VNC in High-Latency Environments and Techniques for Improvement

Taurin Tan-atichat
Dept. of Computer Science and Engineering
University of California, San Diego
La Jolla, CA, USA
ttanatic@cs.ucsd.edu

Joseph Pasquale
Dept. of Computer Science and Engineering
University of California, San Diego
La Jolla, CA, USA
pasquale@cs.ucsd.edu

***Abstract*—VNC (Virtual Network Computing), a form of thin-client computing, offers many advantages over traditional desktop computing including lower costs, higher security, ubiquitous access to resources, and easier maintenance. In this paper, we focus on the problems of using VNC to display video over a high-latency network, and how to solve them. VNC performance is inherently capped at 1 frame per round-trip time (between the client and server) due to its client-pull communication style. We describe *VNC-HL*, our extension of VNC, which improves frame rate performance by employing 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 over a network with 500 ms of latency.**

I. Introduction

Thin-client computing has grown in popularity due to a number of benefits it provides over traditional desktop 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.

One of the more popular thin-client systems is Virtual Network Computing, or VNC [1]. 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.

The VNC system is broken up into two separate components, the VNC client (also known as the VNC viewer) and the VNC server, as shown in Fig. 1. 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.

Key to VNC's operation is its *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.

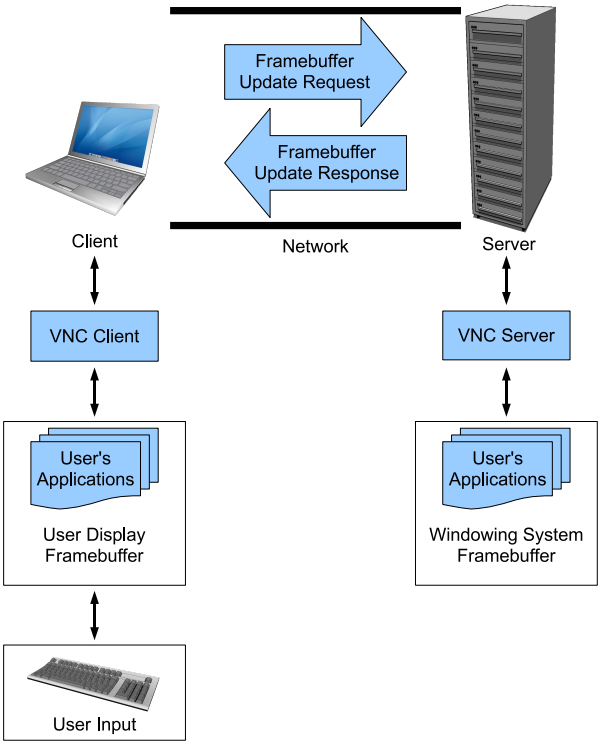

Fig. 1. VNC system overview.

In a high-latency environment this pull technology suffers a significant performance penalty by having to wait at least one full round-trip time (RTT). This implies that VNC's frame rate has an upper bound of $\frac{1}{RTT}$ frames per second (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.

In this paper, we introduce VNC-HL, our solution to improve frame rate performance, and provide an evaluation of VNC-HL in high-latency environments with respect to frame rate performance. Our approach is to extend VNC's simple

design and protocol, allowing us to 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, and Linux. VNC also supports a wide variety of additional platforms including personal digital assistants (PDAs) and mobile phones [2]. Additionally, no modifications to existing applications are required. For these reasons, and given an already large base of users, VNC is worthy of investigation and improvement.

The remainder of this paper is organized as follows: Section II describes the design and implementation of VNC-HL. Section III contains experimental results that show VNC-HL delivers performance up to an order of magnitude better than VNC in high-latency environments and discusses the reasons for it. Section IV describes related work. Finally, we conclude in Section V.

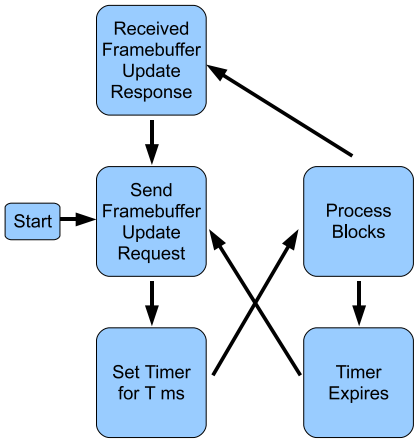

Fig. 2. VNC-HL pre-request diagram with a PRP of T ms.

II. Design and Implementation

We 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 unchanged. 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. We do not address the problem of user input latency, as it would require significant structural changes to the VNC protocol and is beyond the scope of this work.

There are several available versions of VNC such as RealVNC, TightVNC, and UltraVNC [3]–[5]. 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.

A. Pre-Requesting

To provide good VNC performance given high latencies, one must ensure that a framebuffer be transferred from the server to the client soon after any update occurrence. Since VNC is a client-pull system, it 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 above, a traditional VNC client can only have up to one framebuffer update request outstanding. In a high-latency network, this results in a long delay from the time of the client's request until the time the client has fully received the server's response. One can improve the client by modifying it to make multiple requests before receiving a response, carefully considering that 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).

In Fig. 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 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. The timer guarantees that a framebuffer update request is sent with some period T, in addition to regular requests. However, sending framebuffer update requests solely based on the timer does not work well, yielding worse performance in our experiments, and we discuss reasons for not 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 while using 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 and the time a client receives and displays the update. We discuss tuning the PRP parameter in Sections III-D and III-E.

B. 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.

III. 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.

A. Methodology

Experimental tests were selected to be representative of real world conditions with a practical application. The two main parameters tested were latency (RTT) and the VNC-HL pre-request period (PRP). Because of these two dimensions of variability we have opted to display many of the results with contour maps. Each test suite consisted of hundreds of individual tests that were each allocated a testing time period of 1 minute for capturing results that were normally in the 15 FPS range. Each individual test was conducted at least 5 times and then averaged. The repeated test results had very little variation among each other: we typically measured standard deviations of 0.13 (less than 1%) for means in the 15 FPS range.

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. The inherent latency is negligible and so our discussion of latency is the additional latency above and beyond the minimal latency that exists in our system. Latency was simulated with the *tc* command. The VNC implementation used was the RealVNC 4.1.2 source distribution for UNIX platforms. VNC-HL was based off of the same package.

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 one minute. Realistic usage was simulated with a video playing in a web browser (YouTube). We chose this scenario because it is a common computing use case that requires a high FPS rate. The data collected included the number of framebuffers received and the number of framebuffer requests sent. Parameters that were varied included the VNC-HL PRP and the latency between the two computers.

B. FPS Performance

A high FPS rate is crucial for smooth video playback. Fig. 3 shows a contour map of the FPS of VNC-HL across all PRPs and network latencies tested. VNC-HL achieved 15 FPS or higher nearly all the way across the board of latencies for PRPs of 50 ms or faster.

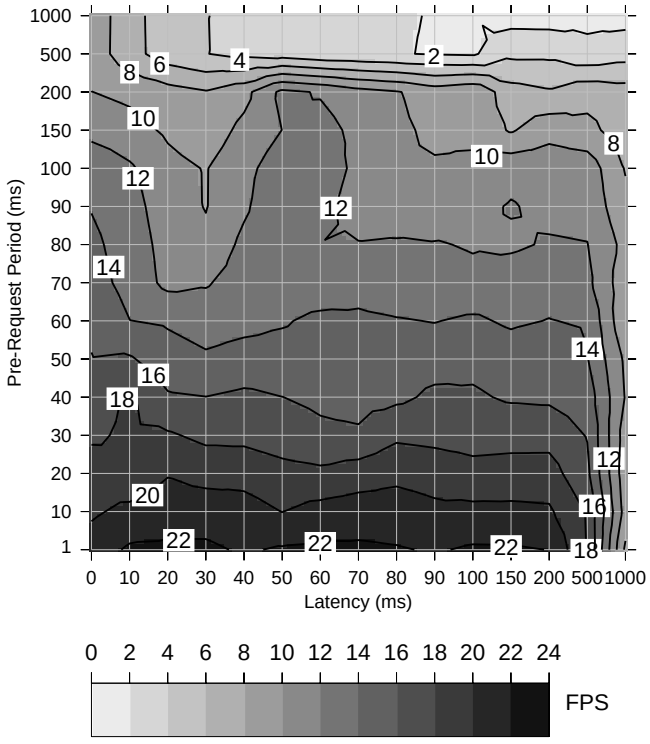

Fig. 3. Contour map of VNC-HL FPS.

A comparison of VNC to a select number of VNC-HL PRPs is shown in Fig. 4. VNC-HL significantly outperformed VNC in all latencies for all PRPs below 500 ms. For PRPs beyond 500 ms, VNC-HL behaved the same as VNC, so very little/no performance improvement occurred at 500+ ms PRPs. VNC-HL behaved the same because its timers would never expire, never causing pre-requests. Framebuffer update responses would arrive approximately every RTT, each time resetting the timer to another value that would not expire before the next framebuffer update response.

A PRP of 50 ms or faster yielded an order of magnitude of improvement in latencies beyond 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 the FPS with a 1 ms PRP in a 1000 ms latency and is discussed in Section III-C.

C. 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. Table I summarizes 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. The FPS suddenly dropped when using a 1 ms PRP. We

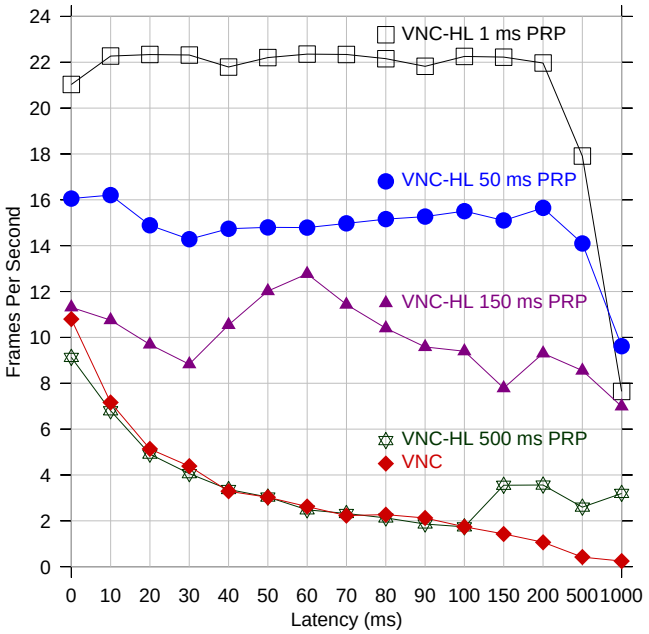

Fig. 4. FPS performance comparison of VNC and VNC-HL.

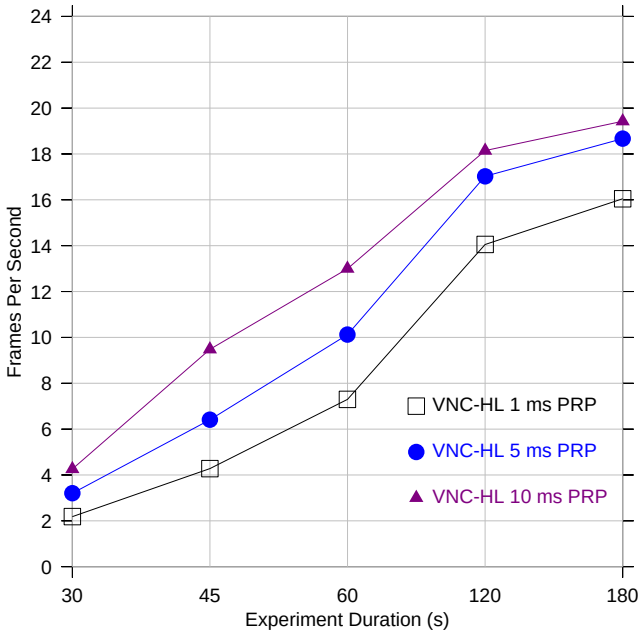

Fig. 5. FPS performance comparison of experiment durations in a 1000 ms latency for inactive screen.

TABLE I
VNC-HL PRE-REQUESTS PER SECOND (PRPS).

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

investigated this and found the cause to be an overloading of the server that is discussed in the next section.

D. 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 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 above, 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. Fig. 5 shows that after a duration of a couple minutes, the FPS recovered back to high levels. Note that this experiment was done on an inactive screen to highlight the drastic change in FPS over time. It is best not to overload the server in the first place but instead send framebuffer update requests at a rate that is roughly what we expect the server to be able to handle.

E. 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 other factors, 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 a PRP based on the desired FPS. For example, in our setup the maximum FPS supported by our system in

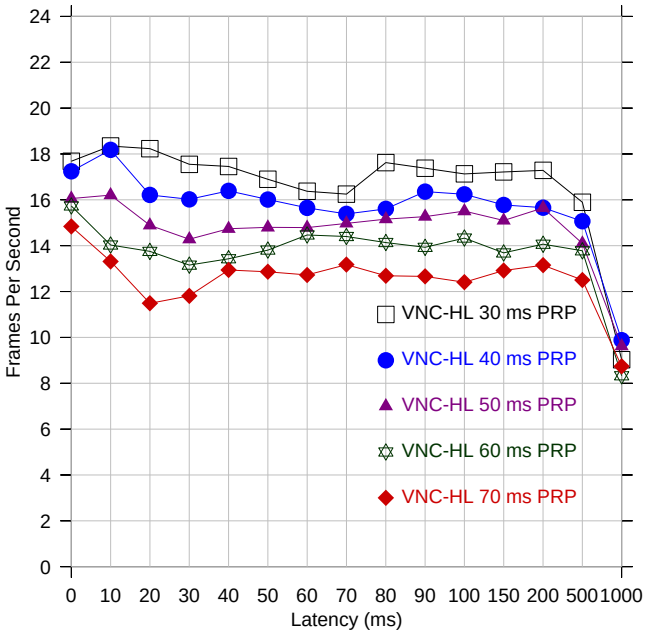

Fig. 6. FPS performance comparison of VNC-HL PRPs.

ideal conditions was around 20 - 25 FPS. 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 in a 0 ms latency network. Fig. 6 shows that 40 ms and 50 ms PRPs performed best over all latencies for a video; a PRP of 30 ms had degraded FPS at 1000 ms latency.

IV. Related Work

While much work has been done in the research area of thin-client computing, relatively little work focuses on high-latency environments. Most other thin clients perform poorly in high latency.

Baratto, Kim, and Nieh presented THINC, a thin-client system that utilized a virtual display interface at the video device driver level [6]. 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.

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.

V. 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 when it experiences just 50 ms of latency. By employing frequent pre-requests for the framebuffer, VNC-HL can display 14 FPS with latencies of all the way up to 500 ms; this frame rate is an order of magnitude better than unmodified VNC.

Our contributions are as follows. We profiled VNC in high-latency environments and explained its behavior. We introduced VNC-HL and demonstrated how its PR dramatically increased performance. We found that extremely fast PRPs will yield poor performance due to server overloading. Finally, we showed that VNC-HL can sustain high FPS performance in high-latency environments with a PR rate of the desired FPS rate.

References

[1] T. Richardson, Q. Stafford-Fraser, K. R. Wood, and A. Hopper, "Virtual network computing," *IEEE Internet Computing*, vol. 2, no. 1, pp. 33–38, 1998.

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

[3] RealVNC, "Realvnc," http://www.realvnc.com/, July 2008.

[4] TightVNC, "Tightvnc," http://www.tightvnc.com/, July 2008.

[5] UltraVNC, "Ultravnc," http://www.uvnc.com/, July 2008.

[6] R. A. Baratto, L. N. Kim, and J. Nieh, "Thinc: a virtual display architecture for thin-client computing," in *SOSP '05: Proceedings of the Twentieth ACM Symposium on Operating Systems Principles*. New York, NY, USA: ACM, 2005, pp. 277–290.

[7] A. Lai and J. Nieh, "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*. New York, NY, USA: ACM, 2002, pp. 228–239.