the TSE kernel. Although Microsoft documentation states that the typical clock interval for NT 4.0 running on a Pentium processor is 15ms, we found, as Endo et al. did, small regular CPU spikes at 10ms intervals in both TSE and NT, suggesting that the clock interrupts are handled every 10ms [5]. This discrepancy is unexplained.

Beyond clock interrupt handling, TSE seems to perform a number of other activities at regular intervals that NT does not. These can be attributed to the addition of the Terminal Service and Session Manager which

listen for and handle incoming client connections, and whatever additional overhead there is in the idle-state for per-session state management in the NT Virtual Memory, Object, and Process Managers.

X Windows on Linux

Unlike TSE, Unix has long had multi-user and local and remote graphical display capabilities courtesy of the X Windows System. Unix operating systems can also run in single-user mode, like NT, but seldom do, and usually only for the purposes of crash diagnosis and recovery.

Figure 1 also shows idle-state CPU activity for the Linux kernel running in multi-user mode. Clearly, the Linux kernel spends much less CPU time handling tasks when idle than do either NT or TSE. This contributes less compulsory load, which, as we will see in the next section, translates to less latency.

4.2 From Load to Latency

Next we discuss how system load translates to user-perceived latency. We examine how both compulsory and dynamic load can increase latency and how intelligent operating system design decisions can minimize those impacts.

4.2.1 Compulsory Latency

Figure 2 compares the cumulative latency of the three systems (NT, TSE, and Linux) in the idle state. The bulk of CPU activity under NT is attributable to events that are 100ms or shorter in duration. The TSE idle state sees these same events, plus a number of additional events lasting 250ms and 400ms. Linux, contrastingly, sees few idle events of significant latency. In the aggregate, TSE generates about three times the idle-state load that NT Workstation does, and about seven times that of Linux.

Even when the systems are idle, any user input activity that intersects with these events will experience delay. The scheduler design determines just how much delay there must be. In particular, the quantum (NT parlance for time-slice) is often manipulated to adjust the responsiveness of a system, but we find that the choice of quantum length is something of a “latency catch-22.”

Consider a round-robin scheduler and several non-blocking, ready-to-run threads with equal priority. The longer the quantum, the longer some thread three or four deep in the queue will have to wait until it can run. In contrast, if the quantum is made shorter, this inter-quantum waiting is reduced, but the full, run-to-block execution time of each thread becomes fragmented across more distinct quanta. If there are a large number of threads in the ready queue, then this problem of execution fragmentation can easily overwhelm the benefits of

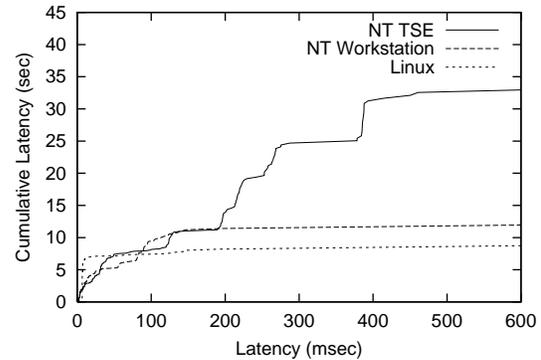


Figure 2: Idle-state profiles of the NT Workstation, TSE, and Linux kernels. The profiles are displayed as cumulative latency histograms. Large changes in the y-axis values indicate a significant contribution to overall latency by the events at the corresponding x-coordinate.

shorter quanta.

Consider a user operation that takes 100ms to complete with no other competing activity. Scheduler quanta on systems like NT and Linux are on the order of 10ms, meaning this operation is fragmented into as many as 10 quanta. If the system is busy with background processing, the original 100ms completion time can be extended considerably. While dynamic priority boosting for GUI-related and foreground threads may help alleviate this problem, it does not help when the competing threads are also foreground and/or GUI-related, as we would often find on a thin client server. Next, we discuss how effectively the TSE and Linux schedulers deal with this problem. For the following discussion, note that greater numeric priorities are better on NT and TSE, while lower are better on Unix systems.

NT and TSE Scheduling

The NT and TSE kernels share the same scheduling code and differ only in their default priority assignments for processes and threads. NT Workstation and TSE both have a 30ms quantum on Intel Pentiums and higher. While TSE is based more directly on NT Server, which has a 180ms quantum, it uses the 30ms quantum found in NT Workstation, ostensibly to improve interactive responsiveness. The default priority level for foreground threads is 9 and for other threads it is 8.

The NT/TSE scheduler implements two mechanisms to further improve user interaction. The first, “quantum stretching,” allows the system administrator to multiply the quantum for all foreground threads. The allowed stretch factors are one, two, and three. The second mechanism is “priority boosting” for waiting GUI threads. GUI threads associated with an interactive ses-

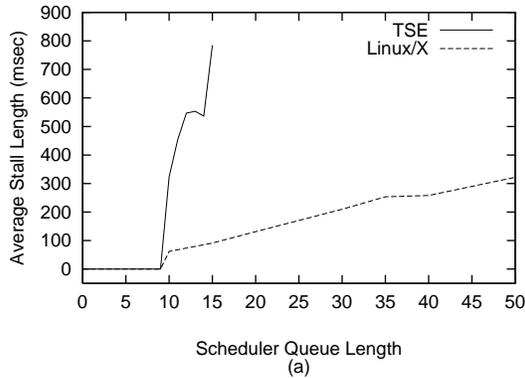


Figure 3: User-perceived keystroke handling latency as a function of server CPU load. Linux/X scales more gracefully than does TSE. TSE became unusable at a load of 15.

sion get their priority boosted to level 15 after waking up to service a user input event. This boost lasts for two quanta.

In TSE, the Session Manager and Terminal Service have a priority of 13. So, if the idle activity we saw in TSE is associated with these processes, GUI thread priority boosting should theoretically prevent this background activity from increasing user-visible latency since the boosted priority for the GUI thread is higher than that of the idle activity (15 vs. 13). However, the boost lasts for only two quanta, and even assuming they are stretched by three, this boost benefits the GUI thread for at most 180ms. After these 180ms, a GUI thread’s priority drops back to 9, and cannot run again until all priority 13 threads yield or block. In the previous section, we saw idle-state events in TSE of up to 400ms. If a GUI thread intersects with such an event, its completion time would be extended by 400ms in spite of the scheduler’s help.

When the system is dominated by foreground activity, a likely scenario on a thin client server hosting a large number of concurrent users, priority boosting provides no benefit for foreground threads with respect to one another. In this case, GUI threads can run for at most 90ms before being pre-empted by other threads in the queue.

The lesson here is that quantum stretching and priority boosting can only eliminate latency when the net boosted-priority “grace period” is long enough to complete the interactive operation. In the case of NT and TSE, this threshold is 180ms when competing with background activity, and 90ms when competing with other foreground activity.

How do these thresholds compare with typical interaction completion times? Endo et al. report that the majority of interactive events last between 30 to 45ms for Microsoft Notepad, 60 to 250ms for Microsoft PowerPoint, and 50 to 100ms for Microsoft Word. They also

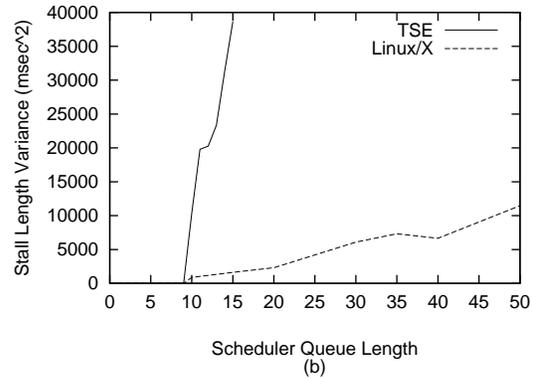


Figure 4: User-perceived keystroke handling jitter as a function of server CPU load. As with average latency, jitter is less noticeable on Linux/X than on TSE.

report a 500ms completion time for the window maximize operation. We highlight this operation because it is representative of user interface elements meant to “enhance” the user experience, but are designed with fast, local, desktop systems in mind instead of thin client environments.

Note that Endo et al.’s values were collected on a 100Mhz Pentium machine, so these completion times are likely to be on the order of 5 to 10 times faster on today’s processors. Nevertheless, continuing increases in user interface complexity, marked by more sophisticated graphics and the introduction of animated elements, makes such thresholds a continuing concern.

Linux Scheduling

The Linux kernel supports FIFO, round robin, and other scheduling classes, with priority values between -20 and +20 in each class. Most processes run in the round robin class with a quantum of 10ms. There is no provision for changing the quantum length and no facility for automatic priority boosting on GUI-related or foreground processes. The first implication of this design is that any user input event that is greater than 10ms, which is a fairly low threshold, risks being fragmented across quanta. In a high load scenario, this can increase latency for that event considerably. And because the quantum is so small, the level of fragmentation is greater than in TSE.

Moreover, Linux provides no help for interactive processes. NT and TSE can easily target and boost foreground threads because of the tight integration of the graphics subsystem into the kernel. However, X Windows is a user-level graphics subsystem and there is no well-defined method for passing GUI-related information into the kernel.

But in 1993, Evans et al. of SunSoft did exactly that

when optimizing the System V, R4 scheduler for interactivity [8]. Their approach traded a clean user-level/kernel separation for scheduler access to application-level information on process interactivity. They showed that in a control system running the SVR4 kernel, keystroke handling latency increases as the scheduler queue length (or load) grows because there is no provision for protecting interactive processes from CPU-intensive processes. Then they demonstrated a prototype SVR4 kernel modified with an interactive scheduler for which keystroke handling latency remains constant and small, even as load approaches 20.

In spite of this work, years later no Unix-like kernels implement such improvements, probably due to the separation between kernel development and user-level X Windows development. Although the modifications made by Evans et al. should be readily adaptable to the Linux and BSD kernels, such an implementation faces several challenges. First, user-applications must be modified in order to take advantage of priority boosting. Kernel modifications and new system calls do not benefit existing X applications. Furthermore, because GUI services are provided by the user-level Xlib library and not the kernel, the kernel has no straightforward way of preventing arbitrary processes and threads from claiming they are GUI-related even when they are not. This challenge represents an opportunity for further research stemming from this work.

4.2.2 Dynamic Latency

If scheduler deftness is important in minimizing compulsory latency, it is even more critical in the dynamic case when the processor is heavily loaded and many threads are queued, ready to run. Our experiments show that both TSE and Linux perform poorly when compared to Evans et al.'s modified SVR4, and surprisingly, TSE performs much more poorly than does Linux, despite its mechanisms for favoring interactive foreground processes.

Methodology

We wrote a simple C program called *sink* that is a greedy consumer of CPU cycles. Since *sink* never voluntarily yields the processor, each running instance should increase the scheduler queue length by one. We used this program to control the load level on the server.

At varying levels of load, we ran a simple text editing application at the client. Under TSE it was Notepad and under X Windows *vim*. The tester held down a key in the application to engage character repeat on the client machine, the rate of which was set at 20Hz.

Under no load, we expect the server to respond every 50ms with a screen update message to draw a new

character. We measured message inter-arrival times by examining the timestamps on the corresponding network packets using *tcpdump*. Under increasing load, we expect some or all of these inter-arrival times to rise above 50ms as the server handles other computations. We call each instance of this an “interactive stall,” with the length of the stall defined as the inter-arrival time minus 50ms. A stall is therefore the duration of time when the server’s processor was occupied with other tasks and the user therefore ended up waiting.

For each load level we recorded message inter-arrival times for 60 seconds and report the average.

Results

With no load on the server, both kernels performed as expected, sending a message to the client every 50ms. But figures 3 and 4 show that as the number of *sink* processes increased, so too did the length and variability of stalls. The relationship between load and latency exhibited by TSE and Linux are still similar to that of the 1993 vintage SVR4 scheduler.

TSE exhibits poor behavior, with latency increasing sharply around 10 load units. The data for TSE stops at 15 load units because at that point the system became barely usable at the console. These TSE results are inexplicable without access to NT source code. Linux, on the other hand, while still not protecting interactive processes properly, handles increasing load with more grace, with latency increasing linearly but more slowly with respect to load. This confirms the expectations based on our analysis of the scheduler.

These levels of latency and jitter are quite high and certainly well within the range of human perception. Subjectively, these interactive stalls felt like irregular “hiccups” in the system. That is, while some keystrokes would generate immediate character echo, the echo for other keystrokes was delayed by up to one second in extreme cases. The inconsistency of these stalls, or latency jitter, further contributed to an unsatisfying user experience.

5 Memory

In this section, we examine memory consumption and memory induced latency in a thin client context. First, we consider how varying user behavior consumes memory at different rates. Second, we consider the latency consequences of increasing memory usage on the server.

5.1 From Behavior to Load

5.1.1 Compulsory Load

Compulsory memory load has two components. The first is the dynamic memory usage of the kernel and the user-level services necessary to support graphical, multi-user, remote login. This is simply the amount of memory that is unavailable to user applications when the system is idle with no user sessions. In our configuration, memory load in this state was roughly comparable between the two systems, 17MB for Linux and 19MB for TSE.

The second component is the memory usage of each user session. This usage is governed by what is considered to be a minimal login with no additional user activity. The following tables list the processes associated with minimal logins for each system. For each process, we conservatively give the amount of private, per-user memory consumption, excluding any amortized shared code page costs for executable text or mapped, shareable memory.

Process	Typical
<i>in.rshd</i>	204 KB
<i>xterm</i>	372 KB
<i>bash</i>	176 KB
<i>Total</i>	752 KB

a. Linux/X

The *in.rshd* is a daemon process that accepts and processes client session initiation requests. It is somewhat analogous to the Terminal Service in TSE. *xterm* is the X-enabled application that draws a graphical terminal window to the client's screen. *bash* is the shell that gives the user access to operating system services and allows other processes to be launched.

Process	Typical	Light
(<i>shell</i>)	<i>explorer.exe</i> 1,368 KB	<i>command.com</i> 224 KB
<i>csrss.exe</i>	452 KB	452 KB
<i>nddeagnt.exe</i>	300 KB	300 KB
<i>winlogon.exe</i>	700 KB	700 KB
<i>Total</i>	2,820 KB	1,676 KB

b. NT TSE

csrss.exe is the subsystem that provides Win32 API services, *nddeagnt.exe* provides network dynamic data exchange (DDE) services, and *winlogon.exe* manages interactive user logons and logoffs.

In fairness, *explorer.exe* does offer a fully featured graphical file navigation and program launching interface while *bash* does not. We should note that TSE

clients have the option of dispensing with the Explorer and running just a specific application in a user session (but it cannot be empty). This admits the possibility of using a lighter alternative more comparable to *bash*. The DOS Prompt (*command.com*), for example, requires only 224KB of private, unshared memory, bringing the minimum compulsory memory load per-user for TSE down to 1,676KB.

5.1.2 Dynamic Load

Dynamic load is the memory utilization due to user behavior above and beyond the simple act of logging in to the system. This issue is no different for thin client servers than for other types of operating systems. Assuming that the operating system supports code page sharing, the smaller the set of active applications and the smaller their user-specific stack and heap areas, the lower the dynamic memory load.

Of the three hardware resources discussed in this paper, memory is the most difficult for which to make generalizing comments regarding load. Memory utilization is highly dependent upon the applications used.

5.2 From Load to Latency

The latency consequences of increasing memory utilization are well-known. As the active data set of a system exceeds the size of each level of the memory hierarchy, average data access latency increases roughly in steps [2].

The two most dramatic steps occur when the data set falls out of the cache and into main memory and when it falls out of main memory onto disk. Processes running on thin client system are by no means restricted to being interactive programs. In fact, multi-user systems like Unix are often used to run backgrounded compute or memory intensive jobs while the owner attends to other tasks.

As we saw in Section 4, poor resource scheduling of the processor can allow such greedy processes to severely impact perceived latency for interactive users. Likewise, poor resource management for memory can do the same. As observed by Evans et al., certain types of non-interactive, streaming memory jobs will typically force all other non-active processes to be paged to disk. They give as examples large data copies over NFS, creation of large temporary files in `/tmp`, and various stages of program compilation.

This behavior can be particularly damaging to interactivity in the following scenario. An interactive user may load a document into an application but then stop interacting with it for several minutes while he reads the document on-screen. During this time, a non-interactive

process on the same server with high page demand may force his application out to disk. When he goes to scroll down, there will be significant lag as his process is paged back into memory. We next demonstrate that TSE and Linux both perform poorly in this scenario.

In our tests, we opened a simple text editing application remotely. In Windows we used Notepad and in Linux *vim*. We then started on the server a process that touches enough memory to force pages of the edit application’s memory to be swapped to disk. Then we input a single keystroke and measured the time it took for the server to respond with a screen update. As we saw in an earlier experiment, the response should come in less than 50ms. However, because the application’s memory must be paged back from disk, significant latency is introduced. We report ranges and averages over ten runs for each operating system:

OS		Process Pages	
		In-Memory	Paged Out
Linux	<i>min</i>	50ms	330ms
	<i>avg</i>	50ms	1,170ms
	<i>max</i>	50ms	3,000ms
TSE	<i>min</i>	50ms	2,430ms
	<i>avg</i>	50ms	4,026ms
	<i>max</i>	50ms	11,850ms

These values are quite high and well into the range of perceptible latency. The latencies generated in TSE average about 40 times the threshold of human perception, while in Linux they average 11 times this limit.

Frequent paging of semi-active processes like we describe here indicates that the server has insufficient memory. A simple solution, of course, is to add more memory. Nevertheless, operating system mechanisms that mask scarcity-induced latency still provide at least two non-trivial benefits. First, a system that protects the pages of interactive processes is capable of hosting non-interactive batch jobs that have arbitrary memory requirements. A system without this protection cannot host such jobs without drastically degrading interactive performance. Second, paging protection guards against disruptive delays during spikes in memory utilization that are transient and do not warrant permanent additional resources.

Evans et al. also demonstrated in their prototype kernel a solution to this problem, which is non-interactive process throttling in high load situations. They demonstrated that their SVR4 kernel modified with throttling eliminated this pathology. The modifications they made could be adapted to today’s TSE, Linux, and BSD kernels. But as with the scheduling modifications, the task would be easier in TSE than in Linux and BSD because the Unix/X platforms must rely on user-level applications to pro-

vide information about which threads and processes are latency-sensitive.

6 Network

In this section, we consider the impact of the network on latency and the role therein of operating system abstractions for display and input service.

First, we consider how user behavior generates network load. We compare the ability of RDP, X, and LBX to minimize network traffic for any given user behavior. The comparison includes a typical application workload, and an examination of the impact of trends in user interface design, particularly the growing usage of animation. Second, we discuss how network load translates to user-perceived latency, underscoring the importance of network protocol efficiency.

To simplify our discussion, we first define two terms. Let a “channel” simply be a directed stream of network messages between the client and server. We call the stream from the server to the client the display channel because it carries messages instructing the client to display application interface elements. The stream from the client to the server we call the input channel because it carries keystroke and mouse input information to the application.

6.1 From Behavior to Load

Thin client servers export user interfaces over network links to remote users. Therefore, load generated on the network resource depends heavily on the design and implementation of the user interfaces of the applications being run remotely.

Perhaps the most visible user application trend over recent years has been the increasing richness and sophistication of graphical interfaces. As a result, the typical user behavior in a thin client environment is becoming increasingly network intensive.

One of the strengths of TSE’s design is that it allows existing Windows applications to run unmodified in a remote access environment. In fact, TSE would not be commercially viable if users could not use the same applications on which they currently rely. However, Windows software developers have typically designed their interfaces assuming fast, local graphics acceleration and therefore many applications that will be run on TSE will potentially consume unfriendly amounts of bandwidth.

Likewise, applications written for the X environment are growing more like modern Windows applications in their appearance and functionality, particularly with Linux’s growing popularity on the desktop. In the following subsections, we investigate the network loads generated by typical modern graphical applications.

6.1.1 Compulsory Load

Compulsory network load includes both the quantity of bytes exchanged between the client and the server for session negotiation and initialization, and any network traffic that is exchanged after session setup but while the user is idle.

Session setup costs in our configurations were 45,328 bytes and 16,312 bytes for TSE and Linux/X, respectively. These costs are rare and ephemeral, and are typically not major contributors to latency. In terms of idle load, neither system requires data to be exchanged when no user activity is present. So compulsory load is a relative non-issue with the network on these two systems. The real contributor to latency is dynamic load generated by application usage, which we discuss next.

6.1.2 Dynamic Load

To gain a broad understanding of the relative performance of RDP, X, and LBX, we compared their behavior on a typical application workload. Thanks to the growth in popularity of Linux, we were able to develop a workload with cross-platform applications that run on Windows and Linux/X. These were Corel WordPerfect, a word processor, the Gimp, an open-source photo-editing package, and Netscape Navigator, a web browser.

For each network protocol, we performed a predefined set of user interactions: editing a WordPerfect document, creating a simple bitmap with the Gimp, and surfing several websites with Netscape. We collected data during these trials using *prototap*, our own protocol tracing software based on the *tcpdump pcap* packet sniffing library.

The following table shows byte and message counts for each channel and for each protocol. We also report the average message size for each protocol.

		RDP	LBX	X
Bytes	<i>input</i>	196,343	1,478,341	2,749,328
	<i>display</i>	2,049,132	8,137,445	14,183,842
	<i>total</i>	2,245,475	9,615,786	16,933,170
Msgs	<i>input</i>	1,155	18,888	19,345
	<i>display</i>	2,422	74,618	20,797
	<i>total</i>	3,577	93,506	40,142
Avg. msg size		627.75	102.84	421.83

RDP is clearly the most efficient protocol, generating less than 25% of the byte traffic of LBX and less than 15% of X. This is due in large part to the the small number of messages it sends relative to X and LBX. However, RDP also has the largest average message size, suggesting that RDP messages encode a higher level of graphics semantics than do those of X and LBX. The message size

advantage of LBX over X is due to message compression. Note that this savings comes at the expense of a 133% increase in display message count over X. This, however, does not seem to adversely affect the overall performance of LBX.

The average message size among the three protocols is just 209 bytes, which is much smaller than the interface MTU on our systems (1500 bytes). For such small messages, the overhead imposed even by just 20 byte IP headers is significant. In non-routed deployment environments, a scheme like the *x-kernel* virtual-IP (VIP) network stack could reduce overhead by omitting the IP header [11]. The following table gives the potential byte savings of omitting the IP header.

	RDP	X	LBX
<i>Normal Bytes</i>	2,245,475	16,933,170	9,615,786
<i>Bytes w/ VIP</i>	2,169,435	16,096,790	7,745,666
<i>Savings</i>	3.38%	4.94%	19.45%

Because LBX has the smallest average message size, it stands to benefit most from a VIP-like scheme. However, even with this VIP optimization, LBX would still be less than half as efficient than RDP.

6.1.3 Animations

As discussed earlier, application interfaces have steadily grown more sophisticated and active. In particular, we observe an increasing use of animation in user interface design.

Animation is often employed to improve the user experience by creating the illusion of reduced latency through visual contiguity. Ironically, the use of smooth and effective animation in a thin client environment can produce considerable network load, yielding, on balance, a negative impact on user-perceived latency. Moreover, animations often run asynchronously of user interaction, meaning that their activity is not limited by the rate of user events such as keystrokes or mouse movements.

Simple animations like blinking cursors and progress bars generate a harmless amount of traffic, generally less than 10KBps for short durations. Other types of animation, however, can be quite costly. In particular, today's web pages are replete with animated GIF advertisements and Java and HTML based stock and news tickers. The remote display of the MSNBC homepage produces sustained average network loads well in excess of 1Mbps on RDP, X, and LBX.

Such levels of network activity make multi-user service over 10Mbps Ethernet unfeasible. If just five users open their browsers to a page like this, the network link becomes saturated. Although many administrators of interface service environments may, by policy, prohibit

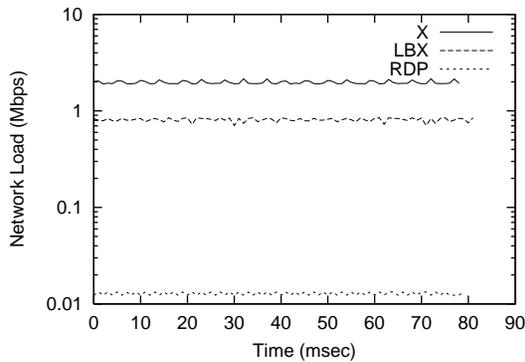


Figure 5: Network load generated by displaying a 100 by 100 pixel, 10-frame, 20Hz animated GIF in Netscape Navigator over X, LBX, and RDP.

the use of web browsers or enforce the disabling of webpage animations, this remains an important issue to consider when developing a “realistic” behavior profile for a user base. 100Mbps of faster Ethernet connectivity is virtually required to support this type of user behavior.

Taming Animation: Bitmap Caching

While on the animation-intensive MSNBC page all the protocols perform similarly, on a less intensive synthetic page, RDP significantly outperforms LBX and X.

Figure 5 shows the network load for displaying on Netscape a synthetic animated GIF. The animation is a 100 by 100 pixel rotating MSNBC logo that has 10 total frames and runs at 20 frames per second. Under LBX, bandwidth utilization is 0.8Mbps, while under X it is 2.0Mbps, and under RDP just 0.01Mbps.

This disparity between RDP and the X-based protocols suggests the presence of a client-side bitmap cache in TSE large enough to store all the frames of the synthetic animation. But for the animations on the more intensive MSNBC page, their competing frames overflow the cache such that each miss generates a full bitmap transfer over the network. When the frames do fit into the cache, we presume that display data not need to be transferred, so that only small “swap bitmap” messages are exchanged.

Indeed, according to Microsoft’s product literature, the TSE client reserves, by default, 1.5MB of memory for a bitmap cache using an LRU eviction policy [5]. The cache is typically used to store icons, button images, and glyphs. Storing these items can be especially bandwidth effective since users often spend much of their time in just a few applications each with a finite number of icons and images.

X, and consequently LBX, does not support bitmap caching, and the consequences are clearly shown in Fig-

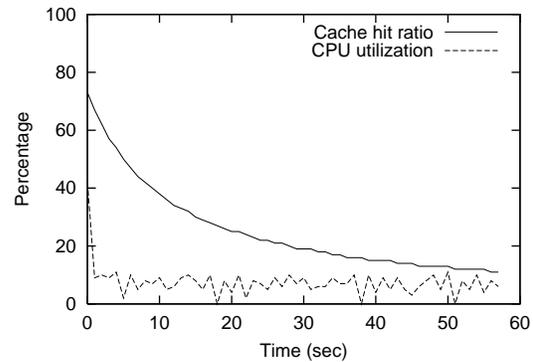


Figure 6: CPU utilization and bitmap cache hit ratio for a 66-frame animation that overflows the bitmap cache. The CPU hovers around 10% utilization. The hit ratio starts at 75% due to cache hits on screen elements displayed before the beginning of the test. During the test, the oversize animation overflows the cache, preventing any hits, and resulting in an asymptotically declining cumulative cache hit ratio.

ure 5. If there were a cache of any appreciable size (which there is not) it is not being used. Each frame of the animation requires the full bitmap to be transferred across the network.

The difference in performance between RDP and X/LBX again reinforces the importance of considering the design and implementation of the operating system abstractions for hardware resources. The inclusion of a bitmap cache in TSE’s abstraction for display and input allows it to handle behavior that includes animation with much less load. That, in turn, means that TSE can support more concurrent users with this behavior before exhibiting noticeable latency.

Cache Effectiveness and CPU Load

The effectiveness of the cache is not only critical to reducing network load, but also processor load at the server. In these tests, we used the Session object in the Microsoft Performance Monitor to measure the various metrics associated with the client-side bitmap cache.

Figure 6 shows the effect on CPU Utilization and Bitmap Cache Hit Ratio of a 66-frame animation that overflows the cache. The CPU Utilization starts at around 10% as it transmits frames for the first time. However, it never falls, because the server must continue to send the frames that fall out of the cache just before being needed, which is all of them. The Cache Hit Ratio which is cumulative, begins around 70%, and falls asymptotically toward zero with each subsequent miss.

Cache Pathology

However, bitmap caching has its limitations. Loop-

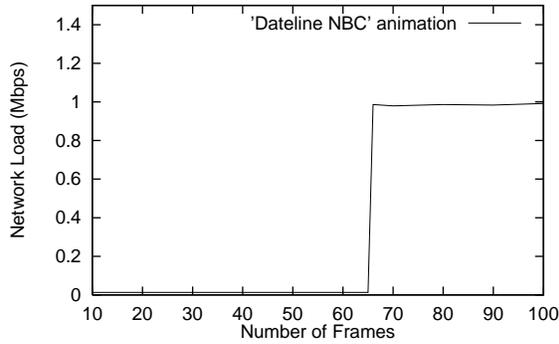


Figure 7: Network load generated as IE4 displays animations of varying frame counts over RDP. For frame counts less than 65, the animation fits in the cache, and network load is negligible. For 66 or more frames, the animation overflows the cache and every frame must be transferred over the wire, generating considerable load.

ing animations defeat LRU bitmap caches in the same way that sequential byte range accesses defeat LRU disk caches, which is a well-known phenomenon in file systems research [15].

To demonstrate, we created a series of animations whose frame counts range from 25 to 100. Figure 7 shows that for values 25 through 65, bandwidth utilization is 0.01Mbps, but for all values above 65, bandwidth utilization is 0.96Mbps.

Clearly, while LRU may be the appropriate eviction scheme for typical usage, it is exactly the wrong scheme for handling looping animations. A more intelligent scheme capable of dealing with such animations might somehow detect loop patterns and adjust its eviction behavior accordingly.

X Windows could certainly benefit from the introduction of client-side bitmap caching, and even TSE could benefit from a larger cache. Even though Microsoft claims that “testing has found that 1.5 MB is the optimal cache size,” our results show that modest amounts of interface animation can render a cache of this size useless [5]. When available, additional memory should be allocated to enlarge the bitmap cache. As a rule of thumb, a 256-color, 468 by 60 pixel banner advertisement with 10 seconds of 20 frame per second animation requires about 5.6 MB of memory to cache the entire animation. This figure does not account for any compression that is possible on the frames, and so serves as an upper bound for this amount of animation.

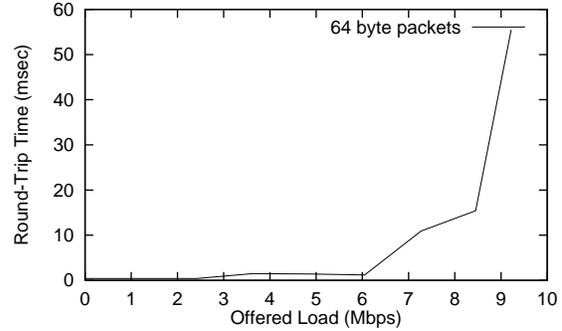


Figure 8: Latency increases above human-perceptible levels as the network fabric approaches saturation.

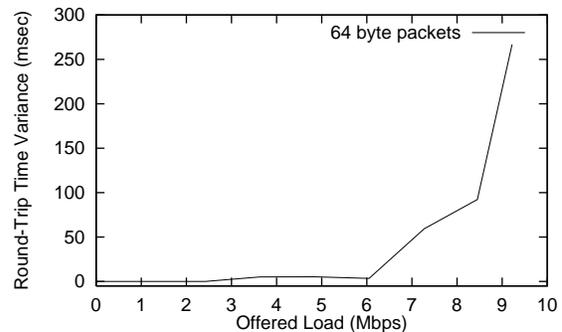


Figure 9: Jitter also increases considerably as the network fabric approaches saturation.

6.2 From Load to Latency

We have just shown how various user behaviors generate network load. The next step in the framework is mapping network load levels to user-perceived latency.

To investigate this relationship, we produced synthetic TCP/IP network load on our experimental testbed. The load was generated by two simple C programs that establish a TCP connection and send and receive random data at various rates. Figures 8 and 9 show the effect of load on network latency and jitter. For each load level, we ran *ping* for 60 seconds and took the average and variance in round-trip time (RTT) for all packets sent. We used the default packet size in *ping*, which is 64 bytes. 64 bytes is roughly the size of a typical input channel message, such as a keystroke. Therefore, the latencies we observe here give a realistic lower-bound for latencies that would be observed by a user.

Figure 9 shows that while the network is not saturated, RTT remains low and almost perfectly consistent. However, as the network nears saturation, performance suffer-

s dramatically. The 55ms delay induced at 9.6Mbps load is considerable with respect to known levels of human latency tolerance [7]. The inconsistency of the latency, a phenomenon known as jitter, only compounds the negative impact of network saturation.

7 Related Work

There is relatively little other work directly relevant to our broad discussion of operating system design impact on thin client performance. What there is, we now discuss and analyze in light of our approach and findings.

Endo et al. and Evans et al. have emphasized the importance of latency, and were discussed in detail in Sections 4 and 5.

Danskin published several papers on profiling the X protocol [6]. His work, in terms of our framework, focused primarily on determining the load generated by typical applications of his day. While the typical X behavior profile has changed considerably since then, his methodology provides the inspiration for our *prototap* tool. Danskin also did work on characterizing application-specific idioms used on the display channel. Finally, he came to the same conclusion as we did that small message size makes TCP/IP an inefficient network substrate for protocols like RDP, X, and LBX.

Schmidt et al. introduced a new thin client wire protocol called SLIM, which is embodied in the Sun Microsystems SunRay product [19]. While SLIM has the advantage of being more platform independent than X or RDP, their results show it to be roughly equivalent in performance to X, placing it still behind RDP and LBX in network load efficiency. Their paper also touches upon the issue of latency induced by server-side resource contention, but omits analysis of how the server-side operating system can be tuned to reduce latency. We contend that in the long view, network protocol efficiency is just one piece of a high-performance, low-latency thin client user experience. VNC, from AT&T Laboratories Cambridge, is another network protocol that is similar to SLIM [16].

Sirer et al. describe an architecture for extruding the components of a virtual machine over multiple hosts in a network [22]. Their work further emphasizes the benefits of factoring system services out of clients and onto network servers. Their DVM architecture differs from what are colloquially called thin client architectures in that it draws the line where the network is spliced in a different place, factoring out services such as verification, security enforcement, compilation and optimization, but not execution. Moreover, their work emphasizes throughput and not user interaction latency, and focuses more on the network bandwidth impact of code transfers rather than user interface input and display traffic.

On the performance of TSE, the only documents we have been able to find are server sizing white papers published by Microsoft and various hardware vendors who market TSE servers [14, 10, 3]. These white papers are remarkably similar, defining typical user profiles and reporting the load generated by these profiles. They uniformly ignore, however, the issue of user-perceived latency. We also believe the network load characterizations in these papers are overly optimistic, ignoring the increasingly dynamic and rich nature of user interfaces.

8 Conclusion

Latency is the paramount performance criterion for operating system support for thin client service, or interactive, graphical, multi-user, remote access.

We have presented an approach for evaluating thin client environments which is founded on latency and highlights important issues relevant to thin client performance. These include the influence of user-specific behavior, the translation of that behavior into resource load, the importance of operating system abstraction implementation therein, and the translation of resource load into user-perceived latency.

This approach guided our resource-by-resource comparison of TSE and X Windows on Linux, two popular implementations of thin client services. Our investigation reveals that resource scheduling for both the processor and memory in these systems is not well optimized for heavy, concurrent, interactive use. In common cases of resource saturation, both latency and jitter rise well above human-perceptible levels. We also performed a detailed comparison of the RDP, X, and LBX protocols and found that RDP is generally more efficient in terms of network load, particularly in handling animated user interface elements.

9 Acknowledgements

We would like to thank our shepherd, Christopher Small, and the anonymous members of the program committee for their help in shaping this paper. We also thank Mike Smith and H.T. Kung for their review and commentary on early incarnations of this work.

References

- [1] Beck, M., Bohme, H., Dziadzka, M., Kunitz, U., Magnus, R., Verworner, D. *Linux Kernel Internals: Second Edition*, Addison-Wesley Longman, 1998.
- [2] Chen, J.B., Endo, Y., Chan, K., Mazieres, D., Dias, A., Seltzer, M., Smith, M. "The Measured Performance of Personal Computer Operating Systems."

- ACM Transactions on Computer Systems, Vol. 14, No. 1, February 1996, Pages 3-40.
- [3] Compaq Corp. "Performance and Sizing of Compaq Servers with Microsoft Windows NT Server 4.0, Terminal Server Edition". White Paper. July 1998.
- [4] Conner, B., Holden, L. "Providing A Low Latency User Experience In A High Latency Application". Proceedings of the 1997 Symposium on Interactive 3D Graphics.
- [5] Cumberland, C., Carius, G., Muir, A. *Microsoft Windows NT Server 4.0 Terminal Server Edition Technical Reference*, Microsoft Press, 1999.
- [6] Danskin, J., Hanrahan, P. "Profiling the X Protocol". In Proceedings of the 1994 SIGMETRICS Conference on Measurement and Modeling of Computer Systems.
- [7] Endo, Y., Wang, Z., Chen, B., Seltzer, M. "Using Latency to Evaluate Interactive System Performance". Proceedings of the 1996 Symposium on Operating System Design and Implementation(OSDI).
- [8] Evans, S., Clarke, K., Singleton, D., Smaalders, B., "Optimizing Unix Resource Scheduling for User Interaction," 1993 Summer USENIX Conference.
- [9] Fulton, J., Kantarjiev, C. "An Update on Low Bandwidth X(LBX): A Standard for X and Serial Lines." Proceedings of the 7th Annual X Technical Conference.
- [10] Hewlett-Packard Corp. "Performance and Sizing Analysis of the Microsoft Windows Terminal Server on HP NetServers". White paper. July 1998.
- [11] Hutchinson, N., Peterson, L., Abbott, M., O'Malley, S. "RPC in the x-Kernel: Evaluating New Design Techniques". SOSP 1989: 91-101.
- [12] Li, K., and Hudak, P., "Memory Coherence in Shared Virtual Memory Systems." ACM Transactions on Computer Systems, November 1989.
- [13] MacKenzie, I., and Ware, C. "Lag as a Determinant of Human Performance in Interactive System." Proceedings of the ACM InterCHI '93.
- [14] Microsoft Corp. "Microsoft Windows NT Server 4.0, Terminal Server Edition — Capacity Planning". White Paper. June 1998.
- [15] Patterson, R., Gibson, G., Ginting, E., Stodolsky, D., Zelenka, J., "Informed Prefetching and Caching," Proceedings of the Fifteenth Symposium on Operating Systems Principles, December 1995, pp. 79-95.
- [16] Richardson, T., Stafford-Fraser, Q., Wood, K., Hopper, A., "Virtual Network Computing", IEEE Internet Computing, Jan/Feb 1998.
- [17] Sandberg, R., Goldberg, D., Kleiman, S., Walsh, D., Lyon, B., "Design and Implementation of the Sun Network Filesystem," Proceedings of the Summer 1985 USENIX Conference.
- [18] Scheifler, R., Gettys, J. "X Window System: Core and Extension Protocols". Butterworth-Heinemann. 1997
- [19] Schmidt, B., Lam, M., Northcutt, J., "The interactive performance of SLIM: a stateless, thin client architecture," Operating Systems Review, 34(5):32-47, December 1999.
- [20] Shneiderman, B. "Designing the User Interface". Addison-Wesley. 1992.
- [21] Solomon, D. *Inside Windows NT: Second Edition*, Microsoft Press, 1998.
- [22] Sirer, E.M., Grimm, R., Gregory, A., Bershad, B, "Design and implementation of a distributed virtual machine for networked computers," SOSP 1999: 202-216.
- [23] Wang, Z., Rubin, N. (1998) "Evaluating the Importance of User-Specific Profiling," Proceedings of the 3rd USENIX Windows NT Symposium.