THINC: A Virtual Display Architecture for Thin-Client Computing

Ricardo A. Baratto, Leonard N. Kim, and Jason Nieh
Department of Computer Science
Columbia University, New York, NY, USA
{ricardo, lnk2101, nieh}@cs.columbia.edu

ABSTRACT

Rapid improvements in network bandwidth, cost, and ubiquity combined with the security hazards and high total cost of ownership of personal computers have created a growing market for thin-client computing. We introduce THINC, a virtual display architecture for high-performance thin-client computing in both LAN and WAN environments. THINC virtualizes the display at the device driver interface to transparently intercept application display commands and translate them into a few simple low-level commands that can be easily supported by widely used client hardware. THINC's translation mechanism efficiently leverages display semantic information through novel optimizations such as offscreen drawing awareness, native video support, and server-side screen scaling. This is integrated with an update delivery architecture that uses shortest command first scheduling and non-blocking operation. THINC leverages existing display system functionality and works seamlessly with unmodified applications, window systems, and operating systems.

We have implemented THINC in an X/Linux environment and compared its performance against widely used commercial approaches, including Citrix MetaFrame, Microsoft RDP, GoToMyPC, X, NX, VNC, and Sun Ray. Our experimental results on web and audio/video applications demonstrate that THINC can provide up to 4.8 times faster web browsing performance and two orders of magnitude better audio/video performance. THINC is the only thin client capable of transparently playing full-screen video and audio at full frame rate in both LAN and WAN environments. Our results also show for the first time that thin clients can even provide good performance using remote clients located in other countries around the world.

Categories and Subject Descriptors: C.2.4 Computer-Communication-Networks: Distributed Systems – client/server

General Terms: Design, Experimentation, Performance

Keywords: thin-client computing, remote display, virtualization, mobility

1. INTRODUCTION

In the last two decades, the centralized computing model of mainframe computing has shifted toward the more distributed model of desktop computing. However, as these personal desktop computers become prevalent in today's large corporate and government organizations, the total cost of owning and maintaining them is becoming unmanageable. The use of mobile laptops and handheld devices to store and process information poses additional administration and security issues. These devices often contain sensitive data that must be carefully secured, yet the devices themselves must travel in insecure environments where they can be easily damaged, lost, or stolen. This management and security problem is particularly important for the medical community, given the increasing use of computing in medicine, the urgent need to comply with HIPAA regulations [17], and the huge privacy consequences for lost patient data.

Thin-client computing offers a solution to the rising management complexity and security hazards of the current computing model by leveraging continued improvements in network bandwidth, cost, and ubiquity to return to a more centralized, secure, and easier-to-manage computing strategy. A thin-client computing system consists of a server and a client that communicate over a network using a remote display protocol. The protocol allows graphical displays to be virtualized and served across a network to a client device, while application logic is executed on the server. Using the remote display protocol, the client transmits user input to the server, and the server returns screen updates of the user interface of the applications to the client.

The thin-client approach provides several significant advantages over traditional desktop computing. Clients can be essentially stateless appliances that do not need to be backed up or restored, require almost no maintenance or upgrades, and do not store any sensitive data that can be lost or stolen. Mobile users can access the server from any client and obtain the same persistent, personalized computing environment. Server resources can be physically secured in protected data centers and centrally administered, with all the attendant benefits of easier maintenance and cheaper upgrades. Computing resources can be consolidated and shared across many users, resulting in more effective utilization of computing hardware. It is not surprising that the popularity of thin clients continues to rise [12] and the market for thin-client systems is expected to grow substantially over the next five years [38, 43].

Moreover, the key remote display functionality used in thin clients to decouple display from application execution

SOSP'05, October 23–26, 2005, Brighton, United Kingdom.
Copyright 2005 ACM 1-59593-079-5/05/0010 ...$5.00.

over a network enables a myriad of other benefits. Remote users can travel and access their full desktop computing environment from anywhere [16, 37]. Applications written for one platform can be remotely displayed on a completely different one without rewrite. Scientists can gain full access at their desktops to specialized computer-controlled scientific instrumentation located at remote locations. Since display output can be arbitrarily redirected and multiplexed over the network, screen sharing among multiple clients becomes possible. Thus, groups of users distributed over large geographical locations can seamlessly collaborate using a single shared computing session. Furthermore, by mirroring local display output and redirecting it over the network, instant technical support can be provided [11, 14, 16] with the ability to see exactly what the user sees on the desktop, enabling problems to be diagnosed and corrected much more quickly. Finally, virtualized computing infrastructure such as virtual machines can leverage remote display to avoid display device dependencies and display anywhere, further decoupling their execution from underlying hardware.

However, thin clients face a number of technical challenges before achieving mass acceptance. The most salient of these is the need to provide a high fidelity visual and interactive experience for end users across the vast spectrum of graphical and multimedia applications commonly found on the computing desktop. While previous thin-client approaches have focused on supporting office productivity tools in LAN environments and reducing data transfer for low bandwidth links such as ISDN and modem lines, they do not effectively support more display-intensive applications such as multimedia video, and they are not designed to operate effectively in higher latency WAN environments. WAN performance is particularly important given the growing number of thin-client application service providers attempting to provide desktop computing services over the Internet [5, 16, 33].

We introduce THINC (*THin-client InterNet Computing*), a virtual display architecture for thin-client computing that provides high fidelity display and interactive performance in both LAN and WAN environments. THINC leverages the standard video driver interface, a well-defined, low-level, device-dependent layer that exposes the video hardware to the display system. Instead of providing a real driver for a particular display hardware, THINC introduces a simple virtual display driver that intercepts drawing commands at the device layer, packetizes them, and sends them over the network to a client device to display. Using a standard interface enables THINC to work seamlessly with existing unmodified applications, window systems, and operating systems. As a result, THINC avoids reimplementing existing display functionality in window servers and can leverage any continuing advances in window server technology.

THINC's remote display protocol consists of a small set of efficient low-level commands that mirror the video display driver interface and are easy to implement and accelerate using widely-available commodity client video hardware. THINC avoids the complexity and overhead of directly implementing higher-level graphics commands used by applications. Instead, it transparently maps them to its protocol command set. This is done by taking advantage of information available at the video display driver interface and and preserving that information throughout display processing using transformation optimizations such as offscreen drawing awareness and native video support. These transformations encode display updates in a manner that is more computationally and bandwidth efficient than commonly used compression-based approaches [41, 16].

THINC carefully partitions functionality between client and server using low-latency mechanisms to provide fast performance even in high latency WAN environments. THINC introduces a shortest-job-first display command scheduling to improve response time for interactive applications, a low-latency push display update model that minimizes synchronization costs between client and server, and a non-blocking drawing pipeline that integrates well with and maximizes the performance of today's single-threaded window servers. THINC also provides server-side screen scaling, which minimizes display bandwidth and processing requirements for small display handheld devices.

We have implemented THINC as a virtual display driver in the X window system and measured its performance on real applications. We have compared our THINC prototype system against the most current and widely-used thin-client systems, including Citrix MetaFrame XP [10], Microsoft Remote Desktop [11], Sun Ray [36], VNC [41], GoToMyPC [16], X [35], and NX [28]. All of these systems are commercially-deployed, continue to be heavily developed and supported by substantial engineering efforts, and have recently released versions available. Rather than just discussing the benefits of THINC compared to other approaches, we conducted a direct comparison with highly-tuned commercial and open-source thin-client systems to quantitatively measure the performance of THINC versus the current state-of-the-art.

Our experimental results on web and multimedia applications in various network environments demonstrate that THINC can provide an order of magnitude better performance than many of these other approaches. These results illustrate the importance of not just the choice of display commands used, but also how the mapping of application-level drawing commands to protocol primitives can affect performance. THINC's approach results in superior overall application performance and network bandwidth usage. Most notably, it is the only system capable of providing audio/video playback with full-screen video at full frame rate. As a result, THINC provides a key technology for enabling application service providers and utility computing with modern, multimedia-oriented applications [4].

This paper presents the design and implementation of THINC. Section 2 provides background and discusses related work. Section 3 presents an overview of the system architecture and display protocol. Section 4 discusses the key translation mechanisms used in THINC. Section 5 presents THINC's scheduling mechanisms to improve system interactivity. Section 6 describes THINC's screen scaling support for heterogeneous display devices. The implementation of THINC as a virtual display driver in the X window system is discussed in Section 7. Section 8 presents experimental results measuring THINC performance and comparing it against other popular commercial thin-client systems on a variety of web and multimedia application workloads. Finally, we present some concluding remarks.

2. BACKGROUND AND RELATED WORK

Because of the importance of developing effective thin-client systems, many alternative designs have been proposed. These approaches can be loosely classified based on three related design choices: (1) where the graphical user interface of applications is executed, (2) how display commands from applications are intercepted so that display updates can be sent from server to client, and (3) what display primitives are used for sending display updates over the network.

Older thin-client systems such as Plan 9 [29] and X [35] provide remote display functionality by pushing all user interface processing to the client computer. Application-level display commands are not processed on the server, but simply forwarded to the client. This division of work is more apparent in X, where, the client is referred to as the "X server", and applications running on the server are called "X clients". X applications perform graphics operations by calling library functions in charge of forwarding application-level display commands over the network to the X server. X commands present a high-level model of the overall characteristics of the display system, including descriptions of the operation and management of windows, graphics state, input mechanisms, and display capabilities of the system. By running the user interface on the client, user interface interactions that do not involve application logic can be processed locally without incurring network latencies. The use of high-level application display commands for sending updates over the network is also widely thought to be bandwidth efficient.

However, there are several important drawbacks to this approach. First, since application user interfaces and application logic are usually tightly coupled, running the user interface on the client and application logic on the server often results in a need for continuous synchronization between client and server. In high-latency WAN environments, this kind of synchronization causes substantial interactive performance degradation [22]. Second, the use of high-level application display commands, such as those used by X, in practice turns out to be not very bandwidth efficient [22, 36]. Third, the window server software used to process application user interfaces is large and complex. Maintaining that software on the client requires frequent updates and software maintenance costs contrary to the zero administration goals of thin clients. Fourth, as application user interfaces become richer, they impose more complex processing requirements. Running the user interface on the client necessitates replacing clients at the same high frequency as traditional desktop PCs in order to run the latest applications effectively. Otherwise, they grow outdated quickly as X-based terminals did when web browsers came out in the early 1990s. Finally, storing and managing all display state at the client makes it difficult to support seamless user mobility across locations.

Proxy extensions such as low-bandwidth X (LBX) [44] and NoMachine's NX [28] have been developed to address some of these problems and improve X performance. LBX has been shown to have poor performance [21] compared to other thin-client systems [42]. NX is a more recent development that provides good X protocol compression and reduces the need for network round trips to improve X performance in WAN environments. Neither of these systems address the maintenance costs associated with executing a complete window system on the client.

More recent thin-client systems such as Citrix MetaFrame [10], Microsoft Remote Desktop [11] which comes standard with Windows, Sun Ray [36], and VNC [32], run the graphical user interface of applications at the server, avoiding the need to maintain and run complex window server software at the client. The client functions simply as an input-output device. It maintains a local copy of the framebuffer state used to refresh its display and forwards all user input directly to the server for processing. When applications generate display commands, the server processes those commands and sends screen updates over the network to the client to update the client's local framebuffer. The server maintains the true application and display state, while the client only contains transient soft state.

This approach provides several important benefits. First, synchronization overhead across the network between the user interface and applications can be eliminated since both components run on the server. Second, no window server software needs to run on the client, allowing for less complex client implementation. Third, client processing requirements can scale with display size instead of graphical user interface complexity, enabling clients to be designed as fixed-function devices for a given display resolution. Fourth, since all persistent state resides on the server, mobile users can easily obtain the same persistent and consistent computing environment by connecting to the server from any client.

Achieving these benefits with good system performance remains a key challenge. Citrix MetaFrame, Tarantella [34], and Microsoft Remote Desktop translate application display commands into a rich set of low-level graphics commands that encode display updates sent over the network. These commands are similar to many of the commands used in X. However, performance studies [22, 45] of these systems indicate that using a richer set of display primitives does not necessarily provide substantial gains in bandwidth efficiency, particularly in the presence of multimedia content. Furthermore, the added overhead of supporting a complex set of display primitives results in slower responsiveness and degraded performance in WAN environments. MetaFrame recently added video support by taking advantage of the Windows media architecture to capture the encoded media stream from the application, transmit it over the network, and decode and playback on the client. However, video support is limited only to certain video formats and applications that use the necessary Windows media framework. While this approach reduces network utilization, it increases client complexity, as media decoding and support for different formats relies heavily on local software components, which need to be bundled with and maintained on the client. A specialized version of Remote Desktop in Windows Media Center Edition takes a similar approach, but requires a specialized Windows server and hardware clients; video support is not available in standard Remote Desktop.

Sun Ray uses simpler 2D drawing primitives for sending updates over the network. The original command set developed was simple and easy to implement, and was thoroughly evaluated [36], which motivated the use of a similar command set for THINC. Sun Ray has since evolved and is now in its third major product version. However, it lacks efficient and transparent mechanisms to translate application display commands into its command set. For example, some application commands need to be reduced to pixel data then sampled to determine which drawing primitives to use. This can be difficult to do effectively, and processing overhead can adversely affect overall performance. Applications which generate display commands that Sun Ray cannot efficiently translate need to be explicitly modified to deliver adequate performance. In particular, Sun Ray lacks transparent support for video playback. Sun Ray intercepts application commands using a customized X server, which causes difficulty in keeping up with continuing advances in more widely supported window server implementations, such as XFree86 and X.org.

VNC [41] and GoToMyPC [16] reduce everything to raw pixel values for representing display updates, then read the resulting framebuffer pixel data and encode or compress it, a process sometimes called screen scraping. Other similar systems include Laplink [24] and PC Anywhere [37], which have been previously shown to perform poorly [25]. Screen scraping is a simple process, and decouples the processing of application display commands from the generation of display updates sent to the client. Servers do the full translation from application display commands to actual pixel data, while clients can be very simple and stateless, allowing for maximum portability of the system across client platforms. However, display commands consisting of raw pixels alone are typically too bandwidth-intensive. For example, using raw pixels to encode display updates for a video player displaying at 30 frames per second (fps) full-screen video clip on a typical 1024x768 24-bit resolution screen would require over 0.5 Gbps of network bandwidth. As a result, the raw pixel data must be compressed. Many compression techniques have been developed for this purpose, including FABD [15], PWC [3], and TCC [7, 8, 6]. Generating display updates in this manner is computationally intensive since the original application display semantics are lost and cannot be used in the process. However, these inefficiencies may be less important in the context of providing a user with remote access to an otherwise idle PC [16].

The aforementioned systems focus on providing thin-client computing for general-purpose applications. Specialized architectures that provide remote access to specific applications have also been proposed. The topic of remote multimedia access has been extensively explored; the Infopad project [39] created a device optimized for wireless access to multimedia content. Commercial systems such as SGI's VizServer [40] provide remote access to 3D content. Similarly, WireGL and Chromium [18] enable cluster rendering systems that distribute the 3D processing load, but require high bandwidth environments to operate efficiently.

3. ARCHITECTURE AND PROTOCOL

The architecture of THINC virtualizes the display at the video device abstraction layer, which sits below the window server and above the framebuffer. This is a well-defined, low-level, device-dependent layer that exposes the video hardware to the display system. The layer is typically used by implementing a device-specific display driver that enables the use of a particular display hardware. THINC instead introduces a simple virtual display driver that intercepts drawing commands at the device layer, packetizes them, and sends them over the network to a client device to display.

THINC's virtual display approach brings with it some key implementation and architectural advantages. Because the display device layer sits below the window server proper, THINC avoids reimplementing display system functionality already available, resulting in a simpler system that can leverage existing investments in window server technology. In addition, using a standard interface enables THINC to work seamlessly with existing unmodified applications, window systems, and operating systems. In particular, THINC can operate within unmodified window servers, avoiding the need to maintain and update its own window server code base. THINC can also support new video hardware features with at most the same amount of work necessary to support them in traditional hardware-specific display drivers. Finally, as the video device driver layer still provides semantic information regarding display commands, THINC utilizes those semantics to encode application commands and transmit them from the server to the client in a manner that is both computationally and bandwidth efficient.

Command	Description
RAW	Display raw pixel data at a given location
COPY	Copy frame buffer area to specified coordinates
SFILL	Fill an area with a given pixel color value
PFILL	Tile an area with a given pixel pattern
BITMAP	Fill a region using a bitmap image

Table 1: THINC Protocol Display Commands

With this virtual display approach, THINC uses a small set of low-level display commands for encoding display updates, inspired by the core commands originally used in Sun Ray [36]. The display commands mirror a subset of the video display driver interface. The five commands used in THINC's display protocol are listed in Table 1. These commands are ubiquitously supported, simple to implement, and easily portable to a range of environments. They mimic operations commonly found in client display hardware and represent a subset of operations accelerated by most graphics subsystems. Graphics acceleration interfaces such as XAA and KAA for X and Microsoft Windows' GDI Video Driver interface use a set of operations which can be synthesized using THINC's commands. In this manner, clients need only translate protocol commands into hardware calls, and servers avoid the need to do full translation to actual pixel data, reducing display processing latency.

THINC display commands are as follows. RAW is used to transmit unencoded pixel data to be displayed verbatim on a region of the screen. This command is invoked as a last resort if the server is unable to employ any other command, and it is the only command that may be compressed to mitigate its impact on the network. COPY instructs the client to copy a region of the screen from its local framebuffer to another location. This command improves the user experience by accelerating scrolling and opaque window movement without having to resend screen data from the server. SFILL, PFILL, and BITMAP are commands that paint a fixed-size region on the screen. They are useful for accelerating the display of solid window backgrounds, desktop patterns, backgrounds of web pages, text drawing, and certain operations in graphics manipulation programs. SFILL fills a sizable region on the screen with a single color. PFILL replicates a tile over a screen region. BITMAP performs a fill using a bitmap of ones and zeros as a stipple to apply a foreground and background color.

For high fidelity display, all THINC commands are designed to support full 24-bit color as well as an alpha channel, a feature not supported by thin-client systems that execute the graphical user interface of applications on the server. The alpha channel enables THINC to support graphics compositing operations [31] and work with more advanced window system features that depend on these operations, such as anti-aliased text. Although graphics compositing operations have been used in the 3D graphics world for some time, only recently have they been used in the context of 2D desktop graphics. As a result, there is currently a dearth of support for hardware acceleration of these operations, particularly with low-end 2D only cards commonly used in more modest machines.

THINC provides support for graphics composition by taking advantage of available client hardware acceleration support only when it is present. In the absence of such support, THINC's virtual device driver approach allows it to transparently fall back to the software implementation provided by the window system precisely for video cards lacking hardware support. By doing so, THINC guarantees the simplicity of the client while utilizing the faster server CPU to perform the software rendering. In contrast, thin-client systems which push functionality to the client may need to perform the software rendering using the limited resources of the client computer.

4. TRANSLATION LAYER

The key aspect of THINC's design is not the choice of display primitives that it uses, but rather how it utilizes the virtual display approach to effectively and transparently intercept application display commands and translate them efficiently into THINC commands. There are three important principles in how THINC translation is performed.

First, as the window server processes application requests, THINC intercepts display commands as they occur, and translates the result into its own commands. By translating at this point, THINC can use the semantic information available about the command (and which is lost once processing is finished), to identify which commands should be used. In particular, THINC can know precisely what display primitives are used, instead of attempting to infer those primitives after the fact. Translation in many cases becomes a simple one-to-one mapping from display command to the respective THINC command. For example, a fill operation to color a region of the screen a given color is easily mapped to a SFILL command.

Second, THINC decouples the processing of application display commands and their network transmission. This allows THINC to aggregate small display updates into larger ones before they are sent to the client, and is helpful in many situations. For example, sending a display update for rendering a single character can result in high overhead when there are many small display updates being generated. Similarly, some application display commands can result in many small display primitives being generated at the display device layer. Rasterizing a large image is often done by rendering individual scan lines. The cost of processing and sending each scan line can degrade system performance when an application does extensive image manipulation.

Third, THINC preserves command semantics throughout its processing of application display commands and manipulation of the resulting THINC commands. Since THINC commands are not immediately dispatched as they are generated by the server, it is important to ensure that they are correctly queued and their semantic information preserved throughout the command's lifetime. For example, it is not uncommon for regions of display data to be copied and manipulated. If copying from one display region to another is done by simply copying the raw pixel values, the original command semantics will be lost in the copied region. If THINC commands were reduced to raw pixels at any time, semantic information regarding those commands is lost and it becomes difficult to revert to the original commands in order to efficiently transmit them over the network.

THINC's virtual video device translation layer builds on these three design principles by utilizing two basic objects: the *protocol command object*, and the *command queue object*. Protocol command objects, or just *command objects*, are implemented in an object-oriented fashion. They are based on a generic interface that allows the THINC server to operate on the commands, without having to know each command's specific details. On top of this generic interface, each protocol command provides its own concrete implementation.

As previously mentioned, translated commands are not instantly dispatched to the client. Instead, depending on where drawing occurs and current conditions in the system, commands normally need to be stored and groups of commands may need to be manipulated as a single entity. To handle command processing, THINC introduces the notion of a *command queue*. A command queue is a queue where commands drawing to a particular region are ordered according to their arrival time. The command queue keeps track of commands affecting its draw region, and guarantees that only those commands relevant to the current contents of the region are in the queue. As application drawing occurs, the contents of the region may be overwritten. In the same manner, as commands are generated in response to these new draw operations, they may overwrite existing commands either partially or fully. As commands are overwritten they may become irrelevant, and thus are evicted from the queue. Command queues provide a powerful mechanism for THINC to manage groups of commands as a single entity. For example, queues can be merged and the resulting queue will maintain the queue properties automatically.

To guarantee correct drawing as commands are overwritten, the queue distinguishes among three types of commands based on how they overwrite and are overwritten by other commands: complete, partial, and transparent. Partial commands are opaque commands which can be partially or completely overwritten by other commands. Complete commands are opaque commands that can only be completely overwritten. Transparent commands are commands that depend on commands previously executed and do not overwrite commands already in the queue. The command queue guarantees that the overlap properties of each command type are preserved at all times.

4.1 Offscreen Drawing

Today's graphic applications use a drawing model where the user interface is prepared using offscreen video memory; that is, the interface is computed offscreen and copied onscreen only when it is ready to present to the user. This idea is similar to the double- and triple-buffering methods used in video and 3D-intensive applications. Though this practice provides the user with a more pleasant experience on a regular local desktop client, it can pose a serious performance problem for thin-client systems. Thin-client systems typically ignore all offscreen commands since they do not directly result in any visible change to the framebuffer. Only when offscreen data are copied onscreen does the thin-client server send a corresponding display update to the client. However, all semantic information regarding the offscreen data has been lost at this point and the server must resort to using raw pixel drawing commands for the onscreen display update. This can be very bandwidth-intensive if there are many offscreen operations that result in large onscreen updates. Even if the updates can be successfully compressed, this process can be computationally expensive and would impose additional load on the server.

To deliver effective performance for applications that use offscreen drawing operations, THINC provides a translation optimization that tracks drawing commands as they occur in offscreen memory. The server then sends only those commands that affect the display when offscreen data are copied onscreen. THINC implements this by keeping a command queue for each offscreen region where drawing occurs. When a draw command is received by THINC with an offscreen destination, a THINC protocol command object is generated and added to the command queue associated with the destination offscreen region. The queue guarantees that only relevant commands are stored for each offscreen region, while allowing new commands to be merged with existing commands of the same kind that draw next to each other.

THINC's offscreen awareness mechanism also accounts for applications that create a hierarchy of offscreen regions to help them manage the drawing of their graphical interfaces. Smaller offscreen regions are used to draw simple elements, which are then combined with larger offscreen regions to form more complex elements. This is accomplished by copying the contents of one offscreen region to another. To preserve display content semantics across these copy operations, THINC mimics the process by copying the group of commands that draw on the source region to the destination region's queue and modifying them to reflect their new location. Note that the commands cannot simply be moved from one queue to the other since an offscreen region may be used multiple times as source for a copy.

When offscreen data are copied onscreen, THINC executes the queue of display commands associated with the respective offscreen region. Because the display primitives in the queue are already encoded as THINC commands, THINC's execution stage normally entails little more than extracting the relevant data from the command's structure and passing it to the functions in charge of formatting and outputting THINC protocol commands to be sent to the client. The simplicity of this stage is crucial to the performance of the offscreen mechanism since it should behave equivalently to a local desktop client that transfers pixel data from offscreen to onscreen memory.

In monitoring offscreen operations, THINC incurs some tracking and translation overhead compared to systems that completely ignore offscreen operations. However, the dominant cost of offscreen operations is the actual drawing that occurs, which is the same regardless of whether the operations are tracked or ignored. As a result, THINC's offscreen awareness imposes negligible overhead and yields substantial improvements in overall system performance, as demonstrated in Section 8.

4.2 Audio/Video Support

From video conferencing and presentations to movie and music entertainment, multimedia applications play an everyday role in desktop computing. However, existing thin-client platforms have little or no support for multimedia applications, and in particular for video delivery to the client. Video delivery imposes rather high requirements on the underlying remote display architecture. If the video is completely decoded by applications on the server, there is little the thin-client server can do to provide a scalable solution. Real-time re-encoding of the video data is computationally expensive, even with modern CPUs. At the same time, delivering 24fps of raw `RGB` data can rapidly overwhelm the capacity of a typical network. Further hampering the ability of thin-client systems to support video playback are the lack of well-defined application interfaces for video decoding. Most video players use ad-hoc methods for video decoding, and providing support in this environment would require prohibitive per-application modifications.

While full video decoding in desktop computers is still confined to the realm of software applications, video hardware manufacturers have been slowly adding hardware acceleration capabilities to video cards for specific stages of the decoding process. For example, the ability to do hardware color space conversion and scaling (the last stage of the decoding process) is present in almost all of today's commodity video cards. To allow applications to take advantage of these advancements, interfaces have been created in display systems that allow video device drivers to expose their hardware capabilities back to the applications. With its approach, THINC provides a virtual "bridge" between the remote client hardware and the local applications, and allows applications to transparently use the hardware capabilities of the client to perform video playback across the network.

THINC supports the transmission of video data using widely supported YUV pixel formats. A wide range of YUV pixel formats exist that provide efficient encoding of video content. For example, the preferred pixel format in the MPEG decoding process is YV12, which allows normal true color pixels to be represented with only 12 bits. YUV formats are able to efficiently compress RGB data without loss of quality by taking advantage of the human eye's ability to better distinguish differences in brightness than in color. When using YUV, the client can simply transfer the data to its hardware, which automatically does color space conversion and scaling. Hardware scaling decouples the network transfer requirements of the video from the size at which it is viewed. In other words, playing back a video at full screen resolution does not incur any additional overhead over playing it at its original size, because the client hardware transparently transforms the stream to the desired view size.

THINC's video architecture is built around the notion of video stream objects. Each stream object represents a video being displayed. All streams share a common set of characteristics that allow THINC to manipulate them such as their format, position on the screen, and the geometry of the video. In addition, each stream encapsulates information and state for its respective format. The THINC server uses its translation architecture to seamlessly translate from application requests to video commands which are forwarded to the client. Additional protocol messages are used to manipulate video streams, and they allow operations such as initialization and tearing down of a video stream, and manipulation of the stream's position and size.

Audio streams are not as resource intensive as video streams and THINC supports audio by simply applying its virtual display driver approach to the audio device to create a virtual audio driver that takes audio input, packetizes it, and sends it over the network to a client device to display. By operating at the device layer, THINC provides transparent support for audio applications that can use many different audio libraries. THINC timestamps both audio and video data at the server to ensure they are delivered to the client with the same synchronization characteristics present at the server. Due to space constraints and our primary focus here on remote graphical display, further details regarding THINC audio/video synchronization support are beyond the scope of this paper.

5. COMMAND DELIVERY

THINC schedules commands to be sent from server to client with interactive responsiveness and latency tolerance as a top priority. THINC maintains a per-client command buffer based on the command queue structure described in Section 4 to keep track of commands that need to be sent to the client. While the client buffer maintains command ordering based on arrival time, THINC does not necessarily follow this ordering when delivering commands over the network. Instead, alongside the client buffer THINC provides a multi-queue *Shortest-Remaining-Size-First (SRSF)* preemptive scheduler, analogous to Shortest-Remaining-Processing-Time (SRPT). SRPT is known to be optimal for minimizing mean response time, a primary goal in improving the interactivity of a system. The size of a command refers to its size in bytes, not its size in terms of the number of pixels it updates. THINC uses remaining size instead of the command's original size to shorten the delay between delivery of segments of a display update and to minimize artifacts due to partially sent commands. Commands are sorted in multiple queues in increasing order with respect to the amount of data needed to deliver them to the client. Each queue represents a size range, and commands within the queue are ordered by arrival time. The current implementation uses 10 queues with powers of 2 representing queue size boundaries. When a command is added to the client's command buffer, the scheduler chooses the appropriate queue to store it. The commands are then flushed in increasing queue order.

Reordering of commands is possible with guaranteed correct final output as long as any dependencies between a command and commands issued before it are handled correctly. To explain how THINC's scheduler guarantees correct drawing, we distinguish between partial, complete, and transparent commands. Opaque commands completely overwrite their destination region. Therefore, dependency problems can arise after reordering only if an earlier-queued command can draw over the output of a later-queued command. However, this situation cannot occur for partial commands because the command queue guarantees that no overlap exists among these types of commands, as discussed in Section 4. Furthermore, since complete commands are typical of various types of fills such as solid fills, their size is constantly small and they are guaranteed to end up in the first scheduler queue. Since each queue is ordered by arrival time, it is not possible for these commands to overwrite later similar commands.

On the other hand, transparent commands need to be handled more carefully because they explicitly depend on the output of commands drawn before them. To guarantee efficient scheduling, THINC schedules a transparent command T using a two step process. First, dependencies are found by computing the overlap between the output region of T and the output region of existing buffered commands. T will depend on all those commands with which it overlaps. Second, from the set of dependencies, the largest command L is chosen, and the new command is added to the back of the queue where L currently resides. As queues are flushed in increasing order, THINC's approach guarantees that all commands upon which T depends will have been completely drawn before T itself is sent to the client. Although more sophisticated approaches could be used to allow the reordering of transparent commands, we found that their additional complexity outweighed any potential benefits to the performance of the system.

In addition to the queues for normal commands, the scheduler has a *real-time* queue for commands with high interactivity needs. Commands in the real-time queue take priority and preempt commands in the normal queues. Real-time commands are small to medium-sized and are issued in direct response to user interaction with the applications. For example, when the user clicks on a button or enters keyboard input, she expects immediate feedback from the system in the form of a pressed button image. Because a video driver does not have a notion of a button or other high-level primitives, THINC defines a small-sized region around the location of the last received input event. By marking updates which overlap these regions as real-time and delivering them sooner as opposed to later, THINC improves the user-perceived responsiveness of the system.

THINC sends commands to the client using a *server-push* architecture, where display updates are *pushed* to the client as soon as they are generated. In contrast to the *client-pull* model used by popular systems such as VNC [41] and GoToMyPC [16], server-push maximizes display response time by obviating the need for a round trip delay on every update. This is particularly important for display-intensive applications such as video playback since updates are generated faster than the rate at which the client can send update requests back to the server. Furthermore, a server-push model minimizes the impact of network latency on the responsiveness of the system because it requires no client-server synchronization, whereas a client-driven system has an update delay of at least half the round-trip time in the network.

Although a push mechanism can outperform client-pull systems, a server blindly pushing data to clients can quickly overwhelm slow or congested networks and slowly responding clients. In this situation, the server may have to block or buffer updates. If updates are not buffered carefully and the state of the display continues to change, outdated content is sent to the client before relevant updates can be delivered.

Blocking can have potentially worse effects. Display systems are commonly built around a monolithic core server which manages display and input events, and where display drivers are integrated. If the video device driver blocks, the core display server also blocks. As a result, the system becomes unresponsive since neither application requests nor user input events can be serviced. In display systems where applications send requests to the window system using IPC mechanisms, blocking may eventually cause applications to also block after the IPC buffers are filled.

The THINC server guarantees correct buffering and low overhead display update management by using its command queue-based client buffer. The client buffer ensures that outdated commands are automatically evicted. THINC periodically attempts to flush the buffer using its SRSF scheduler in a two-stage process. First, each command in the buffer's queue is committed to the network layer by using the command's flush handler. Since the server can detect if it will block when attempting to write to a socket, it can postpone the command until the next flush period. Second, to protect the server from blocking on large updates, a command's flush handler is required to guarantee non-blocking operation during the commit by breaking large commands into smaller updates. When the handler detects that it cannot continue without blocking, it reformats the command to reflect the portion that was committed and informs the server to stop flushing the buffer. Commands are not broken up in advance to minimize overhead and allow the system to adapt to changing conditions.

6. HETEROGENEOUS DISPLAYS

The promise of ubiquitous computing access has been a major driving force in the growing popularity of thin-client systems. To deliver on this promise, THINC decouples the session's original framebuffer size, from the size at which a particular client may view it. In this way, THINC enables access from a variety of devices by supporting variable client display sizes and dynamic resizing. For instance, to view a desktop session through a small-screen mobile device such as a PDA, THINC initially presents a zoomed-out version of the user's desktop, from where the user can zoom in on particular sections of the display. In sharp contrast to similar client-only approaches in existing thin-client systems, THINC's display resizing is fully supported by the server. After a client reports its screen size to the server, subsequent updates are automatically resized by the server to fit in the client's smaller viewport. When the user zooms in on the desktop, the client presents a temporary magnified view of the desktop while it requests updated content from the server. The server updates are necessary when the display size increases, because the client has only a small-size version of the display, with not enough content to provide an accurate view of the desktop.

Server resize support is designed to minimize processing and network overhead while maintaining display quality and client simplicity. For this reason, resizing is supported differently for each protocol command. `RAW` updates can be easily resized because they consist of pure pixel data which can be reliably resampled, and more importantly, the bandwidth savings are significant. Similarly for `PFILL` updates the tile image is resized. On the other hand, `BITMAP` updates cannot be resized without incurring significant loss of display information and generating display artifacts. Traditionally, anti-aliasing techniques are used to minimize the loss of information from the downsize operation. However, anti-aliasing requires the use of intermediate pixel values which bitmap data cannot represent. In this case, `BITMAP` updates are converted to `RAW` and resampled by the server. While this may increase bandwidth usage, requiring the client to do resizing would be prohibitively expensive. Finally, resizing `SFILL` updates represents no savings with respect to bandwidth or computation, and therefore they are sent unmodified.

As shown in Section 8, our approach provides substantial performance benefits by leveraging server resources and reducing bandwidth consumption, vastly outperforming the client-only support present in other systems. Furthermore, since THINC can use the powerful server CPU to do most of the resize work, it can use high quality resampling algorithms to provide superior display content to the user.

7. IMPLEMENTATION

We have implemented a prototype THINC server in Linux as a video device driver that works with all existing open source X server implementations, including XFree86 4.3, 4.4, and X.org 6.8. We have implemented a number of THINC clients, including a simple X application, a Java client (both as a standalone application and a web browser applet), a Windows client, and a Windows PDA client, demonstrating THINC's client portability and simplicity. THINC seamlessly hooks into X's existing driver infrastructure, and no changes are required to applications or the window system. XFree86 and derived implementations are designed around a single-user workstation model where a server has exclusive access to the computer's display hardware, and multiple server instances are not allowed to be active simultaneously. Because the THINC server does not access local hardware, THINC modifies the window server's behavior from within the video driver to allow multiple servers to be active at the same time.

RAW commands are the only commands where we apply additional compression to mitigate its impact on the network. The current prototype uses PNG[30] for this purpose. To support resizing, we use a simplified version of Fant's resampling algorithm [13], which produces high quality, anti-aliased results with very low overhead. To provide video support, THINC leverages the standard XVideo extension by implementing the necessary XVideo device driver hooks. THINC primarily exports the YV12 format to applications, which we chose not only for its intrinsic compression characteristics, but more importantly, for the wide range of applications supporting it, and its use as one of the preferred formats in MPEG codecs. For audio support, we use a virtualized ALSA audio driver implemented as a kernel module to intercept audio data. The THINC audio driver utilizes ALSA's driver interfaces and multiplexes its resources across multiple THINC users. Applications interact with the driver using ALSA's audio library. The driver works in tandem with a per client daemon which is automatically signaled as audio data becomes available.

For improved network security, THINC encrypts all traffic using RC4, a streaming cipher particularly suited for the kind of traffic prevalent in thin-client environments. Although block ciphers can have a significant effect in the performance of the system, we have found the cost of RC4 to be rather minimal, and the benefits far outweigh any minor overhead in overall system performance. Our prototype also implements authentication using the standard UNIX authentication facilities provided by PAM (Pluggable Authentication Modules). Our authentication model requires the user to have a valid account on the server system and to be the owner of the session she is connecting to. To support multiple users collaborating in a screen-sharing session, the authentication model is extended to allow host users to specify a session password that is then used by peers connecting to the shared session.

8. EXPERIMENTAL RESULTS

To demonstrate the effectiveness of THINC, we conducted a direct comparison with a number of state-of-the-art and widely used thin-client platforms. These were Citrix MetaFrameXP, Microsoft Remote Desktop, GoToMyPC, X, NX, Sun's Sun Ray, and VNC. We follow common practice and

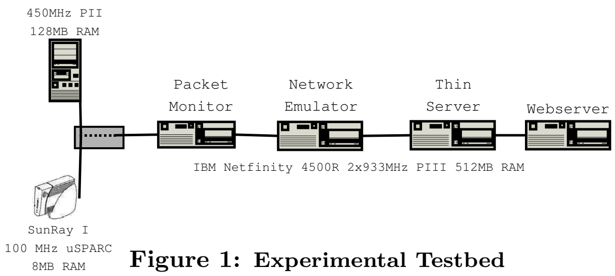

Figure 1: Experimental Testbed

refer to Citrix MetaFrameXP and Microsoft Remote Desktop by their respective remote display protocols, ICA (Independent Computing Architecture) and RDP (Remote Desktop Protocol). Our measurements present the first quantitative performance comparison of these systems. We measured their performance on common web and multimedia audio/video applications in LAN, WAN, and 802.11g wireless network environments. We also used a PC running all applications locally as a baseline for representing today's prevalent desktop computer model. Section 8.1 describes our experimental setup. Section 8.2 describes the application benchmarks used for our studies. Section 8.3 presents our measurement results.

8.1 Experimental Setup

We compared the performance of various thin-client systems using an isolated network testbed, and we measured wide-area THINC performance using PlanetLab [9] nodes and other remote sites located around the world. As shown in Figure 1, our testbed consisted of six computers connected on a switched FastEthernet network: two thin clients, a packet monitor, a network emulator for emulating various network environments, a thin-client server, and a web server used for testing web applications. Except for the thin clients, all computers were IBM Netfinity 4500R servers, with dual 933 MHz Pentium III processors and 512 MB of RAM. The client computers were a 450 MHz Pentium II computer with 128 MB of RAM, and a Sun Ray I with a 100 MHz μSPARC processor and 8 MB of RAM. During each test, only one client/server pair was active at a time. The web server used was Apache 1.3.27, the network emulator was NISTNet 2.0.12, and the packet monitor was Ethereal 0.10.9.

To provide a fair comparison, we standardized on common hardware and operating systems whenever possible. All of the thin-client systems used the PC as the client, except Sun Ray, for we which we used a Sun Ray I hardware thin client. All of the systems used the Netfinity server as the thin-client server. For the three systems designed for Windows (ICA, RDP, and GoToMyPC), we ran Windows 2003 Server on the server and Windows XP Professional on the client. For the systems designed for X-based environments, we ran the Debian Linux Unstable distribution with the Linux 2.6.10 kernel on both server and client, except for Sun Ray, where we encountered a problem with audio playback that required us to revert to a 2.4.27 kernel. We used the latest thin-client system versions available on each platform, namely Citrix MetaFrame XP Server for Windows Feature Release 3, Microsoft Remote Desktop built into Windows XP and Windows 2003 using RDP 5.2, GoToMyPC 4.1, VNC 4.0, NX 1.4, Sun Ray 3.0, and XFree86 4.3.0 on Debian. Since X does not natively support audio, we used it with aRts 1.3.2, a sound server commonly used to provide remote audio.

To minimize application environment differences, we used common thin-client configuration options whenever possible. Client display was set to 24-bit color except for GoToMyPC which is limited to 8-bit color. To mimic realistic usage of the systems over public and insecure networks, we enabled RC4 encryption with 128-bit keys on all platforms which supported it. For those which did not, namely X and VNC, we used ssh to provide a secure tunnel through which all traffic was forwarded. The ssh tunnel was configured to use RC4. Following common practice, we configured X's ssh tunnel to also compress all traffic [20]. Any remaining thin-client configuration settings were set to their defaults for a particular network environment. ICA, RDP, and NX were set to LAN settings when used in the LAN and WAN settings when used in the WAN. Some thin-client systems used a persistent disk cache in addition to a per-session cache. To minimize variability, we left the persistent cache turned on but cleared it before every test was run.

We considered three different client display resolution and network configurations: *LAN Desktop*, *WAN Desktop*, and *802.11g PDA*. LAN Desktop represents a client with a 1024 x 768 display resolution and a 100 Mbps LAN network. WAN Desktop represents a client with a 1024 x 768 display resolution and a 100 Mbps WAN network with a 66 ms RTT, which emulates Internet2 connectivity to a US cross-country remote server [22]. We conducted our WAN experiments using the kind of high-bandwidth network environment that is becoming increasingly available in public settings [1]. 802.11g PDA represents a client with a 320 x 240 display resolution for a PDA-like viewing experience and a 24 Mbps network, which emulates an idealized 802.11g wireless network [2]. We chose 802.11g over 802.11b to reflect 802.11g's emergence as the next standard for wireless networks. The added bandwidth capacity provides a more conservative comparison for bandwidth intensive applications, such as video playback. Since the purpose of the test was to measure performance on small-screen displays, 802.11g PDA did not emulate the additional latency and packet loss characteristics typical of wireless networks, and results are only reported for those systems with support for small client screens.

GoToMyPC is only offered as an Internet service that connects the client and server using an intermediate hosted server through which all traffic is routed. As a result, we were unable to fully control the network configuration used. Our measurements show a 70 ms RTT between the intermediate GoToMyPC server used and our testbed, resulting in similar network latencies as our emulated WAN environment. We measured GoToMyPC performance without network emulation and referred to it as WAN Desktop. We repeated the same measurements with small-screen display and network emulation limiting bandwidth to 24 Mbps and referred to it as 802.11g PDA. The display resolution for the GoToMyPC 802.11g PDA was set to 640x480 because it does not support smaller displays.

We also measured thin-client performance in WAN environments by running the server in our local testbed, but running the client on PlanetLab [9] nodes and other remote sites located around the world. Table 2 lists the sites used. Since the PlanetLab machines run User-Mode Linux, we were unable to run X-based thin-client servers on these machines, and the use of Linux precluded any testing of Windows-based thin-client systems. We were also prohibited from making significant modifications to the Linux in-

Name	PlanetLab	Location	Distance
NY	yes	New York, NY, USA	5 miles
PA	yes	Philadelphia, PA, USA	78 miles
MA	yes	Cambridge, MA, USA	188 miles
MN	yes	St. Paul, MN, USA	1015 miles
NM	no	Albuquerque, NM, USA	1816 miles
CA	no	Stanford, CA, USA	2571 miles
CAN	yes	Waterloo, Canada	388 miles
IE	no	Maynooth, Ireland	3185 miles
PR	no	San Juan, Puerto Rico	1603 miles
FI	no	Helsinki, Finland	4123 miles
KR	yes	Seoul, Korea	6885 miles

Table 2: Remote Sites for WAN Experiments

stallations at the non-PlanetLab sites. To measure THINC performance, we developed an instrumented headless version of the THINC client that could process all display and audio data but did not output the result to any display or sound hardware. We deployed this client on the remote sites and ran the same experiments as the WAN configuration.

Since most of the thin-client systems tested used TCP as the underlying transport protocol, we were careful to consider the impact of TCP window sizing on performance in WAN environments. Since TCP windows should be adjusted to at least the bandwidth delay product size to maximize bandwidth utilization, we used a 1 MB TCP window size in our testbed WAN environment and with remote sites whenever possible to take full advantage of the network bandwidth capacity available. However, PlanetLab nodes were limited to a window size of 256 KB due to their preconfigured system limits.

8.2 Application Benchmarks

We evaluated thin-client systems on web browsing and audio/video playback, two dominant applications used on the desktop. Web browsing performance was measured by running a benchmark based on the Web Page Load test i-Bench benchmark suite [19]. The benchmark consists of a sequence of 54 web pages containing a mix of text and graphics. Once a page has been downloaded, a link is available on the page that can be clicked to download the next page in the sequence. This mouse clicking operation was done using a mechanical device we built to press the mouse button in a precisely timed fashion. The mechanical device enabled us to better simulate a user browsing experience and ensure that the test could be easily repeated on different thin-client systems without introducing human timing errors. For remote site experiments with THINC, the headless client read a script of timed mouse coordinates and clicks to run the web benchmark. We used the Mozilla 1.6 browser set to full-screen resolution for all experiments to minimize application differences across platforms.

Audio/video playback performance was measured by playing a 34.75 s MPEG-1 audio/video clip, with the video being of original size 352x240 pixels and displayed at full-screen resolution. We measured combined audio/video playback performance except for GoToMyPC and VNC for which we only report video playback results since they do not support audio. The audio/video (A/V) player used was MPlayer 1.0pre6 for the Unix-based platforms, and Windows Media Player 9 for the Windows-based platforms.

Since many of the thin-client systems are closed and proprietary, we measured their performance in a noninvasive manner by capturing network traffic with a packet monitor and using a variant of slow-motion benchmarking [26, 23]. Our primary measure of web browsing performance is page download latency. Using slow-motion benchmarking, we captured network traffic and measured page latency as the time from when the first packet of mouse input is sent to the server until the last packet of web page data is sent to the client. We ensured that a long enough delay was present between successive page downloads so that separate pages could be disambiguated in the network packet capture. However, this measure does not fully account for client processing time. To account for client processing time, we also instrumented the client window system to measure the time between the initial mouse input and the processing of the last graphical update for each page. We could only do this for X, VNC, NX, and THINC as we did not have access to client window system code for the other systems. Thus, our results provide a conservative comparison with Windows-based thin clients and Sun Ray for which we cannot fully account for client processing time.

A/V performance is measured using slow-motion benchmarking based on A/V quality [26], which accounts for both playback delays and frame drops that degrade playback quality. For example, 100% A/V quality means that all video frames and audio samples were played at real-time speed. On the other hand, 50% A/V quality could mean that half the A/V data was dropped, or that the clip took twice as long to play even though all of the A/V data was played. We used a combined measure of A/V quality since many of the closed platforms tested transmit both audio and video over the same connection, making it difficult to disambiguate packet captures to determine which data corresponds to each media stream. Since all data is treated equally and video data is generally much larger than audio data, our quality measure effectively weighs the impact of video quality more heavily than audio quality. However, weighing video more than audio quality is useful in this context given that the primary focus in this paper is on remote display.

8.3 Measurements

Figures 2 to 4 show web browsing performance results. Figure 2 shows the average latency per web page for each platform. For platforms in which we instrumented the window system to measure client processing time, the solid color bars show latency measured using network traffic, while the cross-hatched bars show a more complete measure by including client processing time. For example, Figure 2 shows that client processing time is a dominant factor for local PC web browsing performance since the web browser needs to process the HTML on the client. As shown in Figure 2, most of the systems did well in both LAN and WAN environments, having latencies below the one second threshold for users to have an uninterrupted browsing experience [27].

Figure 2 shows that THINC provides the fastest web page download latencies of all systems. THINC is up to 1.7 times faster in the LAN and up to 4.8 times faster in the WAN versus other systems. THINC outperforms the local PC by more than 60% because it leverages the faster server to process web pages more quickly than the web browser running on the slower client. Figures 2 shows that THINC does not suffer much performance degradation going from LAN to WAN, where it still outperforms all other platforms. In contrast, a higher-level approach such as X experiences the largest slowdown, performing about two and a half times

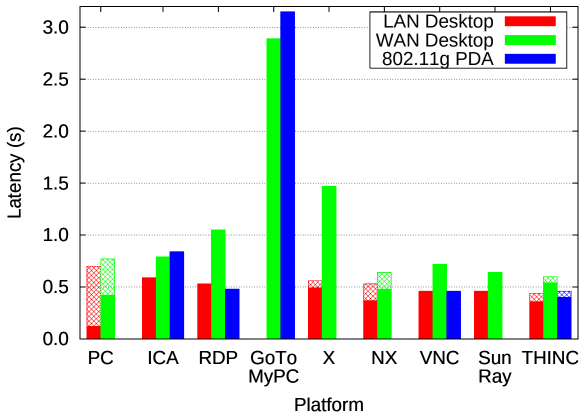

Figure 2: Web Benchmark: Average Page Latency

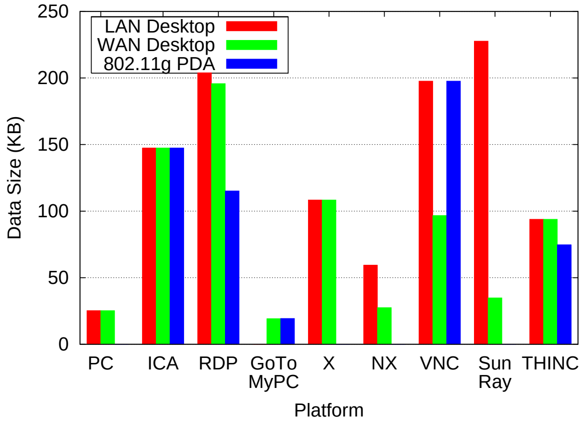

Figure 3: Web Benchmark: Average Page Data Transferred

worse due to the tight coupling required between applications on the server and the user interface on the client. While still slower than THINC, NX is much faster than X, indicating that some of these problems can be mitigated through careful X proxy design. Figure 2 shows that even though we excluded client processing time for ICA, RDP, GoToMyPC, and Sun Ray, THINC including client processing time is faster than all of them. GoToMyPC takes almost three seconds on average to download web pages. Figure 3 shows that GoToMyPC's slow performance is not due to its data requirements as it sends the least amount of data. The measurements suggest that GoToMyPC employs complex compression algorithms to reduce its data requirements at the expense of high server utilization and longer latencies. GoToMyPC's use of an intermediate server most likely also affects its performance, but enables it to provide ubiquitous service even in the presence of NATs and firewalls.

Figure 4 shows results using remote PlanetLab nodes and other sites as THINC clients, demonstrating that THINC maintains its fast performance under real network conditions even when client and server are located thousands of miles apart. THINC provides sub-second web page download times for all sites except for when the client is running in Korea, which is almost seven thousand miles away from the server in New York. Figure 4 shows that THINC's web page download latencies increased by less than 2.5 times in going from running the client in the local LAN testbed to running the client in Finland while the corresponding network RTTs increased by more than two orders of magnitude. These measurements show for the first time a thin client that can provide excellent web browsing performance even when clients are located on another continent.

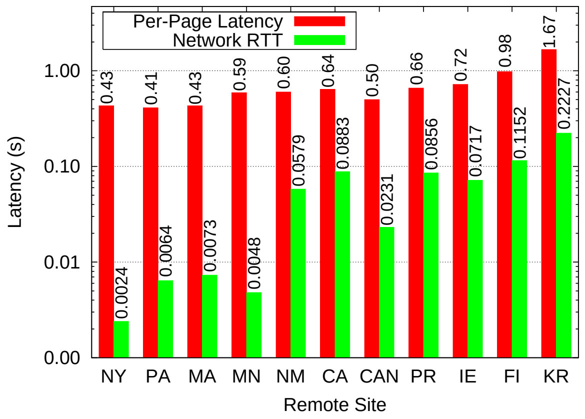

Figure 4: Web Benchmark: THINC Average Page Latency Using Remote Sites

Figure 3 shows the average data transferred for each web page and demonstrates that THINC achieves fast performance with only modest data transfer requirements. The local PC is the most bandwidth efficient platform for web browsing, but THINC is better than all other thin clients for LAN Desktop except NX. Surprisingly, GoToMyPC had the smallest data transfer requirements of the thin clients for WAN Desktop despite its low-level pixel-based display approach. While this is an unfair comparison since GoToMyPC only supports 8-bit color, it demonstrates that compression algorithms can be effective at reducing raw pixel data at great computational expense. A number of systems show significant reductions in data size when going from the LAN to the WAN environment. NX has specific user settings for this type of environment which causes it to use more aggressive data compression techniques. Sun Ray and VNC use adaptive compression schemes which change its encoding settings according to the characteristics of the link. This adaptive mechanism also accounts for the significant decrease in Sun Ray's data requirements, as more complex and cpu-intensive compression schemes are used.

Comparing Sun Ray and THINC provides a measure of the effectiveness of THINC's translation architecture, as both systems use a similar low-level protocol. Although we could not instrument the Sun Ray hardware client to measure client processing time, we can use the network measurements as a basis of comparison between the systems. Both systems perform well, but THINC outperforms Sun Ray by 22% and 16% in the LAN and WAN environments, respectively. Sun Ray incurs higher overhead because it lacks THINC's translation mechanisms, especially offscreen drawing which is used heavily by Mozilla. As a result, it lacks semantic information originally present in the application display commands and must attempt to translate back into its protocol from raw pixel data. Similarly, comparing VNC and THINC provides a measure of the efficiency of THINC's encoding approach versus VNC's pixel data compression approach. THINC is faster than VNC for the LAN Desktop while sending almost half the data. This suggests that THINC's small set of command primitives and translation layer provides significant performance efficiency compared to relying on a single compression strategy for all types of display data. These results show the importance of an effective translation layer, not just a good command set.

Comparing these systems as well as NX on a page-by-page basis provides further insight based on how different web page content contributes to performance differences. Except for THINC, Sun Ray, VNC, and NX were the fastest systems. Compared with these systems, THINC was faster on all web pages except those that primarily consisted of a single large image. For those pages, THINC resorted primarily to its RAW encoding strategy combined with simple, off-the-shelf compression, given the lack of additional semantic information. In the LAN, Sun Ray's lack of compression and VNC's simple compression strategy both sent more data but provided faster processing of those pages compared to THINC. In the WAN, the more advanced compression used in NX and Sun Ray reduced the data size significantly, allowing them to transfer the pages much faster. This breakdown indicates that THINC's performance on pages with mixed web content (text, logos, tables, etc.) was even better than what is shown in Figure 2 when compared with these other systems. These results suggest two important observations. First, not only is THINC's low-level translation approach faster than a pixel-level approach as embodied by VNC, but it is also faster than a high-level encoding approach as embodied by NX, even on non-image content. Second, although optimized compression techniques were not a central focus in the current THINC prototype, the results suggest that better compression algorithms such as used in NX and adapting compression based on network performance as used by VNC and Sun Ray can provide useful performance benefits when displaying large image content.

Figures 2 and 3 also show results for 802.11g PDA for ICA, RDP, GoToMyPC, VNC, and THINC, the only systems that support a client display geometry different than the server's. THINC has the best performance overall, providing the fastest web page download times and the smallest data transfer requirements among the 24-bit color systems. GoToMyPC transfers less data, but only supports 8-bit color and is roughly eight times slower than THINC. Compared to the other 24-bit color systems, THINC is up to 2.75 times faster while transferring as little as one third of the data.

Support for small screen devices can be divided in two models: systems which clip the client's display and systems which resize the contents of the display. RDP and VNC fall within the first category, requiring users to scroll around the display. Citrix, GoToMyPC, and THINC fall within the second category though THINC differs in that the server does all of the resizing work. As shown by our results, this approach achieves the best performance across all the architectures. Since all of the updates are resized before being sent, THINC's bandwidth utilization is reduced by more than a factor of two while only marginally affecting the latency of the system. In contrast, ICA's and GoToMyPC's client-only resize approach noticeably increase latency with no improvement in bandwidth consumption. Furthermore, our latency measure underestimates their latency since it does not fully account for client processing time. In the CPU and bandwidth-limited environment of mobile devices, this approach adversely affects the overall user experience.

It is worth mentioning the large difference in quality of THINC's resized display compared with ICA's. THINC's resize algorithm appropriately interpolates pixel data and uses anti-aliasing to provide high quality results such that the web page is still readable even when displaying the 320x240 client window on a computer with a resolution of 1280x1024. On the other hand, ICA's resized display version is barely readable and appears to be mostly for locating portions of the screen in which to zoom. Clearly, ICA's choice of resizing algorithm is restricted by the client's computational power, a limitation not present in THINC's server-side approach.

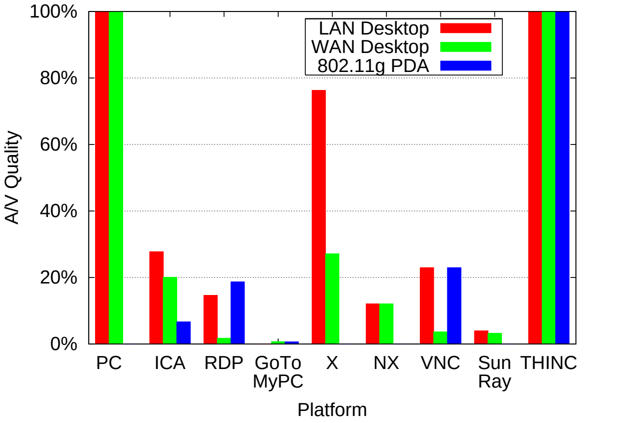

Figure 5: A/V Benchmark: A/V Quality (GoToMyPC and VNC are video only)

Figures 5 to 7 show A/V playback performance results. Results for VNC and GoToMyPC are for video playback without audio since those platforms do not provide audio support. We also ran the same benchmark on all platforms with video only and no audio. The results were similar to the A/V playback results. For platforms that supported audio, we also ran the same benchmark with audio only and no video. Most of the platforms with audio support provided perfect audio playback quality in the absence of video.

Figure 5 shows that THINC is the only thin client that provides 100% A/V quality in all network environments and is the only system that provides 100% A/V quality even in the 802.11g PDA configuration. THINC's A/V quality is up to 8 times better than the other systems for LAN Desktop and up to 140 times better for WAN Desktop. Other than THINC, only the local PC provides 100% A/V quality in any of the configurations tested. From a qualitative standpoint, THINC A/V playback was consistently smooth and synchronized and indistinguishable from A/V playback on the local PC. On the other hand, A/V playback was noticeably choppy and jittery for all other thin clients. In particular, playback on RDP and ICA was marked by lower audio fidelity due to compression and frequent drops.

Figure 5 shows quantitatively that all other thin clients deliver very poor A/V quality. NX has the worst quality for LAN at only 12%, and GoToMyPC has the worst quality for WAN at less than 2%. These systems suffer from their inability to distinguish video from normal display updates, and their attempts to apply ineffective and expensive compression algorithms on the video data. These algorithms are unable to keep up with the stream of updates generated, resulting in dropped frames or extremely long playback times. VNC has poor video performance for these same reasons, and drops quality by half for the WAN Desktop because of its client-pull model. The VNC client needs to request display updates for the server to send them. This is problematic in higher latency WAN environments in which video

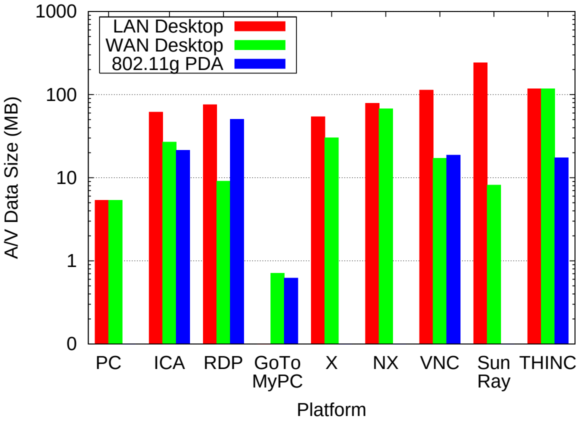

Figure 6: A/V Benchmark: Total Data Transferred (GoToMyPC and VNC are video only)

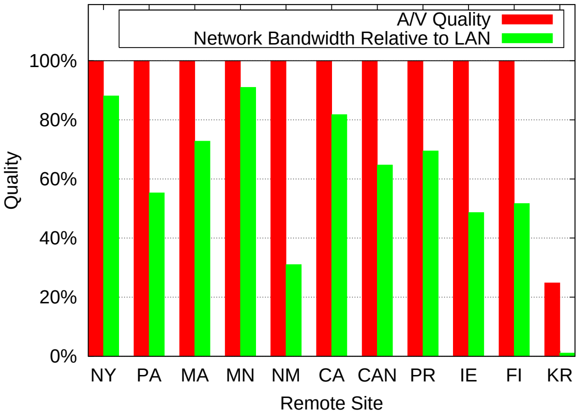

Figure 7: A/V Benchmark: THINC A/V Quality Using Remote Sites

frames are generated faster than the rate at which the client can send requests to the server. In contrast, THINC's server push model and its native audio/video support provide substantial performance benefits over the other systems.

The effects of ICA's support for native video playback are not reflected in Figures 5 to 7. Its playback mechanism only supports a limited number of formats, and the widely-used MPEG1 format used for the A/V benchmark is not one of them. We conducted additional experiments with the video clip transcoded to DivX, a supported format, and surprisingly found the results to be only slightly better. ICA relies on the Windows Media Player installed on the client to do the video playback, and in turn the player had hardware requirements beyond what the client could support. The client was unable to keep up with the desired playback rate, resulting in poor video quality.

Figure 6 shows the total data transferred during A/V playback for each system. The local PC is the most bandwidth efficient platform for A/V playback, sending less than 6 MB of data, which corresponds to about 1.2 Mbps of bandwidth. THINC's 100% A/V quality requires 117 MB of data for the LAN Desktop and WAN Desktop, which corresponds to bandwidth usage of roughly 24 Mbps. Several other thin clients send less data than THINC, but they do so because they are dropping video data, resulting in degraded A/V quality. For example, GoToMyPC sends the least amount of data but also has the worst A/V quality.

Figure 7 shows results using remote PlanetLab nodes and other sites as THINC clients, demonstrating that THINC maintains its superior A/V playback performance under real network conditions even when client and server are located thousands of miles apart. THINC provides perfect A/V quality for all remote sites except for Korea. Figure 7 also shows the relative bandwidth available from each remote site to the local THINC server compared to the bandwidth available in our local LAN testbed. These measurements were obtained using Iperf. The bandwidth measurements show that THINC does not perform well for Korea due to insufficient bandwidth. The lack of bandwidth in this case was not due to network link itself, but due to the TCP window size configuration of the Korea PlanetLab site, which we were not allowed to change. For other distant non-PlanetLab remote sites such as Puerto Rico, Ireland, and Finland in which a sufficiently-sized TCP window was allowed, Figure 7 shows that THINC provides 100% A/V quality.

Figures 5 and 6 also show 802.11g PDA small-screen results for ICA, RDP, GoToMyPC, and THINC. The results again demonstrate the benefits of THINC's server resize mechanism. THINC still performs at 100% A/V quality, demonstrating the minimum overhead incurred on the server by resampling the video data while significantly reducing bandwidth consumption to 3.5 Mbps, well below any of the other systems. We have conducted additional tests demonstrating THINC's ability to also provide perfect video playback over an 802.11b wireless network, which cannot be done by any of the other thin-client systems. ICA's client-side resize mechanism slows the client and further reduces its low A/V quality from above 20% for LAN Desktop and WAN Desktop to only 6% for 802.11g PDA. RDP and VNC's clipping mechanisms are not particularly useful for video playback since the user only sees the section of the video that intersects with the client's viewport. A user could potentially watch the video at a smaller size and make the video window fit within the client's display. However, adding such awkward constraints to the user interface is detrimental to the overall usability of the system.

9. CONCLUSIONS

THINC is a new virtual display system for high-performance thin-client computing built around a virtual device driver model. THINC introduces novel translation optimizations that take advantage of semantic information to efficient convert high-level application requests to a simple low-level protocol command set. THINC leverages client hardware capabilities to provide native support for audio/video playback, and supports small screen devices with server-side scaling of display updates. THINC's virtual display approach enables it to leverage continuing advances in window server technology and work seamlessly with unmodified applications, window systems, and operating systems.

We have measured THINC's performance on common web and video applications in a number of network environments and compared it to existing widely used thin-client systems. Our experimental results show that THINC can deliver good interactive performance even when using clients located around the world. THINC provides superior web performance over other systems, with up to 4.5 times faster response time in WAN environments. THINC's audio/video support vastly outperforms existing systems. It is the only thin client able to provide transparent, format-independent, full screen audio/video playback in WAN environments. Our results demonstrate how THINC's unique mapping of application level drawing commands to protocol primitives and its command delivery mechanisms significantly improve the overall performance of a thin-client system. Going beyond thin-client computing, THINC provides a fundamental building block for a broad range of remote display applications.

10. ACKNOWLEDGEMENTS

J. Duane Northcutt provided insightful suggestions during the early stages of this work. S. Jae Yang implemented the hardware mouse device used for our web experiments. William Caban, Teemu Koponen, Monica Lam, Barak Pearlmutter, and Constantine Sapuntzakis, kindly provided remote machine resources for our experiments. Shaya Potter and Stefan Savage provided helpful comments on earlier drafts of this paper. This work was supported in part by NSF ITR grants CCR-0219943 and CNS-0426623, an IBM SUR Award, and Sun Microsystems.

11. REFERENCES

[1] 100x100 Project. `http://100x100network.org/`.

[2] 802.11 Wireless LAN Performance. `http://www.atheros.com/pt/atheros_range_whitepaper.pdf`.

[3] P. Ausbeck. A Streaming Piecewise-constant Model. In *Proceedings of the Data Compression Conference (DCC)*, Mar. 1999.

[4] R. Baratto, S. Potter, G. Su, and J. Nieh. MobiDesk: Mobile Virtual Desktop Computing. In *Proceedings of the 10th Annual ACM International Conference on Mobile Computing and Networking (MobiCom)*, Sept. - Oct. 2004.

[5] Charon Systems. `http://www.charon.com`.

[6] B. O. Christiansen and K. E. Schauser. Fast Motion Detection for Thin Client Compression. In *Proceedings of the Data Compression Conference (DCC)*, Apr. 2002.

[7] B. O. Christiansen, K. E. Schauser, and M. Munke. A Novel Codec for Thin Client Computing. In *Proceedings of the Data Compression Conference (DCC)*, Mar. 2000.

[8] B. O. Christiansen, K. E. Schauser, and M. Munke. Streaming Thin Client Compression. In *Proceedings of the Data Compression Conference (DCC)*, Mar. 2001.

[9] B. Chun, D. Culler, T. Roscoe, A. Bavier, L. Peterson, M. Wawrzoniak, and M. Bowman. PlanetLab: An Overlay Testbed for Broad-Coverage Services. *SIGCOMM Comput. Commun. Rev.*, 33(3):3–12, 2003.

[10] Citrix Metaframe. `http://www.citrix.com`.

[11] B. C. Cumberland, G. Carius, and A. Muir. *Microsoft Windows NT Server 4.0, Terminal Server Edition: Technical Reference*. Microsoft Press, Redmond, WA, 1999.

[12] Dan Tynan, InfoWorld. Think thin. `http://www.infoworld.com/article/05/07/14/29FEthin_1.html`.

[13] K. M. Fant. A Nonaliasing, Real-Time Spatial Transform Technique. *IEEE Computer Graphics and Applications*, 6(1):71–80, Jan. 1986.

[14] Fog Creek Copilot. `http://www.copilot.com`.

[15] J. M. Gilbert and R. W. Brodersen. A Lossless 2-D Image Compression Technique for Synthetic Discrete-Tone Images. In *Proceedings of the Data Compression Conference (DCC)*, Mar. - Apr. 1998.

[16] GoToMyPC. `http://www.gotomypc.com/`.

[17] Health Insurance Portability and Accountability Act. `http://www.hhs.gov/ocr/hipaa/`.

[18] G. Humphreys, M. Houston, R. Ng, R. Frank, S. Ahern, P. Kirchner, and J. T. Klosowski. Chromium: A Stream Processing Framework for Interactive Rendering on Clusters. In *Proceedings of the 29th International Conference on Computer Graphics and Interactive Techniques (SIGGRAPH)*, 2002.

[19] i-Bench version 1.5. `http://etestinglabs.com/benchmarks/i-bench/i-bench.asp`.

[20] Jim Gettys. Personal communication, July 2004.

[21] Keith Packard. An LBX Postmortem. `http://keithp.com/~keithp/talks/lbxpost/paper.html`.

[22] A. Lai and J. Nieh. Limits of Wide-Area Thin-Client Computing. In *Proceedings of the International Conference on Measurement & Modeling of Computer Systems (SIGMETRICS)*, June 2002.

[23] A. Lai, J. Nieh, B. Bohra, V. Nandikonda, A. P. Surana, and S. Varshneya. Improving Web Browsing on Wireless PDAs Using Thin-Client Computing. In *Proceedings of the 13th International World Wide Web Conference (WWW)*, May 2004.

[24] LapLink, Bothell, WA. *LapLink 2000 User's Guide*, 1999.

[25] J. Nieh, S. J. Yang, and N. Novik. A Comparison of Thin-Client Computing Architectures. Technical Report CUCS-022-00, Department of Computer Science, Columbia University, Nov. 2000.

[26] J. Nieh, S. J. Yang, and N. Novik. Measuring Thin-Client Performance Using Slow-Motion Benchmarking. *ACM Trans. Computer Systems*, 21(1):87–115, Feb. 2003.

[27] J. Nielsen. *Designing Web Usability.* New Riders Publishing, Indianapolis, IN, 2000.

[28] NoMachine NX. `http://www.nomachine.com`.

[29] R. Pike, D. Presotto, S. Dorward, B. Flandrena, K. Thompson, H. Trickey, and P. Winterbottom. Plan 9 from Bell Labs. *Computing Systems*, 8(3):221–254, Summer 1995.

[30] Portable Network Graphics (PNG). `http://www.libpng.org/pub/png/`.

[31] T. Porter and T. Duff. Compositing Digital Images. *Computer Graphics*, 18(3):253–259, July 1984.

[32] T. Richardson, Q. Stafford-Fraser, K. R. Wood, and A. Hopper. Virtual Network Computing. *IEEE Internet Computing*, 2(1), Jan./Feb. 1998.

[33] Runaware.com. `http://www.runaware.com`.

[34] Tarantella Web-Enabling Software: The Adaptive Internet Protocol. SCO Technical White Paper, Dec. 1998.

[35] R. W. Scheifler and J. Gettys. The X Window System. *ACM Trans. Gr.*, 5(2):79–106, Apr. 1986.

[36] B. K. Schmidt, M. S. Lam, and J. D. Northcutt. The Interactive Performance of SLIM: A Stateless, Thin-Client Architecture. In *Proceedings of the 17th ACM Symposium on Operating System Principles (SOSP)*, Dec. 1999.

[37] PC Anywhere. `http://www.pcanywhere.com`.

[38] Thin-Client market to fatten up, IDC says. `http://news.com.com/2100-1003-5077884.html`.

[39] T. E. Truman, T. Pering, R. Doering, , and R. W. Brodersen. The InfoPad Multimedia Terminal: A Portable Device for Wireless Information Access. *IEEE Transactions on Computers*, 47(10):1073–1087, Oct. 1998.

[40] SGI OpenGL Vizserver. `http://www.sgi.com/software/vizserver/`.

[41] Virtual Network Computing. `http://www.realvnc.com/`.

[42] A. Y. Wong and M. Seltzer. Operating System Support for Multi-User, Remote, Graphical Interaction. In *Proceedings of the USENIX Annual Technical Conference*, San Diego, CA, June 2000.

[43] Worldwide Enterprise Thin Client Forecast and Analysis, 2002-2007: The Rise of Thin Machines. `http://www.idcresearch.com/getdoc.jhtml?containerId=30016`.

[44] X Web FAQ. `http://www.broadwayinfo.com/bwfaq.htm`.

[45] S. J. Yang, J. Nieh, M. Selsky, and N. Tiwari. The Performance of Remote Display Mechanisms for Thin-Client Computing. In *Proceedings of the USENIX Annual Technical Conference*, June 2002.