UNIVERSITY OF CALIFORNIA, SAN DIEGO

Behavior of VNC in High-Latency Environments and Techniques for Improvement

A thesis submitted in partial satisfaction of the requirements for the degree
Master of Science

in

Computer Science

by

Taurin Pete Tan-atichat

Committee in charge:

Professor Joseph C. Pasquale, Chair
Professor William G. Griswold
Professor Geoffrey M. Voelker

2008

The Thesis of Taurin Pete Tan-atichat is approved, and it is acceptable in quality and form for publication on microfilm and electronically:

_____

_____

_____

Chair

University of California, San Diego

2008

DEDICATION

To my parents Eddie and Suchada Tan-atichat, for providing all of their love and support throughout the years and always going out of their way and helping me in any situation.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

ACKNOWLEDGEMENTS

I owe a great deal of thanks to my advisor Joe Pasquale. He always made time in his busy schedule to meet and check up on how I was doing whether he was in or out of town. He provided guidance and new directions when I believed I had hit a wall. He led me to avoid definite dead ends and let me know when I was too ambitious. His words of encouragement never ceased to help me pull through.

I also want to acknowledge my committee members Bill Griswold and Geoff Voelker for their long-time support and introduction of different approaches that had not occurred to me.

Special thanks to Cynthia Taylor for numerous helpful discussions on VNC, Fallon Chen for providing motivation and fantastic company in the lab, Jim Hong for some much needed distractions, and Dana Dahlstrom for releasing his LaTeX style classes which have proved most useful.

ABSTRACT OF THE THESIS

Behavior of VNC in High-Latency Environments and Techniques for Improvement

by

Taurin Pete Tan-atichat
Master of Science in Computer Science

University of California, San Diego, 2008

Professor Joseph C. Pasquale, Chair

Thin-client computing offers many advantages over traditional PC computing including lower costs, ubiquitous access to resources, higher security, and easier maintenance. Unfortunately thin clients rely upon the services of other networked computers to properly function. Because of this tight-knit relationship, high-latency environments can render thin clients practically unusable. With the *VNC* thin-client system, performance can be capped at 1 frame per *round-trip time* (RTT) due to its *client pull* communication style. We extend VNC to build *VNC-HL* and show how to improve frame rate performance by employing *PR*, a pre-requesting technique, to periodically request updates, even with several previous requests pending. Experimental results demonstrate up to an order of magnitude of improvement. VNC-HL achieved 14 FPS of multimedia computing in a 500 ms latency network with negligible additional resource consumption. In this thesis we demonstrate that VNC can be improved to sustain high frame rate performance in high-latency environments by employing a periodic pre-request at the desired FPS rate.

Chapter 1

Introduction

Thin-client computing has grown in popularity within recent years due to a number of benefits it provides over traditional PC computing. Advantages include lower costs, higher security, ubiquitous access to resources, and easier maintenance. A thin client is a lightweight device that primarily serves the functions of graphical display output and user input. The bulk of the user's computation is executed at a remote server. Communication between the thin client and the server is transferred over a computer network.

1.1 Popular Thin Clients

There have been a number of popular commercial thin-client systems in recent years including VNC, the X Window System (X11), and Microsoft Terminal Services [12, 14, 3]. Each system represents a different thin-client architecture.

VNC uses a low-level client pull technique that transfers remote desktop images to a thin client. A VNC client is stateless and simply requests an image of the user's computing environment and then displays it. A VNC server maintains all state regarding the user's computing session and replies back with an image of it when requested by the client.

The X Window System uses a high-level display protocol that allows applications running on a server to project their display output to a thin client's windowing system. The terminology of client and server is reversed, as the thin client is required to run an X Window Server that acts as a full windowing system. Applications must be modified into X Window Clients that perform all logical operations on the server and then send all graphical display system calls to the X Window Server on the thin client. A single X Window Server on the thin client can service multiple X Window Clients (applications) from remote servers.

Microsoft Terminal Services uses a proprietary protocol that allows users to access various resources on a remote computer, including the display. Since the protocol is proprietary it is difficult to detail its architecture, as many components are Microsoft-specific. A thin client using this protocol can access a remote file system, sound, printer, and other services.

These systems are a representative sample of the popular thin-client technologies currently on the market. We next draw analogies to different video distribution systems in order to help the reader frame these systems with more well-known concepts.

1.1.1 Thin Client Analogies

The VNC system can be thought of as a Netflix (online DVD rental) subscription. Users must first request a DVD; the DVD is then sent to the user; the user consumes the DVD with a commodity DVD player; and the process repeats from the beginning.

The X Window System can be thought of as digital cable television that requires a digital television set-top box at the user's end. This set-top box is designed to communicate exclusively with the service provider to obtain content. It then renders that content locally to the user. For example, the content could be rendered with picture-in-picture (multiple channels simultaneously) or as a preview inside a television channel guide (framed inside a table of television programming schedules).

Microsoft Terminal Services can be thought of as a movie theater. The entire

experience is first-rate but it is only available at select locations. Additionally, there is tight control over the distribution of movies to movie theaters.

1.1.2 Thin Client Conclusion

In the area of thin-client computing we believe it is ideal to put minimal requirements on the thin client, not require any modifications to existing software, and be able to support a large number of platforms. Because of these objectives we focus on VNC in this thesis. VNC has the best architecture to build off of as it embraces the thin-client philosophy of minimizing the amount of processing the thin client performs and allowing the thin client to control how much data it wishes to receive and process. We now give a more thorough introduction of VNC and its weakness in high-latency environments.

1.2 VNC Overview

Virtual Network Computing (VNC) is a system that uses an extremely thin, stateless software client that displays output of graphical images sent over a network from a server. The server provides a full computing environment for the user, including a windowing system. When in use, the client requests an image update from the server. The server transfers the current image of the user's desktop to the client that then renders the image for the user. This process repeats indefinitely. When a user provides input at the client, an input event is generated and the server processes the request appropriately. When a user is finished, the client simply disconnects from the server and the computing session is preserved on the server. A subsequent computing session is resumed in the same state as the user last left it.

1.3 High-Latency Environments

In this thesis we use the terms latency and round-trip time (RTT) interchangeably. We define RTT to be the transit time for a message to traverse across a network from one host to another host summed with the time for a reply back to the first host. High latency is when RTT is a long period of time, loosely in the hundreds of milliseconds.

VNC uses a *client pull* communication style. The client must initiate a request to the server and then wait for the server to respond. The server's sole responsibility is to process and reply to a client's request; it is not allowed to contact the client unsolicited. In a high-latency environment this pull technology suffers a significant performance penalty by having to wait at least one full RTT. This implies that VNC's frame rate has an upperbound of $\frac{1}{RTT}$ FPS since there can be at most one request/response pair in progress at any given time. In high-latency networks such as mobile phone and satellite networks the latency is typically in the hundreds of milliseconds, implying single-digit FPS. Clearly this limitation is a very undesirable characteristic.

1.4 Thesis and Organization

The objective of this thesis is to (1) investigate the behavior of VNC in high-latency environments, (2) introduce VNC-HL, our solution to improve frame rate performance, and (3) provide a comprehensive evaluation of both VNC and VNC-HL in high-latency environments with respect to frame rate performance, CPU consumption, and network utilization. Our thesis is:

> *The VNC thin-client system can be improved to sustain high performance, measured in frames per second (FPS), in high-latency environments by employing a periodic pre-request at the desired FPS rate.*

We validate this statement by extending VNC, performing realistic use experiments, and showing significant improvements in high-latency environments.

This thesis is organized as follows: Chapter 2 provides more details of the VNC thin-client system and of the growing problem of high-latency environments. Chapter 3 describes the design and implementation of VNC-HL. Chapter 4 contains experimental results that show VNC-HL delivers performance up to an order of magnitude higher than VNC in high-latency environments and discusses the reasons for it. Chapter 5 and Chapter 6 describe related and future work respectively. Finally, we conclude in Chapter 7.

Chapter 2

Background

VNC has been released for over 10 years and is installed by default on Ubuntu Linux, one of the most popular Linux distributions with over 8 million users [5]. Unfortunately there are no major books written on VNC and thus no comprehensive authoritative documentation guide. This chapter is intended to provide details on a few key aspects of VNC and then discuss the implications of high latency and networking protocols.

2.1 VNC Details

Virtual Network Computing (VNC) was first introduced in 1998 as an ultra-thin-client system that was platform-independent [12]. VNC provides a desktop environment that can be accessed from any computer with network connectivity to the VNC server. It essentially allows a user to access a remote computer as if that user were sitting right in front of the actual remote machine. Additionally, the user may disconnect and reconnect to the server from another computer and have the user's desktop environment state perfectly preserved between sessions.

VNC achieves this extremely flexible method of mobile computing by using a simple protocol that operates at the framebuffer level. All modern operating systems

have a framebuffer that contains information of what should be graphically displayed to the output device. This framebuffer can be thought of as the pixels of what the user normally sees. The image can be transferred by sending all pixel values line by line. This crude technique is known as *raw encoding*. VNC also uses other more efficient strategies such as *Zlib Run-Length Encoding* (ZRLE) and *Hextile* that compress graphical updates. This thesis exclusively utilizes the *raw encoding* for simplicity but our work can be generalized to other encodings with minimal changes.

2.1.1 Two Separate Components

Figure 2.1 shows that the VNC system is broken up into 2 separate components, the VNC client (also known as the VNC viewer) and the VNC server. The VNC client is located on the thin-client computer and provides access to resources at another computer that is running the VNC server. The client regularly requests an update of the framebuffer from the server, the server sends a response, and then the client processes and displays it. The client also notifies the server of any cursor or keyboard input. The server simply listens for requests by clients and fulfills them.

2.1.2 Protocol

The VNC protocol is divided up into 2 main phases. The first phase includes initialization and negotiation between the client and the server. This includes exchanging protocol version numbers, security and authentication, resolution size, and pixel format. The second phase consists of the client continuously requesting the server for framebuffer updates as well as sending input device requests. More specifically, clients can ask for incremental updates or specify arbitrary screen areas for updates. Incremental updates are used to decrease the amount of data needed to be transferred by only sending parts of the framebuffer that have been modified since the last update. It should be noted that a framebuffer update request occurs only after the last update has been rendered, consequently allowing only one framebuffer update request to be outstanding.

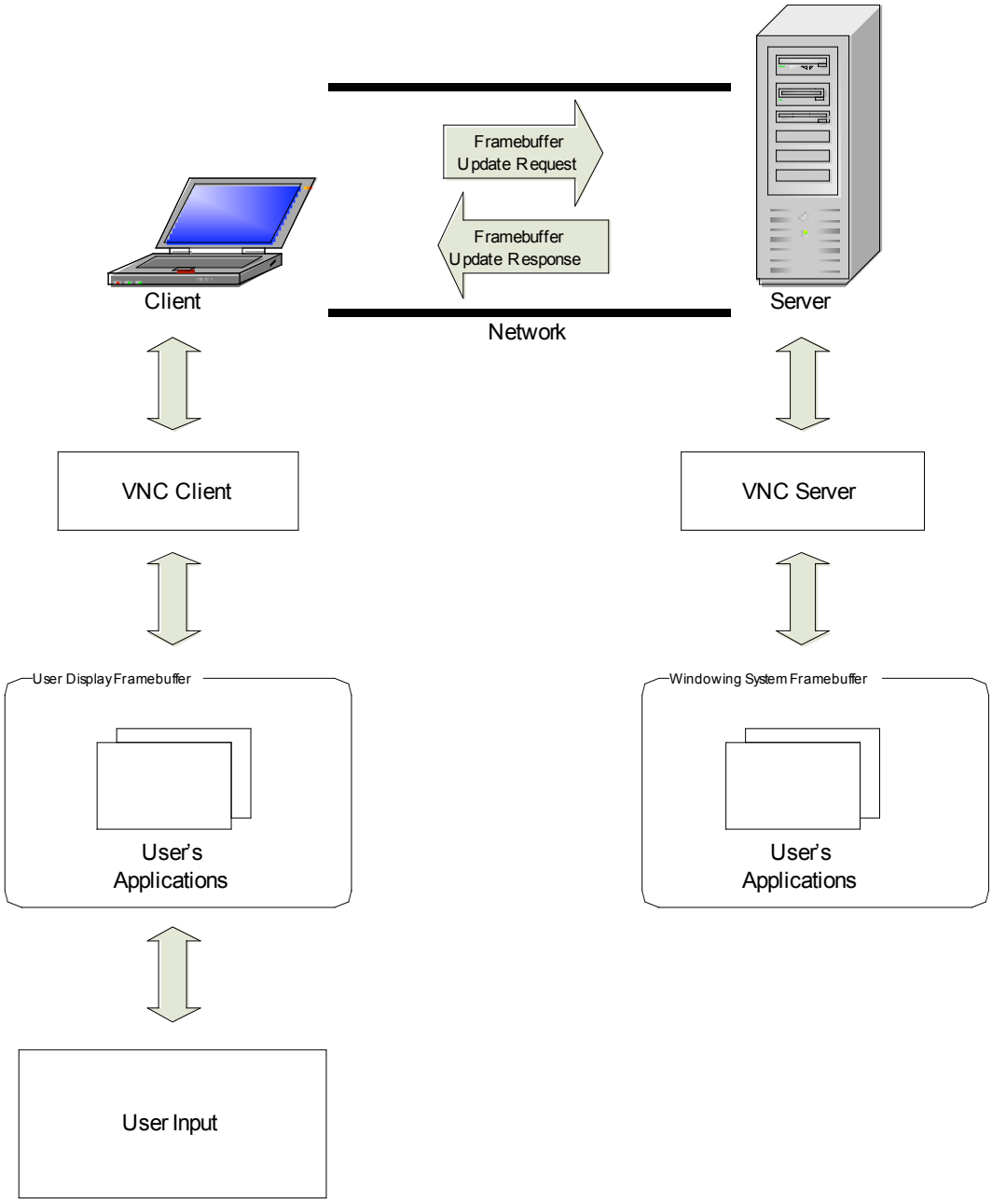

Figure 2.1 VNC System Overview.

Additionally, there are other tasks performed by the client including the timing of frame updates in order to estimate available bandwidth and select appropriate encodings. This thesis focuses on modifying client behavior to improve frame rate performance.

2.1.3 An Ideal Foundation

The main advantage of extending this simple design and protocol is that we can leverage the existing VNC client and server implementations that span a wide variety of operating systems and platforms. VNC supports both clients and servers for major operating systems including Microsoft Windows, Mac OS X, UNIX, and Linux. VNC also supports countless additional platforms including personal digital assistants (PDAs) and mobile phones [15]. Additionally, no modifications to existing applications are required. With an already large base of users that we can only assume will continue to grow, we believe VNC is an ideal thin-client system to investigate, research, and extend.

2.2 High-Latency Environments

Bandwidth and latency are the two most important components that characterize computer networks. Both components have dramatically improved in recent years. For example the Ethernet family has gone from 100 Mbps in 1995 to 1000 Mbps in 1999 and even to 10000 Mbps in 2003 [9]. However, latency has not attained nearly the same levels of improvement.

Bandwidth can be improved with the use of additional parallel network links or compression. There have been latency improvements, but they are slow in coming. And despite these improvements, a plot of bandwidth and latency over time shows that bandwidth improves at a rate that is at least the square of the improvement of latency [9].

These trends lead us to presume that latency will be the area of largest concern in our networks in the future, if not already. A thin-client system that is available world-

wide would need to account for a round-trip time (RTT) around the world, which is, at minimum, in the hundreds of milliseconds. Systems that require synchronous communication could be adversely affected due to the number of long RTT time periods in between messages.

Examples of commonly deployed high-latency environments include mobile phone networks and satellite connections. Latencies for both of these run in the hundreds of milliseconds [8]. Although interplanetary networking is currently being planned with projects such as InterPlanetary Internet [2], we limit the scope of high-latency environments to our global network with traditional protocols.

2.3 Network Fundamentals

VNC requires a network that can support a reliable transport protocol. Transmission Control Protocol (TCP) used over Internet Protocol (IP) is the de facto standard set of protocols. Special attention must be paid to TCP since it controls and regulates the amount of data that the sender can transmit without an acknowledgement from the receiver.

The most important relevant TCP concept is the TCP window. The TCP *advertised window* is used for flow control from the sender to the receiver. It is defined to be the number of bytes that the receiver is willing to accept from the sender. The TCP *congestion window* is another mechanism to limit network use in order to prevent congestion. It is constantly adjusted but typically starts small and increases to a value that is near the network limit. It presumes that the loss of a packet indicates a network overload and will then dramatically lower the window to prevent further loss.

These two windows have huge consequences in high-latency environments. For VNC to perform well, it behooves the server to send an entire framebuffer update without having to receive an acknowledgement. First, the advertised window must be set to be at least as large as the size of one framebuffer but ideally should be as large as the *bandwidth-delay product*, calculated by multiplying the bandwidth with the RTT. With

an advertised window as large as the bandwidth-delay product, a sender can utilize the full throughput of the network because the sender will receive acknowledgements from the receiver before the advertised window becomes full. If a default advertised window is set too low, a framebuffer update will take over 1 RTT to complete because the sender will have to stop transmitting and wait for at least one acknowledgement from the receiver before finishing the transmission of the rest of the framebuffer. Second, the congestion window typically undergoes exponential growth during the beginning of the connection to quickly determine an estimate of the network's capacity, e.g. *slow-start* [16]. But this exponential growth is a function of the RTT; the network transmission sizes only increase after every successful acknowledgement. This is certainly undesirable in the case of high latency.

This thesis focuses only on the TCP advertised window. The TCP congestion window is a large and complex topic that is out of the scope of this work. We point out that the *CUBIC* TCP implementation, currently used in the latest Linux kernel, does recover quickly after a dropped packet in high-latency networks because its window growth function is based on units of real time, not units of RTT [11]. We assume that our network is sufficient to handle all required traffic. VNC's traffic is typically on the order of 10 Mbps so we feel this assumption should be safe on modern networks.

2.4 Goals

We attempt to take advantage of the design of VNC to develop a system to provide improved frame rate performance. We do so by modifying the VNC client, keeping the VNC server as little changed as possible. This allows for backwards compatibility that is extremely important since there already exists a large number of VNC server implementations across a myriad of platforms.

2.5 Chapter Summary

In this chapter we have provided the pertinent details of VNC, qualified the motivation to extend VNC, described the growing problem of high-latency environments, and introduced basic TCP behavior and characteristics.

Chapter 3

Design and Implementation

There are several available versions of VNC such as RealVNC, TightVNC, and UltraVNC [10, 17, 18]. RealVNC was created by the original developers of VNC. Other implementations typically focus on improving VNC by supporting more efficient encodings to save bandwidth or providing additional security. To the best of our knowledge there is no implementation of VNC that focuses on improving performance in high-latency environments. We designed and developed VNC-HL with this one goal in mind.

3.1 Pre-Requesting

In order to provide good VNC performance over high latencies, one must ensure that a framebuffer be transferred from the server to the client soon after any update occurrence. Unfortunately VNC is a *client pull* system that does not allow servers to initiate commands. The goal of maximizing frame rate can be achieved if a client's framebuffer update request arrives at the server immediately following a change in the framebuffer. We attempt to emulate this behavior by having a client periodically pre-request framebuffer updates.

As mentioned in Section 2.1.2, a client can only have up to one framebuffer update request outstanding. On a high-latency network, this results in a long delay from

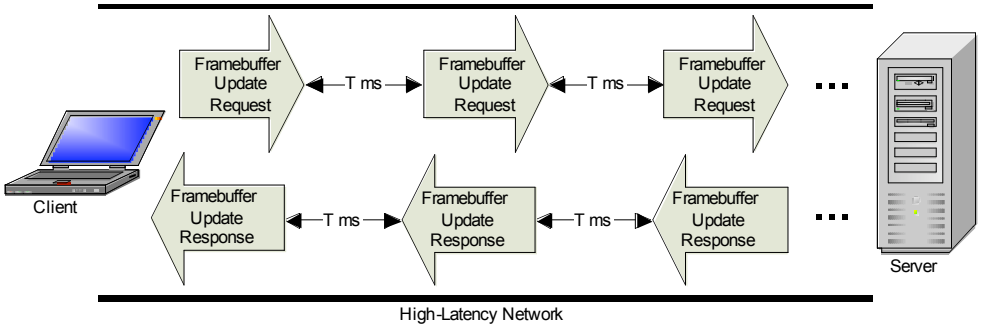

Figure 3.1 VNC-HL System Diagram with a PRP of T ms.

the time of the client's request until the time the client has fully received the server's response. This can be thought of as a stop-and-wait protocol. One can attempt to improve the client by modifying it to make multiple requests before receiving a response but it is not clear what sort of scheme would result in good performance. Sending too many requests could overload the network, server, or both, while sending too few could result in suboptimal performance. We chose to employ a timing scheme to periodically send a framebuffer update request to the server. We refer to this pre-requesting technique as *PR* and the time period between pre-requests to be the *pre-request period* (PRP). Figure 3.1 illustrates the VNC-HL system.

In Figure 3.2 we illustrate the behavior of our timing scheme. Every time a framebuffer update is received and rendered, a VNC client will normally send a framebuffer update request and then block while it waits for a response from the server. VNC-HL (our system) also behaves in the same way except a timer is set for PRP time units at the time the process starts to block. If the timer expires, the process wakes up and an additional framebuffer update request is sent and then the same steps are repeated.

A reasonable question to ask is why should VNC-HL send a framebuffer update request immediately after a framebuffer update response is received and rendered. The timer should guarantee that a framebuffer update request should be sent periodically. However, preliminary tests showed that sending framebuffer update requests solely based on the timer yielded worse performance and we discuss reasons for not

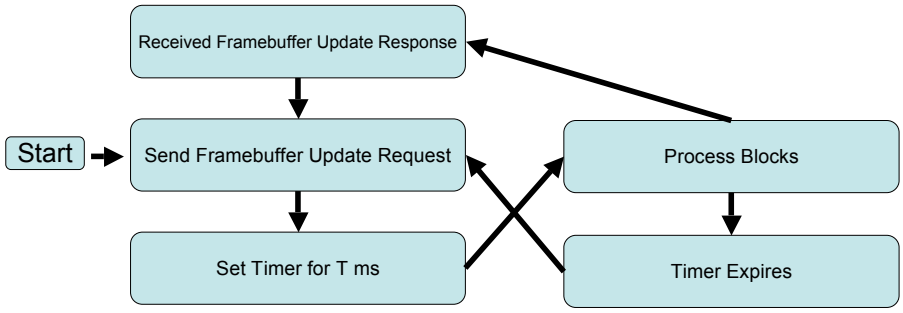

Figure 3.2 VNC-HL Pre-Request Diagram with a PRP of T ms.

doing so.

A timer is only started when a client is idle and waiting for a response from the server. When framebuffer updates are very large it takes a significant amount of time to receive the update and then render the update. If a client happens to take 20 ms to receive and render an update and has a PRP of 40 ms, a framebuffer update request will only be sent every 60 ms. Another reason for immediately requesting a framebuffer update after just receiving and rendering one is because we expect VNC-HL to reach a steady state where enough framebuffer requests have been injected into the system that not many more additional requests are needed. As long as the server is not overloaded, clients should receive the same periodic server responses and render output at a smooth, high FPS. Using this scheme, there should be minimal delay between the time a change in the framebuffer occurs until the time a client receives and displays the update. We discuss tuning the PRP parameter in Sections 4.7 and 4.8.

3.2 PR Implementation

We extended the RealVNC 4.1.2 source distribution for UNIX platforms. We implemented PR, the pre-request mechanism, by adding a timer to a *select()* loop that waited for incoming network activity. If no activity occurred, the timer expires, then a new framebuffer update request is submitted and we repeat the loop. Other modifica-

tions included commenting out optimization code that would cause the process to block permanently until user input or network activity occurred. This code slightly interfered with PR but could be modified and added back if desired. Instrumentation code was also added.

3.3 Chapter Summary

In this chapter we described the VNC-HL system, our rationale for the design of PR, the pre-requesting mechanism used by VNC-HL, how PR behaves, why timers should not solely be used for sending framebuffer update requests, and how VNC-HL was implemented.

Chapter 4

Experimental Results and Discussion

We subjected both VNC and VNC-HL to a test suite that consisted of a full battery of experiments designed to provide insight into the behavior of both systems with respect to frame rate performance, CPU consumption, and network utilization. Frame rate performance was the most important metric measured because it directly affected the user experience. Standard video playback is often at 30 FPS so we arbitrarily chose half of that value (15 FPS) as a threshold for sufficient performance.

4.1 Methodology

Experimental tests were selected to be representative of real world conditions and practical applications. The two main parameters tested were latency (RTT) and the VNC-HL pre-request period (PRP). Because of these 2 dimensions of variability we have opted to display many of the results with contour maps for ease of viewing. Each test suite consisted of hundreds of individual tests that were each allocated a generous testing time period of 1 minute for capturing results that were normally in the 15 FPS range. To ensure confidence in accuracy, each individual test was conducted 5 times and then averaged. The 5 individual test results had very little variation among each other: 15 FPS average means had a standard deviation of about 0.13. Assuming a normal

distribution, the probabilities (p-values) for observing VNC-HL's performance results while expecting VNC's performance results were essentially 0.

4.1.1 Hardware

A Dell Optiplex 320 running an Intel®Core™2 Duo CPU E4400 @ 2.00GHz with 1 GB of RAM was set up to act as a VNC server. A Dell Dimension 4400 running an Intel®Pentium®4 CPU @ 2.00GHz with 1 GB of RAM was set up to act as a VNC/VNC-HL client. Both ran Ubuntu 7.10 with a Linux 2.6.22-14 kernel. They were connected on a Gigabit Ethernet LAN that had latencies on the order of 0.1 ms between hosts. This thesis assumes that this inherent latency is negligible and our discussion of latency is the additional latency above and beyond this minimal latency that exists on our system. The VNC implementation used was the RealVNC 4.1.2 source distribution for UNIX platforms. VNC-HL was based off of the same package.

4.1.2 Experimental Setup

The experimental tests consisted of establishing a 24-bit color 1024 X 768 resolution VNC session from the VNC/VNC-HL client to the VNC server for approximately 1 minute. Realistic usage was simulated with 2 scenarios: a video playing in a web browser (YouTube) and an inactive screen. We chose these 2 scenarios because they are the extreme cases of when large CPU and network resources are required and when little CPU and network resources are required. They are also common computing use cases, for example when a user is watching media and when a user is busy reading the text of some web content. The data collected included the number of framebuffers received, the number of framebuffer requests sent, the CPU consumption, and the average network utilization. Parameters that were varied included the VNC-HL PRP, the latency between the two computers, and the TCP window sizes.

Table 4.1 Comparison of VNC and VNC-HL FPS for video with default TCP window.

Latency (ms)	VNC-HL 1 ms PRP FPS	VNC-HL 50 ms PRP FPS	VNC-HL 150 ms PRP FPS	VNC-HL 500 ms PRP FPS	VNC FPS
0	20.8	17.8	13.7	11.6	10.4
50	8.1	6.8	6.3	2.8	2.8
150	2.9	2.4	2.3	1.6	1.0
500	0.8	0.8	0.7	0.7	0.3

4.2 Default TCP Window Size

The default TCP window size in many systems is set to approximately 85 kB and can grow up to twice that size, 170 kB. Initially, both VNC and VNC-HL were tested with this configuration since it would be the most common situation. The first experiment was the scenario of a user watching a video in a web browser. The results were poor for both systems when viewing the video over a high-latency network. The trend was for FPS to proportionally decrease as latency increased. For example, a latency of 40 ms yielded roughly 7-8 FPS; a latency of 80 ms yielded roughly 3-4 FPS; and a latency of 150 ms yielded roughly 2-3 FPS. This was because the default TCP window size was not large enough. The VNC server could not send the entire framebuffer without receiving acknowledgements part of the way through the transmission. The required acknowledgements forced the time for the transmission to complete to increase at least one RTT, a long duration of time in high latency, as described in Section 2.3. Figure 4.1 shows the contour map of the FPS of VNC-HL across all PRPs and network latencies tested. Any latency over 100 ms caused FPS to drop below 4 no matter which PRP used. Table 4.1 compares major data points of VNC and select VNC-HL PRPs. Although VNC-HL did outperform VNC for all latencies and all PRPs considered, we consider performance to still be poor since performance was less than 3 FPS with a 150 ms latency.

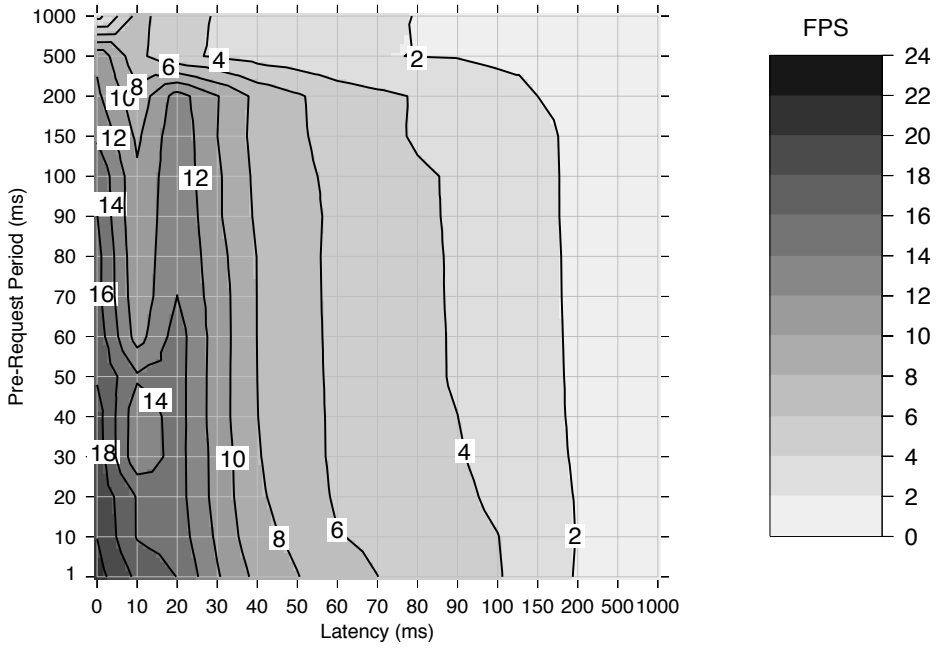

Figure 4.1 Contour map of VNC-HL FPS for video. The TCP window size is set to the default, approximately 85 kB - 170 kB.

One framebuffer update of video consumed approximately 325 kB of network resources. With a TCP window size that is up to 170 kB, framebuffer requests/responses should take at least 2 RTT, one RTT for the client request and the first server response of 170 kB, one additional RTT for the client acknowledgement and the rest of the server response. Bandwidth constraints and overhead at the client and server consume additional time. The highest FPS achieved on a 0 ms latency network was 20.8, or 48 ms per frame. So we estimate that the period between frames should be about *2 RTT + 48 ms*.

This estimate correlates well with our observed results so we can reasonably conclude that our TCP window was being filled, resulting in frequent stagnation of traffic flow between the two computers. After this experiment, the TCP window size was changed to 32 MB. This value should support the transfer of 100 video framebuffer updates before an acknowledgement. Even at 1-second latency, this figure is beyond the 20 FPS our system can achieve so the TCP window size should not serve as a bottleneck.

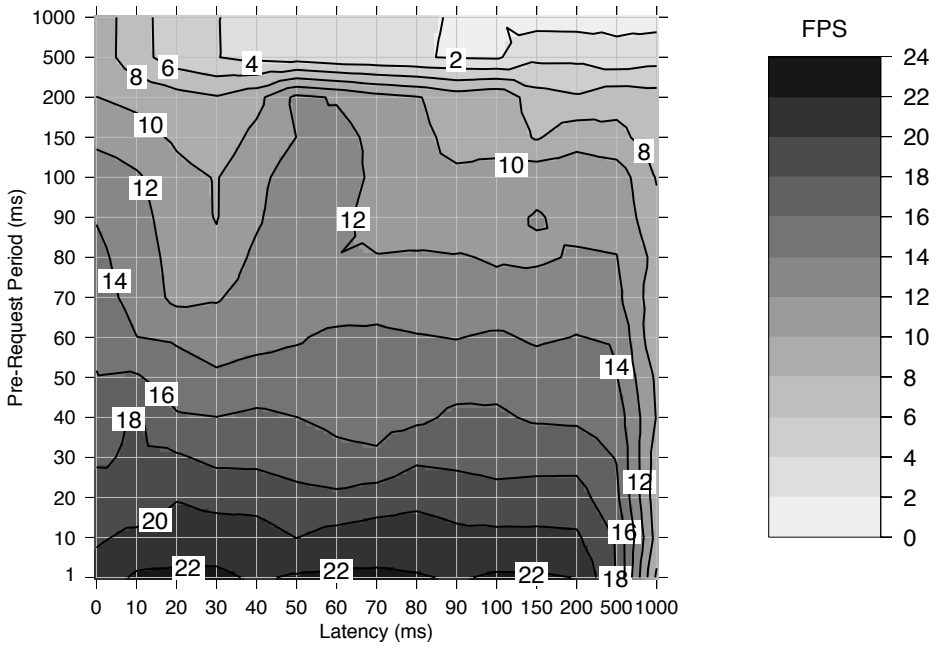

Figure 4.2 Contour map of VNC-HL FPS for video. The TCP window size is set to 32 MB.

This figure is also still reasonable for thin-clients to support since current thin-clients have RAM on the order of hundreds of MBs and even GBs [6]. All further results are based using this new window size.

4.3 FPS Performance

A high FPS rate is crucial for smooth video playback. Figure 4.2 shows the results of the same batch of tests performed in the last section for VNC-HL, with the only change being an increased TCP window size. VNC-HL achieved 15 FPS or higher nearly all the way across the board of latencies for PRPs of 50 ms or faster. We attribute the increase in performance to the unclogging of the TCP window bottleneck that was discussed in the last section.

A comparison of VNC to a select number of VNC-HL PRPs is shown in Figure

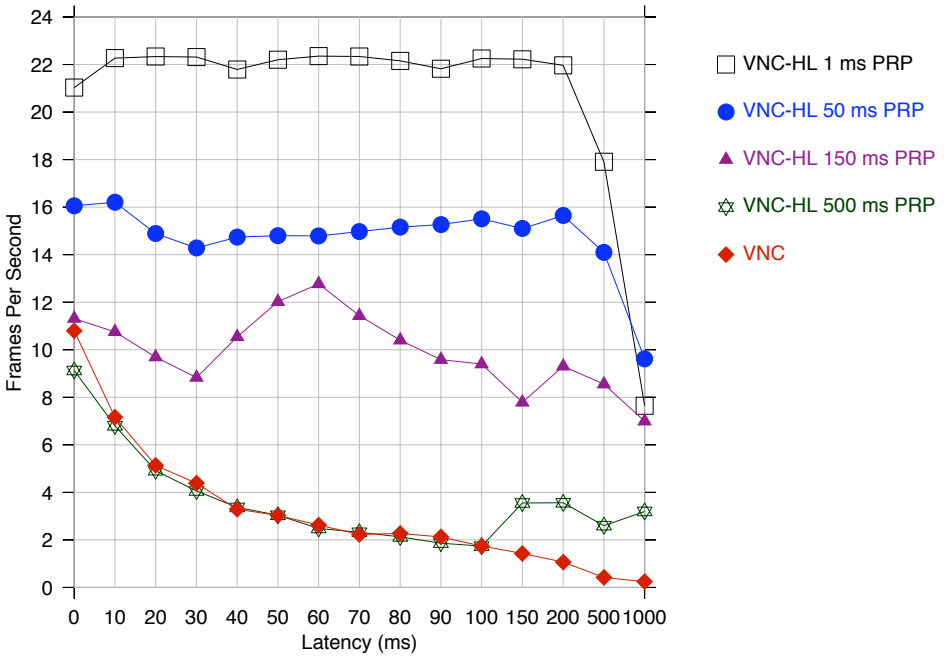

Figure 4.3 FPS performance comparison of VNC and VNC-HL for video.

4.3. VNC-HL significantly outperformed VNC at all latencies for all PRPs below 500 ms. For PRPs of 500 ms and slower, framebuffer update requests would occur only after framebuffer update responses were received and never from expiring timers, yielding the same behavior as VNC. This happened because framebuffer update request/response cycles would normally take about RTT time, usually much shorter than the 500 ms and longer timer length.

A PRP of 50 ms or faster yielded an order of magnitude of improvement at latencies slower than 150 ms. We attribute the increase in performance to VNC-HL's PR that keeps multiple framebuffer update requests outstanding to create a steady flow of framebuffer update responses. There is a noticeable drop in FPS with a 1 ms PRP at 1000 ms latency and is discussed in Section 4.6.

The experiment was repeated for the scenario of an inactive computing session. The resulting contour map is shown in Figure 4.4. At low latency, PRP did not appear to be a factor on performance. We attribute this to the little CPU and network requirement

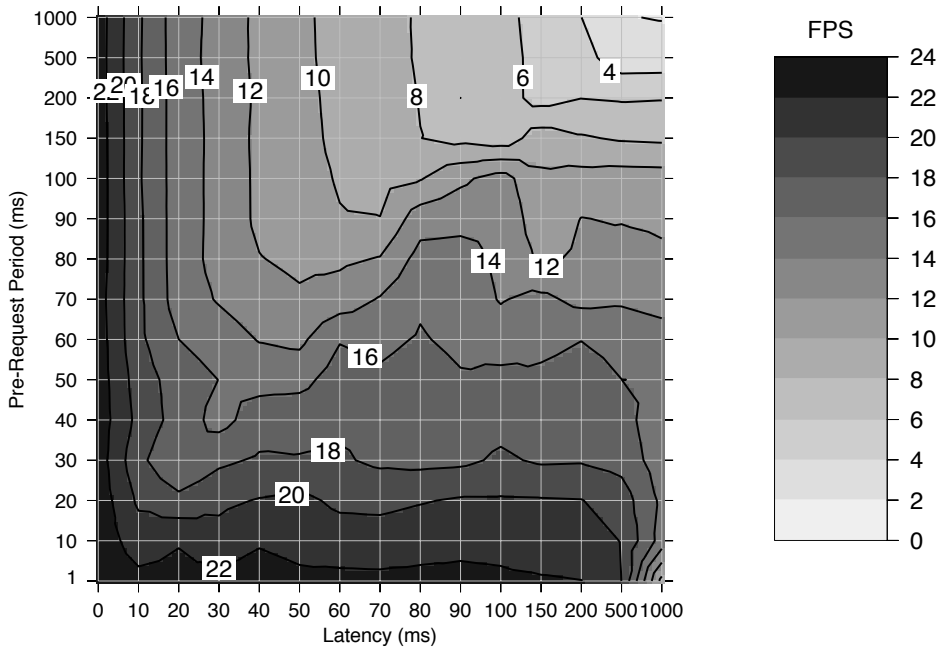

Figure 4.4 Contour map of VNC-HL FPS for inactive screen.

for inactive computing as discussed further in the following subsection. A comparison between VNC and VNC-HL is shown in Figure 4.5. VNC faired better in this inactive screen test than the video test. VNC and VNC-HL provided similar performance until around 20 ms of latency; after that VNC-HL offered significantly higher FPS. At latencies of 500 ms and slower VNC-HL with fast PRPs performed roughly an order of magnitude higher. Again we attribute the increase in performance to VNC-HL's PR.

4.3.1 Inactive Screen and Video Differences

VNC is designed to save network bandwidth by transmitting updates only for regions of the screen that have changed since the last update. This allows a VNC session that has a fairly inactive screen to use very little network resources. It also indirectly reduces CPU consumption, having less network traffic to process and display regions to update.

Using VNC to view video requires much more bandwidth than an inactive screen.

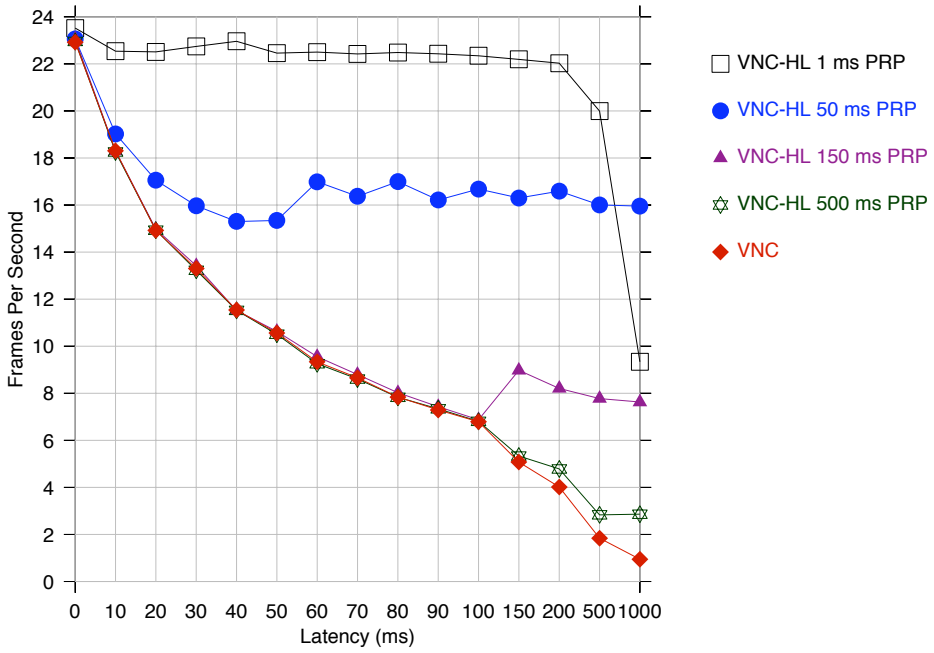

Figure 4.5 FPS performance comparison of VNC and VNC-HL for inactive screen.

During the tests we found that one framebuffer update of video was approximately 325 kB and one framebuffer update of an inactive screen was 25 kB. Even though the two computers were on a Gigabit Ethernet LAN that should be able to transfer 325 kB in less than 3 ms, there is overhead to determine which area of the screen needs to be transmitted, encoding it, decoding it, and displaying it on the thin client. The time to process this overhead is directly added to the time period of frame-to-frame displays. When latency is introduced, even more delay is directly added to the period between frames and VNC suffers tremendously.

This overhead also caused VNC-HL to perform poorly at low latency and long PRPs. When VNC-HL uses a PRP that is longer than the latency of the network it behaves very much like VNC. Since VNC performed fairly well at low latencies on with an inactive screen, VNC-HL performed well at low latencies regardless of PRP.

When PRPs are shorter than the latency of the network VNC-HL pipelines the processing queue at both the server and client so that they both work efficiently and

Table 4.2 Comparison of VNC and VNC-HL CPU % for video.

Latency (ms)	VNC-HL 1 ms PRP CPU % (FPS)		VNC-HL 50 ms PRP CPU % (FPS)		VNC-HL 150 ms PRP CPU % (FPS)		VNC-HL 500 ms PRP CPU % (FPS)		VNC CPU % (FPS)	
0	8.5	(21.0)	5.8	(16.1)	4.2	(11.3)	3.0	(9.1)	4.2	(10.8)
50	8.4	(22.2)	5.2	(14.8)	4.0	(12.0)	0.8	(3.0)	1.0	(3.0)
150	8.2	(22.2)	5.2	(15.1)	2.6	(7.8)	1.0	(3.6)	0.0	(1.4)
500	5.4	(17.9)	4.2	(14.1)	2.8	(8.6)	1.0	(2.6)	0.0	(0.4)

consequently provide a much higher FPS. This is best demonstrated on the video tests. By keeping the server's queue full with rapid firing framebuffer update requests VNC-HL delivers nearly optimal FPS across all latencies.

4.4 CPU Consumption

Thin clients are designed to consist of low resources that are adequate for their needs and not much more. While it is desired to produce a system that achieves optimal performance, one must be aware of the resource consequences and lean towards a good compromise when tuning parameters. Figure 4.6 shows a contour map of CPU consumption for video. It has a strong resemblance to the FPS contour map. The reason for this correlation is because every framebuffer update requires a certain amount of CPU time to process and update the screen. Table 4.2 summarizes and compares CPU consumption and FPS for the viewing of a video for both VNC and VNC-HL. CPU consumption is highly correlated to FPS. VNC-HL required more CPU time than VNC with respect to latency because VNC-HL's PR resulted in higher FPS. There is no significant difference in CPU time when comparing VNC to VNC-HL when both are performing at the same FPS. This implies the efficiency of VNC-HL is not significantly worse than VNC.

Both VNC and VNC-HL consumed negligible CPU time for the inactive screen test as shown in Figure 4.7 and Table 4.3. This is due to the small amount of processing

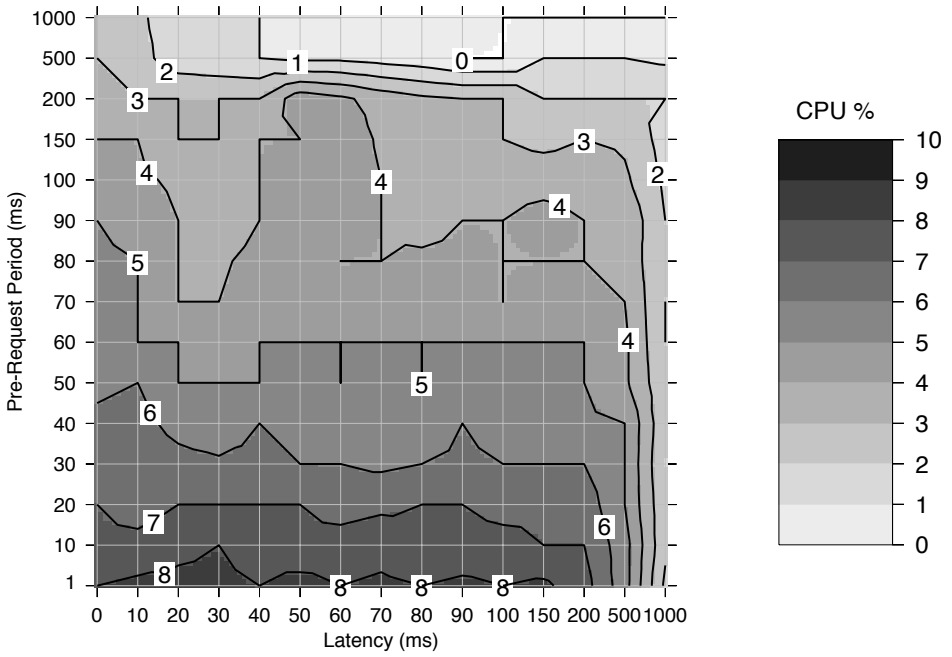

Figure 4.6 Contour map of VNC-HL CPU % for video.

of framebuffer update responses required for inactive screens. Note that CPU time granularity was only available to the nearest integer and that 5 tests were averaged together to result in 0.2 unit precision.

Table 4.3 Comparison of VNC and VNC-HL CPU % for inactive screen.

Latency (ms)	**VNC-HL 1 ms PRP CPU % (FPS)**		**VNC-HL 50 ms PRP CPU % (FPS)**		**VNC-HL 150 ms PRP CPU % (FPS)**		**VNC-HL 500 ms PRP CPU % (FPS)**		**VNC CPU % (FPS)**	
0	0.2	(23.2)	0.2	(23.1)	0.4	(23.0)	0.4	(23.0)	0.4	(22.9)
50	0.2	(22.5)	0.0	(15.3)	0.0	(10.6)	0.0	(10.5)	0.0	(10.6)
150	0.2	(22.2)	0.0	(16.3)	0.0	(9.0)	0.0	(5.3)	0.0	(4.0)
500	0.0	(20.0)	0.0	(16.0)	0.0	(7.8)	0.0	(2.8)	0.0	(1.8)

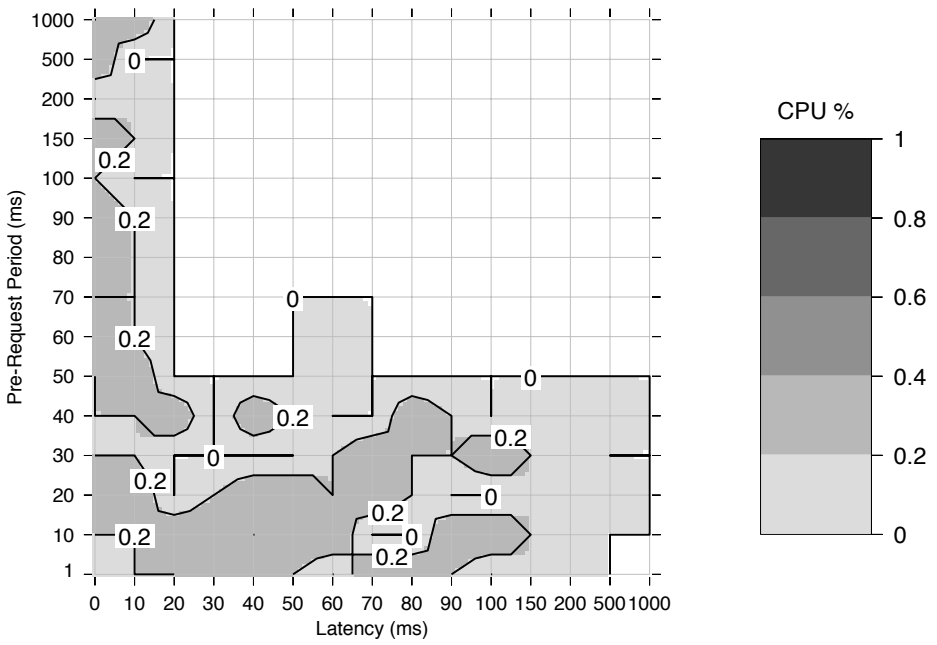

Figure 4.7 Contour map of VNC-HL CPU % for inactive screen.

4.5 Network Utilization

Results for network receive throughput and network send throughput for video are summarized in Tables 4.4 and 4.5. Throughput was directly proportional to the FPS for each test. This was as expected since each framebuffer update request/response used a consistent amount of network resources. Higher FPS implied more framebuffers transferred over the network and lower FPS implied less.

The network utilization for received traffic mainly comprised of framebuffer update data. For video, each framebuffer update was approximately 325 kB. When multiplied with the FPS we find that the network utilization was very efficient. For example, VNC-HL with a 50 ms PRP yielded 14.8 FPS and utilized 5.04 MB/s at 50 ms latency. The raw framebuffer update data is estimated to be 14.8 FPS X 325 kB = 4.70 MB/s, over a 93% efficiency rate. The efficiency rate for an inactive screen is lower since each framebuffer update is small, 25 kB, and overhead, such as packet headers, becomes a larger portion of traffic.

Table 4.4 Comparison of VNC and VNC-HL network traffic for video.

Latency (ms)	**VNC-HL 1 ms PRP**	**VNC-HL 50 ms PRP**	**VNC-HL 150 ms PRP**	**VNC-HL 500 ms PRP**	**VNC**
	recv (MB/s) sent (kB/s) (FPS)	**recv (MB/s) sent (kB/s) (FPS)**	**recv (MB/s) sent (kB/s) (FPS)**	**recv (MB/s) sent (kB/s) (FPS)**	**recv (MB/s) sent (kb/s) (FPS)**
0	7.10 160.17 (21.0)	5.42 107.21 (16.1)	3.84 77.91 (11.3)	3.12 60.95 (9.1)	3.65 64.81 (10.8)
50	7.50 162.31 (22.2)	5.04 93.67 (14.8)	4.10 77.91 (12.0)	1.10 22.83 (3.0)	1.10 20.65 (3.0)
150	7.51 163.19 (22.2)	5.12 97.40 (15.1)	2.70 51.25 (7.8)	1.27 25.36 (3.6)	0.55 12.09 (1.4)
500	6.10 107.95 (17.9)	4.77 86.12 (14.1)	2.97 56.73 (8.6)	0.96 20.08 (2.6)	0.21 7.07 (0.4)

Table 4.5 Comparison of VNC and VNC-HL network traffic for inactive screen.

Latency (ms)	**VNC-HL 1 ms PRP**	**VNC-HL 50 ms PRP**	**VNC-HL 150 ms PRP**	**VNC-HL 500 ms PRP**	**VNC**
	recv (kB/s) sent (kB/s) (FPS)	**recv (kB/s) sent (kB/s) (FPS)**	**recv (kB/s) sent (kB/s) (FPS)**	**recv (kB/s) sent (kB/s) (FPS)**	**recv (kB/s) sent (kb/s) (FPS)**
0	653.13 31.54 (23.5)	625.56 12.67 (23.1)	661.37 12.49 (23.0)	640.23 12.66 (23.0)	612.95 12.32 (22.9)
50	652.52 32.61 (22.5)	455.51 11.20 (15.3)	329.71 7.83 (10.6)	326.87 7.83 (10.5)	328.29 7.77 (10.6)
150	646.41 32.88 (22.2)	481.90 11.86 (16.3)	286.88 7.30 (9.0)	189.58 5.58 (5.3)	182.79 5.44 (5.1)
500	590.55 29.48 (20.0)	478.95 12.09 (16.0)	259.74 7.11 (7.8)	127.69 4.25 (2.8)	102.44 3.59 (1.8)

Table 4.6 VNC-HL Pre-Requests Per Second (PRPS) for video.

Latency (ms)	**VNC-HL 1 ms PRP PRPS (FPS)**	**VNC-HL 50 ms PRP PRPS (FPS)**	**VNC-HL 150 ms PRP PRPS (FPS)**	**VNC-HL 500 ms PRP PRPS (FPS)**	**VNC PRPS (FPS)**
0	217.35 (21.0)	4.91 (16.1)	3.01 (11.3)	0.00 (9.1)	- - - (10.8)
50	212.74 (22.2)	9.78 (14.8)	1.00 (12.0)	0.00 (3.0)	- - - (3.0)
150	213.82 (22.2)	8.90 (15.1)	2.46 (7.8)	0.22 (3.6)	- - - (1.4)
500	218.16 (17.9)	7.44 (14.1)	1.47 (8.6)	1.00 (2.6)	- - - (0.4)
1000	**197.86 (7.6)**	7.51 (9.6)	1.81 (7.0)	0.43 (3.2)	- - - (0.2)

The network utilization for sent traffic mainly comprised of framebuffer update requests. Each request was only 10 bytes in length so the majority of the traffic generated was due to overhead in the packet headers and the TCP acknowledgements for received data. Although this is a huge proportion of overhead, in absolute terms it is still a small amount of network utilization.

4.6 Pre-Request Frequency

In order to gain more insight into the performance of VNC-HL, the number of pre-requests per second was measured during the experiments. Tables 4.6 and 4.7 summarize the results. Typically the faster the PRP is set to, the higher the FPS becomes. This is as expected since more requests should yield more responses but there is an anomaly when latency was increased to 1000 ms. FPS suddenly dropped when using a 1 ms PRP on an inactive screen. We investigated this and found the cause to be an overloading of the server that is discussed in the next section.

4.7 Fast PRPs

A naive strategy for obtaining the highest FPS would be to request a framebuffer update as quickly and as often as possible. Experimental results show that this did

Table 4.7 VNC-HL Pre-Requests Per Second (PRPS) for inactive screen.

Latency (ms)	**VNC-HL 1 ms PRP PRPS (FPS)**	**VNC-HL 50 ms PRP PRPS (FPS)**	**VNC-HL 150 ms PRP PRPS (FPS)**	**VNC-HL 500 ms PRP PRPS (FPS)**	**VNC PRPS (FPS)**
0	239.18 (23.5)	0.85 (23.0)	0.02 (23.0)	0.00 (23.0)	- - - (22.9)
50	239.14 (22.5)	11.46 (15.3)	0.01 (10.6)	0.00 (10.5)	- - - (10.6)
150	237.94 (22.2)	11.81 (16.3)	1.88 (9.0)	0.01 (5.3)	- - - (5.1)
500	235.79 (20.0)	12.32 (16.0)	1.13 (7.8)	1.04 (2.8)	- - - (1.8)
1000	**208.41 (9.3)**	12.32 (16.0)	1.76 (7.6)	0.33 (2.9)	- - - (0.9)

provide good FPS for nearly all latencies but there came a point when the server got overloaded. With a 1 ms PRP, VNC-HL fired framebuffer update requests at roughly 200 times per second. The VNC server is designed to ignore requests when the current state of the framebuffer is the same as the last update sent to the requesting client so it was unlikely that the framebuffer would change fast enough so that the server would send an update for each of those requests.

The main reason for the performance degradation was due to an overloaded server. With high latency and fast PRP, VNC-HL actually filled the TCP receive window of the VNC server with hundreds if not thousands of framebuffer update requests. When the window was full, the VNC-HL client had to halt the sending of additional requests. However, as mentioned in the previous paragraph, the VNC server would send one update and then quickly throw the vast majority of the requests away since there most likely had been no update to the framebuffer. Now with an empty queue, the VNC server had to wait until the VNC client received the sent update, processed it, displayed it for the user, and then send another barrage of framebuffer update requests that the VNC server received approximately 1 RTT after the last update was sent.

We note that after a longer duration of time, framebuffer update requests became spread out more evenly, which caused higher, smoother FPS performance. Figure 4.8 shows that after a duration of a couple minutes, FPS recovered back to high levels. However we believe it is best not to overload the server in the first place but instead

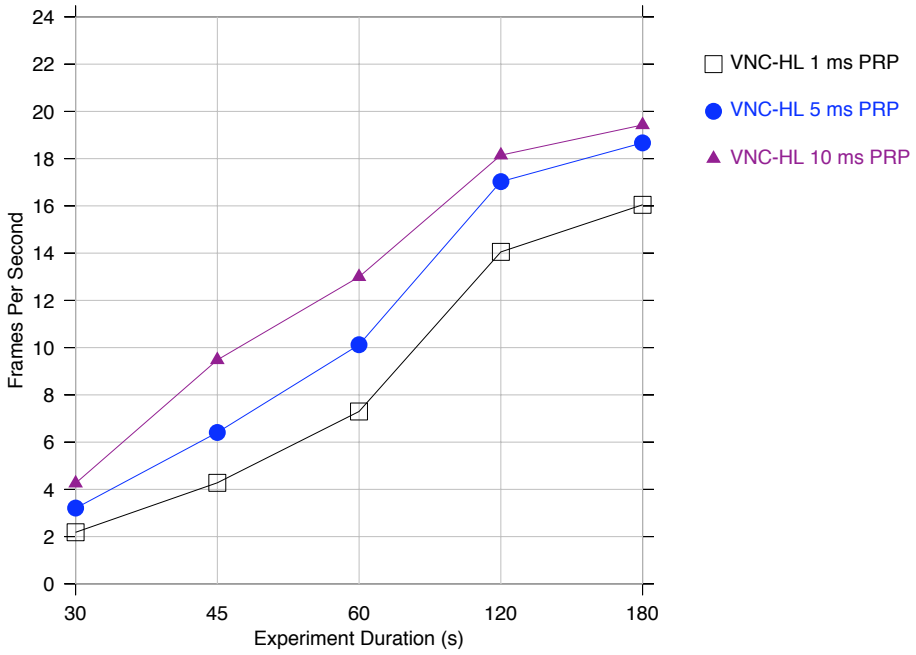

Figure 4.8 FPS performance comparison of experiment durations in 1000 ms latency for inactive screen.

send framebuffer update requests at a rate that is roughly what we expect the server to be able to handle.

4.8 A Good PRP

In an ideal environment, a VNC-HL client would only need to periodically send PRs during the first RTT. Afterwards, each incoming framebuffer update from the server would cause another framebuffer update request to fire off at a good, stable rate, keeping the FPS smooth and high. Because of uncontrollable and unpredictable process scheduling, network congestion, or pure flukes, VNC-HL must continuously request additional framebuffer updates to keep the FPS high. Based on experimental results we believe that a good rule of thumb is to use the desired FPS as the PRP. For example, in our setup the maximum FPS supported by our system in ideal conditions was around 20 - 25 FPS.

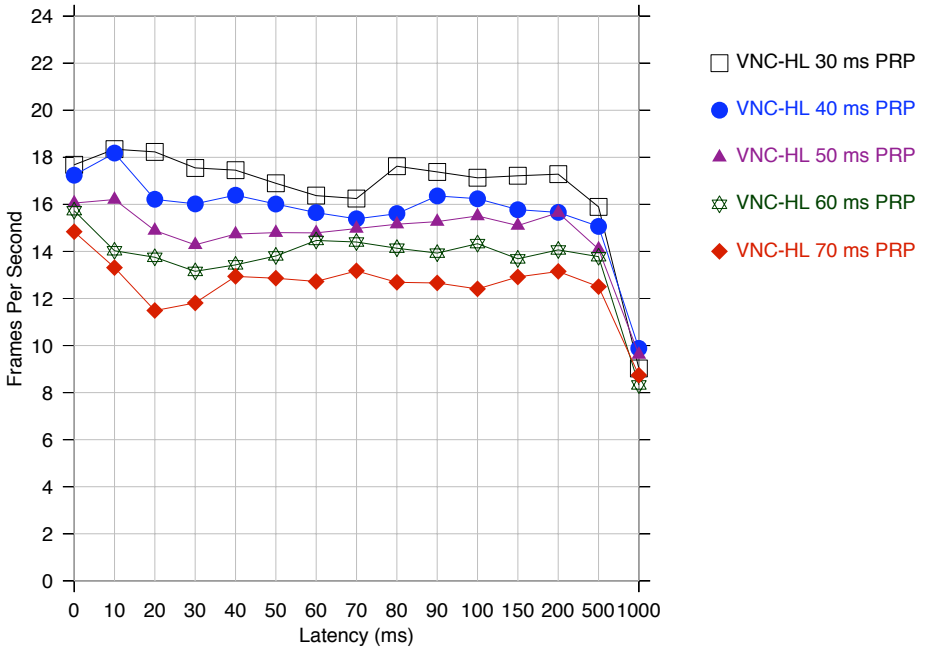

Figure 4.9 FPS performance comparison of VNC-HL PRPs for video

So a client should send around 20 - 25 PRs per second, or in other words a PRP of 40 - 50 ms should be used. Intuitively this makes sense as well since the server is receiving about as many requests as it would be if the client were connected on a 0 ms latency network. Figure 4.9 shows that 40 ms and 50 ms PRPs performed best over all latencies for a video. A 60 ms PRP provided slightly lower FPS throughout most latencies. 30 ms and 70 ms PRPs started to degrade at a 1000 ms latency.

4.9 Chapter Summary

In this chapter we have presented the methodology used to perform the experiments and results of frame rate performance, CPU consumption, and network utilization benchmarks for 2 different real world scenarios. We found that VNC-HL performed up to an order of magnitude higher than VNC in both video and inactive screen test with sufficient PRPs and high latencies. We very nearly reached our 15 FPS goal. We at-

tributed VNC-HL's superior performance to its PR and VNC's weak performance in video tests to its sequential framebuffer processing. CPU consumption and network utilization were very reasonable when considering the increase in FPS performance. Extremely fast PRPs in very high latency actually degraded performance by overloading the server. A good compromise is to send framebuffer update requests at roughly the same rate as the desired FPS.

Chapter 5

Related Work

There is a great breath of related work in the research area of thin-client computing but little work has been done in the realm of high-latency environments. Most other thin clients perform poorly in high latency.

5.1 THINC

Baratto et al. presented a thin-client system that utilized a virtual display interface at the video device driver level [1]. They showed impressive video performance across both LAN and WAN networks when compared to several widely used thin-client systems including *VNC*, *The X Window System*, and *Sun Ray*. The increased performance was attributed to providing a good mapping between high-level application display requests to their low-level protocol and their ability to optimize these translations by aggregating requests and suppressing activity that is off-screen.

THINC does however suffer in high-latency environments. A screen refresh in a sub-millisecond latency network took 0.43 seconds while a 200 ms latency network took 1.67 seconds to do the same screen refresh. These results can partially be attributed to a conservative default 256 kB TCP window. Although we cannot directly compare this to VLC-HL, we note that our system was able to sustain screen refreshes of video

every 0.50 seconds (2 FPS) in a 200 ms latency network with a smaller default TCP window.

5.2 The X Window System

The X Window System is an unusual system in that the *X Window Server* is run on the thin client while the *X Window Client* is run on the server [14]. Graphical applications run the bulk of their computation on the X Window Client. When a display command is required, the X Window Client calls upon the X Window Server to adjust the graphical output appropriately. State must be maintained at both ends of an X connection and if that X connection is interrupted, the application will not be able to recover. Some caching and other optimizations can occur at the X Window Client.

In preliminary tests, we found that the X Window System suffered tremendously in high-latency environments. This was due to the intimate link between the client and server that often uses synchronous function calls that can dramatically hinder performance. *Low Bandwidth X* (LBX) attempted to solve this problem with an intermediary proxy and compression but it never received wide use since its release [4]. Additionally the system requires a heavier weight client with a full-blown windowing system.

5.3 Microsoft Terminal Services

Microsoft Terminal Services is a proprietary thin-client system that offers a vast number of features including a remote desktop interface with file system sharing, sound support, printer support, port redirection, and a shared clipboard [3]. The server pushes updates to the client, depending on the amount of refresh activity. A significant amount of caching and Windows-specific optimizations occurs. However, there are clients available for Mac OS X and UNIX-like operating systems by Microsoft and other third-party projects.

Microsoft Terminal Services currently gives one of the best remote computing

user experiences available with its numerous features. Although some Microsoft Terminal Services clients can run on lightweight devices such as PDAs, the system is primarily designed for normal thick clients running Microsoft Windows. Another disadvantage is the proprietary nature of its protocol.

5.4 Wide-Area Thin-Client Computing

Lai and Nieh benchmarked a number of thin-client systems with environments similar to those encountered across the Internet [7]. They came to the conclusion that latency is often the limiting factor of thin-client performance. Guidelines to designing thin-client systems include minimizing synchronization between the server and client, using simple display primitives, and pushing display updates. VNC-HL fully embraces these principles with slight exception to the last. Since VNC, is natively a *client pull* system, VNC-HL attempts to emulate a *server push* system by constantly sending requests for the server to fulfill.

5.5 Web Browsing on Thin Clients

Yang et al. compared web browsing performance over a lossy wireless network between thin clients and PC-based thick clients [19]. It was determined that thin clients were actually faster and more resilient. The main reason for the thick clients' poor performance was due to the numerous HTTP connections opened by the web browsers that would continually be interrupted, causing each TCP connection to timeout and have its window size decrease. Short-lived HTTP connections typically have an initial timeout default of a rather long 3 seconds. In contrast, thin clients require only 1 long-lived TCP connection that has a much more appropriate timeout length since TCP adjusts its timeout based on previous activity. This leads to a much more graceful recovery. This also suggests that VNC-HL should be able to tolerate networks with a lossy endpoint better than traditional PC systems.

5.6 Chapter Summary

In this chapter we described several different thin-client architectures and how they perform in high-latency environments. To the best of our knowledge no thin clients are capable of delivering a similar frame rate performance to VNC-HL's when in high latency.

Chapter 6

Future Work

Our implementation of VNC-HL was a proof of concept that attempted to leverage as much of the existing RealVNC codebase as possible yet provide significant performance gains. This approach prevented large architectural changes and led to certain assumptions for the purposes of our experimental testing. Future work includes the adoption of a true server push communication style, more refined thin-client resource requirement specifications, and a more fault tolerant design.

6.1 Server Push

As mentioned in Section 5.4, a good design principle for thin-client systems is to push updates from server to client only when required. This minimizes the network usage as well as the delay between the actual screen update and when user perceives the update. VNC uses a client pull technique but due to the simplicity of the VNC protocol a server push technique would not be difficult to implement. We would however have to be diligent and ensure VNC clients could turn down the frequency of updates to keep with the thin-client philosophy of allowing the thin client to control the amount of incoming data.

6.2 TCP Window Sizes

In this thesis we used a 32 MB TCP window size for the majority of our experiments. This value was sufficient for our needs but different screen resolutions and latencies could require larger window sizes. It would be ideal to be able to dynamically adjust the window size depending on current needs.

6.3 Fault Tolerance

We want to provide a higher degree of reliability in VNC-HL. With long latency also comes more risk of problems in between two communicating hosts. Since VNC-HL has a stateless client, the temporary loss of a network connection can be easily recovered from. VNC-HL clients could automatically reconnect to the VNC server upon any type of error or chose another VNC server to connect to. These errors include server failure, network failure, and client exceptions. Redundant VNC servers or proxies would need to be developed. A protocol would also be needed to allow the discovery of nearby proxy servers and for the communication with them.

Chapter 7

Conclusion

Thin-client computing has many benefits over traditional PC computing such as lower costs, higher security, ubiquitous access to resources, and easier maintenance. The main disadvantage is that thin clients must be in constant communication with a server. High-latency environments render most popular thin clients on the market practically unusable. VNC's web browser video performance drops to 3 FPS at just a 50 ms latency. By employing frequent pre-requests for the framebuffer, VNC-HL can display 14 FPS all the way to 500 ms latencies, an order of magnitude better than VNC.

There are several contributions we have provided in this thesis. We profiled VNC in high-latency environments and explained its behavior. We introduced VNC-HL and demonstrated how its PR dramatically increased performance with reasonable CPU consumption and efficient network utilization. We attributed the different FPS performances of video and inactive screens to processing time at the server and client. We found that extremely fast PRPs will yield poor performance due to server overloading. We showed that our thesis statement is correct, that VNC-HL can sustain high FPS performance in high-latency environments with a PR rate of the desired FPS rate.

We conclude with an analogy [13]. The future of computing consists of huge and powerful cloud computers in our networks. Connectivity will be plentiful and provide high bandwidth. A computing device that thrives in this kind of environment is a thin

client, not a PC. A thin client is like a wallet and a PC is like a Swiss Army knife. A wallet allows you to function effectively in a civilized area, providing many times the resources than it is physically, intrinsically worth. A Swiss Army knife is the ultimate survival tool for when you have nothing else with you. It has numerous useful functions, but none that perform nearly as well as dedicated tools. We live in a world that is transforming into a place where a Swiss Army knife of a computer is of little use and where a thin client can grant us access to state of the art resources, but possibly over a long communications link. In this thesis we have filled that long pipeline with useful communications data.

References

[1] Baratto, R. A., Kim, L. N., and Nieh, J., 2005: Thinc: a virtual display architecture for thin-client computing. In *SOSP '05: Proceedings of the Twentieth ACM Symposium on Operating Systems Principles*, 277–290. ACM, New York, NY, USA. ISBN 1-59593-079-5. doi:http://doi.acm.org/10.1145/1095810.1095837.

[2] Burleigh, S., Hooke, A., Torgerson, L., Fall, K., Cerf, V., Durst, B., Scott, K., and Weiss, H., 2003: Delay-tolerant networking: an approach to interplanetary Internet. *Communications Magazine, IEEE*, **41**(6), 128–136.

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

[4] Fulton, J., and Kantarjiev, C. K., 1993: An update on low bandwidth x (lbx). *X Resource*, (5), 251–266. ISSN 1058-5591.

[5] Guy, T. V., 2008: Canonicals ubuntu linux tops 8 million users. http://www.thevarguy.com/2008/09/04/canonicals-ubuntu-linux-tops-8-million-users/.

[6] HP, 2008: Thin clients - comparison results small & medium business - hp. http://h10010.www1.hp.com/wwpc/us/en/sm/WF04a/12454-12454-321959-338927-89307.html.

[7] Lai, A., and Nieh, J., 2002: Limits of wide-area thin-client computing. In *SIGMETRICS '02: Proceedings of the 2002 ACM SIGMETRICS International Conference on Measurement and Modeling of Computer Systems*, 228–239. ACM, New York, NY, USA. ISBN 1-58113-531-9. doi:http://doi.acm.org/10.1145/511334.511363.

[8] Oppenheimer, P., 2004: *Top-down Network Design*. Cisco Press.

[9] Patterson, D. A., 2004: Latency lags bandwidth. *Commun. ACM*, **47**(10), 71–75. ISSN 0001-0782. doi:http://doi.acm.org/10.1145/1022594.1022596.

[10] RealVNC, 2008: Realvnc. http://www.realvnc.com/.

[11] Rhee, I., and Xu, L., 2005: CUBIC: A New TCP-Friendly High-Speed TCP Variant. *PFLDNet05*.

[12] Richardson, T., Stafford-Fraser, Q., Wood, K. R., and Hopper, A., 1998: Virtual network computing. *IEEE Internet Computing*, **2**(1), 33–38. ISSN 1089-7801. doi:http://dx.doi.org/10.1109/4236.656066.

[13] Satyanarayanan, M., Gilbert, B., Tolia, N., Lagar-Cavilla, H., Surie, A., Nath, P., Wolbach, A., Harkes, J., Toups, M., Kozuch, M., et al., 2006: To Carry or To Find? Footloose on the Internet with a zero-pound laptop. Technical report, Tech Report CMU-CS-06-158, Carnegie Mellon University, School of Computer Science.

[14] Scheifler, R. W., and Gettys, J., 1986: The x window system. *ACM Trans. Graph.*, **5**(2), 79–109. ISSN 0730-0301. doi:http://doi.acm.org/10.1145/22949.24053.

[15] Shizuki, B., Nakasu, M., and Tanaka, J., 2002: VNC-based Access to Remote Computers from Cellular Phones. In *CSN '02: Proceedings of the IASTED International Conference on Communication Systems and Networks*, 74–79.

[16] Stevens, W. R., and Wright, G. R., 1994: *TCP/IP Illustrated: The Implementation*. Addison-Wesley.

[17] TightVNC, 2008: Tightvnc. http://www.tightvnc.com/.

[18] UltraVNC, 2008: Ultravnc. http://www.uvnc.com/.

[19] Yang, S. J., Nieh, J., Krishnappa, S., Mohla, A., and Sajjadpour, M., 2003: Web browsing performance of wireless thin-client computing. In *WWW '03: Proceedings of the 12th International Conference on World Wide Web*, 68–79. ACM, New York, NY, USA. ISBN 1-58113-680-3. doi:http://doi.acm.org/10.1145/775152.775163.