# Evaluating Windows NT Terminal Server Performance

Alexander Ya-li Wong, Margo Seltzer
*Harvard University, Division of Engineering and Applied Sciences*
{aywong,margo}@eecs.harvard.edu

## Abstract

With the introduction of Windows NT, Terminal Server Edition (TSE), Microsoft finally brings to Windows the "thin-client" computing model the X Window System has offered Unix for a decade. TSE's two most salient features are the provision of multi-user login service and the provision of that service remotely, over a network link. These features distinguish TSE from previous Microsoft operating systems not only by functionality, but also by how its performance ought to be measured. Because TSE's primary service is interactive, user-perceived latency is more important than ever. In this paper, we examine the resource consumption and latency characteristics of shared usage on a TSE system. We find that the introduction of remote, multi-user access has added to the minimal level of resource consumption, that the efficiency of TSE's RDP protocol is generally good but degrades on dynamic user interface elements, and that TSE can exhibit poor latency performance when subjected to high processor, memory, and network load.

## 1 Introduction

With Microsoft's introduction of Windows NT, Terminal Server Edition (TSE), the Windows platform has acquired the multi-user, remote access capabilities that have been available for Unix since X-Windows appeared a decade ago. Microsoft seems to have awoken to the possibility that "thin-client" computing is a viable and even desirable alternative to their vision of "Windows on Every Desktop". Report after industry report has revealed a thirst among IT managers for the lower total cost of ownership offered by thin-client computing. Microsoft hopes to quench that thirst with TSE, which they bill as a way to extend the life of older hardware, bring Windows applications to non-Windows desktops, and rein in per-user management costs [11, 12].

### 1.1 Inside TSE

The TSE environment is composed of three major components [13]:

**Terminal Server** - The TSE core is a fork of the NT 4.0 Server codebase diverging sometime after the release build of 4.0 Server. Key parts of the OS were modified to support multiple concurrent users. Kernel objects are virtualized across user sessions, the Virtual Memory Manager supports per-session mappings of kernel address space, and the kernel supports code page sharing across sessions. The "Terminal Service" was introduced to manage user sessions.

**Remote Display Protocol (RDP)** - RDP runs over TCP/IP and makes user sessions available remotely. It includes control messages (e.g., keystrokes, mouse events) sent from the clients to the server and display messages sent from the server to the clients. Server-side RDP support includes per-user replacements for display, keyboard, and mouse drivers, which bind to an RDP stack instead of local hardware devices to enable remote display and control.

**Terminal Clients** - Terminal clients communicate with the Terminal Server using RDP. They display graphics delivered from the server on local video hardware and capture local keyboard and mouse input for delivery back to the server. Clients are available for Windows for Workgroups, CE, 95, 98, NT 3.51, and NT 4.0.

### 1.2 Evaluating TSE

Along with the new functionality offered by TSE come new requirements for measuring its performance. As a multi-user, interactive, remote access operating system, TSE cannot be evaluated with the same metrics used for either its predecessors or for other operating systems.

For example, it is not sufficient to assign a simple Winstone, SYSmark, or SPECmark value to a machine running TSE. Ultimately, those interested in deploying TSE need to know the maximum number of concurrent users their servers can support, and they need benchmark results to be expressed in these terms. Moreover, the particular characteristics of the TSE environment make it incompatible with most existing benchmarks. TSE is:

**Multi-User** - TSE is designed to support heavy, concurrent use on today's hardware [1]. 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 only 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.

**Interactive** - Unlike HTTP or database servers for which throughput is critical, TSE's primary service is interactive login. As argued by Endo et. al, latency, not throughput, is the paramount performance criterion for this type of system [5]. Any useful metric should yield information necessary to gauge whether the system satisfies the latency demands of users.

**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 the TSE environment, the video subsystem at the server is irrelevant and the GUI is instead constrained by network bandwidth, the efficiency of RDP, and the video hardware at the terminal.

This paper analyzes the resource demands of TSE's multi-user and remote display capabilities. In particular, we evaluate resource consumption of processor cycles, memory, and network bandwidth. Then, we examine the effect of load on TSE's ability to maintain low latency to yield a smooth user experience.

In Section 2, we describe our experimental environment. In Section 3, we consider resource consumption of the processor, memory, and network. In Section 4, we evaluate TSE's latency characteristics under load. In Section 5, we review related work, and we conclude in Section 6.

## 2 Experimental Environment

Our testbed consisted of a server, a client, and a network listening host connected to a NetGear EN104TP 10Mbps Ethernet hub. The server was a Celeron-333, Intel 440EX AGPset, 48MB SDRAM, 4GB IDE IBM DCAA-34330, and a NetGear FA-310 NIC. The client was a Pentium II-400, Intel 440BX AGPset, 128MB SDRAM, 11GB IDE Maxtor 91152D8, and a 3Com 3C905B NIC. The listening host was a Pentium-233, Intel 440TX PCIset, 96MB EDO RAM, 2GB IDE IBM DTNA-22160, and a 3Com 3C589C NIC. The server ran TSE build 419, the client ran Windows 98 with TSE client build 419, and the listening host ran the 2.0.36 Linux kernel. All NIC's were set to 10Mbps operation.

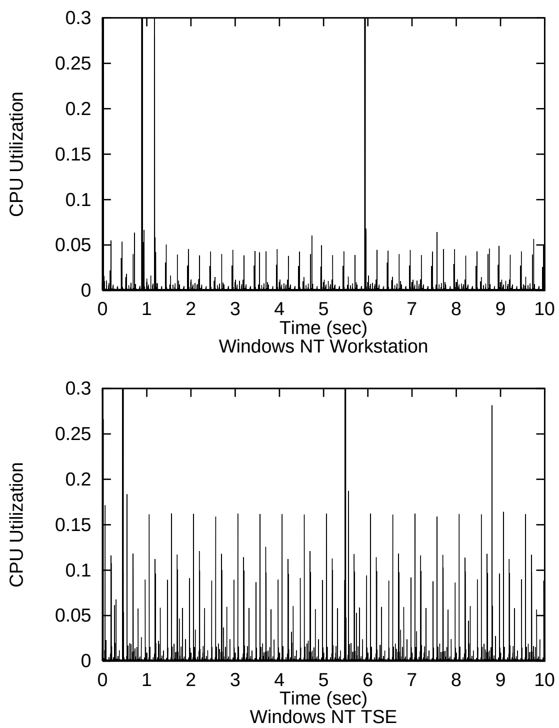



Figure 1: A comparison of the idle-state processor activity in Windows NT Workstation 4.0 and TSE.

## 3 Resource Consumption

In a server operating system, resource sharing is typically defined in terms of concurrent processes, transactions, or queries. For TSE, the sensible unit for quantifying resource consumption is the user. In this section, we evaluate TSE's scalability by examining per-user consumption of server-side processor cycles, memory, and network bandwidth. Resource consumption is highly dependent on the particular applications being used, so we provide consumption data as generically as possible to allow extrapolation to scenario-dependent usage.

### 3.1 Processor

The changes made to the core of NT to enable multi-user, remote access would likely increase processor overhead, which can translate to user-perceptible latency even under no server load.

Endo et. al introduced a technique for measuring user-perceived latency they call "measuring lost time" [5]. Using 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 technique can also be used to generate detailed visualizations of CPU utilization.

Their paper reported idle-state CPU activity profiles for Windows 95, NT 3.51, and NT 4.0. To quantify the baseline activity introduced in the transition from NT to TSE, we reproduce their experiment for NT 4.0 and TSE. Figure 1 shows that TSE is indeed more active than NT when idle. The difference is due primarily to a large number of events in TSE that last between 200 and 400ms. This increased activity can probably be attributed to the additional services necessary to support multi-user, remote access. Viewing the latencies over time as a cumulative distribution reveals that TSE has 71.0% more idle-state activity than NT.

## 3.2 Memory

In a single-user operating system, the user runs a number of applications, each of which consumes memory for the executable image and the dynamic stack and heap. In a multi-user system, many users may share the same applications. Code page sharing, which is supported in TSE, reduces total memory usage by backing only a single copy of the code text and mapping it read-only into the address space of each application instance [13].

Therefore, memory is consumed by applications in TSE in two ways. First, each new instance of an application consumes at least as much memory as the size of its stack and heap. Second, the greater the number of distinct applications used, the greater the combined memory usage for all of the applications' shared code pages.

### 3.2.1 User Sessions

We used Microsoft Process Viewer to collect memory consumption data on TSE and a number of popular Windows applications.

| Process | Cold | Warm | Disconnect |
|---|---|---|---|
| *csrss.exe* | 360 | 452 | 452 |
| *explorer.exe* | 732 | 1,368 | 1,368 |
| *loadwc.exe* | 424 | 424 | 424 |
| *nddeagnt.exe* | 300 | 300 | 300 |
| *winlogin.exe* | 396 | 700 | 700 |
| *Total* | 2,212 | 3,244 | 3,244 |

This table lists the processes automatically started with each unique user login session and the amount, in kilobytes, of private, committed memory in their address spaces. "Private" means that the memory is not mapped and is visible only to the local process. "Committal" is the Microsoft term for the allocation of backing store to virtual memory pages. Memory may be "reserved" without committal, and reserved memory may be released without ever having been committed [16]. Throughout our discussion, we conservatively include only private, committed memory in our definition of memory utilization. We do not include amortized shared code page costs for executable text, nor mapped, committed memory that may or may not be shared.

The "Cold" column reports values for a login with no subsequent user activity. The total for the Cold column is therefore a lower bound on per-user memory usage because each new user login will always consume *at least* this much memory. The "Warm" column reports values after modest usage. We see that the stack and heap size for the `csrss` (client-server runtime subsystem) and `explorer` (the shell) processes increases somewhat, while it does not for the other processes. The "Disconnect" column reports memory usage when the user has disconnected from the session (which is not a full logoff and allows a later reconnect). Clearly, memory is not explicitly conserved upon disconnect. We assume that TSE relies on the pagefile for this.

We should note that TSE clients have the option of dispensing with the default shell, the Explorer, and running just a specific application in a user session. This admits the possibility of using a lighter alternative such as the DOS Prompt (command.com), which at 224KB of cold, private, committed memory, is less than one-third the size of the Explorer.

### 3.2.2 Applications

This table presents memory usage statistics in kilobytes for a sampling of popular Windows applications. Measurements were taken immediately after the application was launched with the closest approximation of a null document. Actual mileage will vary.

| Application | Mapped | Private |
|---|---|---|
| *Notepad* | 1,068 | 148 |
| *Word 97* | 1,796 | 932 |
| *Excel 97* | 1,788 | 284 |
| *PowerPoint 97* | 1,816 | 300 |
| *Access 97* | 2,324 | 1,176 |
| *Outlook '97* | 6,532 | 1,356 |
| *IE 4.0* | 1,368 | 800 |
| *Netscape 4.08* | 1,460 | 1,396 |
| *WordPerfect 8* | 1,276 | 1,212 |
| *Visual C++ 5.0* | 1,046 | 1,316 |

All memory reported is committed. The first column reports the amount of memory that is mapped and is potentially shared. This provides an estimate for the amount of shareable code text in the application that will not be duplicated for each running instance. The second column reports the amount of private, committed memory, which is most likely the stack and heap space. This gives a conservative lower-bound for the per-user memory utilization of each application. These values surely

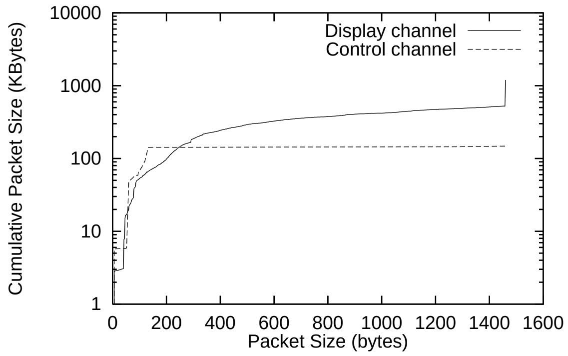



Figure 2: RDP control and display channel packet size distribution. Note the log-scaling.

rise as users open documents and otherwise begin to actually use the applications. Exactly how much depends highly on the application and the documents.

## 3.3 Network

Network load generated by TSE is a function of the user interface richness and complexity of the applications in use and the efficiency with which RDP encodes the dynamics of those interfaces.

### 3.3.1 RDP Efficiency

There is no published specification for RDP. Therefore, we are not yet able to offer detailed explanations for all of our RDP observations. An RDP reverse-engineering effort is part of our ongoing work.

First, we investigated the network packet size distribution of a live RDP message stream. We had a user open a TSE session from *client* to *server* and work for several minutes, composing a document in Word 97, browsing the NT Event Log, and navigating the filesystem using Explorer. Figure 2 shows the cumulative distribution of both control and display channel packets.

The data indicate that the control channel is dominated by packets on the order of 100 bytes. For these small control messages, the overhead imposed by 20 byte IP headers is considerable. In non-routed TSE environments, a scheme like the *x-kernel* virtual-IP (VIP) network stack could reduce overhead by omitting the IP header [9]. The total number of control bytes transferred in this test was 282,258. Without IP headers, only 232,158 would be transferred, yielding a 17.6% savings in overhead.

Unfortunately, the benefit of such an optimization is partially mitigated by traffic on the display channel where the average packet size is much larger. Accounting for both channels, a VIP-like optimization would reduce total bytes transferred from 1,642,337 to 1,533,797, a savings of only 6.6%. Regardless, a VIP scheme could still measurably improve latency performance. Because display messages are typically sent only in response to control messages, reducing the average packet size on just the control channel can reduce overall latency.

### 3.3.2 Decompositional Analysis

In the previous experiment, we showed that the packet size distribution of a "typical" TSE session was amenable to optimization. However, the efficacy of the optimization depends greatly on the distribution. A user running a different mix of applications could produce a dramatically different distribution.

This illustrates the importance of considering user behavior and application mix in the TSE world. A common attribute of previous papers on TSE performance is a definition of what the authors believe to be "typical" user behavior. Because these definitions of behavior strongly influence resource utilization results, it is important to analyze their validity and realism.

Therefore, we performed decompositional analysis on RDP in terms of smaller, more discrete operations. This helped us to gain some understanding of how RDP works from a microscopic view. All values reported here are in bytes per second. For comparison, keep in mind that 10Mbps Ethernet has a bandwidth of 1,250 bytes per second.

**Keystrokes and Typing**

Trace data from *tcpdump* reveals that a single keystroke made at a TSE client generates 2 RDP messages on the control channel:

```
client.1972 > server.3389: P 708:768(60)
client.1972 > server.3389: P 768:828(60)
```

These two server-bound messages are presumably *KeyDown* and *KeyUp*, and each appears to be 60 bytes. Note that 60 is the size of the TCP/IP packet's data payload. In reality, delivery of a 60 byte RDP message requires link-layer transmission of 118 bytes since each message is wrapped with 40 bytes of TCP/IP headers and 18 bytes of Ethernet framing. A single keystroke, then, generates 232 bytes of traffic on the control channel.

Using this information, we can derive the theoretical control channel bandwidth utilization of typing. If we assume 5.29 characters per word (as in the present work), then for every 10wpm a typist is fast, he will generate

$$\frac{10 \text{ words}}{\text{min}} \cdot \frac{5.29 \text{ chars}}{\text{word}} \cdot \frac{232 \text{ bytes}}{\text{char}} \cdot \frac{1 \text{ min}}{60 \text{ sec}} = 0.20\text{KBps}$$

of network traffic. A 75wpm typist therefore generates 1.51KBps of load.

This theoretical value is borne out by empirical observation. In our test, a user typed at approximately 75wpm into the Notepad application. Bandwidth was measured on the control channel only and was found to be 1.56KBps.

Unfortunately, we cannot apply this same generalizing analysis to derive display channel utilization because the size and number of display messages sent in response to keystrokes varies from application to application. We did, however, measure overall bandwidth utilization for "typical" typing behavior in two Windows word processors, Word 97 and WordPerfect 8. Both word processors were set to 10pt Times Roman font and 100% magnification page layout view. For a user typing at 100wpm, Word generated 6.26KBps of display traffic and WordPerfect generated 11.66KBps.

Typing under RDP is not particularly bandwidth-intensive. On a 10 Mbps Ethernet, one should be able to support on the order of 100 users composing simple text documents in Word or WordPerfect.

**Mouse Movement**

If the cursor is handled at the client, we would expect mouse movement to generate little or no RDP traffic. But, because the Win32 GUI can change cursors depending on location, the server-side application logic must be constantly informed by the client of the cursor's location. However, the server sends no display messages in response to mouse movement, except in the case of a cursor change. The TSE client software presumably handles the overlay of the cursor image over the display data received from the server.

We observe that continuous movement of the mouse in random directions on an open area of the desktop (to avoid cursor changes) generates 1.95KBps of network traffic. The simple behavior of moving the mouse appears unlikely to produce bandwidth utilization above perhaps 2KBps.

**Menu Navigation**

Another typical operation performed by Windows users is menu navigation. To measure the bandwidth consumption of this operation, we performed a simple human-driven depth-first traversal of the "Start Button" menu tree on our system. This activity generated a network load of 3.12KBps. Note that this value includes the 1.95KBps of traffic generated by mouse movement alone. This means that approximately 1.17KBps of traffic was generated by the movement of the menu highlight and the periodic display of newly activated cascade menus.

We also performed a similar human-driven test in Word and WordPerfect. In this test, the user hit the 'Alt' key to select the menu bar, and then held down the right arrow key, causing each top-level menu to be displayed briefly. WordPerfect generated 17.13KBps and Word 39.82KBps. We attribute this difference to the fact that Word's menus are larger and include graphical icons. Although menu navigation behavior is rarely sustained, 40KBps is a realistic estimate of burst rate.

**Scrolling**

Scrolling through a document is another common operation. Although the display message behavior of scrolling is highly application-dependent, we can measure bandwidth utilization in a realistic usage scenario.

The test we performed had a user scroll though the same multi-page document in Word and WordPerfect by holding down the *PgDn* key. The document view was set to Page Layout at 100% magnification. The body of the document consisted almost entirely of 10pt Times Roman text, interrupted periodically by a small table.

The average bandwidth over the duration of the scroll operation was 59.24KBps for Word and 192.54KBps for WordPerfect. Word seems to render rapid text movement more efficiently, at least with respect to the primitives used in RDP. Although scrolling is an intermittent operation, in a realistic TSE deployment scenario, one can easily imagine seeing prolonged bursts of up to 60KBps in Word and almost 200KBps in WordPerfect as users scroll through long documents.

**Summary**

Typical use of Windows applications seems to consume modest amounts of bandwidth. Users rarely, if ever, perform any of the above described operations simultaneously. Therefore, the peak bandwidth utilization of a typical user will generally be the maximum utilization among all of the operations. That is, such users will generally produce, at most, 60KBps-200KBps of traffic (from intensive scrolling).

### 3.3.3 Animations

Of course, user interaction with Windows and Windows applications is not limited to just the operations described above. Moreover, user interfaces have steadily grown more rich and sophisticated over time.

Basic user activity in a non-TSE Windows environment rarely saturates system resources for extended lengths of time, often leaving the CPU idle. Modern Windows programmers have, very reasonably, sought to exploit this idle CPU capacity to enhance the user experience. We identify two relevant such exploits here.

The first is the use of animation. Although the motivation for such enhancements is often intuitive, previous work has experimentally shown that animation can create the illusion of reduced latency through visual conti-

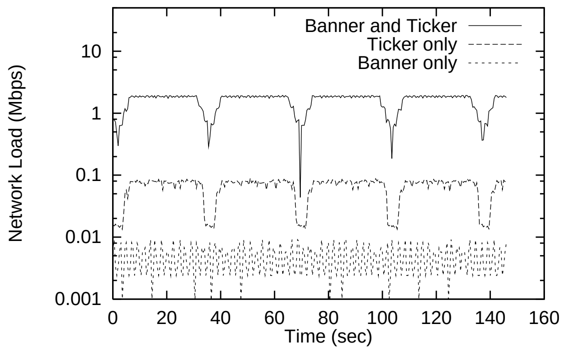



Figure 3: Comparison of network utilization of just the marquee, just the banner, and both at once.

guity [2]. Examples include windows that visually shrink and expand when minimized and maximized, and Office 97/Windows 98 style animated menus that "unfold" or "slide out" upon activation.

A second avenue of idle CPU exploitation is asynchronous multi-threading. Examples include background file copying and spell-checking. Multi-threading has also appeared in user interfaces in the form of asynchronous animations. They can be used to inform users of status, as the fidgety "Office Assistant" characters do for users of Office 97. Often, the animation itself is the application, such as screensavers or animated banner advertisements on webpages.

Animation arguably makes the user experience less static and more engaging, but can exact a high bandwidth cost. The rapid frame rates required for smooth and effective animation can generate a large amount of RDP display traffic. Continuous asynchronous animations also generate "background noise" on the network. So, in the course of exploiting idle CPU and video subsystem resources, Windows developers have created applications whose user interfaces can be potentially ill-behaved in a remote computing environment such as TSE.

**Ancillary Animation**

First, we quantify the network utilization of some simple interface-enhancing animations. Windows supports blinking cursors and even permits the user to adjust the blink rate in the Control Panel. At the maximum setting, a single blinking cursor in the Notepad application generates a consistent 0.50KBps of network activity. Office 97/Windows 98 style menu animations increase the bandwidth utilization of the previously discussed menu navigation test in Word from 39.82KBps to 48.88KBps. The Office 97 Assistant character ClipIt, even when idle, blinks his eyes periodically. This produces a 0.30KBps level of activity. When he becomes more animated, bandwidth usage also increases. When a user saves a file in Word 97, ClipIt folds his paper-clip body into a box and tosses his eyeballs in, indicating that he is storing something valuable. This animation produces a peak rate of 5.00KBps and lasts for approximately 7 seconds.

**Animation as the Application**

Fortunately, these levels of additional network traffic are relatively harmless. Other types of animation, however, can be quite costly. In particular, today's web pages are filled with animated GIF advertisements and Java- and HTML-based stock and news tickers.

To study such media-intensive webpages more carefully, we created a synthetic webpage that included one animated 468x60 pixel GIF banner advertisement and a one-line scrolling news ticker. The page was served by Apache on the *listen* host to Internet Explorer running on *server* but being displayed via RDP on *client*.

This type of animated page is not uncommon on today's web, and might even be considered modest. But, as shown by the top bandwidth trace in Figure 3, simply displaying it in Internet Explorer produces a sustained average network load of 1.60Mbps. The plateaus of higher activity average 1.89Mbps. The periodicity of the trace is due to the periodicity of the scrolling news ticker.

Such levels of network activity make multi-user service over standard 10Mbps Ethernet unfeasible. If just 5 users open their browsers to a page like this, the network link is already totally saturated. Although many TSE administrators may, by policy, prohibit the use of web browsers or enforce the deactivation of webpage animations, this remains an important issue to consider when developing a "realistic" user behavior profile.

**Managing Animation: The Bitmap Cache**

Interestingly, when the two elements of our synthetic webpage are displayed separately, the network load characteristics are markedly different. Figure 3 also shows load traces for displaying the banner advertisement and news ticker each in isolation.

Average bandwidth for the ticker alone is 0.07Mbps, and for the banner alone it is 0.01Mbps. These values do not sum to 1.89Mbps, or even 1.60Mbps, so the network load behavior of RDP is clearly non-linear with respect to the complexity and quantity of this animation.

This behavior implies the presence of a client-side bitmap cache large enough to store all the frames of one animation or the other, but not both. When the two animations are combined, their competing frames overflow the cache and every miss generates a full bitmap transfer over the network. When the frames do fit into the cache, we presume that display data need not be transferred, and only small "swap bitmap" messages are exchanged.

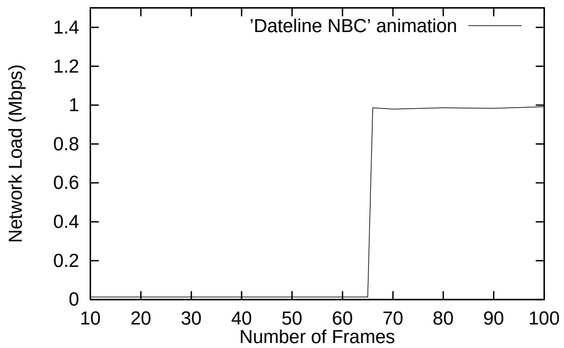



Figure 4: Network load for displaying animations of various frame counts illustrating the size of the bitmap cache.

Indeed, this is the case. 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 [13]. It is used to store icons, button images, glyphs, and the occasional animation frame.

To further study bitmap cache handling of animations, we created a set of custom controlled animations. The GIF Construction Set allowed us to specify the number of frames in the animation as well as the delay between frame displays. Our test files were spinning animations of a "Dateline NBC" logo downloaded from the web. They were 119x119 pixels in size with a color depth of 8bpp. The animations in our test set differed from one another only in the total number of frames and in the delay between frames.

**Overflowing the Cache**

First, we show that as long as cache occupancy is below 100%, RDP activity remains extremely low, but when the cache overflows, RDP activity increases sharply and remains high.

Suppose that the cache is $n$ bytes large, and LRU is the eviction policy. Suppose further that each frame of our animation is, accounting for any compression done by RDP, $m$ bytes. An animation with fewer than $n/m$ frames should therefore fit entirely within the cache and we should see very little network load.

It should also be the case that an animation with $n/m + 1$ frames will generate substantially higher load. If there is one more frame than will fit in the cache, and the eviction policy is LRU, then because of the linear-loop access pattern to the frames, every bitmap request should miss in the cache. At each frame request, the frame needed should have just been evicted to make space for the frame immediately preceding it.

To verify this hypothesis, we created a series of animations whose total number of frames ranged from 25 to 100. Figure 4 supports our hypothesis, showing that while load is just 0.01Mbps for frame counts 10 through 65, it rises sharply to 1.00Mbps for all frame counts above 65.

While LRU may be the appropriate eviction scheme for typical usage, it is exactly the wrong scheme for handling looping animations. Although the cache has room for $n/m$ frames, none of them are in the cache when they are needed. A more intelligent scheme might somehow detect loop patterns and dynamically adjust eviction behavior appropriately.

**Bypassing the Cache**

Flipbook-style animations using a set of bitmapped frames is not the only way to achieve user-interface animation. Animation may also be generated using graphical primitives like line draws. Animated elements of this type will not be aided by the bitmap cache and can generated considerably more traffic.

As a simple test, we ran the "Beziers" screensaver in Windows which draws a pixel-wide loop composed of 10 bezier curves. The loop is continuously bent and twisted as it is translated across the screen. Although we cannot verify with complete certainty that line draw primitives are being used and that the bitmap cache is of no help, the data seem to support such a conclusion. Activating this screensaver generates 96.42 KBps of network load.

While administrators of TSE will almost certainly want to disable screensaver usage, this experiment shows that any dynamic user interface element relying not on bitmaps but on vector- or primitive-based graphics will likely generate considerable load on the network.

## 4 Latency Characteristics

In the previous section, we examined the resource utilization characteristics of TSE. To end-users, load is an abstract concept only relevant to them when it begins to affect their interaction with the system. In this section, we identify and quantify situations in which server and network load translate to user-perceived latency. Specifically, we discuss the latency produced when the processor, memory, and network are saturated.

### 4.1 Processor

First, we evaluate TSE's ability to maintain interactivity under heavy processor load. We generated server-side load using a C program executing a trivial tight loop. Since the code never yields the processor, each concurrent instance increases the scheduler queue length by one. We then opened a single remote session and started Notepad. The tester held down a key, engaging character repeat at a rate of 20cps. Using *tcpdump*, we recorded

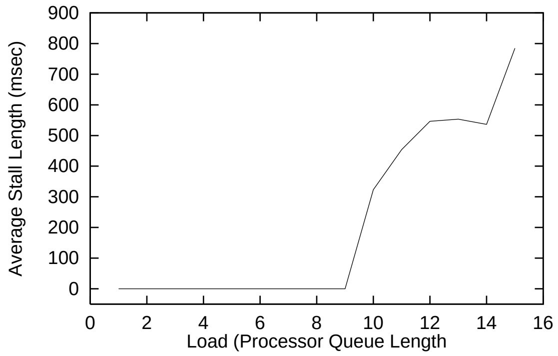



Figure 5: Average stall length experienced by a user given varying CPU loads at the server.

timestamps on the network packets associated with the keystroke and character echo messages.

With a keystroke input rate of 20cps, we would expect the server to respond every 50ms with a screen update message to draw a new character. In our test, we measured the delay between each consecutive screen update message and define the difference between this value and the expected 50ms to be a stall. Figure 5 shows, for varying loads, the average stall length over the course of 60 seconds.

At no load and small loads, TSE performs perfectly, delivering a screen update every 50ms. However, as load approaches 10, the average stall length rises dramatically. At a load of 10, the user spent more than 50% of his time waiting for the server to respond.

In 1993, Evans et. al at SunSoft optimized the System V, R4 scheduler with an eye towards interactivity rather than fairness or real-time requirements [6]. Using a methodology similar to ours, they first demonstrated that in a system based on an unmodified SVR4 kernel, keystroke handling latency increases with load just as TSE does here. They attribute this effect in SVR4 to the lack of protection for interactive processes in the scheduler. Their modified kernel, however, handled load more gracefully, with keystroke handling latency remaining constant and small as load increased. A similar modification to the NT kernel might help it retain good interactive characteristics even when processor utilization is at 100%.

## 4.2 Memory

High page demand on the server will typically force nonactive processes to be paged to disk. 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 as he thinks and reads the document on-screen. During this time, high page demand on the server may force his application out to disk. When he reaches again for the keyboard to scroll down, there will be significant lag as his process is paged back into memory.

To measure this effect in TSE, we opened an instance of Notepad remotely. We then started and let run for 30 seconds on the server a process that sequentially touches a heap of bytes whose total size exceeds the available physical memory. After those 30 seconds, we input a single keystroke into Notepad and measured the time it took for the server to respond with a screen update. We report the results for two different load conditions in the table below.

| Page Demand | Warm-start delay (ms) Min | Max | Avg |
|---|---|---|---|
| *less than 100%* | 50 | 50 | 50 |
| *more than 100%* | 2,400 | 11,900 | 4,000 |

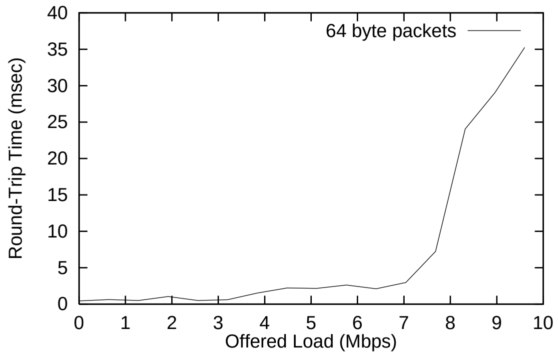
Under heavy page demand, TSE as it runs on our *server* configuration took as long as 12 seconds to reload the process and service the request. With the amount of memory shipping in typical system configurations, today's users of single-user operating systems probably rarely encounter paging behavior, and waiting for a process to reload from disk is likely to be a foreign and highly irritating experience.

Evans et. al also discussed and demonstrated this effect in the SVR4 kernel. Their solution was to throttle non-interactive processes with high page demand, favoring interactive processes instead. Clearly, a memory scheduling scheme better adapted to interactive usage would help the TSE kernel perform better under high memory load.

## 4.3 Network

With TSE's introduction, Windows programmers must consider the fact that their user interfaces may be exported to remote clients over a network.

The mere introduction of a network link introduces latency. A 10Mbps link can transmit 1.25KB per millisecond. Previous work found that latencies exceeding 100ms begin to be noticeable [15]. Therefore, visually discrete screen updates that cannot be expressed in RDP in 125KB may appear sluggish. Our subjective experience with TSE was that even with no network load, remote interaction was noticeably more sluggish than local interaction at the console. On anything less than full Ethernet, users can expect poor performance. A 56Kbps modem connection, for example, can only transmit 7 bytes per millisecond. The average display message size in our trace from Section 3.3.1 was 702 bytes, and visually discrete screen updates are rarely composed of just one



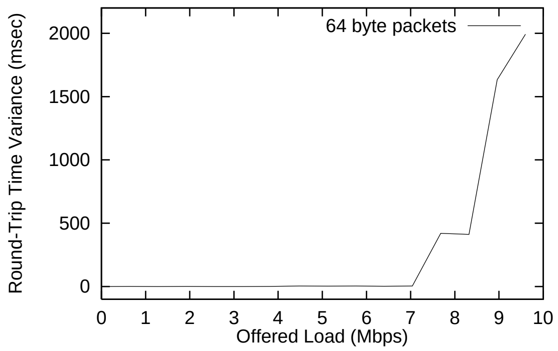




Figure 6: Host-to-host round-trip times (latency) and host-to-host round-trip time variance (jitter) as functions of load.

display message. Therefore, few, if any, operations over such a connection will appear smooth.

Latency also increases dramatically when the network is saturated. To demonstrate, we produced synthetic TCP/IP traffic on our testbed. Figure 6 shows the effect of load on packet latency and jitter (latency variance). 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 a packet size of 64 bytes, which is roughly the size of a keystroke control message. So, the latencies we observed give a realistic lower-bound on the additional latency introduced by network saturation.

As the figure shows, while the network is not saturated, RTT remains low and almost perfectly consistent. However, as the network nears saturation, performance suffers dramatically. The 35ms delay induced at 9.6Mbps load is considerable with respect to known levels of human latency tolerance [15]. The inconsistency of the latency, a phenomenon known as jitter, only compounds the negative impact of network saturation.

## 5 Related Work

We found three technical documents on TSE performance. All are white papers, one published each by Microsoft, Hewlett-Packard, and Compaq [1, 8, 11]. They are all similar, reporting maximum concurrent usage limits for varying combinations of user profiles and server configurations. Their results have limited utility for three reasons. First, they provide little explanation as to the underlying principles of how load is generated, making it difficult to extrapolate limits for configurations other than the few they list. Second, their analyses focus almost exclusively on throughput rather than latency. Third, their user behavior profiles do not include increasingly common tasks like web browsing, which we have shown can produce substantially higher network load.

The Compaq paper stands apart from the HP and Microsoft papers by giving some justification as to why the proffered usage limits are relevant. They identify latency as the key metric for performance:

> The maximum number of clients at an acceptable performance level was determined to exist just prior to a delay in the response time at the client. That is, when the script was executing, there was a pause between entries of characters. Processor queue length was an essential performance factor that was critical in fixing this delay. Processor queue length is the instantaneous length of the processor queue in units of threads. A processor queue length of 12 corresponded to a delay in typing and was consequently used to gauge maximum client support.

They seem to say here that the usage limit is reached when adding one more client pushes typing latency above some threshold.

Harwood and Mathers and Genoway authored books aimed at system administrators on configuring, using, and testing TSE [7, 10]. Danskin and Hanrahan wrote several papers on profiling the X protocol, which provides functionality similar to RDP for Unix systems [3, 4]. Endo et. al and Evans et. al, as discussed earlier, both wrote papers of a rare breed in OS research that address the issue of user-perceived latency [5, 6].

## 6 Conclusions

We have analyzed TSE's resource consumption and latency characteristics.

The transition from NT to TSE has introduced additional compulsory load. TSE has 71% more CPU activity when idle than NT Workstation 4.0 and each user session consumes a minimum of 2.5MB of memory. Typical applications, even with code page sharing, consume at least 1MB for each instance. While RDP is efficient for simple operations like typing and mouse movement, more sophisticated user interaction like web browsing and document scrolling can generate surprisingly high network loads. On 10Mbps Ethernet, this behavior can limit concurrent use to as few as 10 sessions.

In terms of latency, the TSE kernel does not protect interactive processes against high CPU and memory load as well as it could. Evans et. al suggest potential remedies. Network latency even with no load degrades the user experience. Given the average RDP message size, TSE remote sessions would likely be difficult to use over modem links with speeds on the order of tens of Kbps. Without improving some of the latency characteristics of TSE, adopters will need to readjust the interactivity expectations they developed while using single-user systems with copious memory and fast, local video.

## 7 Acknowledgements

We would like to thank our shepherd, Sam Leffler, the anonymous referees of the program committee, Mike Jones, and Chin-hoa.

## References

[1] Compaq Corp. "Performance and Sizing of Compaq Servers with Microsoft Windows NT Server 4.0, Terminal Server Edition," `http://www.compaq.com/support/techpubs/whitepapers/ecg0680698.html`

[2] B. Conner, L. Holden. "Providing A Low Latency User Experience In A High Latency Application," *Proceedings of the 1997 ACM Symposium on Interactive 3D Graphics*, pp. 45-48.

[3] J. Danskin, P. Hanrahan. "Profiling the X Protocol," *Proceedings of the 1994 ACM Conference on Measurement and Modeling of Computer Systems*, pp. 272-273.

[4] J. Danskin, P. Hanrahan. "Higher Bandwidth X," *Proceedings of the 1994 ACM Multimedia Conference*.

[5] Y. Endo, Z. Wang, B. Chen, M. Seltzer. "Using Latency to Evaluate Interactive System Performance," *Proceedings of the 1996 USENIX Symposium on Operating System Design and Implementation*, pp. 185-199.

[6] S. Evans, K. Clarke, D. Singleton, B. Smaalders. "Optimizing Unix Resource Scheduling for User Interaction," *Proceedings of the 1993 USENIX Summer Technical Conference*.

[7] T. Harwood. *Windows NT Terminal Server and Citrix MetaFrame*, New Riders, 1999.

[8] Hewlett-Packard Corp. "Performance and Sizing Analysis of the Microsoft Windows Terminal Server on HP NetServers," `http://www.hp.com/netserver/techlib/perfbriefs/`

[9] N. Hutchinson, L. Peterson, M. Abbott, S. O'Malley. "RPC in the x-Kernel: Evaluating New Design Techniques," *Proceedings of the 1989 ACM Symposium on Operating Systems Principles*, pp. 91-101.

[10] T. Mathers, S. Genoway. *Windows NT Thin Client Solutions: Implementing Terminal Server and Citrix MetaFrame*, MacMillan Technical Publishing, 1999.

[11] Microsoft Corp. "Comparing Microsoft Windows NT Server 4.0, Terminal Server Edition, and UNIX Application Deployment Solutions," `http://www.microsoft.com/ntserver/zipdocs/ts_unix.exe`

[12] Microsoft Corp. "Microsoft Windows NT Server 4.0, Terminal Server Edition: Bringing Windows to Desktops That Can't Run Windows Today," `http://www.microsoft.com/ntserver/zipdocs/tsoverview.exe`

[13] Microsoft Corp. "Microsoft Windows NT Server 4.0, Terminal Server Edition: An Architectural Overview," `http://www.microsoft.com/ntserver/zipdocs/tsarchitecture.exe`

[14] Microsoft Corp. "Microsoft Windows NT Server 4.0, Terminal Server Edition — Capacity Planning," `http://www.microsoft.com/ntserver/zipdocs/tscapacity.exe`

[15] B. Shneiderman. *Designing the User Interface*, Addison-Welsey, 1992.

[16] D. Solomon. *Inside Windows NT: Second Edition*, Microsoft Press, 1998.