ADAPTATION FRAMEWORK FOR WIRELESS THIN-CLIENT COMPUTING

By

MOHAMMAD AL-TURKISTANY

A DISSERTATION PRESENTED TO THE GRADUATE SCHOOL
OF THE UNIVERSITY OF FLORIDA IN PARTIAL FULFILLMENT
OF THE REQUIREMENTS FOR THE DEGREE OF
DOCTOR OF PHILOSOPHY

UNIVERSITY OF FLORIDA

2006
Copyright 2006

by

Mohammad Al-Turkistany
To my mother, and my lovely wife Mona
ACKNOWLEDGMENTS

First and foremost, my thanks and deepest gratitude go to Allah, God Almighty, for enabling me and sustaining me during my doctoral studies at the University of Florida.

My deep appreciation goes to my advisor, Dr. Abdelsalam Helal, who provided unlimited assistance and excellent mentoring during my graduate studies at the University of Florida. His strong support was essential in order that I could complete my research.

Also, I would like to express my gratitude to Dr. Shigang Chen, Dr. Janise McNair, Dr. Tan Wong, and Dr. Ye Xia for serving on my supervisory committee and for their invaluable comments and suggestions.

Last, but not least, I would like to thank my mother, Maymuna, for her moral support and her continuous prayers. Also, very special thanks go to my lovely wife, Mona, for sharing this journey with me and for raising our joyful children, Alaa, Abdullah, Maryam, and Razan.
# TABLE OF CONTENTS

ACKNOWLEDGMENTS ........................................................................................................ iv

LIST OF TABLES............................................................................................................ viii

LIST OF FIGURES ......................................................................................................... ix

ABSTRACT ...................................................................................................................... xi

CHAPTER

1 INTRODUCTION ....................................................................................................... 1
   Motivation .................................................................................................................. 3
      Resource Variability in Wireless Environment .................................................... 3
      Mobility and Performance Adaptability ............................................................. 4
      Organization of the Dissertation ...................................................................... 5

2 RELATED WORK ................................................................................................... 7
   Transcoding of Multimedia Web Objects ............................................................... 7
   Performance Comparison of Thin-client Systems .................................................... 8
   Limits of Thin-client Computing in Wide Area Networks ......................................... 9
      Latency vs. Bandwidth Efficiency ..................................................................... 10
      Eager vs. Lazy Screen Updates ..................................................................... 10
      Server-push vs. Client-pull Model .................................................................. 11
   Wireless Web Access Performance of Thin-clients .................................................. 12
      TCP Connections Usage .............................................................................. 12
      Thin-client Display Updating ........................................................................ 13
   Thin-clients Optimization for Wireless Active-media Applications ....................... 13
   Internet Suspend Resume .................................................................................... 15
   Fuzzy Rule-based Control Systems .................................................................. 16

3 THIN-CLIENT COMPUTING MODEL ................................................................ 19
   Thin-client System Architecture ........................................................................ 20
   Advantages of Thin-client Computing ................................................................ 20
   Support for Mobile Computing .......................................................................... 20
   Central Application Maintenance and Data Management .................................... 21
Fuzzification Module.................................................................50
Fuzzy Rule Base ...........................................................................51
Fuzzy Inference Engine...............................................................52
Defuzzification Module.............................................................53

7 EXPERIMENTAL EVALUATION.................................................55

Experimental Setup..................................................................55
Fuzzy Controller Tuning............................................................60
Fuzzy Fluctuation Effect............................................................62
Fuzzy Rule Base Reduction.........................................................62
Adaptive Performance under Variable Processing Speed...........64
Screen Quality-latency Trade-offs..............................................66
Size-based Distortion Optimization..........................................68

8 SUMMARY AND FUTURE WORK .............................................69

Summary....................................................................................69
Future Work..............................................................................70

LIST OF REFERENCES................................................................72

BIOGRAPHICAL SKETCH ..........................................................76
## LIST OF TABLES

<table>
<thead>
<tr>
<th>Table</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>6.1</td>
<td>Fuzzy rule base for the adaptation mechanism</td>
<td>50</td>
</tr>
<tr>
<td>7.1</td>
<td>Reduced fuzzy rule base</td>
<td>63</td>
</tr>
</tbody>
</table>
# LIST OF FIGURES

<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>2.1</td>
<td>ISR client</td>
<td>15</td>
</tr>
<tr>
<td>2.2</td>
<td>Fuzzy controller</td>
<td>16</td>
</tr>
<tr>
<td>2.3</td>
<td>Fuzzy controller’s stages</td>
<td>18</td>
</tr>
<tr>
<td>3.1</td>
<td>Thin-client system architecture</td>
<td>20</td>
</tr>
<tr>
<td>4.1</td>
<td>Virtual network computing components for UNIX platforms</td>
<td>27</td>
</tr>
<tr>
<td>4.2</td>
<td>Normal interaction between the VNC server and a thin-client</td>
<td>28</td>
</tr>
<tr>
<td>5.1</td>
<td>Performance model using three M/M/1 queues</td>
<td>37</td>
</tr>
<tr>
<td>5.2</td>
<td>Service rates in a wireless thin-client system</td>
<td>42</td>
</tr>
<tr>
<td>6.1</td>
<td>Thin-client adaptation framework</td>
<td>47</td>
</tr>
<tr>
<td>6.2</td>
<td>Fuzzy controller for the wireless thin-client system</td>
<td>49</td>
</tr>
<tr>
<td>6.3</td>
<td>Evaluation of fuzzy inference rule</td>
<td>52</td>
</tr>
<tr>
<td>6.4</td>
<td>Center of area method of defuzzification</td>
<td>52</td>
</tr>
<tr>
<td>7.1</td>
<td>GWIC’s decoding time</td>
<td>56</td>
</tr>
<tr>
<td>7.2</td>
<td>Setup for performance evaluation under variable bandwidth</td>
<td>57</td>
</tr>
<tr>
<td>7.3</td>
<td>Compression level control action</td>
<td>58</td>
</tr>
<tr>
<td>7.4</td>
<td>Bit rate control action</td>
<td>59</td>
</tr>
<tr>
<td>7.6</td>
<td>Tuned membership functions for fuzzy variables</td>
<td>61</td>
</tr>
<tr>
<td>7.7</td>
<td>Fuzzy controller’s response to bandwidth decrease</td>
<td>62</td>
</tr>
<tr>
<td>7.8</td>
<td>Effect of reducing the number of fuzzy inference rules</td>
<td>63</td>
</tr>
<tr>
<td>7.9</td>
<td>Setup for performance evaluation under variable processor speed</td>
<td>64</td>
</tr>
</tbody>
</table>
7.10 Adaptive performance under variable CPU frequency .............................................65
7.11 Adaptive performance of tuned controller ..........................................................65
7.12 Effect of the quality factor on compression level control ......................................66
7.13 Experimental relationship between latency and the quality factor .......................67
7.14 Decoding time and rectangle size relationship .....................................................68
This dissertation presents two novel contributions in the area of wireless thin-client computing. The first contribution is a mathematical performance model for a wireless thin-client system. This model identifies the factors that affect the performance of the system. Also, the model helps to analyze and identify adaptation strategies to maintain a certain quality of service. The second contribution is a proxy-based adaptation framework for wireless thin-client systems. This framework adapts dynamically the performance of a wireless thin-client using dynamically discovered contexts. The framework offers application adaptability by employing a fuzzy rule-based engine that adapts dynamically to wireless bandwidth variability and client processing power variability. The fuzzy rule-based inference engine uses context information to trade off among the different qualities of service parameters offered to end-users. The adaptation framework uses a highly scalable wavelet-based image coding technique to provide scalability of quality of service that degrades gracefully. This adaptation framework shields the user from the ill effects of high variability of the wireless network quality and the mobile device’s resources. This
adaptation framework improves the performance of active applications in which the
display changes rather frequently. Active applications behavior may result in high
transmission latency of screen updates. The transmission latency of a large amount of
screen update data adversely affects the user perception of quality of service and results
in poor interactivity and slow screen updates.
CHAPTER 1
INTRODUCTION

Today, wireless networking technologies are in widespread use, and a geographical
hierarchy of wireless network services exists. These wireless services vary greatly in their
coverage area and quality of service. The wireless communication infrastructure coverage
range includes personal-cell, Pico-cell, Micro-cell, Macro-cell, and global cell. The
wireless LAN (IEEE 802.11) standard is an example for Pico-cell wireless
communication technology. The Bluetooth standard is an example of personal-cell
wireless communication technology that provides very short-range wireless connectivity
with a very small power requirement. Wireless cellular networks, such as GSM, GPRS,
EDGE, CDMA, CDMA2000, and WCDMA, are evolving from providing just voice
services to offering users a variety of services, including data, mobile video services, and
Web access. Each wireless network service exhibits different parameters for service
quality offered in its coverage area. For instance, Wireless LAN and cellular WAN
wireless networks have different bandwidth, network latency, loss rate, and usage cost.
Bandwidth may increase by orders of magnitude between different wireless networks.
For instance, CDPD networks offer a bandwidth of 19 kbps while Wireless LAN offers
11 Mbps.

Portable information appliances come in every size and shape. Today, users enjoy a
wide variety of computing devices that promise to satisfy every imaginable need. These
devices include PDAs and handheld computers, smart phones, MP3 players, portables
video players, and light notebooks. Wireless communication networks offer users the
benefits of mobile computing and global connectivity to the Internet. The core standard of the Internet is the TCP/IP protocol, which enables all types of computing devices to communicate, share data, and perform computing tasks. This new paradigm of mobile computing is made possible by the technological innovations in wireless communications and portable computing devices. Mobile computing poses many challenges to the traditional client-server model of computing over the Internet. User mobility requires that applications and data follow the user anywhere. But mobile computing appliances, by their nature, are limited in computing and storage resources. It is therefore essential that they have access to the virtually unlimited computing resources on the network. This emphasizes the fact that the network is the computer.

The importance and utility of the thin-client model of computing are clear. In contrast to the client-server model, the thin-client model puts all computing tasks on the shoulders of a powerful server that resides in the network. The server is responsible for processing the application logic and storing and managing data. The client is responsible only for receiving screen updates and rendering them on its display. This is a very attractive model of computing since it is easy to manage and maintain data and applications on a central server. Also, this model is more secure than workstation-based computing. More importantly, it is attractive for mobile computing because it requires minimal computing resources on the mobile device. In addition, all data and application state information are on the server, and any network disconnection or device failure or loss does not affect the state on the central server. When the client reestablishes network connectivity, the user would be able to resume his tasks from the point at which he stopped.
The thin-client model suffers from its total dependence on the network. For interactive application, network latency and bandwidth limitation may render the system useless due to excessive delays and unresponsiveness to user activity. We present an adaptation framework that dynamically controls latency and adapts performance to the variability of the wireless network quality of service and the variability of the mobile device’s resources. Next, we introduce some of the factors that degrade the performance of the wireless thin-client system.

Motivation

In a wireless networking environment, there are different levels of network connection quality, which are location dependent. This location dependence directly affects the operation and the performance of computer applications. Therefore, an essential need exists for adapting applications behavior to the variability of wireless connection conditions and the device’s resource [13]. It is necessary for communication stack protocols, especially application level protocols, to be able to adapt their behavior according to available wireless connection quality and achieve optimal use of computing resources including available wireless network resources.

Resource Variability in Wireless Environment

Noise and interference in wireless networks play a crucial role in determining error rate and available bandwidth to client applications. There are many sources of wireless noise and interference, for example, multi-path fading, impulse noise, and environmental noise. Wireless interference is often dependent on the location of the mobile host relative to the transmission station, which affects the signal-to-noise ratio (SNR). Consequently, application layer protocols on a mobile client observe variable wireless bandwidth and
packet loss rate. Bandwidth variability represents a major challenge to application-level quality of service control.

A fast-moving client device in a wireless network presents a challenging case to the goal of maintaining a quality of service level. The difficulty stems from the fact that the bandwidth of wireless service offered to a mobile client depends on its location and speed. In a cellular wireless network, a fast-moving client has to deal with many hand-off operations between base stations, which reduce the effective data rate. Therefore, the need arises again for application protocols adaptability to the mobility of client devices.

**Mobility and Performance Adaptability**

The concept of application-level adaptation allows mobile applications to make trade-off decisions to favor certain qualities of service (QoS) over others, using application semantics or knowledge. Such QoS parameters include bandwidth, network latency, error rate, and usage cost, to mention just a few. For a given mobile device with limited resources and for a specific wireless link quality, mobile applications should be able to adapt and manifest different requirements to the underlying system, based on available computing and communication resources. For instance, wireless resource variation could be due to a sudden source of signal interference or an increase in the number of users sharing resources at a wireless cell location.

To enable dynamic adaptation of mobile applications, communication protocols need to discover wireless service parameters and available processing resources. Wireless context discovery allows applications to adjust their behavior to achieve near optimal performance. For instance, a thin-client system server should be able to dynamically change the type or level of compression used to transmit screen updates to remote thin-client over a wireless link. This compression scalability enables the wireless thin-client
system to control the data traffic generated on the wireless link. This is particularly important since thin-client systems generate excessive traffic load on wireless links when compared to the standard client-server computing model. Compression level scalability also allows the thin-client server to control the transmission latency of screen rectangles. Transmission latency is the main factor that degrades the perceived performance of the wireless thin-client system.

Furthermore, usage cost is incentive for optimal utilization of wireless bandwidth since many wireless data service providers charge customers based on the amount of data they transmit and receive over the network. Battery energy constraint of mobile and portable devices is the single most important constraint on wireless mobile computing. It is therefore important to optimize the thin-client usage of battery power by using optimized encoding schemes to send screen updates. It is essential for these encodings to have scalable and low computational complexity as much as possible to control energy consumption. This scalability enables trade-offs between energy consumption and quality of screen updates offered by an encoding scheme.

**Organization of the Dissertation**

In this dissertation, we describe a proxy-based adaptation framework for wireless thin-client systems. Our work builds on the Virtual Network Computing (VNC) thin-client system from AT&T Cambridge Labs [3]. We describe our framework and show how it adapts to dynamically discovered context information. The system uses a fuzzy rule-based inference engine to control the compression of screen updates sent to remote client. We first present the related works in Chapter 2. We then discuss the thin-client computing model in Chapter 3 and the VNC system in Chapter 4. In Chapter 5, we present a wireless thin-client performance model by which we analyze the performance
of the adaptation framework and inference engine. The adaptation framework and its components are given in Chapter 6. Experimental evaluation of the effectiveness of the framework and its adaptive behavior are presented in Chapter 7. Finally, Chapter 8 contains a summary and possible future research.
CHAPTER 2
RELATED WORK

Transcoding of Multimedia Web Objects

The Transend system of the University of California at Berkeley [7] is a proxy-based adaptation system that uses dynamic transcoding of multimedia Web objects. The system employs lossy compression of images to reduce bandwidth usage and to enable low-latency Web surfing on handheld devices such as Palm OS devices. The Transend system demonstrated that on-demand adaptation using transformational (transcoding) proxy is practical and economic. The major outcome of that project is addressing the need to adapt to network and client variations.

Data-specific transcoding enables application-level network resources management by controlling the transcoding level. In theory, Transend’s proxy architecture could support dynamic adaptation to changing network conditions. Berkeley researchers point out that their system has potentially the best performance when utilizing a network connection monitor. They suggest an automatic adaptation mode where a network monitor discovers effective bandwidth and roundtrip latency. However, they did not present an implementation and performance results for the suggested automatic adaptation mode.

Compared to the Transend project, our adaptive system is not limited to Web browsing and it is applicable to any application running on the server. In addition, our approach enables user-transparent adaptation of active media presentations (such as active Flash presentation or Java applet). We use the virtual bandwidth concept, which
represents the combined effect of wireless bandwidth and client processing speed. Based on this information, a fuzzy controller fires its inference rules and decides on the compression level that is needed to achieve target latency and quality of screen updates. Our approach does not need to measure directly the available link bandwidth or the client processing speed.

**Performance Comparison of Thin-client Systems**

A research project, conducted by Lai et al. at the Network Computing Laboratory (NCL) of Columbia University [20], evaluated the performance of several thin-client systems and demonstrated experimentally that bandwidth efficiency-improving techniques, such as screen updates compression, may degrade the overall performance in wide area networks (WAN). This effect is due to computational overhead needed for decoding screen updates at a resource-limited thin-client. The NCL group reports on quantitative measurements that show the impact of WAN latency on thin-client computing performance. The NCL’s experiments demonstrate the feasibility of thin-client computing in the WAN (Internet 2) environment. Their experimental results suggest that optimizing for network latency is more important than optimizing bandwidth usage of the thin-client system in WAN environment.

The NCL group experimentally demonstrated [42] that thin-client systems could provide good performance for Web and multimedia applications in LAN environments. The performance evaluation shows that an eager server-push update policy, such as the one adopted by X protocol, results in better overall performance than the lazy update policy adopted by AT&T’s VNC system for multimedia video applications. In the lazy update model, the server saves bandwidth by discarding intermediate display updates. Optimizing for bandwidth efficiency therefore degrades the performance of multimedia
applications in the LAN environment. The NCL group points to the need for adaptive usage of available bandwidth that balances computational overhead of decoding against possible bandwidth saving. They indicated that client processing time is more important in a high bandwidth environment, and bandwidth efficiency is more important in low bandwidth situations.

The NCL group evaluated the performance of thin-client Web browsing in a wireless LAN environment [41]. The group investigated Web browsing performance of thin-client model of computing under high packet loss rates. The experimental evaluation shows that wireless thin-client Web browsing under a low network quality condition (i.e., high packet loss rate) has a faster response time (less Web page download times) than a local Web browser running on a fat client.

Packet loss rate increases as the wireless network quality deteriorates. The error correction mechanism, which is handled by the TCP protocol in most thin-client systems, has a great impact on Web browsing performance. A thin-client maintains only one connection to the server while a local Web browser on a fat-client often uses many TCP connections. Setting up and maintaining these TCP connections in the presence of packet loss errors introduce excessive overheads and latency compared to thin-client browsing. The group’s experimental results show that thin-clients indeed perform better in wireless Web browsing compared to a local browser on wireless PDA clients because thin-client-based browsing exhibits much lower response time than local browser.

Limits of Thin-client Computing in Wide Area Networks

Columbia University researchers evaluated the performance of thin-client computing in high-latency networks [42]. They suggest that thin-client computing will be widely used in high-bandwidth networks by extrapolating the current trend of wireless
bandwidth growth. They evaluated the performance of different thin-client computing systems in delivering server-based computing over Internet2. They found that using the thin-client model in future high-speed network environments provides acceptable performance. In addition, they found that thin-client systems performance has a great deal of variability. They showed experimentally that network latency is the critical limiting factor that degrades the performance of a thin-client system over Internert2. These researchers reported that improving bandwidth efficiency might lead to overall performance degradation in high-latency networks.

**Latency vs. Bandwidth Efficiency**

The NCL group’s experimental results emphasize the need to consider both minimizing bandwidth and minimizing latency ill effects when designing thin-client systems. They argue that propagation latency of a network is the dominating factor which determines the overall performance of a thin-client system. Sun Ray’s thin-client platform provides good performance because its display encoding makes the right trade-off between computing and communication latencies. In a network with high bandwidth and high network latency, low-complexity encodings provide better overall performance. This is very clear from our performance mathematical model, which will be introduced later in this dissertation.

**Eager vs. Lazy Screen Updates**

In an eager update policy, the occurrence of server window commands triggers encoding and sending display updates to the client. The Sun Ray thin-client system uses an eager update policy. In a lazy update policy, window system commands effects are stored in a frame buffer, and the server keeps track of the regions of the frame buffer that changed since the last update that was sent to the client. Intermediate changes to the
frame buffer are lost, and only the latest changes to the frame buffer are encoded and sent to the client. The server sends screen updates at predetermined intervals. The VNC and Citrix ICA thin-client systems use the lazy update policy. This policy enables thin-clients to minimize bandwidth usage. However, it leads to quality degradation since some intermediate screen updates are discarded.

**Server-push vs. Client-pull Model**

In the server push model, the server sends screen updates without the need for an explicit request from the client. The server decides how often it sends screen updates. In the client-pull model, the client must explicitly ask the server for screen updates, usually after it received and processed the previous screen update. The best example for the client-pull model is the VNC system. Most other thin-client platforms use the server-push model. The experimental performance evaluation indicates that the server-push model can better cope with WAN latencies than the client-pull model because the server does not need to wait for client screen update requests. Consequently, the server push-model significantly reduces the impact of network latency. In contrast, in the client-pull model, the server cannot send next screen updates until it receives an explicit request from the client. Over future Internet protocols and wide area networks, that results in around 70 ms round-trip latency. Regardless of client processing power, the maximum rate at which the server is able to send screen updates is limited by this round-trip latency in the client-pull model. Therefore, the performance of active media applications is severely degraded under the client-pull model and does not allow a full frame rate presentation at the client device. Under the current trend of ever-increasing bandwidth of wireless networks, multimedia applications are poised to have much better performance under the server-push policy than the client-pull one. On the other hand, the client-pull model could still
be beneficial in situations where sharing bandwidth and minimizing network traffic are major objectives.

**Wireless Web Access Performance of Thin-clients**

Wireless networks are characterized by high packet loss rates. This property can adversely affect the performance of wireless Web access on mobile devices. The thin-client computing model has superior performance over traditional fat-client-based Web access when using a lossy wireless network [41]. Thin-clients are faster and cope better with packet loss in wireless LAN. This better performance is due mainly to the simplicity of the thin-client-based Web access. We will discuss the main factors that contribute to the superior performance.

**TCP Connections Usage**

The main reason for the performance difference is the number of TCP connections each model uses. The thin-client-based Web browsing needs only one TCP connection while a fat-client running a local browser may open many TCP connections to view a Web page. Several connections may be needed because a Web page may contain many objects, and each object may need a new TCP connection to a different Web server.

For instance, VNC needs only one TCP connection to the server and uses it to receive all screen updates from the server. In contrast, the fat-client may open many TCP connections while downloading a Web page from a Web server. Under high packet loss rates, the control packets used to establish new TCP connections get lost and must be retransmitted causing long delays due to long time-out periods used for control packets. Therefore, when packet loss rates increase, the fat-client would need to open and maintain many TCP connections. The retransmission delays become longer and severely impact the perceived performance by the user.
**Thin-client Display Updating**

The thin-client model has a fundamental advantage: The client does not need to be concerned with application logic that is running on the server. It only needs to be able to receive screen updates from the server, decode them, and render them on its display. Packets lost over the lossy wireless network do not therefore affect the ongoing Web transactions between the thin-client server and the Web server. It only leads to losing intermediate screen updates, and subsequent screen updates sent to the client would update the client’s display to the correct state.

The VNC server keeps tracking the display frame buffer and sends the most up-to-date screen state to the client only when explicitly requested by the client. So if the client loses intermediate screen updates, then data traffic generated on the wireless network would be reduced. Lost packets retransmissions increase delays between the time when an update request is sent to the server and the time when the screen update is received by the client. This latency reduces the rate at which the client requests display updates from the server. This behavior under increasing packet loss rates leads to displaying Web pages in their final state and skipping intermediate states. Also, it leads to less data traffic on the network due to the reduced rate of sending screen updates to the client. In contrast, in the fat-client approach, increasing packet loss rates would disrupt Web transactions between the fat-client and the Web server, and would lead to degraded performance perceived by the fat-client user due to excessive retransmission latencies.

**Thin-clients Optimization for Wireless Active-media Applications**

Mobile computing devices are generally poor in resources when compared to stationary computers. Thin-client systems enable resource-limited devices to access applications on powerful servers over wireless networks. However, high network latency
could cause severe performance degradation. To mitigate the ill effects of high latency of wireless networks on the performance of thin-client systems, application localization can be used to deal with performance degradation, especially in active media applications [1].

The concept of application localization in thin-client computing can be implemented by transferring some of the application logic to the thin-client. Basically, this process amounts to making the thin-client more complex by forcing it to perform some of the application processing locally on the mobile device. The degree of localization may be varied from a pure thin-client to a fat-client that has a local copy of the application. Application localization is not needed in a fully connected mode (i.e., high bandwidth and low latency). The degree of localization may increase as the network connection quality degrades. Server tasks that could be target for localization include local processing of events generated by the client, such as keyboard and mouse events. This eliminates the latency that occurs when events are sent to the server and their effects on the screen state are sent back to the client as screen updates. Localization can be application-specific or application-transparent. Application-specific localization targets a certain application and localizes some of its logic to the thin-client device. Application-specific localization requires more development effort to localize various applications. Also, it requires that applications present interfaces and information that help to decide the most suitable localization policies for each application. On the other hand, application-transparent localization works for any application running on the server regardless of its functionality. This mode is more appealing since it provides a more generic localization solution and does not require changing existing legacy applications.
Internet Suspend Resume

The Internet Suspend resume (ISR) is a collaboration research project between Intel and CMU [15, 16]. ISR enables seamless mobility of the user’s computing environment from one location to another. The ISR enables mobility without sacrificing the workstation experience in which users enjoy low-latency interactive computing. After the user suspends his computing environment at one place, he would be able to resume at another place. While he is on the way to his new location, the state of his computing environment has migrated from his old location to the new one. These functionalities are made possible because of two well understood technologies: virtual machine technology and distributed file systems. Figure 2.1 shows a virtual machine (VM) that is running inside the Linux operating system. A virtual machine is a software abstraction of real hardware. The VM technology provides the flexibility of running multiple VMs and operating systems on the same hardware. The computing environment’s state includes applications, data files, and current execution state. The ISR technology offers the user a low-latency computing environment because the user works directly on a local interactive machine that runs his computing environment (virtual machine).

<table>
<thead>
<tr>
<th>Application</th>
</tr>
</thead>
<tbody>
<tr>
<td>Guest OS (Win XP)</td>
</tr>
<tr>
<td>Virtual Machine</td>
</tr>
<tr>
<td>Linux</td>
</tr>
<tr>
<td>x86 Hardware</td>
</tr>
</tbody>
</table>

Figure 2.1 ISR client
The ISR employs the Coda distributed file system [14, 31] to store and transfer the virtual machine’s state data files. Coda system can handle huge data files necessary to store VM state. Also, Coda supports data hoarding and reintegration. Coda is a flexible experimental file system. The ISR stores large VM state files as small files in Coda. The RAM state file is stored in a compressed format since it is not always full with data. A kernel module acts as a device driver for the VM virtual disk. Another module redirects I/O requests to the user-space module that manages the mapping and transfer of VM files.

**Fuzzy Rule-based Control Systems**

Figure 2.2 shows a closed-loop feedback control system using a fuzzy controller. The main characteristic of this control system is the use of a fuzzy rule-base and inference engine.

![Fuzzy controller diagram](image)

**Figure 2.2 Fuzzy controller**

Fuzzy control is based on fuzzy logic. Fuzzy logic is an extension of the standard set theory. Membership in a fuzzy set can take any real value between zero and one. Fuzzy control can be viewed as a control system that uses natural language instead of mathematical equations. A fuzzy controller uses experienced expert knowledge to drive controls rules. The fuzzy rule base consists of many fuzzy if-then rules. For instance, the
following statement is a fuzzy rule: If error is Pos and change in error is Zero then output is NM.

Neg, Pos, and Zero are called *linguistic variables*. A rule base expresses a control strategy that is stated in natural language. In Figure 2.2, the process output is compared against a reference value. If these values differ, the controller takes action to eliminate the difference. Since fuzzy controllers are considered non-linear, fuzzy controller stability is an open problem in fuzzy control. There are no definite rules that can be used to study a fuzzy controller’s stability. Stability is achieved when the system is progressively getting close to an equilibrium point.

Each fuzzy rule has condition and conclusion. A possible source from which we can extract the fuzzy rule base is expert knowledge. For instance, we can extract if-then rules from plant operators by extracting from them control actions they take when they operate the plant. Finally, another method is to make the fuzzy controller discover the rules by learning them from an online process and identifying the rules that work.

A fuzzy controller usually uses a preprocessing stage that includes actions such as normalization or scaling of crisp input variables. This stage may involve removing noise, differentiation, and integration of inputs.

Figure 2.3 shows the major components of a fuzzy controller. A fuzzification process converts each input value to different degrees of membership in fuzzy sets. The value of a membership function for each fuzzy set varies from 1-degree membership to 0-degree membership in a gradual way. A fuzzy set is a collection of ordered pairs: the element and degree of membership. Activation of a rule is the deduction of the conclusion. Each fuzzy rule can have a different weight by some factor and called the
degree of confidence. The conclusions of all activated fuzzy rules are combined to form the output fuzzy set using the fuzzy union operation. In the defuzzification process, the resulting fuzzy set is converted to a crisp value that can be applied as a control signal to the process. One of the most popular defuzzification methods is the Center of Gravity method. In post-processing stage, the output value could be scaled or could be amplified by some gain. Fuzzy logic and fuzzy control enjoy widespread use in the industry. Consumer electronic manufacturers are using them in camcorders, washers and dryers, and other products. This is due to the cost-effectiveness of fuzzy controllers. In addition, fuzzy control is easy to learn since it relies a lot on human intuition when making control decisions.

Figure 2.3 Fuzzy controller’s stages
CHAPTER 3
THIN-CLIENT COMPUTING MODEL

Thin-client computing has its origins in the server-based computing model that was popular in the 1960s. In this computing paradigm, a server which used to be called mainframe, has powerful processing capabilities and large storage space. These resources could be accessed from dumb terminals that offer only text display and no graphical user interface. In the 1980s, the workstation computing paradigm dominated because central servers had low reliability and high maintenance costs. Nowadays, server machines are very reliable and cost-effective. Essentially, the thin-client model is an extension of central server-based computing by allowing terminals to present graphical user interface--and not only textual information.

Continuous advances in reliable networking technologies, such as high bandwidth and low-latency computer networks, promise to enable full utilization of the thin-client computing model. These advances improve the performance of thin-clients and allow them to overcome the performance constraints that limit their widespread adoption by users. In addition, the continuing trend of migrating computing services away from end user machines to network servers emphasizes the importance and advantages of thin-client computing. For example, standard desktop applications, such as word processing and spreadsheet applications that used to be run locally on end-user machines, are evolving and are going through transformations to enable offering them as Web services. Alternatively, it is possible to use thin-client computing to provide end-users with a remote computing environment that has all the applications that they may need. This
model supports user mobility since the mobile user needs only a network connectivity
and a trivial client device to access his computing environment from anywhere.

**Thin-client System Architecture**

![Figure 3.1 Thin-client system architecture](image)

The thin-client system consists of a powerful central server and a simple client
(thin-client). The client is responsible only for application presentation to an end-user.
The server is responsible for application processing and data management. The thin-client
communicates with the server over a reliable communication link using a remote display
protocol. This protocol enables the server to update the graphical display of the remote
client device over a communication network. One important feature of the thin-client
model is the tendency to generate high traffic load on the underlying network when
compared to the standard client-server model. This traffic load is proportional to the
characteristics of the thin-client device’s graphical user interface. Specifically, it is
proportional to the resolution and the color depth of the thin-client screen.

**Advantages of Thin-client Computing**

**Support for Mobile Computing**

Generally, an ideal thin-client does not store any state information. This property
greatly facilitates mobile computing since all application’s state information is
maintained at the server. After losing the reliable network connection (such as TCP), the
mobile thin-client needs only to re-establish a new network connection to the server from
its new point of attachment in a foreign network. The user would therefore be able to
resume his computing tasks from the interruption point, and he would observe the last state of his computing environment before the loss of network connectivity. This feature allows thin-clients to tolerate the frequent disconnected mode of computing and the loss of network resources that is common in mobile computing scenarios.

**Central Application Maintenance and Data Management**

Maintaining and upgrading a large number of workstations in a business environment is a complex and time-consuming task. This requires valuable human resource and leads to high operating costs. In contrast, in thin-client systems, software and hardware upgrades are centralized on the server and can be managed very efficiently. This advantage substantially reduces operating and maintenance costs. Consequently, the total cost of ownership is much lower than the cost of operating a group of workstations. In addition, thin-client computing enables making the benefits of hardware and software upgrades immediately available to all users.

**Enhanced Security Model**

The thin-client model of computing offers an enhanced security model when compared to the standard client-server computing model. This is because thin-client machines do not need to store data in local storage devices. Thin-clients send only keyboard and mouse events to the central server to change the application’s state. Therefore, critical data and information are centrally managed, and access is tightly controlled. Furthermore, all security policies and access control are centralized, and the end-user does not play a critical role in setting them. Also, central data management offers the benefits of fault-tolerant backup systems, and it limits the effects of malicious code, such as Internet worms, viruses, Trojan horses, and spying software agents.
**Group Collaboration**

The thin-client model enables application sharing and collaborative group work where all participants view the same computing environment. This collaboration is achieved by multicasting display updates to all participating clients. This multicasting allows remote user collaborations and remote access to shared applications and data. In such an environment, all participants would be able to interact in real-time and contribute to the common task that they are working on.

**Thin-clients Constraints in Mobile Environments**

**Latency in Wireless Networks**

High latency in wireless networks severely affects the performance of the thin-client system. It limits the level of interactivity experienced by users. For instance, if a user is typing a report and the underlying network is characterized by high latency, then the user would experience a very unresponsive system. It would take a long time for keyboard events to get to the server and for the client to receive screen updates. One solution that has been proposed by this research group is localizing some of the keyboard activity during high-latency periods. Solutions to the high-latency problem may require straying away from the true thin-client computing model by requiring the client to store some application state information and perform some application processing.

**Wireless Bandwidth Limitation**

Limited bandwidth wireless networks present great challenges to wireless thin-client computing because wireless bandwidth is scarce and very valuable. Thin-client systems can generate a large amount of traffic load on the network connection between the client and the server. This traffic could be affordable in a wired network but not in a wireless network. Hence, thin-client systems must use the most efficient encoding
schemes to send screen updates to clients. Otherwise, the thin-client model may be useless in low bandwidth wireless networks since the user would experience very high latencies as a result of transmitting a huge amount of screen data over low bandwidth connection. Also, there is usage cost incentive for the optimal utilization of the wireless bandwidth since many wireless data service providers charge customers, based on the amount of data they transmit and receive over their network.

**Energy Limitation of Mobile Devices**

Power limitation of mobile and portable devices is one of the most important constraints of wireless thin-client computing. The thin-client model is based on the idea of updating the client's display by a remote server. Therefore, such system has the potential to consume a lot of network bandwidth compared to standard client-server systems where the client takes care of application processing and communicates with the server to get data. Large amounts of network traffic could overload the mobile device's battery, which is very limited in capacity. Therefore, an essential need exists for optimizing the thin-client’s usage of power by optimizing the encoding schemes used to send screen updates to the client. This optimization can be realized by using screen encodings that highly compress graphical screen data to save wireless transmission and reception power. In addition, these encodings are required to have low computational complexity to minimize processing power usage, especially on resource-poor thin-clients. This may involve trade-offs between computational complexity and compression level of an encoding scheme.

**Mobility and Resource Variability**

Mobility and Quality of Service (QoS) variabilities are important features of a wireless computing environment. The QoS variability is a major challenge to the wireless
thin-client model. The mobility level of a thin-client (how fast it is moving) determines the rate in which the wireless connection quality changes. Therefore, QoS is highly dependent on the location of the mobile client and consequently exhibits a high degree of variability. For instance, the bandwidth observed by the mobile device in a wireless LAN network depends on the distance from the access point. Therefore, depending on the location of the mobile client, available bandwidth may range from 1Mbps to 11 Mbps.

Resources variability calls for dynamic adaptation of thin-client system performance and behavior to achieve optimal use of available wireless resources. Basically, this dynamic adaptation is the core of our research effort regarding wireless thin-client systems. Our main goal is enabling dynamic adaptation of thin-client system behavior to accommodate wireless conditions variability and the client’s resources variability. To achieve dynamic adaptability, the thin-client system needs to sense wireless connection conditions and accordingly adjust its behavior. For example, a thin-client system server would be able to control the compression level used to transmit display updates sent to a remote client.

**Time-sharing Effect on Thin-clients**

The performance of wireless thin-clients may suffer dramatically because of the time-sharing effect that exits in a wireless cell area. In the old paradigm of time-sharing systems, time-sharing ill effects were due to queuing latencies caused by many users sharing the processing power of a central server. Similarly, in a wireless thin-client environment, there is a time-sharing effect, but it is a result of the queuing latency caused by sharing the wireless bandwidth in a cell area. The negative impact of time-sharing is that wireless users will experience excessive latencies and therefore degraded performance.
Bi-directionality of Wireless Traffic

Wireless thin-clients may suffer performance hits because of events traffic being overwhelmed by screen updates traffic from the server. This effect would compound the impact of observed latencies and system unresponsiveness to user events. This problem can be resolved by using a wireless network interface with multiple antennas. Each antenna therefore acts as a separate communication channel, and client events would have a very high probability of being delivered to the server.
CHAPTER 4
VIRTUAL NETWORK COMPUTING THIN-CLIENT SYSTEM

Virtual Network Computing (VNC) is an open-source thin-client system from AT&T [3, 28]. The VNC protocol requires a reliable network connection between the client and the server, such as a TCP connection. The VNC is a platform-independent system and supports Web accessibility using a Java client. This Java client enables mobile computing on different types of devices, such as Web browsers, PDAs, Internet appliances, and cell phones. In addition, the VNC thin-client system supports multiple concurrent accesses of geographically separated users who want to share a common computing environment. The VNC viewer is an ultra thin-client since it does not store any application state information.

The VNC system uses the Remote Frame Buffer (RFB) protocol that enables the VNC server to send frame buffer updates to a remote client. The basic primitive in the RFB protocol is the action of placing a rectangle of screen pixel data at position (x, y) in the client's frame buffer. Each frame buffer update may consist of a number of screen update rectangles. The standard RFB protocol offers poor compression when it handles applications that display complex graphics (e.g., natural images). Consequently, the VNC thin-client system may generate huge amount of traffic on network infrastructure. This dramatically degrades the performance of the wireless thin-client system and makes the user dissatisfied due to excessive transmission latencies. Furthermore, the rate of screen updates requested by the thin-client (operating in a client-pull mode) depends on
connection bandwidth and the client's processing speed. The RFB protocol therefore has some ability to regulate the rate of screen updates according to connection bandwidth.

Other thin-client systems include Citrix MetaFrame and Microsoft's Windows Terminal Server. An important feature of the VNC system, which distinguishes it from other systems, is being an open-source system. This makes VNC a great tool for research and development of thin-client systems. Another important feature is that the VNC protocol works at the frame buffer level, which makes it independent of any windowing system and underlying operating system. Open-source implementations of the server and the client exist for a wide variety of platforms, including Unix platforms, MS Windows, and Apple Mac operating systems.

**VNC System Architecture**

![Figure 4.1 Virtual network computing components for UNIX platforms](image)

An example implementation of the VNC thin-client system for the Unix operating system is shown in Figure 4.1, and it illustrates that it is heavily dependent on the open X Window protocol. As shown in Figure 4.1, X applications view the VNC server as X display. Thus, the VNC server functions as X server so that X clients can display their GUI on it using the X protocol. However, instead of displaying on a physical screen, the VNC server sends the graphical user interface to a remote VNC viewer using VNC's
Remote Frame Buffer (RFB) protocol. Hence, the VNC server has two roles: It acts as the X server to X clients and the RFB server to VNC viewers. The VNC server for the Unix platform is based on the XFree86 code where the hardware-dependent code is replaced by an RFB server code.

**Remote Frame Buffer Protocol**

The RFB protocol enables the VNC server to send frame buffer updates to remote clients. The essential primitive in the RFB protocol is placing a rectangle of pixel data at position \((x, y)\) in the client's frame buffer. Since these rectangles of screen pixels represent graphical information, the RFB protocol supports different types of encodings that are used to send RFB rectangles as efficiently as possible over a network connection. The existence of several supported encodings and the possibility of additional types of encodings offer great flexibility in trading off thin-client system parameters, such as network bandwidth, server processing speed, and client processing speed. Theoretically, a thin-client could negotiate the actual encoding used over a network connection, based on server processing power, client processing power, and the quality of network connection between them. This mechanism is not implemented in the standard VNC thin-client system of AT&T.

![Figure 4.2 Normal interaction between the VNC server and a thin-client](image)
RFB Update Protocol

The frame buffer update action transforms the frame buffer state from one valid state to another. Each frame buffer update may consist of a number of RFB rectangles. Each rectangle may be sent using a different encoding type. The VNC update protocol is a client-pull-based protocol, that is, the server sends frame buffer update only after it receives an explicit update request from the client. The client sends a new update request after it finishes processing and rendering the previous screen update. Upon receiving a request, the server sends the latest valid frame buffer state to the client. The client-pull mode could be helpful in situations that involve slow clients. It regulates the rate at which frame buffer updates are generated by the client. This effect helps with controlling the traffic load on the network and processing load on the client.

Input Protocol

The input protocol is based on a thin-client that has a keyboard and pointing device. The client sends keyboard or pointer events to the server. Then the server relays these events to corresponding applications that use them to change their display state. The application’s state changes cause the frame buffer state to change to reflect the current screen content.

RFB Protocol Encodings

The RFB protocol defines several encoding types to be used in different situations. The built-in encodings in ascending order of computational complexity are copy rectangle, raw encoding, RRE (rise-and-run-length) encoding, CoRRE (Compact RRE) encoding, and Hextile encoding. The RFB protocol could be extended by adding new encoding types. One important feature of the default encoding types is that they are optimized to efficiently compress simple screen areas that have low-complexity graphics
(i.e., large areas of screen that have a single color or few colors). These encodings perform poorly (low compression) when encoding screen rectangles that contain complex graphics or natural images.

**Copy Rectangle Encoding**

Copy rectangle encoding is a simple encoding type used when the server wants to send a screen rectangle that the client already has in its frame buffer. In such a case, the server just needs to send the new position coordinates for the rectangle, which tells the client the new location of a pixel rectangle in the frame buffer. This results in a huge reduction in network traffic. For example, this encoding is very useful when scrolling window content.

**Raw Encoding**

Raw encoding is the simplest encoding type. It is the default encoding type when the server and the client reside on the same machine unless the client requests another one. All VNC clients must therefore support this encoding. When sending a screen rectangle using this encoding, pixels are sent in left-to-right scanline order. Raw encoding does not perform any compression as it simply sends the raw pixel data. It is intended for use when low processing overhead is necessary. Also, raw encoding is used to encode RFB rectangles that do not encode well with other encoding types, such as natural images.

**RRE Encoding**

The rise-and-run-length encoding type is an extension of run-length encoding (RLE) to compress two-dimensional screen graphics data. The advantage of this encoding type is that VNC clients can render it easily since it requires little processing power. The
goal is to avoid slow decompression that adversely affects the interactive performance of the thin-client system. It offers limited compression and targets situations where the VNC client has a low-processing capability. Essentially, RRE is based on partitioning the RFB rectangle into sub-rectangles. Each sub-rectangle consists of pixels of a single color. The RRE encoded rectangle information, which the VNC server sends to clients, contains a background pixel value to represent the most prevalent color in the rectangle followed by a number of single-color sub-rectangles. Each sub-rectangle is defined by color, top-left corner coordinates, width, and height.

**CoRRE Encoding**

The compact run-and-length encoding (CoRRE) is a variation on RRE encoding where the maximum rectangle size is 255x255. Larger rectangles are split into smaller ones. In RRE encoding, the entire rectangle is sent using raw encoding or RRE encoding. In CoRRE, hard-to-encode rectangles (such as image data) are sent using raw encoding while rectangles that encode efficiently are sent using CoRRE. This results in better overall compression because CoRRE encoding offers finer granularity for encoding selection. Decreasing the maximum rectangle size results in better compression. However, there is a trade-off between rectangle size and encoding overhead. Very small rectangles have high encoding overhead, which reduces the overall compression level.

**Hextile Encoding**

Hextile encoding is an improvement of CoRRE encoding. The RFB rectangle is divided into tiles of 16x16 pixels. Tiles that belong to the same RFB rectangle are sent in a predetermined order. They are sent starting with the top-left tile and going in left-to-right, top-to-bottom order. This enables more efficient compression by eliminating the need to send the position and the size of each tile.
Each tile’s encoding type determines whether it is encoded using raw encoding or RRE encoding. Each tile has a background color that represents the most prevalent color. Also, each tile could be split into single-color sub-rectangles. The background color is omitted if it is identical to the background color of the previous tile to save network bandwidth. Hextile encoding is considered good for situations that require relatively high compression, such as low-bandwidth network connection. However, it still has poor compression level when encoding screen rectangle that contains complex graphics or natural image data, which in this case must be sent as raw data. Hextile encoding is the default encoding used between VNC client and server that are running at different hosts.

**VNC Protocol Messages**

We now briefly describe the most important messages in VNC’s RFB protocol. VNC protocol has two main stages; initial handshaking stage followed by normal interaction stage. The initial handshaking consists of the exchange of `ProtocolVersion`, `Authentication`, `ClientInitialisation` and `ServerInitialisation` messages. During the handshaking stage, both the client and server negotiate different session parameters, such as desktop size, color depth, encoding types used, and pixel format. The normal interaction stage consists of the exchange of standard protocol messages. The client usually starts this stage of the protocol by sending `FramebufferUpdateRequest` to which the server replies by sending the `FramebufferUpdate` message.

**Set Encodings Message**

The set encodings message is used by a client to inform the VNC server about RFB encodings that can be supported by the client. It also specifies in what order the client prefers using these encodings to receive screen rectangles data. However, this order is a hint for the server, which can be ignored.
Framebuffer Update Request Message

The FramebufferUpdateRequest message is used by a client to request the latest content of the RFB rectangle in the frame buffer. The message specifies (x, y) coordinates of the screen rectangle and its width and height. The server responds by sending a FramebufferUpdate message. It is possible that the server sends only one FramebufferUpdate in response to several FramebufferUpdateRequest messages from the client. Usually, the client keeps a local copy of the frame buffer areas that it has received previously. Therefore, the server needs to send only incremental updates to the client. This means that the VNC server sends screen pixel data only for screen areas that have changed since the last time the server sent RFB rectangle update. The client can disable this behavior by setting the incremental flag in a FramebufferUpdateRequest message to false. This causes the server to send the entire RFB rectangle content.

Framebuffer Update Message

The FramebufferUpdate message is from the server to the client. It consists of a number of RFB rectangles sent by the server. The client stores them in its frame buffer. Each RFB rectangle could be sent using a different type of encoding, depending on the content of each screen rectangle. The server usually sends this message in response to a FramebufferUpdateRequest message from the client. The VNC protocol does not offer any guarantee about when the client may get a FramebufferUpdate message after sending FramebufferUpdateRequest to the server.

Key Event Message

The KeyEvent message is used by a client to send keyboard events to the VNC server. Specifically, the KeyEvent message informs the server about key press or release event.
**Pointer Event Message**

Similarly, this message is used by a client to send pointing device events to the VNC server. The *PointerEvent* message is used to send pointer movements, and button press or release events to the server.

**VNC Protocol Limitations**

A major limitation of RFB protocol encodings is that they were designed to compress desktop graphical user interface with low complexity graphics. Such computing environment is Microsoft Windows desktop running basic office applications such as Word and Excel. Consequently, RFB protocol encodings offer poor compression when the VNC server handles active media application that contains complex graphics and natural images. Such applications could be Web browser or image-editing application. As a result, the VNC thin-client system can generate a large amount of data traffic over a wireless connection, which is not desirable since wireless bandwidth is usually a very valuable resource. This drastically degrades the performance of the wireless thin-client system and makes the user dissatisfied because of excessive latencies and loss of interactivity.

The RFB protocol does not support effective dynamic adaptation of the wireless thin-client system performance to changing wireless connection conditions. Wireless context variability is a direct consequence of user mobility inside and between wireless cell coverage areas and roaming between different wireless networks. Also, the lack of admission control into a cell area causes competition for resources between users, and it leads to reduction in available bandwidth per user as the number of users increases inside a wireless cell area.
In this dissertation, we propose to adaptively change encoding types or compression level according to client processing capabilities and wireless connection characteristics. The main objective is to provide mobile users with the best possible service quality that can be supported by the wireless network in a user-transparent way. In other words, we want to achieve optimal use of available wireless resources at any given time and location. This idea extends the concept of computing mobility to include service quality mobility. Service quality mobility requires that a mobile user gets a reasonable expectation of quality of service from his computing environment as he moves from one location to another. In this work, we propose an adaptive framework for wireless thin-client systems. This framework enables trading off between different quality of service parameters observed by a wireless thin-client user. This adaptation is done in a user-transparent way using a fuzzy rule-based inference system. The fuzzy inference engine enables controlling the latency observed by the user. It enables latency-screen update quality trade-offs, according to available computing and communication resources.
CHAPTER 5
WIRELESS THIN-CLIENT PERFORMANCE MODEL

In this chapter, we will develop a basic mathematical model, which is based on queuing theory, to understand performance bottlenecks that exist in a wireless thin-client system. This mathematical model allows us to reason numerically and understand the factors that could affect the performance of wireless thin-client systems.

Performance Model Assumptions

First, we lay down the assumptions we made about the wireless thin-client system. We assume that the wireless thin-client system has a very powerful scalable server and can update the display on remote thin-clients. The server runs user’s applications and manages data stored on a reliable network file system. Also, in a wireless computing environment, the thin-client system server is replicated at every wireless access point (or wireless base station). Basically, this replicated server can export the application’s display to remote clients. Server replication at wireless access points allows us to overcome the negative effects of propagation latency. Data packets, which carry screen updates from the server and client’s events travelling between the client and the server, are only charged with propagation latency that is caused by the wireless hop between the thin-client and the server. The essential functions in this approach are the utilization of a distributed network file system to manage user data and replicated computing services at each wireless base station. In this proposed model, the thin-client server could be offered as a service to mobile users, which enables them to access their mobile computing environment.
Performance Mathematical Model

We propose a simple model for the performance of a wireless thin-client system. We focus on three factors that affect the latency observed by the end-user: wireless bandwidth, client processing power, and server processing power.

![Figure 5.1 Performance model using three M/M/1 queues](image)

As shown in Figure 5.1, the system model consists of a simple sequence of three M/M/1 queues. We assume exponentially distributed inter-arrival times and service times for each queue. From queuing theory, the average response time $E(R)$ is the sum of the average waiting time $E(W)$ (queuing time) and the average service time $E(S)$, (i.e., $E(R)=E(W)+E(S)$). Also, as shown in Figure 5.1, the arrival rates at each queue are the same when the system is running under stable conditions (steady-state operation). This means that when the system is stable, the average length of each queue remains the same. Therefore, any traffic entering any queue must leave at the same rate to avoid increasing the queue size.

Our model assumes that the server sends incremental screen updates. Under this assumption, the server needs to send only updates for screen areas that have recently changed. To simplify the model, we assume that server processing power is highly scalable and can be made arbitrarily large compared to thin-client processing power. Therefore, we can ignore the effect of the server part in the latency model. For the wireless communication channel, the average latency (or response time $E(R)$) is
\[
T_d = \frac{1}{B\mu - \lambda} \tag{1}
\]

where \( \lambda \) is arrival rate in rectangles/sec, \( 1/\mu \) is average screen rectangle size in bits/rectangle, and \( B \) is link bandwidth in bps. For the thin-client queue, the average latency (response time) is

\[
T_c = \frac{1}{\alpha D(\alpha) - \lambda} \tag{2}
\]

where \( D(\alpha) \) is the decoding rate in bps, \( 0<\alpha<1 \) is the compression ratio. Therefore, using the queuing theory and treating the system as two separate servers, the average total latency for the wireless thin-client system is the sum of the average response times given by Equation (1) and (2)

\[
T_{total} = T_p + \frac{1}{\mu B - \lambda} + \frac{1}{\mu D(\alpha) - \lambda} \tag{3}
\]

\( T_p \) is the wireless link propagation latency. In Equation (3), the stability condition for the queuing system requires that \( \mu D(\alpha) > \lambda \) and \( \mu B / \alpha > \lambda \). The stability requirement means that the arrival rate at each queue is less than the service rate. Therefore, the utilization factor (the arrival rate divided by the service rate) for each queue is less than 1. This condition prevents an infinite delay of screen rectangles in the thin-client system and guarantees that queue sizes are not growing. In this model, at steady-state (the queues are not growing), the arrival rate is the same at each queue. In addition, the queue with the lower service rate is the bottleneck of performance since the arrival rates are the same for both queues, and the queue with the lower service rate would have a higher utilization factor.
The tuple \((\lambda, \mu)\) represents the application’s screen update characteristics. The screen rectangles arrival rate (generation rate) and average rectangle size are determined by the screen activity characteristics of applications and by user’s activity patterns. For example, a higher level of user activity would increase the screen update rectangle generation rate and consequently increase the arrival rate observed by the wireless thin-client system. Active applications, such as a video player or a 3D graphics application, may generate screen rectangles without user input. Therefore, certain types of application behavior may affect both the average screen rectangle size and the screen rectangles generation (arrival) rate.

Generally, the decoding rate \(D(\alpha)\) is a function of several variables, such as the screen rectangle content, decoding algorithm being used, client’s processing power, and target compression ratio. This function is often non-linear, and it is not easy to model mathematically.

**Operation Mode Examples**

**Wireless Bandwidth Constrained Mode**

Wireless bandwidth constrained mode of operation could occur in practice when a large number of users moves into a wireless LAN coverage area, causing the decrease of effective wireless bandwidth observed by the thin-client device. Therefore, the effective wireless bandwidth observed by the thin-client (bandwidth divided by the compression ratio) may become less than the decoding rate of the thin-client device. Consequently, the service rate of the wireless channel is less than the service rate of the thin-client processor. We conclude from the mathematical model of Equation (3) that the latency
contribution is mainly dominated by the effect of the wireless link bandwidth, and hence it is the performance bottleneck.

**Processing Power-constrained Mode**

A thin-client device may decrease the processor’s speed in response to a battery level alarm. This action may result in forcing the wireless thin-client system to operate under processing power-constrained condition. Therefore, the decoding speed of the thin-client device may become less than the transmission bandwidth of the wireless link. Consequently, the service rate of the client device’s processor is less than the service rate of the wireless channel. Our mathematical model (Equation (3) indicates that the system latency is mainly caused by the constrained decoding rate of the client device and shows that the client processing speed is the performance bottleneck.

**Overloaded Mode of Operation**

The overloaded mode of operation may occur when the system transitions from a stable mode to an unstable mode of operation. For instance, such a situation may occur when an application activity or a user’s activity increases to a level which makes the utilization factor of the wireless channel or the client processor very close to 1 (or greater than 1). This forces one of the queues to operate in an unstable mode where the service rate is not adequate to keep up with the arrival rate of screen update rectangles and the increase in screen rectangle sizes. This mode of operation is undesirable because it leads to a very high latency observed by the user and may require dropping some of the arriving screen update rectangles from the system.
Virtual Bandwidth of Wireless Thin-client

In Equation (3), if \( B/\alpha \gg \lambda \) and \( D(\alpha) \gg \lambda \), which is the case when the system operates in a client-pull mode, then the latency is due only to processing time needed for a screen update rectangle. Hence, Equation (3) becomes

\[
T_{\text{total}} = \frac{1}{\mu} \left( \frac{1}{B/\alpha} + \frac{1}{D(\alpha)} \right) = \frac{1}{\mu} \cdot \frac{1}{BW_{\text{virtual}}} \quad (4)
\]

\[
BW_{\text{virtual}} = \frac{B \cdot D(\alpha)}{\alpha D(\alpha) + B} \quad (5)
\]

The virtual bandwidth, \( BW_{\text{virtual}} \), represents the combined effect of transmission latency and processing latency in the system. The virtual bandwidth is the target of our optimization. By maximizing it, we minimize the system’s total latency. Assuming that the decoding rate function \( D(\alpha) \) is monotonically decreasing over the domain of \( \alpha \) (such that \( D(\alpha) \approx D_0 \) at \( \alpha \approx 0 \)), which is the case for the GWIC wavelet-based decoder [10] used in our wireless thin-client, then \( BW_{\text{virtual}} \leq D_0 \).

Update Quality-Latency Trade-off

Figure 5.2 shows the service rates for the communication channel and the thin-client device assuming a monotonically decreasing decoding rate function. The desirable operating region is on the left of the intersection point in the graph. In that region, the performance bottleneck is the decoding rate while transmission latency is relatively very small. The maximum virtual bandwidth is achievable (best-case latency) when

\( BW_{\text{virtual}} \approx D_0 \) and this happens when \( \alpha \approx 0 \). This condition corresponds to the thin-client’s worst screen update quality and highest compression when using a lossy encoder (e.g., wavelet-based encoder). Our goal is to be able to trade off between \( BW_{\text{virtual}} \) and
screen update quality (i.e., compression ratio $\alpha$). For a lossy wavelet-based encoder, increasing the compression level (i.e., smaller $\alpha$) results in quality deterioration of the thin-client’s screen. We therefore set the target virtual bandwidth according to the quality of service acceptable to the thin-client user. For instance, if we have a screen update quality requirement that dictates a maximum value for target $BW_{virtual}$ (e.g., $BW_{virtual} = 3D_0/4$), then from Equation (5), we get the corresponding value for $\alpha$ (e.g., $\alpha = B/(3D_0)$). Dynamic adaptation is achievable by controlling the compression ratio ($\alpha$) at the server (or the proxy) side using a fuzzy controller that compensates for the thin-client’s processing power and wireless bandwidth fluctuations. For example, if $B/\alpha >> D(\alpha)$ (i.e., client’s processing power is the bottleneck), then we adapt by increasing $\alpha$ until $\alpha \approx B/(3D_0)$. Otherwise, if $B/\alpha << D$ (wireless bandwidth is the bottleneck), then we adapt by decreasing $\alpha$ until $\alpha \approx B/(3D_0)$.

![Service rates in a wireless thin-client system](image.png)
Update Quality-energy Consumption Trade-off

Conceptually, a wireless thin-client device has to have at least a processor, memory, display, and transceiver. Energy consumed by the processor and the network transceiver constitutes a considerable portion of the total consumed energy and could be a target for optimization. This is especially attractive with modern mobile processors that have the ability to dynamically throttle their power consumption by changing the processor's frequency-voltage operating point. Additionally, modern wireless transceivers offer powerful power management functionalities. Those features combined with a highly complexity-scalable wavelet decoder present attractive power management options for wireless thin-clients. Thin-clients can dynamically trade off between the quality of screen updates and power usage. A thin-client device is able to detect available battery energy, and it can accordingly adjust the operating frequency of its processor. The processor’s frequency variations affect the client’s decoding rate $D(\alpha)$. If the processor’s frequency is decreased, then the system should adapt by decreasing the compression ratio (assuming the decoding rate would increase). Consequently, this leads to a lower quality of screen updates observed by the end-user.

The average total energy consumed while processing a single screen rectangle is

$$E_{\text{total}} = \frac{k_c}{\mu D(\alpha)} + \frac{k_d \alpha}{\mu B}$$  \hspace{1cm} (6)

where $k_c$ is the energy cost per unit time for the processor, and $k_d$ is the energy cost per unit time for the transceiver in receiving mode. To derive this equation, we assume that the thin-client system uses a client-pull policy for screen updates. Also, we assume that the wireless transceiver can switch to a low power mode after receiving a screen update.
rectangle. Additionally, a similar assumption applies to the processor after it finishes decoding and rendering a screen rectangle. The transmission energy of the client events is ignored since the size of events data is relatively small compared to data traffic in the other direction from the server to the client. This equation suggests the possibility of trading off between the compression level and energy consumption of the lossy compressor. Increasing the compression level leads to decreased average total energy consumed since both terms in Equation (6) decrease when increasing the compression level. However, that may not be the case for the lossless compressor since the decoding rate decreases as the compression level increases. This result can be explained by observing that a higher level of lossless compression comes at the expense of more compression time (e.g., gzip compression).
CHAPTER 6
FUZZY RULE-BASED ADAPTATION FRAMEWORK

The motivation behind this research is to provide the mobile thin-client user the best quality of screen updates that can be supported by a wireless link in a user-transparent way. In other words, we want to achieve optimal use of available wireless resources at any given time and any given location without user intervention. We focus on one possible optimization, which is to control and minimize the average latency observed by the thin-client user. Other possible optimizations include power optimization and monetary cost optimization of the wireless thin-client.

Proxy-based Adaptation Framework

The objective of the adaptation framework (Figure 6.1) is to achieve minimum total latency of the system by controlling the compression ratio ($\alpha$) of a wavelet-based encoder at the proxy. This goal is achieved by making the virtual bandwidth track a target value ($QD_0$), which is chosen based on the quality of service requirement. Basically, it is a trade-off between total latency and quality of screen updates. Control action takes place in response to dynamic variations in wireless bandwidth and client processing speed. These fluctuations cause the operating point to change and require a new compression ratio to be applied. An error signal (difference between the current virtual bandwidth $BW_{\text{virtual}}$ and the target value) is used to drive a fuzzy controller that outputs a new value for compression ratio ($\alpha$), as shown in Equation (7) where Q is the quality factor.

$$Error = BW_{\text{virtual}} - QD_0$$  \hspace{1cm} (7)
The proposed adaptation proxy for wireless thin-clients is shown in Figure 6.1. The server sends screen update rectangles to the thin-client through an adaptation proxy. The system follows a client-pull protocol where the server sends available screen update only after an explicit client request. The proxy then encodes these update rectangles using wavelet-based coding and sends them to the thin-client over a wireless link. Wavelet-based coding has a superior rate-distortion performance when compared to lossy DCT-based coding systems (such as JPEG standard) since it does not suffer from blocking artifacts at high compression. Also, it is more robust under transmission and decoding errors. Its highly scalable rate control allows the adaptive thin-client system to offer graceful degradation of screen updates quality when trading off different performance parameters. Specifically, wavelet-based coding enables trade-offs between encoded image quality and encoded image size (or the computational complexity of decoding).

**Efficient Resource Discovery**

Figure 6.1 shows the context discovery unit as a subpart of the adaptation proxy. Its function is to discover the context in which the thin-client is operating. This context information may include the characteristics of the wireless network and thin-client resources. Generally, the context discovery unit should be able to elicit the characteristics of the thin-client, such as processing speed, memory size, display size, color depth, and battery energy. In addition, it should be capable of discovering wireless network characteristics, such as bandwidth, latency, and error rate. It feeds collected context information to a fuzzy inference engine that makes decisions by changing control actions on the thin-client system to affect the quality of service offered to the user. Our implementation of this framework targets the case where the thin-client is presenting
applications that have continuously updating active media objects. Examples for such applications could be an animated GIF image file on the Web, Flash Web animation, or Java applet.

![Diagram of Thin-client adaptation framework]

**Figure 6.1 Thin-client adaptation framework**

An important advantage of our approach (using a fuzzy controller) is that it does not need to measure directly the available wireless bandwidth \( B \) or the processing speed of the thin-client device. Bandwidth estimation by itself is an important research area. Bandwidth estimation is costly in many ways. In active estimation, overhead data traffic may need to be injected into the network to have an accurate bandwidth estimation. Also, bandwidth estimation consumes processing power and device memory. Instead, we only need to approximate the virtual bandwidth. Virtual bandwidth represents the combined effect of the wireless link bandwidth, decoding speed of the thin-client, and encoding speed of the proxy. To approximate virtual bandwidth, we measure the time period
between two successive, wavelet-encoded, full screen rectangles sent to the thin-client. This approximates the sum of the transmission latency, the client’s decoding latency, and the proxy’s encoding latency ($T_{\text{total}}$).

**Fuzzy Rule-based Controller**

Fuzzy logic uses imprecise empirical or expert knowledge instead of differential equations to describe dynamic systems. The origins of this paradigm started with simple observations from daily life experiences. We know that a racecar driver does not use mathematical equations while driving to win his races. A racecar driver needs only his accumulated knowledge in the form of approximate reasoning rules. These approximate rules tell the driver what to do in different driving situations to gain an advantage over other drivers in the race.

An important area for applying fuzzy logic is controlling complex and non-linear systems because fuzzy control does not need a mathematical model for the controlled system. Fuzzy control instead employs a fuzzy rule-based inference engine to capture the dynamic behavior of such systems. Another important reason for using a fuzzy controller in our wireless adaptation system is to avoid a direct measurement of available wireless bandwidth and processor operating frequency. Available bandwidth measurement is difficult to implement and an expensive process since a reliable bandwidth estimation consumes processing resources and valuable wireless bandwidth. The fuzzy controller enables us to achieve a high degree of adaptability with minimum overheads.

Table 6.1 shows the fuzzy rule base used to control latency in our wireless adaptation framework. The input fuzzy variables (to the controller) are the virtual
bandwidth and the rate of change in virtual bandwidth. The output of the fuzzy controller is the compression level which also is a fuzzy variable.

![Fuzzy controller for the wireless thin-client system](image)

**Figure 6.2** Fuzzy controller for the wireless thin-client system

As shown in Figure 6.2, the fuzzy controller consists of the following modules: fuzzification module, rule base, inference engine, and defuzzification module. The reference value is the target virtual bandwidth \((Q \cdot D_0)\), which is compared to the observed virtual bandwidth to produce an error signal that derives the controller. The fuzzy input and output variables have the following fuzzy linguistic states (represented by fuzzy sets):

- Neg_Med: negative medium
- Neg_Small: negative small
- Near_Zero: near zero
- Pos_Small: positive small
- Pos_Med: positive medium
Each fuzzy variable has its range covered by these fuzzy sets. Any measured crisp value in the range of an input variable would have certain membership degrees in these fuzzy sets that represent the linguistic states of fuzzy variables.

### Table 6.1 Fuzzy rule base for the adaptation mechanism

<table>
<thead>
<tr>
<th></th>
<th>Neg_Small</th>
<th>Near_Zero</th>
<th>Pos_Small</th>
</tr>
</thead>
<tbody>
<tr>
<td>Neg_Med</td>
<td>Pos_Med</td>
<td>Pos_Med</td>
<td>Pos_Small</td>
</tr>
<tr>
<td>Neg_Small</td>
<td>Pos_Small</td>
<td>Pos_Small</td>
<td>Near_Zero</td>
</tr>
<tr>
<td>Near_Zero</td>
<td>Pos_Small</td>
<td>Near_Zero</td>
<td>Neg_Small</td>
</tr>
<tr>
<td>Pos_Small</td>
<td>Near_Zero</td>
<td>Neg_Small</td>
<td>Neg_Small</td>
</tr>
<tr>
<td>Pos_Med</td>
<td>Neg_Small</td>
<td>Neg_Med</td>
<td>Neg_Med</td>
</tr>
</tbody>
</table>

### Fuzzification Module

In the fuzzification stage, input variables, which include the virtual bandwidth and the rate of change in virtual bandwidth, are converted to membership values in a predetermined collection of fuzzy sets. The crisp values for these variables are used to calculate membership degrees in the fuzzy sets that represent those fuzzy variables. This stage prepares the observed system’s state variables for further processing by the fuzzy inference engine. Figure 6.3 shows the fuzzification of a crisp value of virtual bandwidth in just one linguistic state. Generally, there could be more than one linguistic state that has a non-zero membership degree for the given crisp input value. The figure shows a crisp input value of virtual bandwidth that has a membership degree of 0.75 in the illustrated linguistic state (fuzzy set).
Fuzzy Rule Base

The fuzzy rule base contains the control actions information that is needed to control the system operation. There are several ways to generate fuzzy inference rules. One way is to extract the rules from an experienced human operator who manually controls the system operation. Another way is to actively discover the rules from experimental data and rule identification using trial-and-error methods. Next, we present the initial fuzzy rule base used to control the latency of our wireless thin-client system:

IF $BW_{virtual}$ is Neg_Med AND $BW_{virtual}$ is Neg_Small THEN compression is Pos_Med

IF $BW_{virtual}$ is Neg_Med AND $BW_{virtual}$ is Near_Zero THEN compression is Pos_Med

IF $BW_{virtual}$ is Neg_Med AND $BW_{virtual}$ is Pos_Small THEN compression is Pos_Small

IF $BW_{virtual}$ is Neg_Small AND $BW_{virtual}$ is Neg_Small THEN compression is Pos_Small

IF $BW_{virtual}$ is Neg_Small AND $BW_{virtual}$ is Near_Zero THEN compression is Pos_Small

IF $BW_{virtual}$ is Neg_Small AND $BW_{virtual}$ is Pos_Small THEN compression is Near_Zero

IF $BW_{virtual}$ is Near_Zero AND $BW_{virtual}$ is Neg_Small THEN compression is Pos_Small

IF $BW_{virtual}$ is Near_Zero AND $BW_{virtual}$ is Near_Zero THEN compression is Near_Zero

IF $BW_{virtual}$ is Near_Zero AND $BW_{virtual}$ is Pos_Small THEN compression is Neg_Small

IF $BW_{virtual}$ is Pos_Small AND $BW_{virtual}$ is Neg_Small THEN compression is Near_Zero

IF $BW_{virtual}$ is Pos_Small AND $BW_{virtual}$ is Near_Zero THEN compression is Neg_Small

IF $BW_{virtual}$ is Pos_Small AND $BW_{virtual}$ is Pos_Small THEN compression is Neg_Small
IF $BW_{virtual}$ is Pos_Med AND $BW_{virtual}$ is Neg_Small THEN compression is Neg_small

IF $BW_{virtual}$ is Pos_Med AND $BW_{virtual}$ is Near_Zero THEN compression is Neg_Med

IF $BW_{virtual}$ is Pos_Med AND $BW_{virtual}$ is Pos_Small THEN compression is Neg_Med

**Fuzzy Inference Engine**

![Diagram of Fuzzy Inference Engine]

Figure 6.3 Evaluation of fuzzy inference rule

![Diagram of Defuzzification]

Figure 6.4 Center of area method of defuzzification

The fuzzy inference engine combines measured input variables with triggered fuzzy rules to infer the appropriate value for the output of the controller (compression level). In our system, the estimated virtual bandwidth and its rate of change fire a number of rules,
depending on measured input values. Figure 6.3 shows the Max-Min Inference method. In this method, all rules that have been activated by the values of measured input variables are evaluated. Each rule evaluation produces a fuzzy set. All resulting fuzzy sets are composed using a fuzzy union operation to produce the fuzzy set that represents the controller’s output (compression level). For instance, Figure 6.3 shows the evaluation of the following inference rule:

\[
\text{IF } bw_{\text{virtual}} \text{ is Near_Zero AND } bw_{\text{virtual}} \text{ is Near_Zero THEN compression is Near_Zero.}
\]

The crisp value of virtual bandwidth has a membership degree of 0.75 while the rate of change in virtual bandwidth has a membership degree of 0.4. Since the two parts of the rule condition are connected by fuzzy AND operation, we take the minimum of (0.4, 0.75) and use it to clip the output fuzzy set Near_Zero (for the compression level). The outcome of the evaluation of this fuzzy rule is therefore the output fuzzy set Near_Zero, which is clipped at a membership degree of 0.4. Next, all clipped fuzzy sets, which resulted from the evaluation of fired inference rules, are composed together using the fuzzy union operation. This is achieved by taking the maximum value of all membership functions over the entire range of the output variable (as shown in Figure 6.4).

**Defuzzification Module**

The conclusions of fired fuzzy inference rules are combined using fuzzy union to produce an overall output fuzzy set. To get a crisp value that represents the controller’s output fuzzy set, the *defuzzification* operation is performed on the fuzzy set. The resulting crisp value is used to control the system and change its performance parameters so that the wireless thin-client system can adapt to disturbances in available resources, such as wireless bandwidth fluctuations and processing power variations. A widely used method
for defuzzification is the center of gravity method (center of area). In this method, a crisp value is calculated so that it divides the area under the membership function graph of an irregular fuzzy set into two equal sub-areas.
CHAPTER 7  
EXPERIMENTAL EVALUATION

Experimental Setup

Figure 7.2 shows the basic experimental test bed, which we used to test and evaluate our wireless adaptation framework. The VNC server runs on a Red Hat Linux version 7 machine made by Dell, and has a Pentium 3, 450 MHz processor with 256 MB RAM memory. The adaptation proxy runs on an identical machine and both machines are connected to each other through 100 Mbps LAN. Also, a Cisco Wireless LAN access point is connected to the local area network, and it supports wireless communication speeds up to 11 Mbps.

The thin client device is HP IPAQ hx4705 PDA that has an XScale processor running at 624 MHz. The IPAQ has 64 MB RAM memory and has a 64k color display of size 480x640. It has a built-in Wireless LAN card that supports speeds up to 11 Mbps. Also, this PDA supports Bluetooth technology. The thin-client runs on the IPAQ PDA as a Java application and communicates with the adaptation proxy using the Wireless LAN access point.

Emulating bandwidth variability is achieved by setting the link bandwidth between the thin-client and the proxy to the desired value using CBQ-based traffic control script [9]. This Linux script is very flexible and effective since it allows us to set the link bandwidth to any value between 10 kbps and 10 Mbps. Obtaining this wide range of bandwidth scalability is not feasible using a stock wireless access point.
We used a commercial utility that scales the processor frequency for XScale processors. The utility is XCPUScalar 2004. It allowed us to change the processor frequency to one of the following values; 104, 208, 312, 416, 520 MHz. This utility enabled us to test and evaluate the performance of the adaptive wireless thin-client under variable processing power conditions.

Figure 7.1 shows the decoding time curve for the GWIC wavelet-based decoder. The thin-client was tested on a host with a Pentium 4, 1.8 GHz and 512 MB RAM. The thin-client’s screen size was 800x600 and color depth was 24 bit/pixel. Decoding time information is important to characterize the behavior of the decoding rate function for the wavelet-based decoder. Figure 7.1 shows that the wavelet-based decoding is computationally expensive. It also shows that the GWIC decoder has limited complexity scalability.

![Figure 7.1 GWIC’s decoding time](image-url)
We used information inferred from Figure 7.1 to guide the design of the adaptation framework. The decoding rate is mainly dependent on the wavelet decoder’s computational complexity. Based on decoding time curve, it is safe to assume that decoding time is a linear function of bit rate over the range we considered in our tests. In addition, we needed to estimate the thin-client’s decoding rate ($D_0$). For this purpose, we measured the virtual bandwidth when $\alpha \approx 0$. This effectively eliminates the transmission latency contribution. We implemented this by sending the first couple of screen updates encoded with very small compression ratio (e.g., $\alpha = 1/126$). Therefore, the decoding rate is $D_0 \approx 1/(\mu T_{total})$, which is used to determine the target virtual bandwidth ($Q \cdot D_0$) for the fuzzy controller.

![Figure 7.2 Setup for performance evaluation under variable bandwidth](image)
Figure 7.3 Compression level control action

Figure 7.3 shows the performance of the adaptation mechanism for two machines. The square-marked curve represents the Dell Pentium 4, 1.8 GHz with 512 MB RAM PC while the triangle-marked curve represents the Dell Pentium 3, 450 MHz with 256 MB RAM PC. The client’s screen size is 800x600 and color depth is 24 bit/pixel (true color). This figure shows two adaptation aspects. First, it shows how the system adapts to changes in link bandwidth by controlling the compression level to maintain target total latency. For instance, a sudden decrease in the link bandwidth causes the fuzzy controller to output a higher compression level. The target latency for the Pentium 4 machine is 1.7 seconds, and it is 3.36 seconds for the Pentium 3 machine. The fuzzy engine increases the compression level to adapt to a decrease in link bandwidth. Second, it shows how the system responds to different thin-client processing speeds. For the fast machine (Pentium 4), the fuzzy controller compresses more (which reduces transmission latency) to keep up
with the fast decoding rate of the thin-client. This action prevents transmission latency from being the performance bottleneck.

We emphasize here an important advantage of our adaptation mechanism. It does not need to measure the real link bandwidth. Bandwidth estimation is costly, difficult to implement, and it introduces overheads. We instead rely on the total latency of the system to estimate the virtual bandwidth ($BW_{virtual}$).

Emulating bandwidth variability is achieved by setting the link bandwidth between the thin-client and the proxy to the desired value using CBQ-based traffic control script [9], available on the Linux operating system. We then observe the compression level outputted by the fuzzy controller at the steady-state condition.

![Figure 7.4 Bit rate control action](image)

Figure 7.4 Bit rate control action
Figure 7.4 shows the adaptation action using bit rate notation to express the amount of compression applied to screen update rectangles (bit rate = $24 \cdot \alpha$). It shows a linear relationship between bit rate and link bandwidth.

**Fuzzy Controller Tuning**

Tuning the fuzzy controller optimizes the performance of the controller by adjusting fuzzy membership functions parameters for each fuzzy variable. This tuning is achieved by designing the shape and the number of fuzzy sets for each fuzzy variable. The tuned membership functions are shown in Figure 7.6. The controller’s output gain factor ($K_a$) has great effect on the performance of compression control. We experimentally evaluated the effect of the output gain factor. Higher values of $K_a$ result in better latency control but lead to more fluctuations in the controller’s output. Figure 7.5 shows the effect of the output gain factor.

![Figure 7.5 The effect of the output gain factor](image)
Figure 7.6 Tuned membership functions for fuzzy variables
Fuzzy Fluctuation Effect

A fuzzy controller’s output is subject to fluctuations, which is a familiar characteristic of all control systems. The controller behavior, when subjected to external context variation, was evaluated. For instance, output fluctuations could be a result of a sudden change in available bandwidth. Similarly, sudden change in the battery’s energy level could trigger sudden change in the thin-clients’ processor frequency. Figure 7.7 shows the response of the controller to an abrupt decrease in bandwidth from 100 kbit/s to 30 kbit/s.

![Chart Title](BW from 100 kbit/s to 30 kbit/s)

Figure 7.7 Fuzzy controller’s response to bandwidth decrease

Fuzzy Rule Base Reduction

We evaluated the effect of tuning the fuzzy rule base on the adaptive performance of the wireless thin-client system. This involves finding the set of fuzzy rules that satisfy acceptable performance criteria and adjusting the control surface. Figure 7.8 shows that
although the 15-rules controller has lower total latency, the 7-rules controller has a relatively similar performance to the 15-rules controller. This is a desirable property when processing power is limited. The small number of rules reduces processing delays at the adaptation proxy. Table 7.1 shows the 7-rules fuzzy inference rule base.

Table 7.1 Reduced fuzzy rule base

<table>
<thead>
<tr>
<th></th>
<th>Neg_Small</th>
<th>Near_Zero</th>
<th>Pos_Small</th>
</tr>
</thead>
<tbody>
<tr>
<td>Neg_Med</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Neg_Small</td>
<td>Pos_Small</td>
<td></td>
<td>Near_Zero</td>
</tr>
<tr>
<td>Near_Zero</td>
<td></td>
<td>Near_Zero</td>
<td></td>
</tr>
<tr>
<td>Pos_Small</td>
<td>Near_Zero</td>
<td></td>
<td>Neg_Small</td>
</tr>
<tr>
<td>Pos_Med</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Figure 7.8 Effect of reducing the number of fuzzy inference rules
Adaptive Performance under Variable Processing Speed

We experimentally evaluated the performance of the adaptation framework under variable processor speeds on HP IPAQ PDA. The nominal frequency for the IPAQ is 624 MHz. We used Xscale CPU frequency Scalar utility software to vary the frequency to the desired value. Figure 7.9 shows the experimental setup.

Figure 7.10 shows that the fuzzy controller responds to a decrease in the CPU’s frequency by increasing the compression level, which leads to a higher decoding rate. This control action leads to maintaining the target total latency by countering the effect of a decreased processor frequency. When the quality factor increases (corresponding to lower total latency), the controller compresses more aggressively to maintain target total virtual bandwidth and therefore target total latency. Figure 7.11 shows the improved compression level control for a highly tuned fuzzy controller.

Figure 7.9 Setup for performance evaluation under variable processor speed
Figure 7.10 Adaptive performance under variable CPU frequency

Figure 7.11 Adaptive performance of tuned controller
Screen Quality-latency Trade-offs

Figure 7.12 shows the effect of the quality factor on the performance of the adaptation framework for thin-clients. Generally, increasing the quality factor results in lower total latency of the system, but leads to a greater distortion of screen updates. The fuzzy controller therefore needs to compress more as the quality factor increases, which is observed in Figure 7.12. In other words, increasing the quality factor ($Q$) results in reducing screen updates quality since the curve in Figure 7.12 shifts up.

![Figure 7.12 Effect of the quality factor on compression level control](image)

Figure 7.12 Effect of the quality factor on compression level control

Figure 7.13 shows the experimental relationship between total latency and the quality factor. This relationship is indeed a linear relation. When the quality factor increases, the adaptive control action causes total latency to decrease. The figure shows that both processing latency and transmission latency are handled by the adaptive system.
in the identical manner because the adaptive system is sensitive only to the latency they cause and does not care about the source of that latency.

![Figure 7.13 Experimental relationship between latency and the quality factor](image)

One of the essential characteristics of our approach is its reliance on the trade-off between total latency and screen rectangles quality (distortion). Heuristics can be used to decide the $Q$ value. The ratio $\frac{\lambda}{\mu}$ represents the activity characteristics of each application, and it represents the average traffic rate generated by the application. We suggest assigning higher $Q$ values for active applications ($Q \propto k \cdot \frac{\lambda}{\mu}$). We estimate $\lambda$ and $\frac{1}{\mu}$ for different levels of screen update activity ($k$ is the distortion tolerance of a given application). A higher quality factor ($Q$) results in lower total latency at the cost of increased distortion of screen updates sent to the remote thin-client.
**Size-based Distortion Optimization**

For small size RFB screen update rectangles, high compression levels may be overkill, and wavelet encoding distortion effects on small screen rectangles are more severe than on large rectangles. Therefore, it is desirable to have a compression level that is adjustable based on screen rectangle size. This optimization would improve the perceived presentation of applications on the client side. A possible approach is given by

\[
compLevel = compLevel \cdot \frac{A_{\text{rect}}}{A_{\text{full}}} + c
\]

This formula states that the compression level is proportional to the ratio between rectangle size and the device’s full display size. The linear relationship in Figure 7.14 suggests that the decoding rate does not change with screen update size. This property should allow the fuzzy controller to have more flexibility in processing all screen rectangle sizes.

---

**Figure 7.14 Decoding time and rectangle size relationship**
Summary

We have proposed a proxy-based adaptation framework for wireless thin-client systems. Also, we have presented a mathematical model that can be used to reason about different factors that affect the performance and effectiveness of wireless thin-client systems. This adaptation framework dynamically adapts the performance of wireless thin-client using dynamic context discovery. This context information is used by a fuzzy rule-based inference engine to optimize wireless resources usage and make trade-offs between different qualities of service parameters offered to end-users (e.g., screen update quality, total latency, bandwidth usage). The system uses a highly scalable wavelet-based image coding technique to provide high scalability of the quality of service that degrades gracefully. This framework shields the user from the ill effects of dynamic variability of wireless and mobile device resources.

The thin-client architecture has been shown to offer a promising utility for mobile computing. By delivering any application through a single, small footprint client (the thin-client) on a mobile device, it is possible to mobilize all applications without the need for building wireless application gateways. To this end, the thin-client model is very promising. However, for certain applications in which the display changes rather frequently, sending display updates frequently and inefficiently could challenge the case for thin-clients and their use in mobile computing environments. Such applications behavior would result in performance penalties and costly connection charges. Also, it
results in a high transmission latency of thin-client’s display updates. The effect of active applications could be further compounded by the wide variability of the wireless network and mobile device resource parameters.

Furthermore, the thin-client computing model is not always optimal in terms of using the wireless link bandwidth between the client and server. This is critical in wireless and mobile computing environments where bandwidth is a very valuable commodity in such environments. Sending thin-client screen data inefficiently would translate to accumulating costs. Thin-client remote display protocols usually do not encode complex graphic screens efficiently. The resulting transmission latency of a large amount of screen data adversely affects the user perception of quality of service. Excessive transmission latency results in poor interactivity and slow screen updates. Our proxy-based thin-client adaptation framework utilizes wavelet-based image coding to enable variable and scalable compression of display rectangles. It offers an application-level adaptability by employing a fuzzy rule-based engine that dynamically adapts to wireless bandwidth and client’s processing power variability.

**Future Work**

We propose some promising ideas that can be investigated for future research. The server-push model of screen updates has been shown to have superior performance for active applications when compared to the client-pull model of screen updates. However, the server-push model has the potential to generate the most traffic on the network infrastructure. For this model, the benefits of our adaptive thin-client system are therefore potentially far greater than its benefits for client-pull systems, such as VNC system. Extending the adaptation system to support thin-client systems that employ the server-push mode of operation is therefore a very promising direction for future research.
The GWIC wavelet-based image coder used in our project has very limited scalability. Also, it has a relatively high computational complexity, and therefore it is slow. It would be preferable to try other wavelet coding systems that have low computational complexity and can trade speed for higher image quality. It would also be beneficial if we could develop wavelet coding that has great computational complexity scalability. We therefore suggest investigating the potential of using other wavelet-based encodings that have excellent computational complexity scalability. This possibility could offer much more scalable trade-off options for adaptive thin-client systems.

Battery power of a thin-client device is a valuable resource. Energy consumption of a wireless thin-client is always a great target for optimization. We suggest building on our mathematical model for wireless thin-clients and developing a more comprehensive mathematical model for wireless thin-client system performance. This model would focus more on the energy consumption aspects of wireless thin-client systems.

Finally, we suggest investigating the potential of using high-compression lossless coders such as bzip2 in our wireless adaptation system. These coders have different trade-off characteristics since they trade off between compression level and coding time--the higher the compression level, the slower the coder. It would therefore be desirable to investigate the possibility of trading coding speed for higher image compression. This property may be useful in situations where we need to trade off between energy consumed by the thin-client’s wireless transceiver and the total latency observed by an end-user of the wireless thin-client system. We plan to investigate complexity scalable lossless coding in resource-limited systems.
LIST OF REFERENCES


Communication 1 (1) (1994).


[17] M. Kozuch, M. Satyanarayanan, T. Bressoud, C. Helfrich, S. Sinnamohideen,
Seamless Mobile Computing on Fixed Infrastructure, IEEE Computer 37 (7)
(2004).

International Conference on Wireless Communications, Calgary, Canada, July
1999.

[19] A. Kuznetsov, CBQ.init Traffic Management Script,

International Conference on Measurement and Modeling of Computer Systems,
Marina del Rey, CA, June 2002.

Web Browsing on Wireless PDAs Using Thin-Client Computing, in: 13th

Healthcare Computing, in: 11th World Conference on Medical Informatics, San


Architectures, Technical Report No. CUCS-022-00, Department of Computer
Science, Columbia University, November 2000.

[26] K. M. Passino, S. Yurkovich, Fuzzy Control, Addison-Wesley Longman, Menlo


Mohammad Al-Turkistany received his Bachelor of Science degree in electrical engineering from King Saud University, Saudi Arabia, in 1995. He was then employed by Umm Al-Qura University, Saudi Arabia. He was awarded full scholarship to pursue graduate study leading to the Ph.D. degree in computer engineering. In 1998, he started his graduate studies in the Computer and Information Science and Engineering Department (CISE) at the University of Florida in Gainesville. He received his Master of Science in computer engineering from the University of Florida in 2003. His research interests include mobile computing, pervasive computing, and network security.