Measuring Thin-Client Performance Using Slow-Motion Benchmarking

JASON NIEH, S. JAE YANG, and NAOMI NOVIK
Columbia University

Modern thin-client systems are designed to provide the same graphical interfaces and applications available on traditional desktop computers while centralizing administration and allowing more efficient use of computing resources. Despite the rapidly increasing popularity of these client-server systems, there are few reliable analyses of their performance. Industry standard benchmark techniques commonly used for measuring desktop system performance are ill-suited for measuring the performance of thin-client systems because these benchmarks only measure application performance on the server, not the actual user-perceived performance on the client. To address this problem, we have developed slow-motion benchmarking, a new measurement technique for evaluating thin-client systems. In slow-motion benchmarking, performance is measured by capturing network packet traces between a thin client and its respective server during the execution of a slow-motion version of a conventional benchmark application. These results can then be used either independently or in conjunction with conventional benchmark results to yield an accurate and objective measure of the performance of thin-client systems. We have demonstrated the effectiveness of slow-motion benchmarking by using this technique to measure the performance of several popular thin-client systems in various network environments on Web and multimedia workloads. Our results show that slow-motion benchmarking solves the problems with using conventional benchmarks on thin-client systems and is an accurate tool for analyzing the performance of these systems.

Categories and Subject Descriptors: C.4 [**Performance of Systems**]: Measurement Techniques

General Terms: Experimentation, Measurement, Performance

Additional Key Words and Phrases: Thin-client computing, measurement methodology, client-server, multimedia

1. INTRODUCTION

The rising cost of support and maintenance for desktop systems has fueled a growing interest in thin-client computing. Modern thin-client systems are

Parts of this work appeared as a conference publication in the 2001 USENIX Annual Technical Conference. This work was supported in part by an NSF CAREER Award, NSF grant CCR-0219943, and Sun Microsystems.
Authors' addresses: J. Nieh, Columbia University, 518 Computer Science Building, 1214 Amsterdam Avenue MC0401, New York, NY 10027; email: nieh@cs.columbia.edu; S. Jae Yang, Columbia University, 487 Computer Science Building, 1214 Amsterdam Avenue MC0401, New York, NY 10027; email: sy180@cs.columbia.edu; N. Novik, 511 Computer Science Building, 1214 Amsterdam Avenue MC0401, New York, NY 10027; email: novik@cs.columbia.edu.

designed to provide the same graphical interfaces and applications available on desktop systems while centralizing computing work on powerful servers to reduce administration costs and make more efficient use of shared computing resources.

Although the term thin-client computing has been used to refer to a variety of different client-server computing architectures, the primary feature common to most thin-client systems is that all application logic is executed on the server, not on the client. The user interacts with a lightweight client that is generally responsible only for handling user input and output, such as receiving screen display updates and sending user input back to the server over a network connection. Unlike older client-server architectures such as X [Danskin and Hanrahan 1994], many modern thin-client systems even run the window system on the server. As a result, the client generally does not need many resources, thus requiring fewer upgrades, and can have a simple configuration, reducing support costs. Because of the potential cost benefits of thin-client computing, a wide range of thin-client platforms has been developed. Some application service providers (ASPs) are even offering thin-client service over wide-area networks such as the Internet [Expertcity 2000; Charon Systems; Personable; Runaware].

The growing popularity of thin-client systems makes it important to develop techniques for analyzing their performance, to assess the general feasibility of the thin-client computing model, and to compare different thin-client platforms. However, because thin-client platforms are designed and used very differently from traditional desktop systems, quantifying and measuring their performance effectively is difficult. Conventional benchmarks for desktop system performance cannot be relied upon to provide accurate results when used to measure thin-client systems. Because benchmark applications running in a thin-client system are executed on the server, these benchmarks effectively measure only the server's performance and do not accurately represent the user's experience at the client-side of the system. To make matters more difficult, many of these systems are proprietary and closed-source, making it difficult to instrument them and obtain accurate results.

To address this problem, we introduce slow-motion benchmarking, a new measurement technique for evaluating thin-client systems. In slow-motion benchmarking, performance is measured by capturing network packet traces between a thin client and its respective server during the execution of a slow-motion version of a conventional benchmark application. The slow-motion version can be produced by inserting delays throughout the benchmark to temporally separate discrete visual components. These results can then be used either independently or in conjunction with conventional benchmark results to yield a more accurate and objective measure of user-perceived performance of applications running over thin-client systems.

To demonstrate the effectiveness of slow-motion benchmarking, we evaluated four popular thin-client platforms utilizing widely used industry standard benchmarks and slow-motion versions of these benchmarks. The four platforms we measured were Citrix MetaFrame [Citrix Systems 1998], Microsoft Terminal Services [Microsoft 2000b], AT&T VNC [Virtual Network Computing], and

Sun Ray [Sun Microsystems]. We measured the performance of these systems in running both Web and multimedia applications over various network access bandwidths, ranging from ISDN up to LAN network environments. Our results demonstrate the effectiveness of slow-motion benchmarking as a tool for analyzing thin-client system performance. These results also illustrate the errors that can result from using conventional benchmarks to measure thin-client systems. To further validate the accuracy of the slow-motion benchmarking technique, we directly instrumented the source code for VNC to measure its performance on the benchmarks used for our studies. Our results show that measurements using noninvasive slow-motion benchmarking were within five percent of those obtained through internal instrumentation of VNC.

The rest of this article is organized as follows. Section 2 describes in further detail how thin-client systems operate and then explains the difficulties inherent in measuring the performance of thin-client platforms with conventional benchmarks. Section 3 presents the slow-motion benchmarking technique and discusses how it can be used to measure thin-client performance. Section 4 presents examples of how slow-motion benchmarking can be applied to conventional Web and video benchmark applications so that they can be used to evaluate thin-client systems. Section 5 compares slow-motion benchmarking to conventional benchmarking by presenting experimental results that quantify the performance of popular thin-client systems on Web and multimedia application workloads over different network bandwidths. Section 6 discusses related work. Finally, we summarize our conclusions and discuss opportunities for future work.

2. MEASURING THIN-CLIENT PERFORMANCE

To provide sufficient background for discussing the issues in measuring thin-client performance, we first give a detailed discussion of how thin-client systems operate. In this article, we focus on thin-client systems in which a user's complete desktop computing environment, including both application logic and the windowing system, is entirely run on the server. This is the architecture underlying most modern systems referred to as thin clients, such as Citrix MetaFrame, Microsoft Terminal Services, VNC, and Sun Ray. One of the primary advantages of this approach is that existing applications for standalone systems can be used in such systems without modification.

In this type of architecture, the two main components are a client program that executes on a user's local desktop machine and a server application that executes on a remote system. The end-user's machine can be a hardware device designed specifically to run the client program or simply a low-end personal computer. The remote server machine typically runs a standard server operating system, and the client and server communicate across a network connection between the desktop and server. The client sends input data across the network to the server, and the server, after executing application logic based on the user input, returns display updates encoded using a platform-specific remote display protocol. The updates may be encoded as high-level graphics drawing commands, or simply compressed pixel data. For instance, Citrix MetaFrame encodes display updates as graphics drawing commands whereas VNC encodes display updates as compressed pixel data.

To improve remote display performance, especially in environments where network bandwidth is limited, thin-client systems often employ three optimization techniques: compression, caching, and merging. With compression, algorithms such as zlib or run-length encoding can be applied to display updates to reduce bandwidth requirements with only limited additional processing overhead. With caching, a client cache can be used to store display elements such as fonts and bitmaps, so the client can obtain some frequently used display elements locally rather than repeatedly requesting them from the server. With merging, display updates are queued on the server and then merged before they are sent to the client. If two updates change the same screen region, merging will suppress the older update and only send the more recent update to the client. In merging, display updates are sent asynchronously with respect to application execution, decoupling application rendering of visual output on the server from the actual display of the output on the client. Depending on the merging policy and the network speed, there may be a significant time lapse between the time application rendering occurs on the server and the time the actual display occurs on the client. These optimizations, particularly merging, can make analyzing thin-client performance difficult.

The performance of a thin-client system should be judged by what the user experiences on the client. There are two main problems encountered when trying to analyze the performance of thin-client systems: how to correctly measure performance of the overall system rather than simply the server's performance; and the more subtle problem of how to objectively measure the resulting display quality, particularly given the display update optimizations that may be employed. Three methods that have been used to measure thin-client system performance are internal application measurements with conventional benchmarks, client instrumentation, and network monitoring. We discuss each of these methods below and their limitations.

The first approach, commonly used for traditional desktop systems, is to simply run a conventional benchmark on the system. For instance, a video playback application could be run that reports the frame rate as a measure of performance. However, this does not provide an accurate measure of overall thin-client performance because the application programs are executed entirely on the server. Since application execution is often decoupled from client display, the results reported using internal application measurements might not be an accurate reflection of user-perceived performance on the client. The benchmark program often runs on the server irrespective of the status of the display on the client. A video playback application, for example, would measure the frame rate as rendered on the server, but if many of the frames did not reach the client, the frame rate reported by the benchmark would give an overly optimistic view of performance.

A second, more accurate, measurement method would be to directly instrument the client. If appropriate tracing mechanisms could be inserted into the thin-client system to log input and output display events on the client, very detailed measurements of thin-client performance could be performed. However, many thin-client systems are quite complex and proprietary, thus instrumenting them effectively would not be easy. Even if instrumenting were possible, the invasive modification might alter the systems under measurement. Furthermore, the information that could be obtained within these systems would still not provide a direct measure of user-perceived display quality. Running a video playback application, for example, would result in thousands of display updates. One would still be left with the problem of how to determine how those display updates translate into actual video frames to determine how many video frames were delivered to the client. A more practical problem is that many of the most popular thin-client systems, such as Citrix MetaFrame, Microsoft Terminal Services, Tarantella, and Sun Ray, are proprietary closed systems. Even the specification of the remote display protocol used in most systems is not available.

A third measurement method is network monitoring. Just measuring application performance on the server can be inaccurate and direct thin-client instrumentation is often not possible, however, monitoring network activity between the client and server during the execution of an application can give us a closer approximation to the user-perceived performance of that application on the client-side. We can measure the latency of operations such as Web page downloads as the time between the start and end of client-server communication in response to those operations. However, although this method enables us to more accurately measure the latency of display updates for such operations, we are still left with the problem of determining the resulting display quality. Fast performance is naturally an important metric, but it is insufficient when considered in isolation. The user's interactive experience is also determined by the visual quality of the updates, and in many cases platforms may achieve high-speed screen refresh rates by discarding data, which does not necessarily lead to good visual experience from the user's perspective.

In particular, thin-client systems that use display update merging may drop interim display updates if the client cannot keep up with the rate of display updates generated on the server. A thin-client platform that uses this kind of policy will appear to perform well on benchmarks measuring only the latency of display updates, even at very low bandwidths, because it will simply discard whatever data cannot be transmitted quickly enough. This problem is exacerbated by most conventional benchmarks for measuring graphical performance, which typically execute a rapid sequence of tasks with frequent display changes. For instance, the industry benchmark i-Bench [Ziff Davis Media] measures performance on Web applications with a rapid-fire series of Web page downloads, each page triggering the download of the next page when complete. This technique works for traditional desktop systems, but on a thin-client system, the server can end up finishing one page download and starting the next long before the client has finished displaying the first page. The server may even stop sending the display updates associated with the first page and send the client on to the second regardless of whether the client has finished its display.

Monitoring the amount of data transferred for display updates at different network bandwidths can help to determine when display update merging is occurring, but other problems remain. For one, simple network monitoring cannot quantify the amount of display updates still being discarded at the

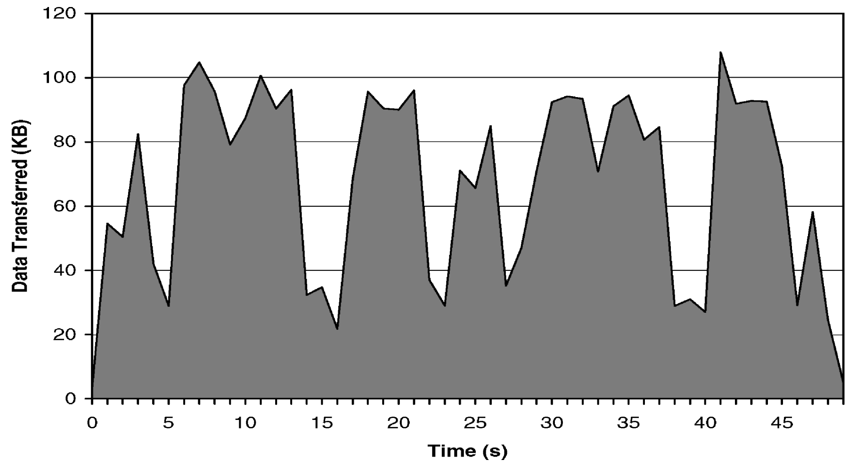

Fig. 1. Data transferred at one-second intervals during a sequence of Web page downloads with no delays under Citrix MetaFrame at 100 Mbps.

highest available bandwidths. In addition, because each thin-client platform encodes display updates in its own proprietary protocol, network monitoring alone cannot determine whether the thin-client platforms are all transmitting the same overall visual display data, making it impossible to effectively compare the performance of the platforms to each other. Monitoring network traffic at the client is an improvement over server-side application measurements, but we still cannot correctly measure overall performance in a way that accounts for both system response time and display quality.

3. SLOW-MOTION BENCHMARKING

To provide a more effective method for measuring thin-client performance, we introduce slow-motion benchmarking. In slow-motion benchmarking, we use network packet traces to monitor the latency and data transferred between the client and the server, but we alter the benchmark application by introducing delays between the separate visual components of that benchmark, such as Web pages or video frames, so that the display update for each component is fully completed on the client before the server begins processing the next one.

Figures 1 and 2 illustrate the difference in network traffic between conventional and slow-motion versions of an i-Bench Web benchmark that downloads a sequence of Web pages. The benchmark is described in Section 4.1. The data presented are from measurements for one thin-client platform, Citrix MetaFrame, with a 100 Mbps network connection between client and server. In the conventional benchmark with no delays, the pages run together and cannot be separately identified, even at this high network bandwidth. In the slow-motion version of the benchmark with delays inserted between each page download, the display update data for each separate page are clearly demarcated.

With slow-motion benchmarking, we process the network packet traces and use these gaps of idle time between components to break up the results on a per-component basis. This allows us to obtain the latency and data transferred

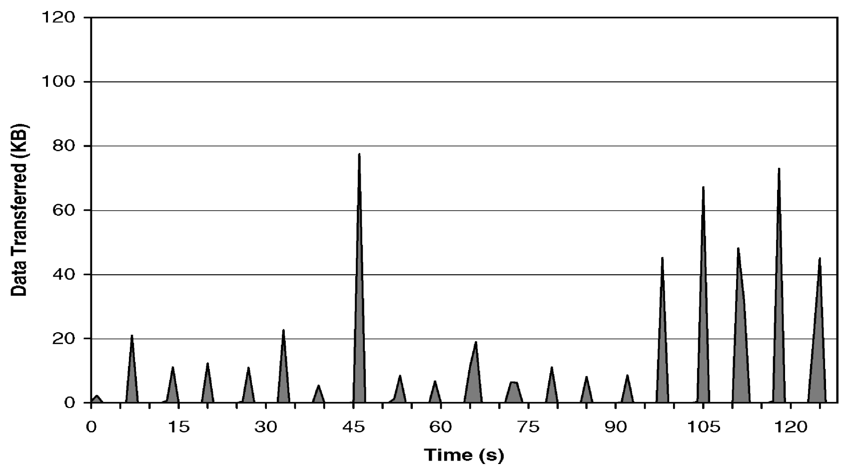

Fig. 2. Data transferred at one-second intervals during a slow-motion version of the same Web page sequence. For visual clarity, only a subset of the full 108-page sequence represented by Figure 1 is shown here.

for each visual component separately. We can then obtain overall results by taking the sum of these results. The amount of the delay used between visual components depends on the application workload and platform being tested. The necessary length of delay can be determined by monitoring the network traffic and making the delays long enough to achieve a clearly demarcated period between all the visual components where client-server communication drops to the idle level for that platform. This ensures that each visual component is discrete and generated completely. In Section 5.2.2, we demonstrate that the amount of delay has no bearing on the results.

Slow-motion benchmarking has many advantages. First and most importantly, it ensures that display events reliably complete on the client so that capturing them using network monitoring provides an accurate and repeatable measure of system performance. Slow-motion benchmarking ensures that clients display all visual components in the same sequence, providing a common foundation for making comparisons among thin-client systems.

Second, slow-motion benchmarking does not require any invasive modification of thin-client systems, which is difficult even for open-source systems such as VNC and nearly impossible for proprietary systems. Moreover, because no invasive instrumentation is required, slow-motion benchmarking does not result in any additional performance overhead for the thin-client system being measured. Although some conventional benchmark applications may need to be modified in order to produce a slow-motion version, others can be used without modification by simply adjusting some application parameters. Examples of each of these cases are discussed further in Section 4.

Third, slow-motion benchmarking provides an effective way to determine when display updates are being discarded. Because the slow-motion benchmarks run at a slower rate, the likelihood that display updates will be discarded is reduced. We can then compare the amount of data transferred for the conventional and slow-motion versions of a given benchmark application to determine whether the display updates are being discarded even at the highest network bandwidths. In Section 4.2, we show how this is particularly useful for measuring the performance of video applications on thin-client systems.

Fourth, slow-motion benchmarking is actually closer to typical user behavior for some applications, notably interactive activities such as Web browsing. Unlike the behavior of Web benchmarks such as those in i-Bench, most users do not click through a hundred Web pages in near-instantaneous succession; they wait for a page to visually complete loading before they move on to the next one.

Finally, the delays introduced with slow-motion benchmarking allow us to obtain results with finer granularity at the level of individual visual components. This is useful for studying the effects of different thin-client remote display mechanisms on different kinds of display data. For instance, some platforms may prove to be better than others at downloading text-only Web pages, whereas others may be superior at graphics-heavy pages. These more detailed results enable us to make better judgments about the various design choices made in each thin-client platform.

In designing slow-motion benchmarking, we made several assumptions. First, we assumed that introducing the delays between visual components would not inherently change the type or amount of data that should be transferred between client and server in a thin-client system. As far as we know, none of the thin-client platforms fundamentally alters the way they encode and send updates based on the amount of time between visual components.

Second, we assumed that there would not be extraneous packets in the data stream that would change our measurements. In particular, we assumed that the delays introduced between visual components would be directly reflected in noticeable gaps in the network packet traces captured. For almost all thin-client platforms and benchmark applications that we measured, there were no packets during the idle periods. However, on certain platforms such as Sun Ray, some small packets were transmitted even during idle periods, possibly to make sure that the connection had not been lost. In such cases, we found that a judicious filtering process based on the volume of idle-time data allowed us to successfully distinguish the data transferred for the pages from the overhead.

Third, we assumed that monitoring network traffic generated by the slow-motion benchmarks was a valid measure of overall performance. Network monitoring allows us to completely measure network and server latency, but may not provide a complete end-to-end measure of client latency. Network monitoring does account for all client latency that occurs between the first and last packet generated for a visual component. However, this technique does not account for any client processing time required for displaying a visual component that occurs before the first network packet or after the last network packet for that component. The impact of this limitation depends on the importance of client latency to the overall performance. If, as we expected, network and server latency were the dominant factors in overall performance, the additional client latency would not be significant. However, if the client latency were a significant component of overall performance, network monitoring might not completely measure end-to-end performance. Client latency would typically be large if the client were heavily loaded. We therefore compensated for this limitation by monitoring client load with standard system monitoring tools such as perfmon and sysload to check whether the client was heavily loaded. We found that client load was generally not an issue and that the network was typically the primary performance bottleneck. The one instance in which this was not the case was for VNC running over high network bandwidths. However, as we discuss in Section 5.2.2, we also instrumented VNC directly and found that the packet traces accounted for all of the client latency for this platform.

The third assumption implies a possibility of achieving exaggerated benchmark performance with slow-motion benchmarking. As with any benchmark, it is possible to demonstrate misleading performance results using slow-motion benchmarking in a way that the results do not correlate with user experience. In particular, if a thin-client system buffers a large amount of data on the client, the performance perceived by monitoring the network traffic may not reflect the latency that would be seen by the user. Although the network monitor may indicate that all of the display data have been transferred, the user may experience substantial additional delay before the display information in the client buffer is actually rendered. A system could be intentionally designed in this way to generate misleading slow-motion benchmarking results, but none of the thin-client systems that we are aware of perform this kind of client buffering in practice.

4. EXAMPLES OF SLOW-MOTION BENCHMARKS

To illustrate how slow-motion benchmarking can be used in practice, we applied the slow-motion benchmarking technique to two existing conventional benchmark applications. The two examples are taken from the Ziff Davis i-Bench benchmark suite version 1.5 [Ziff Davis Media], a benchmarking tool that has been used by numerous computer companies and Ziff Davis Media's eTesting Labs for measuring the performance of a variety of desktop and thin-client systems. The i-Bench benchmarks used were the Web Text Page Load and MPEG1 Video benchmarks, which can be used to measure system performance on Web-based and multimedia applications.

4.1 Web Text Page Load Benchmark

The Web Text Page Load benchmark measures the total time required to download a Javascript-controlled sequence of Web pages from a Web server. In the unmodified benchmark, the Web pages are downloaded one after another without user input using a Javascript-enabled Web browser, such as Netscape or Internet Explorer. Each page contains a small script initiated by the Javascript onLoad handler that immediately begins downloading the next page once the Web browser has finished downloading the current one. The benchmark cycles through a set of 54 unique Web pages twice and then reports the elapsed time in milliseconds on a separate Web page. Not including the final results page which is not included in the time measurements, a total of 108 Web pages is downloaded during this test. The 54 pages contain both text and bitmap graphics, with some pages containing more text and others containing more graphics, with some common graphical elements included on each page.

There are certain problems with using the unmodified Web Text Page Load benchmark for measuring thin-client performance. The first is that since the benchmark would execute in the Web browser on the server of a thin-client system, the Javascript handler may indicate that a given Web page has been completely downloaded on the server and move on to the next page while the given page has not yet been completely displayed on the client. The second problem is that since the benchmark only provides an overall latency measure for downloading an entire sequence of Web pages, it does not provide an accurate measure of per-page download latencies and how performance may vary with page content.

We can address these problems by applying slow-motion benchmarking as follows. The visual components of this benchmark are the individual Web pages. To break up the benchmark into its separate components, we can simply alter the Javascript code used to load the successor page so that it introduces delays of several seconds between Web pages. The delay should be long enough to ensure that the thin client receives and displays each page completely before the thin server begins downloading the next Web page. Furthermore, the delays should be long enough to ensure that there is no temporal overlap in transferring the data belonging to two consecutive pages. A longer delay might be required in systems with lower network bandwidth between client and server. With enough delays to separate the Web pages, we can determine the latency to download each page. We can then sum up the page latencies for each of the 108 Web pages to obtain the total time to download the sequence of Web pages.

By using a slow-motion version of the Web Text Page Load benchmark modified along these lines, we can ensure that each Web page is completely displayed on the client and measure the page load latency and data transferred on a per-page basis. As a result, we can conduct performance comparisons of different thin-client systems knowing that each of them is correctly displaying the same set of Web pages. In addition, we can use the per-page measurements to determine how page download latency and the amount of data transferred vary with page content. By performing these measurements with various network bandwidths between client and server, we can determine how the response time of a thin-client system varies with network access bandwidth. In implementing the Web Text Page Load test, we implicitly assumed a model of user behavior in which users download and display an entire page before moving on to the next one. Although this model is typically true in practice, some more advanced users may at times click ahead to subsequent Web pages without waiting for the current page to display completely. We did not model click-ahead behavior in this implementation of the slow-motion benchmark.

4.2 MPEG1 Video Benchmark

The MPEG1 Video benchmark measures the total time required to play back an MPEG1 video file containing a mix of news and entertainment programming. The video is a 34.75 second clip that consists of 834 352×240 pixel frames with an ideal frame rate of 24 frames per second (fps). The ideal frame rate is the rate a video player would use for playing the video file in the absence of resource limitations that would make this impossible. The total size of the video file is 5.11 MB. In running the video benchmark on a thin-client system, the video player would run on the server and decode and render the MPEG1 video on the server. The remote display protocol of the thin-client system would then send the resulting display updates to the client. Note that unlike streaming MPEG media systems that transmit MPEG video to the client for decoding, thin-client systems first decode the video on the server and then transmit display updates using their own remote display protocol.

There are some problems with using the conventional MPEG1 Video benchmark for measuring thin-client performance. The first is that playback time alone is a poor measure of video performance. Some video players discard video frames when the system is not fast enough to decode and display every frame. The second problem is that in thin-client systems, the system itself may also drop video frames by discarding their corresponding display updates when the network between the client and server is congested. The resulting lower video quality is not properly accounted for by either the playback-time metric or the video player's accounting of dropped video frames.

We can address these problems by applying slow-motion benchmarking as follows. In this case, the visual components of the benchmark are the individual video frames. We can isolate these frames simply by reducing the video playback rate of the benchmark. Because many video playback applications allow a user to select the playback rate, this can be achieved without any alteration to the application or media content. The playback rate should be slow enough to ensure that there is enough time to decode and display each video frame before the next one needs to be processed. Although users would not actually watch video at such a greatly reduced playback rate, the measurements at this reduced playback rate can be used to establish the reference data transfer rate from the thin server to the client that corresponds to a perfect playback without discarded video frames. The data transfer rate can be calculated as the total data transferred divided by the total playback time. We can then compare the data transfer rate at the reduced playback rate with the corresponding full playback rate measurements to determine the video quality achieved at full playback rate. More specifically, we can use the following formula as a measure of video quality *VQ* at a given specified playback rate P.

$$VQ(P) = \frac{\left[\dfrac{\textit{Data Transferred(P)/Playback Time(P)}}{\textit{Ideal FPS(P)}}\right]}{\left[\dfrac{\textit{Data Transferred(slow-mo)/Playback Time(slow-mo)}}{\textit{Ideal FPS(slow-mo)}}\right]}$$

For example, suppose playing a video at an ideal 24 fps rate takes half a minute and results in 10 MB of data being transferred, and playing the video at a slow-motion ideal 1 fps rate takes 12 minutes and results in 20 MB of data being transferred. Then, based on the above formula, the resulting video quality *VQ* at 24 fps will be 0.5 or 50%, which is what one would expect since the 24 fps video playback discarded half of the video data. In our video experiments discussed in Section 5.3, 1 fps was slow enough to allow every frame to be

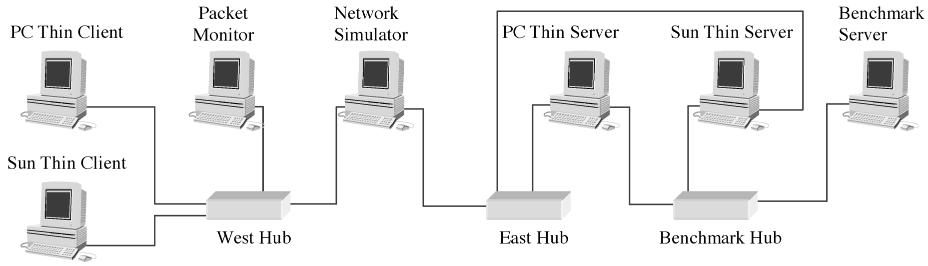

Fig. 3. Testbed configuration.

completely delivered and rendered completely on the client. If that was not the case, the slow-motion frame rate would need to be lowered even further.

This metric provides a useful noninvasive way to measure video quality, but it only accounts for the amount of data discarded and does not account for the fact that some video data may be more important to the overall display quality of a given video sequence than other data. For instance, discarding display updates corresponding to a video frame that looks almost the same as both the previous and next video frames in a sequence would not change the perceived display quality as much as discarding updates corresponding to a video frame that is unlike any of its neighboring frames. At the same time though, as discussed in Section 2, many of the thin-client systems use some form of compression to reduce the data size of their display updates. Compression effectively reduces data size by removing redundant information in the data. If we assume that the amount of unique information in a display update is a measure of its importance, then a compressed display update could be viewed in a rough sense as being scaled according to its importance. In this case, the proposed measure of video quality based on the amount of discarded compressed data effectively accounts for the fact that different display data may be of different importance.

5. EXPERIMENTAL RESULTS

To demonstrate the effectiveness of slow-motion benchmarking, we evaluated four popular thin-client platforms using the conventional and slow-motion versions of the Web and video benchmarks described in Section 4. The platforms we measured were Citrix MetaFrame, Windows Terminal Services, AT&T VNC, and Sun Ray. Section 5.1 describes our experimental design, including the hardware and software testbed we used, the thin-client platform configurations we tested, and the experiments we conducted. Sections 5.2 and 5.3 discuss our measurements and results comparing slow-motion benchmarking against using the unmodified conventional benchmarks. The results also contrast the performance of different thin-client systems on Web and video applications.

5.1 Experimental Design

5.1.1 *Experimental Testbed.* Our testbed, shown in Figure 3, consisted of two pairs of client/server systems, a network emulator machine, a packet monitor machine, and a benchmark server. Only one client/server pair was active

Table I. Summary of Testbed Configurations

Role / Model	Hardware	OS/Window System	Software
PC Thin Client Micron Client Pro	450 MHz Intel PII 128 MB RAM 14.6 GB Disk 10/100BaseT NIC nVidia Riva TNT graphics adapter, 16 MB SDRAM	Windows NT 4.0 Workstation SP6, Caldera OpenLinux 2.4, XFree86 3.3.6, KDE 1.1.2	Citrix ICA Win32 Client, Microsoft RDP5 Client, VNC Win32 3.3.3r7 Client
Sun Thin Client Sun Ray I	100 MHz Sun uSPARC IIep 8 MB RAM 10/100BaseT NIC ATI Rage 128 graphics adapter	Sun Ray OS	N/A
Packet Monitor Micron Client Pro	450 MHz Intel PII 128 MB RAM 14.6 GB Disk 10/100BaseT NIC	Windows 2000 Professional	WildPackets' Etherpeek 4
Network Emulator Micron Client Pro	450 MHz Intel PII 128 MB RAM 14.6 GB Disk 10/100BaseT NIC	Windows NT 4.0 Server SP6a	Shunra Software The Cloud 1.1
PC Thin Server Micron Client Pro (SPEC95 – 17.2 int, 12.9 fp)	450 MHz Intel PII 128 MB RAM 14.6 GB Disk 2 10/100BaseT NICs	Windows 2000 Advanced Server, Caldera OpenLinux 2.4, XFree86 3.3.6, KDE 1.1.2	Citrix MetaFrame 1.8 Win 2000 Terminal Services AT&T VNC 3.3.3r2 for Linux Netscape Communicator 4.72
Sun Thin Server Sun Ultra-10 Creator 3D (SPEC95 – 14.2 int, 12.9 fp)	333 MHz Ultra SPARC IIi 384 MB RAM 9 GB Disk 2 10/100BaseT NICs	Sun Solaris 7 Generic 106541-08, OpenWindows 3.6.1, CDE 1.3.5	Sun Ray Server 1.2_10.d Beta, Netscape Communicator 4.72
Benchmark Server Micron Client Pro	450 MHz Intel PII 128 MB RAM 14.6 GB Disk 2 10/100BaseT NICs	Windows NT 4.0 Server SP6a	Ziff-Davis i-Bench 1.5, Microsoft Internet Information Server
Network Hub Linksys NH1005	3 10/100 5-port Hubs	N/A	N/A

during any given test. The features of each system are summarized in Table I, along with the SPEC95 performance numbers for each server system.

The client/server systems included a Sun Ray thin-client machine and a Sun server, and a PC client and server. The Sun thin server was used only for Sun Ray testing and the PC server was configured as a dual-boot machine to support the various Windows- and Linux-based thin-client systems. The Sun Ray client was considerably less powerful than the PC client, with only a 100 MHz uSPARC CPU and 8 MB of RAM compared to a 450 MHz Pentium II with 128 MB of RAM in the PC client. However, the large difference in client processing power was not a factor in our evaluations, as the client systems were not generally heavily loaded during testing.

As shown in Figure 3, the network emulator machine was placed between the thin-client and thin-server machines to control the bandwidth between them. This emulator ran a software package called The Cloud [Shunra Software], which allowed us to vary the effective bandwidth between the two network interface cards installed in the system. The thin clients and thin servers were separated from one another on isolated 100 Mbps networks. The server-side network was then connected to one of the network interfaces in the network emulator PC, and the client-side network was connected to the other interface.

To ensure that this emulator did not itself introduce extra delay into our tests, we measured round-trip ping times from the client to the server at 100 Mbps, with and without the emulator inserted between the client and the server. There were no significant differences and round-trip ping times were roughly 0.6 ms in both cases.

To monitor the client–server network traffic, we used a PC running Etherpeek 4 [WildPackets, Inc.], a software packet monitor that timestamps and records all packet traffic visible to the PC. As shown in Figure 3, we primarily used the packet monitor to observe client-side network traffic. In order to capture all packet traffic being sent in both directions between the thin client and server, we used hubs rather than switches in our testbed. Because traffic going through a hub is broadcast to all other machines connected to the hub, this enabled us to record network traffic between the client and server simply by connecting the packet monitor to the hub through which the data were passing.

A limitation of this network setup is that the hubs are half-duplex, so that traffic cannot be sent through the hub from client to server and from server to client concurrently. Because most data in these thin-client platforms are traveling from the server to the client in any case, it is unlikely that the half-duplex network added significant delay to our experiments.

Other options are possible, each with its disadvantages. One alternative would be to run a packet monitor on the thin client or thin server, but Etherpeek is highly resource-intensive and would undoubtedly adversely affect performance results. Furthermore, in the case of the Sun Ray thin-client device, it is not possible to run a packet monitor locally on the client, because the device is an appliance that does not allow for any installation of additional software. Another alternative would be to use port-mirroring switches to support full-duplex network connections, but mirroring typically would only allow monitoring of either client to server traffic or vice versa, not both at the same time, as mirroring a duplex port in both directions simultaneously can result in packet loss unless the switch is a high-end type with adequate performance [Curtis 1998].

Finally, we also had a separate benchmark server, which was used to run our modified version of the Web page benchmark described in Section 4.1. To ensure that network traffic from the benchmark server did not interfere with the network connection between thin client and thin server, the benchmark server was connected to the testbed using a separate hub, as shown in Figure 3. Each thin server had two 100 Mbps network interfaces, one connected to the network emulator and through that to the client, the other connected to the benchmark server on a separate channel.

Table II. Thin-Client Platform Configurations

Platform	Citrix MetaFrame (Citrix Win2K)	Terminal Services (RDP Win2K)	VNC Linux	Sun Ray
Display	1024 × 768, 8-bit	1024 × 768, 8-bit	1024 × 768, 8-bit	1024 × 768, 24-bit
Transport	TCP/IP	TCP/IP	TCP/IP	TCP/IP
Options	Disk cache off Memory cache on Compression on	Disk cache off Memory cache on Compression on	Hextile encoding Copyrect on	N/A

5.1.2 *Thin-Client Platforms.* The versions of the four thin-client systems tested are shown in the last column of Table I. Citrix MetaFrame and Terminal Services were run with Windows 2000 servers and VNC and Sun Ray were run with UNIX servers, Linux and Solaris. It was necessary to use different server operating systems because Terminal Services is part of Windows 2000, VNC performs much better on UNIX than Windows [Virtual Network Computing], and Sun Ray only works with Solaris. However, to minimize system differences across thin-client platforms, all platforms except for Sun Ray used the exact same server hardware and same client OS and hardware.

The thin-client platform configurations used for our experiments are listed in Table II. To minimize application environment differences, we used common thin-client configuration options and common applications across all platforms whenever possible. Where it was not possible to configure all the platforms in the same way, we generally used default settings for the platforms in question.

For all of our experiments, the video resolution of the thin client was set to 1024 × 768 resolution with 8 bit color, as this was the lowest common denominator supported by all of the platforms. However, the Sun Ray client was set to 24 bit color, because the Sun Ray display protocol is based on a 24 bit color encoding. Displaying in 8 bit color requires the Sun Ray server to convert all pixels to a pseudo 8 bit color stored in 24 bits of information before they are sent over the network. As a result, displaying in 8 bit color reduces the display quality and increases the server overhead, but does not reduce the bandwidth requirements.

5.1.3 *Benchmarks.* We ran the benchmarks described in Section 4 on each of the four thin-client platforms. We measured the platforms using both the conventional benchmarks and their respective slow-motion versions. We used the network emulator to vary the network bandwidth between client and server to examine the impact of bandwidth limitations on thin-client performance. We measured performance at four network bandwidths: 128 Kbps, 1.5 Mbps, 10 Mbps, and 100 Mbps, roughly corresponding to ISDN, DSL/T1, and LAN network environments, respectively.

To run the Web Text Page Load benchmark, we used Netscape Navigator 4.72, as it is available on all the platforms under study. The browser's memory cache and disk cache were cleared before each test run. In all cases, the Netscape browser window was 1024 × 768 in size, so the region being updated was the same on each system. Nevertheless, Netscape on Windows 2000 performs somewhat differently from Netscape on Linux and Solaris. For instance,

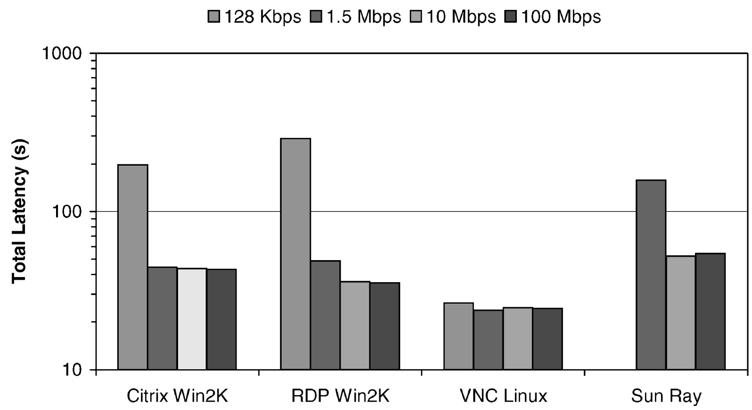

Fig. 4. Total latency for conventional Web benchmark. Using Sun Ray, the benchmark did not complete at 128 Kbps.

in the UNIX version, fonts appear smaller by default and a blank gray page appears between page downloads. These effects would tend to increase the amount of data that would need to be transferred on screen updates using a UNIX-based thin-client platform. Our experience with various thin-client platforms indicates that these effects are minor in general, but should be taken into account when considering small thin-client performance differences across UNIX and Windows systems.

To run the MPEG1 Video benchmark, we used Microsoft Windows Media Player version 6.4.09.1109 for the Windows-based thin clients and MpegTV version 1.1 for the UNIX-based thin clients. In order to facilitate a fair comparison between all platforms despite using two different players, we configured the two players so they had the same size video window and otherwise appeared as similar as possible. As the only portion of the display that is updated is the video window, both UNIX- and Windows-based thin clients are effectively performing the same tasks.

5.2 Web Benchmark Results

5.2.1 *Conventional Benchmark Results.* Figures 4 and 5 show the results of running the unmodified Web Text Page Load benchmark on each of the thin-client platforms. Figure 4 shows the total latency for the unmodified benchmark on each platform. To provide some context for these results, a per-page latency of less than one second has been shown to be desirable to ensure that the flow of a user's browsing experience is not interrupted [Nielsen 1994]. Given the 108 Web pages in the Web Text Page Load benchmark, a total latency of less than 108 seconds is necessary for good performance.

At first glance, it appears that VNC performs extremely well, maintaining the same low latency across all bandwidths and outperforming the other platforms, 46% faster than its nearest competitor, RDP, at 100 Mbps, whereas Sun Ray appears to perform much worse than the other platforms, 20% slower than

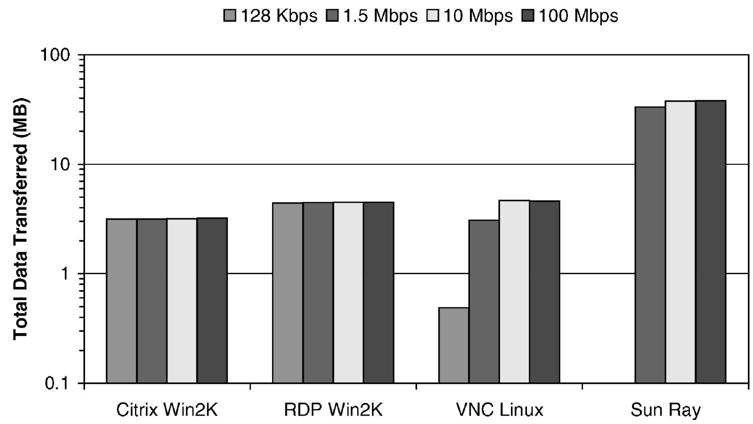

Fig. 5. Total data transferred for conventional Web benchmark.

RDP at 100 Mbps. In addition, both Citrix and VNC still appear to be performing well on the benchmark even at 128 Kbps with average per-page download speed of less than 1 second. However, examining the data-transferred results in Figure 5 shows that VNC discards a substantial amount of display data at lower bandwidths, whereas the other platforms transmit a consistent amount of data and slow down playback as necessary. VNC discards the display data in order to maintain good performances in handling user interactions even at low bandwidth environments.

This highlights the problems with the results from the conventional benchmark. Because we do not know exactly how the data are being encoded and compressed under each platform, we have no way of establishing a baseline for the amount of data that should be transferred to the client by each system. As a result, we have no way of knowing whether the pages are being fully transferred to the client, even at 100 Mbps. We also cannot be sure that each platform is transmitting updates corresponding to the same pages, so the data transfer results are not an accurate measure of the relative efficiency of the platforms. As a result, we cannot draw conclusions about the relative performance of the systems when they are effectively being tested on different sequences of pages.

Visual observation of the platforms during the course of the test revealed another weakness of the conventional benchmark. The pages stream by at such a fast rate that the sequence is not a realistic model of Web-browsing behavior. Real users typically do not interact with a browser in a rapid-fire manner but rather wait for a page to load before clicking on to the next page. This rapid rate causes a pipelining effect that hides the latency that results when each page is loaded from a standing start, which would be experienced in typical use.

5.2.2 *Slow-Motion Benchmark Results.* Figures 6 and 7 show the results of running the slow-motion version of the Web Text Page Load benchmark on the four thin-client platforms. Figure 6 shows the total latency for downloading the 108 Web pages, calculated as the sum of the individual page download latencies. A progressive improvement in performance with increased bandwidth is now

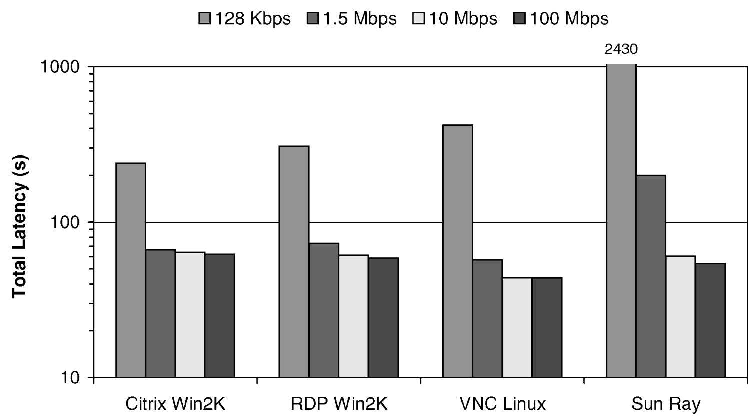

Fig. 6. Total latency for slow-motion Web benchmark.

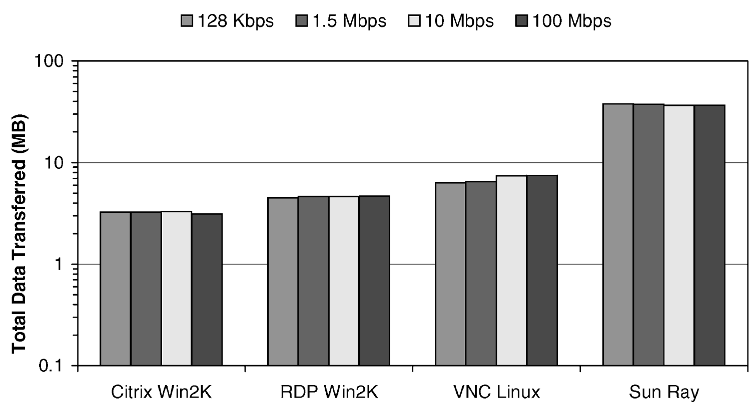

Fig. 7. Total data transferred for slow-motion Web benchmark.

visible for all of the platforms, even VNC Linux, which shows exaggerated performance at lower bandwidths with the conventional benchmark.

As shown in Figure 7, the amount of data transferred now remains almost constant for all of the platforms across all bandwidths. However, we note that VNC transmits slightly less data at lower bandwidths because it uses a client-pull update policy in which each display update is sent in response to an explicit client request. At low network bandwidths, each display update takes longer to transmit, resulting in the client sending fewer update requests and receiving fewer display updates. The unsent interim updates are merged by the server. This does not affect the overall results as we are only interested in the total per-page latency for displaying the entire viewable Web page. The absence of interim updates received at high bandwidths when the client can send more update requests does not affect the final visual quality or per-page download latency.

Comparing Figures 4 and 6, the measurements show that the total latency for the slow-motion benchmark is from 10% (for Sun Ray) to 63% (for VNC) higher than for the conventional benchmark. There are several reasons for the difference in latency. First, none of the thin-client platforms discards display updates for the slow-motion benchmark that affect visual quality. A comparison of Figures 5 and 7 shows that VNC no longer discards numerous display updates for pages in the slow-motion benchmark as it did for the conventional benchmark. VNC transfers more data in the slow-motion case even at 100 Mbps, indicating that VNC was discarding data even at the highest bandwidth when running the conventional benchmark. Second, in using the slow-motion benchmark, each Web page is downloaded from a standing start after the previous page is completely downloaded. None of the latency is hidden by pipelining page downloads. Third, for Citrix and RDP, there were two Web pages, pages 23 and 49 in the second iteration of downloading the pages, that consistently took 3 to 4 seconds to download for the slow-motion benchmark that did not take as long in the conventional benchmark. We discovered that the long delays were due to an unusual interaction between Netscape and these two thin-client platforms. Although these extra delays were not present when using the conventional benchmark, the slow-motion benchmark provides a more realistic measurement of Web-browsing performance.

Figure 6 shows that all of the thin-client platforms deliver acceptable Web browsing performance at LAN network bandwidths and that all of the platforms except Sun Ray provide subsecond per-page response times at 1.5 Mbps as well. As shown in Figure 7, because Sun Ray provides higher-quality 24 bit displays as opposed to the 8 bit displays of the other platforms, it consumes much more network bandwidth, resulting in lower performance at low bandwidths. Aside from the color depth, Sun Ray's large data size is also due to the use of 2-D draw primitives in encoding the display. VNC also employs 2-D draw primitives. Compared to Sun Ray and VNC, Citrix and RDP based on higher-level graphics encoding generated many fewer data. Note that none of the platforms provides good response time at 128 Kbps.

Overall, VNC and Sun Ray were faster at higher network bandwidths, whereas Citrix and RDP performed better at lower network bandwidths. This suggests that the more complex optimizations and higher-level encoding primitives used by Citrix and RDP are beneficial at lower network bandwidths when reducing the amount of data transferred significantly reduces network latency. However, the simpler architectures of VNC and Sun Ray have lower processing overhead and hence perform better when bandwidth is more plentiful and data transfer speed is not the dominant factor.

Slow-motion benchmarking also allows us to obtain actual per-page results. Figure 8[1] shows a subset of the per-page latency results for one of the platforms, VNC. For all pages, VNC provides excellent Web-browsing performance with page download latencies well below a second. Much information about the way the different platforms handle different types of pages is hidden by the aggregate results, but with the conventional benchmark it is impossible to obtain the per-page data. With slow-motion benchmarking, the transferred data size per page can also be obtained. Figure 9 shows more graphically complex pages yielding significantly more data per page.

[1]This graph is identical to Figure 8 in Yang et al. [2001] except for the latency for page 10, which was shown incorrectly in Yang et al. [2001] and is corrected here.

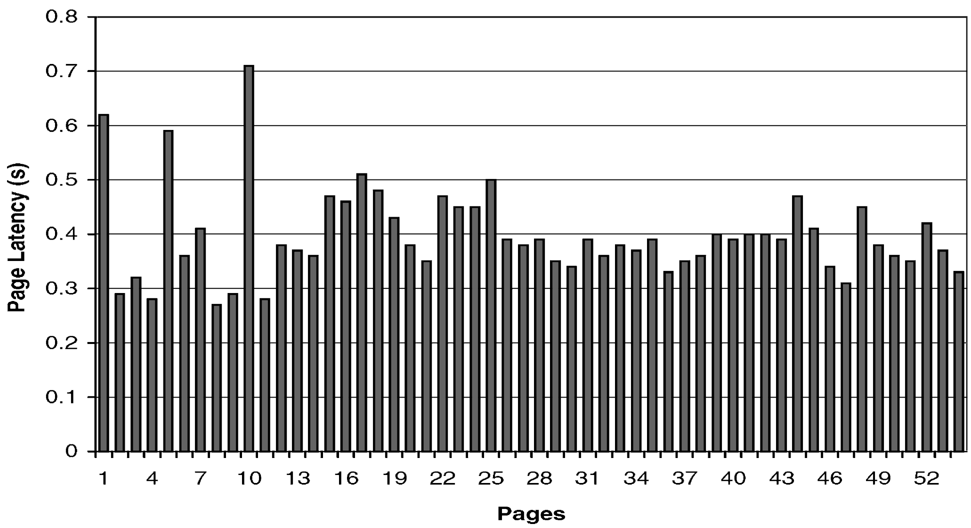

Fig. 8. Per-page latency for VNC Linux running the slow-motion benchmark at 100 Mbps.

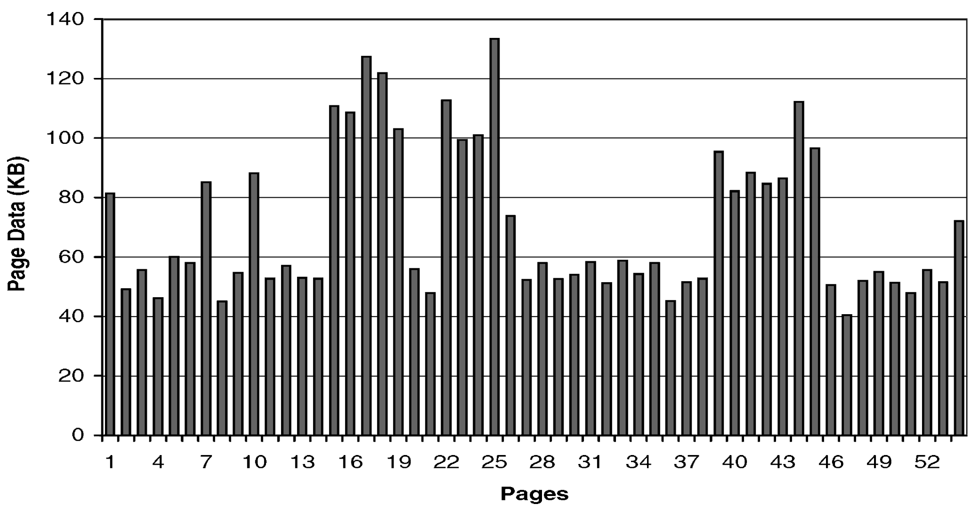

Fig. 9. Per-page data for VNC Linux running the slow-motion benchmark at 100 Mbps. The correlation coefficient between the per-page latency and data is 0.62 at 100 Mbps.

To examine the relationship between the per-page latency and per-page data, we computed the correlation coefficient of the two sets of data. We used Pearson's correlation coefficient, which is the most commonly used measure of linear relationship between two variables [University of Cambridge]. The coefficient can range from −1 to +1, where −1 signifies a perfect negative correlation, and +1 the maximum positive correlation. The coefficient value of 0 represents the lack of correlation.

In examining the corresponding pages in Figures 8 and 9, we can visually detect some correlation between the per-page data and latency. At 100 Mbps,

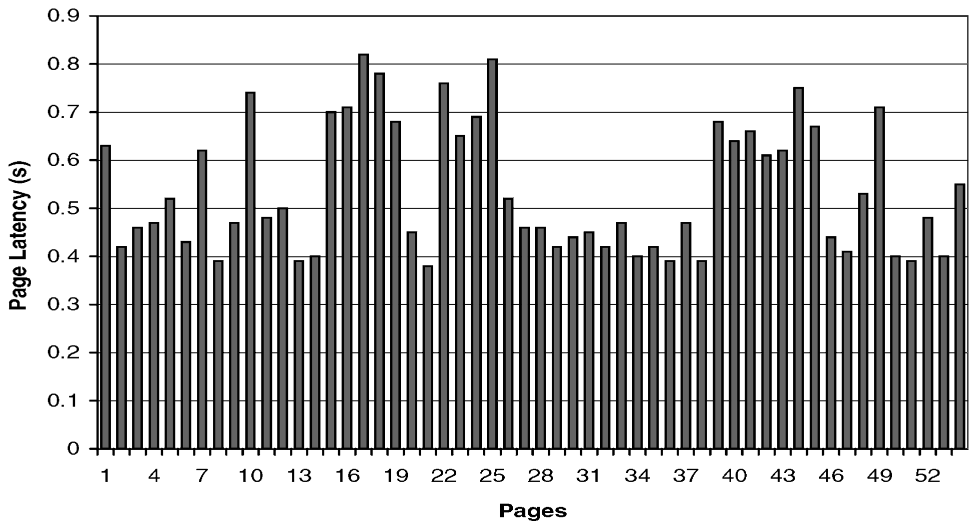

Fig. 10. Per-page latency for VNC Linux running the slow-motion benchmark at 1.5 Mbps.

the correlation coefficient was 0.62. Roughly, pages with more display data took longer to display on screen. One notable anomaly was page 10, which required an exceptionally longer time to display even though its data size was not extraordinarily greater. Upon closer examination, we determined that the latency was due to the delay in transferring the Web page from the i-Bench server to the thin server. Although the portion displayed onscreen was not unusually complex graphically, the Web page had multiple bitmap images below the part that is displayed. At 100 Mbps, the latency results are not purely due to the network bandwidth but also in part affected by the delay caused by the application. When the available bandwidth was reduced, however, the per-page latency and data size showed much stronger correlation.

In Figures 10 and 11, which illustrate the per-page latency and data at 1.5 Mbps, the correlation is much more pronounced. Precisely, the correlation coefficient between the two sets of numbers is 0.93. As the bandwidth between the thin server and client was reduced, the latency depended more strongly on the amount of display data. The delay due to the application performance, which was exposed when the available bandwidth was high, is now hidden in the latency due to the slow network.

To further validate the accuracy and appropriateness of the slow-motion benchmarking technique, we internally instrumented the open-source platform VNC. By instrumenting VNC, we could obtain end-to-end latency measurements that also completely included any client latency. We repeated the experiments with the instrumented version of VNC and compared the results with the packet capture data. On average, the slow-motion results using network monitoring were verified to be within 1.7% of the instrumented VNC results in measuring the total data transferred and within 1.1% in recording the total latency. Furthermore, all of the individual pages showed little difference in latency. As an example, Figure 12 illustrates the per-page latency difference at

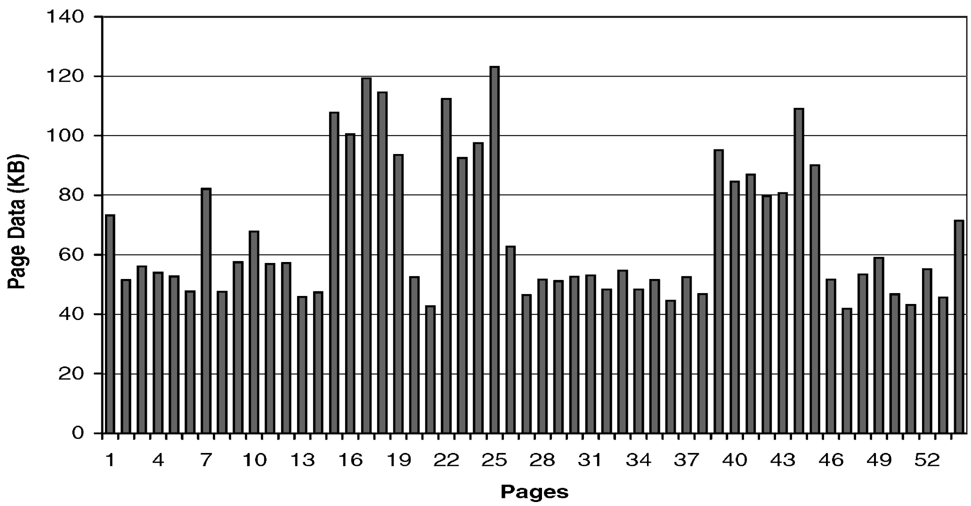

Fig. 11. Per-page data for VNC Linux running the slow-motion benchmark at 1.5 Mbps. The correlation coefficient between the per-page latency and data is 0.93 at 1.5 Mbps.

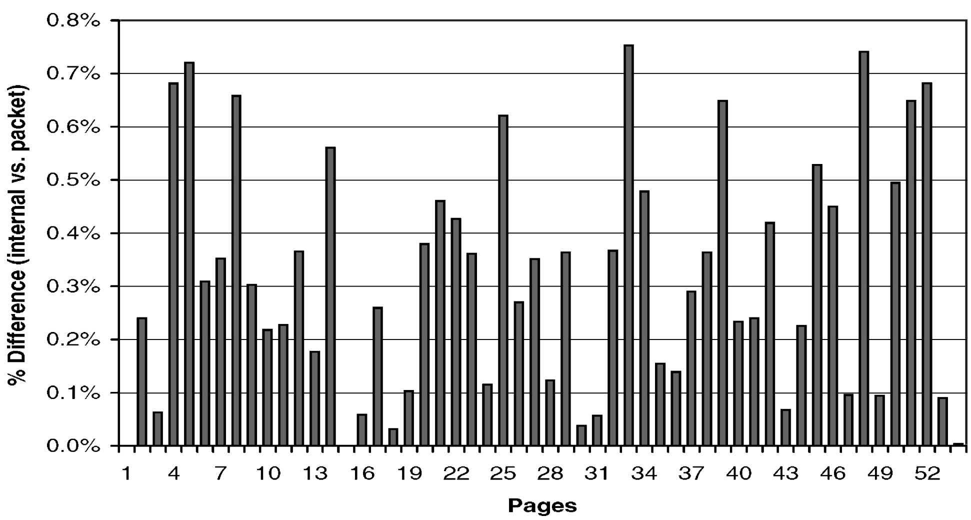

Fig. 12. Percentage difference in per-page latency measured internally and by packet at 1.5 Mbps.

1.5 Mbps. Similarly, Figure 13 shows that there was little variation between the internal measurement and the packet capture method in measuring the transferred data size corresponding to each individual page. The data transfer measurements differed because the internal measurement used did not include the size of packet headers. Of all the thin-client platforms measured, VNC had the highest client load and yet the slow-motion network monitoring results and internal instrumentation results showed little difference. The main reason for this is that the VNC client sends a message back to the server when it has finished processing the latest display update. As a result, the packet traces completely capture the client latency without direct client instrumentation.

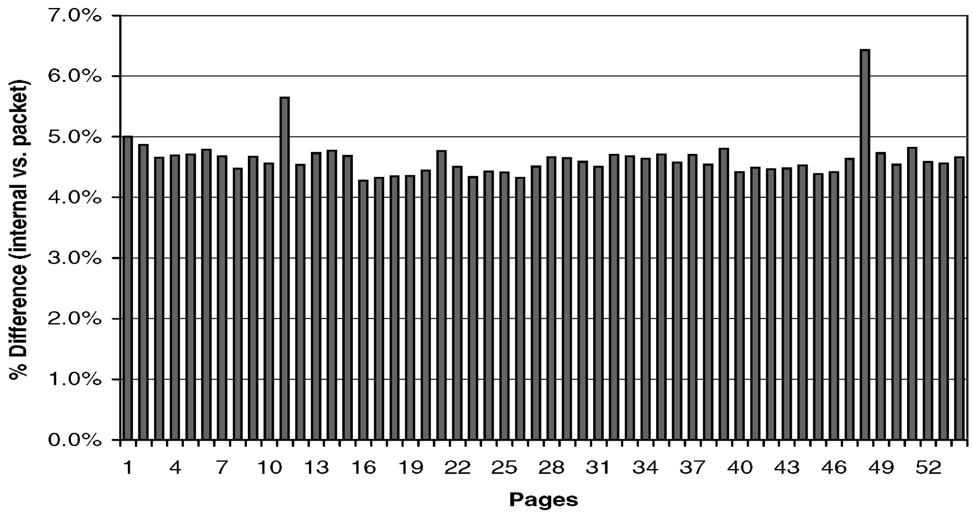

Fig. 13. Percentage difference in per-page data transferred measured internally and by packet capture at 1.5 Mbps.

An important benefit of slow-motion benchmarking for measuring interactive responsiveness is the reproducibility of the results. One way to measure interactive performance is to monitor actual user activity, but it is essentially impossible for a user to repeat the exact same set of experiments with the exact same timing characteristics. In contrast, slow-motion benchmarking can be used to provide better reproducibility of results. We gauged the reproducibility of the slow-motion benchmark data by calculating the standard deviation after five trials of slow-motion Web benchmark at 100 Mbps for each platform. For the latency measurements, the largest standard deviation observed was 5.6% of the mean produced by Sun Ray, but others yielded 2% or lower as shown in Figure 14. The reason for Sun Ray's larger standard deviation may be in part due to the continuous background packets which may cause some random perturbation in the system. The data size measurement showed little variance for all four platforms.

As discussed in Section 3, we have used varying lengths of delay depending on the network bandwidth used for testing. To justify our methodology, we verified that the length of delay has no bearing on the results in the slow-motion Web benchmark. As an experiment we varied the length of delay from three seconds to six in running the slow-motion benchmark at 100 Mbps, and none of the platforms tested varied more than 1% in the total latency.

5.3 Video Benchmark Results

5.3.1 *Conventional Benchmark Results.* Figures 15 and 16 show the results of running the conventional MPEG1 Video benchmark on the four thin-client platforms. Figure 15 shows the playback time for the MPEG video benchmark at the ideal frame rate of 24 fps. Unfortunately, playback time remained

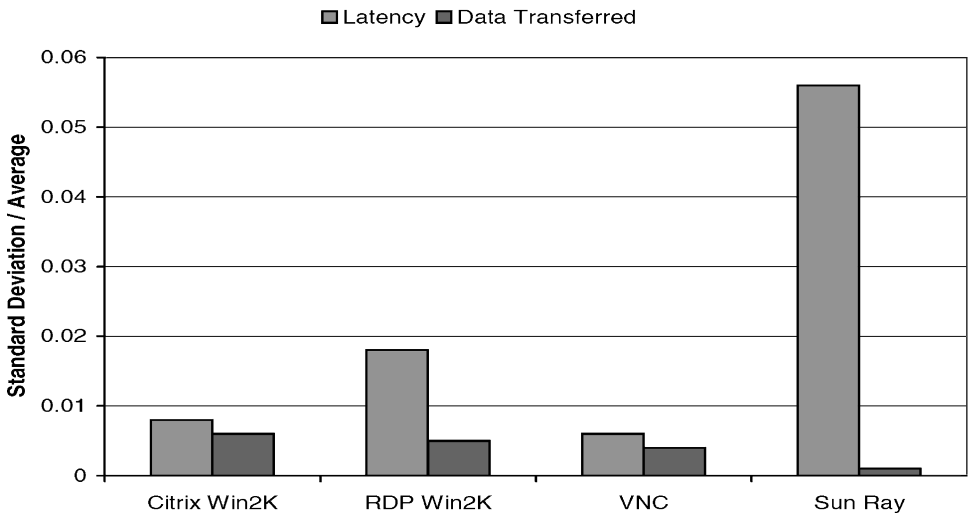

Fig. 14. Standard deviation to the average in latency and transferred data size over five runs of slow-motion benchmark.

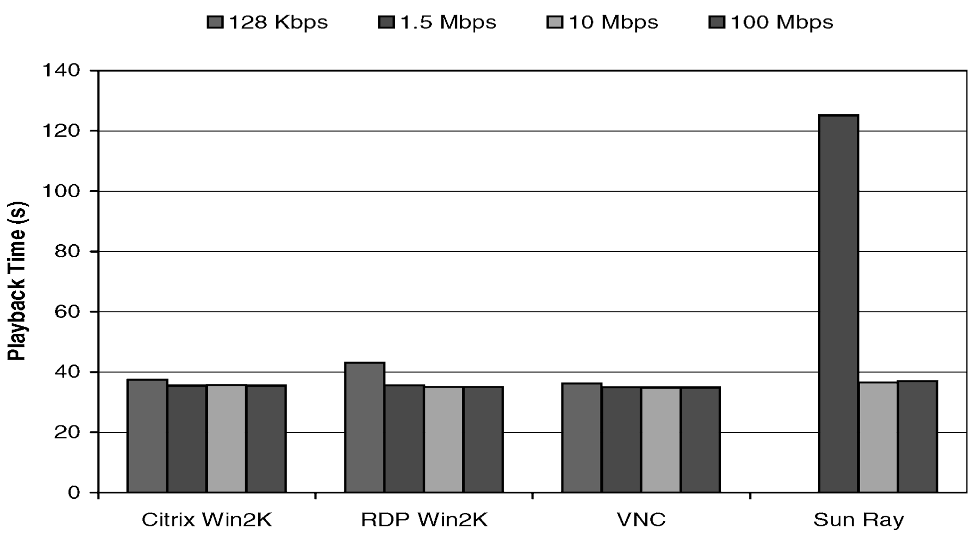

Fig. 15. Playback time for conventional video benchmark. Using Sun Ray, the benchmark did not complete at 128 Kbps.

relatively static on all of the platforms and did not correspond with the subjective performance, which degraded rapidly at lower bandwidths. Because not every frame was delivered to the thin clients in the conventional benchmark, the playback time is ineffective in gauging the performance.

This subjective observation is supported by the data transfer measurements. Figure 16 shows the data transferred during playback, which degrade rapidly at lower bandwidths even when playback time remains low. For instance, the data transferred by VNC at 100 Mbps are 30 times greater than those transferred at

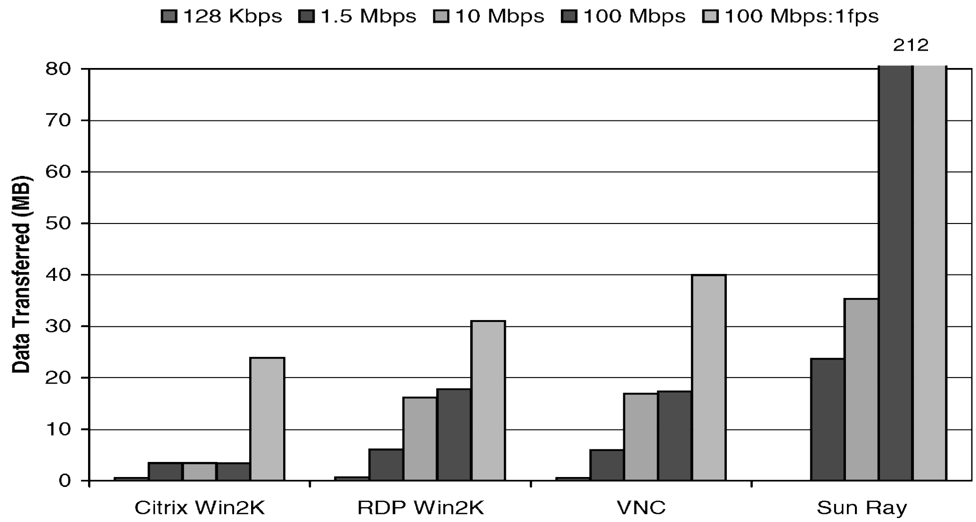

Fig. 16. Total data transferred in conventional video benchmark at 24 fps, and in the slow-motion video benchmark at 100 Mbps bandwidth and 1 fps. Sun Ray data transferred at 100 Mbps were equivalent at both frame rates.

128 Kbps despite the near-constant playback time. Clearly, we cannot use the playback time alone as a measure of the video quality because not all the frames are being fully displayed. The amount of data transferred must be incorporated into any metric of video quality.

We could represent the video quality as a percentage of the ideal data transfer rate. However, this ideal data transfer rate cannot be determined with the conventional benchmark. If we assumed that the 100 Mbps rate was the ideal, we might conclude that all of the platforms perform well at both 100 and 10 Mbps: they maintain the ideal playback time and transmit roughly the same amount of data at both bandwidths. This does not correlate with the subjective performance: visually, only Sun Ray achieved good performance even at 100 Mbps.

5.3.2 *Slow-Motion Benchmark Results.* Slow-motion benchmarking again allows us to clarify the picture. Figure 16 also shows the amount of data transferred when the benchmark was run in slow-motion at a frame rate of 1 fps with network bandwidth of 100 Mbps. At this frame rate, bandwidth limitations were not an issue and each frame of the video was transmitted separately and fully displayed on the client before the subsequent frame was begun. This yields a baseline by which to measure the results from the conventional benchmark, using the formula for video quality described in Section 4.2.

Figure 17 shows this measure of video quality for each of the platforms. We can now obtain a clearer picture of how well each of the platforms performs at high bandwidths and in comparison to each other. This was not possible in looking at the nearly level playback time seen in Figure 15 which obscured the actual performance.

Out of all the thin-client platforms, Sun Ray alone achieves good performance, with 96% video quality at 100 Mbps despite the fact that it sends an

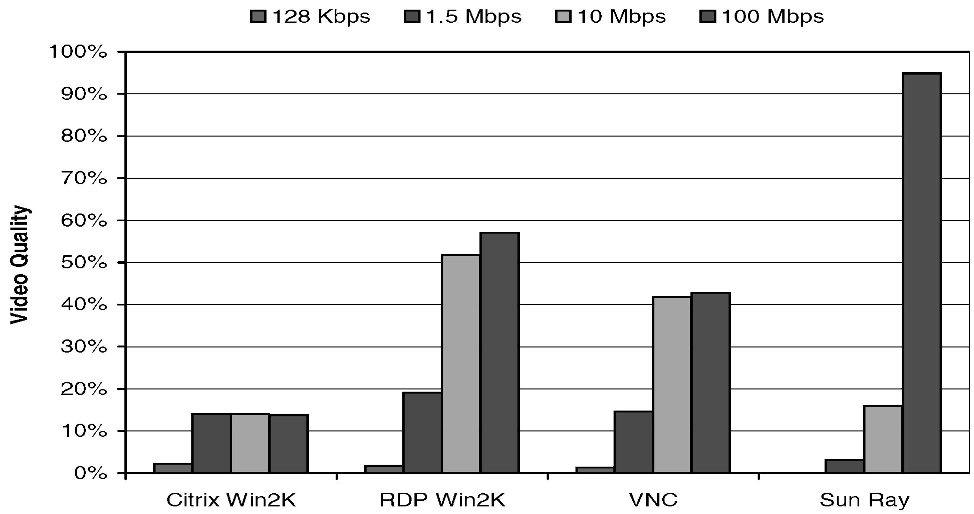

Fig. 17. Video quality as percentage of data transferred in the slow-motion video benchmark.

order of magnitude more data than any other platform at 24 fps. None of the other platforms has good performance even at LAN bandwidths. The fact that Sun Ray sends many more data than any other platform indicates that the poor performance of these other platforms at 100 Mbps is not due to bandwidth limitations but is rather due to their display update mechanisms, which are poorly suited to video applications.

6. RELATED WORK

In this article, we have focused on thin-client systems in which both applications and the window system are completely executed on the server. These systems are the most popular thin-client systems today and many of them have been developed [Citrix Systems 1998; Cumberland et al. 1999; Mathers and Genoway 1998; Microsoft 2000a; Schmidt et al. 1999; Shaw et al. 2000; Sun Microsystems; Tarantella, Inc. 1998; Virtual Network Computing].

Three other types of systems that are sometimes referred to as thin-client systems are network window systems, browser-based systems, and remote control computing systems. The most notable example of a network window system is the X Window system [Scheifler and Gettys 1986]. Unlike the systems discussed in this article, X runs the window system logic on the client and as a result requires more substantial client resources in order to perform well. To run X applications over lower bandwidth networks, a low-bandwidth X (LBX) proxy server extension [Broadway] was developed and released as part of X11R6.3. Browser-based systems employ a Web browser client as a user interface to an application server. These systems require applications to be modified to support a Web-based interface. Remote control computing systems such as Laplink [LapLink.com, Inc. 1999] and PC Anywhere [PC Anywhere] enable users to remotely control a PC by sending screen updates to remote client PCs.

They also run all application and window system logic on the server, but they do not support multiple users at the same time.

There have been several studies of thin-client performance that have focused on evaluating one or two systems. Danskin and Hanrahan [1994] conducted an early study of the X protocol. Schmidt et al. [1999] examined the performance of Sun Ray. Citrix and Microsoft have published some performance data based on studies of their respective thin-client products [Expand Networks 2000; Microsoft 2000b]. All of these studies relied on source code access for internal system instrumentation. Wong and Seltzer [1999] studied the performance of the Windows NT Terminal Server for office productivity tools and Web browsing by monitoring network traffic generated from a real user session. This provides a human measure of user-perceived performance, but makes repeatable results difficult. Tolly Research [2000] measured the performance of Citrix MetaFrame on various scripted application workloads, however, the study suggests that problems in using conventional scripted application workloads as described in this article were not properly considered.

A few performance studies have compared a wider range of thin-client systems. Some of our previous work led to the development of slow-motion benchmarking [Nieh et al. 2000]. Howard [2000] has presented performance results for various hardware thin clients based on tests from the i-Bench benchmark suite. This work suffers from the same problems in measurement technique that we described in Section 2. It relies on the results reported by the conventional benchmarks, which only measure benchmark performance at the server-side. In addition, the work was based on Microsoft Internet Explorer 5.01, which does not properly interpret the Javascript onLoad handler used in the i-Bench Web Text Page Load benchmark. This causes successive pages to begin loading before the previous pages have fully displayed, resulting in unpredictable measurements of total Web page download latencies. Netscape Navigator 4.7 does not suffer from this problem, which is one of the reasons we used this browser platform for our work.

7. CONCLUSIONS AND FUTURE WORK

We have introduced slow-motion benchmarking, a new measurement technique that requires no invasive instrumentation and yet provides accurate measurements for evaluating thin-client systems. Slow-motion benchmarking introduces delays into conventional benchmark applications to isolate the visual components of those benchmarks. This ensures that the components are displayed correctly on the client when the benchmark is run, even when the client display is decoupled from the server processing as in many thin-client systems. Slow-motion benchmarking utilizes network traffic monitoring at the client rather than relying on application measurements at the server to provide a more complete measure of user-perceived performance at the client.

We have demonstrated the effectiveness of slow-motion benchmarking on a wide range of popular thin-client platforms. Our quantitative results show that slow-motion benchmarking provides far more accurate measurements than conventional benchmarking approaches that have been used for evaluating thin-client systems. Our comparisons across different thin-client systems indicate that these systems have widely different performance on Web and video applications. Our results suggest that current remote display mechanisms used in thin-client systems may be useful for Web browsing at lower network bandwidths. However, these same mechanisms may have an adverse impact on the ability of thin-client systems to support multimedia applications.

We are currently using slow-motion benchmarking to evaluate a wide range of thin-client platforms in different network environments. As ASPs continue to increase in popularity, one important area of research is evaluating the performance of thin-client computing in wide-area network environments. Slow-motion benchmarking provides a useful tool for characterizing and analyzing the design choices in thin-client systems to determine what mechanisms are best suited for supporting future wide-area computing services.

ACKNOWLEDGMENTS

We thank Haoqiang Zheng for developing the instrumented version of VNC used in our experiments. Haoqiang, Albert Lai, Rahul Joshi, and Carla Goldburg all assisted with many of the thin-client performance measurements and helped set up the thin-client testbed. Allyn Vogel of Ziff Davis Media, Inc. provided us with valuable information on i-Bench.

REFERENCES

BROADWAY. Broadway / X web FAQ. Available at http://www.broadwayinfo.com/bwfaq.htm.

CHARON SYSTEMS. Available at http://www.charon.com.

CITRIX SYSTEMS 1998. *Citrix MetaFrame* 1.8 *Backgrounder, Citrix White Paper*. Boca Raton, Fla.

CUMBERLAND, B. C., CARIUS, G., AND MUIR, A. 1999. *Microsoft Windows NT Server* 4.0*, Terminal Server Edition: Technical Reference*. Microsoft Press, Redmond, Wash.

CURTIS, J. 1998. Port mirroring: The duplex paradox. *Network World Fusion*.

DANSKIN, J. AND HANRAHAN, P. 1994. Profiling the X protocol. In *Proceedings of the SIGMETRICS Conference on Measurement and Modeling of Computer Systems*, SIGMETRICS (Nashville, Ten.).

EXPAND NETWORKS 2000. *Enterprise Caching and Citrix ICA: Performance Benchmark Test*. Expand Networks, Roseland, N.J.

EXPERTCITY. 2000. *Desktop Streaming Technology and Security, Expertcity White Paper*. Santa Barbara, Calif.

HOWARD, B. 2000. Thin is back. *PC Magazine, Ziff Davis Media 19,* 7 (Apr.).

LAPLINK.COM, INC. 1999. *LapLink 2000 Users Guide*. LapLink.com, Inc., Bothell, Wash.

MATHERS, T. W. AND GENOWAY, S. P. 1998. *Windows NT Thin Client Solutions: Implementing Terminal Server and Citrix MetaFrame*. Macmillan Technical, Indianapolis.

MICROSOFT. 2000a. *Microsoft Windows NT Server* 4.0*, Terminal Server Edition: An Architectural Overview, Technical White Paper*. Microsoft Corporation, Redmond, Wash.

MICROSOFT. 2000b. *Windows* 2000 *Terminal Services Capacity and Scaling, Technical White Paper*. Microsoft Corporation, Redmond, Wash.

NIEH, J., YANG, S. J., AND NOVIK, N. 2000. A comparison of thin-client computing architectures. Tech. Rep. CUCS-022-00, Department of Computer Science, Columbia University, November.

NIELSEN, J. 1994. *Usability Engineering*. Morgan Kaufman, San Francisco.

PC ANYWHERE. Available at http://www.symantec.com/pcanywhere.

PERSONABLE. Available at http://www.personable.com.

RUNAWARE. Available at http://www.runaware.com.

SCHEIFLER, R. W. AND GETTYS, J. 1986. The X Window System. *ACM Trans. Graph. 5*, 2 (Apr.).

SCHMIDT, B. K., LAM, M. S., AND NORTHCUTT, J. D. 1999. The interactive performance of SLIM: A stateless, thin-client architecture. In *Proceedings of the Seventeenth ACM Symposium on Operating Systems Principles*.

SHAW, A., BURGESS, K. R., PULLAN, J. M., AND CARTWRIGHT, P. C. 2000. Method of Displaying an Application on a Variety of Client Devices in a Client/Server Network. US Patent US6104392.

SHUNRA SOFTWARE. The Cloud. Available at http://www.shunra.com.

SUN MICROSYSTEMS. Sun Ray 1 Enterprise Appliance. Available at http://www.sun.com/products/sunray1.

TARANTELLA, INC. 1998. *Tarantella Web-Enabling Software: The Adaptive Internet Protocol, A SCO Technical White Paper*. Tarantella, Inc.

TOLLY RESEARCH 2000. *Thin-Client Networking: Bandwidth Consumption Using Citrix ICA*. Tolly Research.

University of Cambridge. Maths Thesaurus. Available at http://thesaurus.maths.org.

VIRTUAL NETWORK COMPUTING. Available at http://www.uk.research.att.com/vnc.

WILDPACKETS, INC. Etherpeek 4. Available at http://www.wildpackets.com.

WONG, A. Y. AND SELTZER, M. 1999. Evaluating Windows NT terminal server performance. In *Proceedings of the Third USENIX Windows NT Symposium* (USENIX, Seattle).

YANG, S. J., NIEH, J., AND NOVIK, N. 2001. Measuring thin-client performance using slow-motion benchmarking. In *Proceedings of USENIX 2001 Annual Technical Conference* (USENIX, Boston).

ZIFF DAVIS MEDIA. i-Bench version 1.5. Available at http://www.i-bench.com.

Received August 2001; revised September 2002; accepted October 2002