WWW BROWSING USING A TRANSCODING PROXY

By

Abdulbaset A. Gaddah

A thesis submitted to the Faculty of Graduate Studies and Research in partial fulfillment of the requirements for the degree of

Master of Computer Science

Ottawa-Carleton Institute for Computer Science

School of Computer Science

Carleton University

Ottawa, Ontario

December 2000

© Copyright

2000, Abdulbaset A. Gaddah

The undersigned recommend to the Faculty of Graduate Studies and Research
acceptance of the thesis

WWW BROWSING USING A TRANSCODING PROXY

Submitted by **Abdulbaset Gaddah**, High Diploma,
in partial fulfillment of the requirements for the
degree of Master of Computer Science

Dr. Thomas Kunz,
Thesis Co-supervisor

Dr. Roshdy H. M. Hafez,
Thesis Co-supervisor

Dr. Frank Dehne, Director of School of Computer Science

Carleton University

December 2000

ABSTRACT

With advances in wireless networking technology and portable information appliances, a new paradigm of computing, called *mobile* or *nomadic* computing, has generated much attention in recent years. This new platform will make it possible for users who carry portable devices to browse WWW information regardless of their physical location or movement behavior. Such a new environment, however, faces many technical challenges such as limited wireless bandwidth and high error rate. In addition, mobile devices widely differ with respect to display size, color depth, processing power, battery life, and the ability to handle different data formats. Since images usually present the highest rate of WWW traffic, image transcoding could greatly reduce web page retrieval latency and match both the device and channel capabilities. This thesis presents a proxy transcoding process that enhances image transmission to a variety of mobile hosts. The central idea in our work is to place a transcoding proxy between the generic WWW servers and the diversity of client devices in order to adapt to a greatly changing bandwidth on the proxy-client link and to manage the heterogeneity of small-screened mobile devices. The proxy utilizes transcoding functions based on the available bandwidth and the device characteristics to achieve the smallest file size and acceptable quality.

ACKNOWLEDGMENT

I have had the great fortune to be supervised by Dr. Thomas Kunz. Together we have held hands to build an energy that has made this thesis possible. I would like to take this opportunity to express gratitude to him for his practical guidance, continuous support, and helpful contributions. My thanks and appreciations also go to Dr. Roshdy Hafez for his helpful suggestions and valuable ideas in completing my thesis.

I would like to acknowledge the financial support of Nortel Company, National Science and Engineering Research Council (NSERC), and Carleton University. Many thanks go to all members of the Canadian Bureau of International Education (CBIE) especially to the program manager Anna Mastelloto for her generous help and directions.

I am very grateful to Pamela Lawes for her hospitality and assistance in English Language matters. A special thanks goes to all my colleagues for many useful discussions we have had on the subject.

I wish to thank my family, especially my mother, for their support, inspiration, encouragement, and love always.

Above all, I praise Allah (God) who gave me the capability to achieve the objective of my study and made things easier for me.

Most surely in the creation of the heavens and the earth and the alternation of the night and the day, and the ships that run in the sea with that which profits men, and the water that Allah sends down from the cloud, then gives life with it to the earth after its death and spreads in it all (kinds of) animals, and the changing of the winds and the clouds made subservient between the heaven and the earth, there are signs for a people who understand. (Quran 2.164)

TABLE OF CONTENTS

CHAPTER 1: INTRODUCTION 1

1.1 Motivation 1

1.2 Classic Client-Server Model 4

1.3 Thesis Objectives 6

1.4 Thesis Contributions 7

1.5 Thesis Organization 8

CHAPTER 2: THE CHALLENGES OF MOBILE COMPUTING 10

2.1 Wireless Communications Network 10

2.1.1 Limited Bandwidth 11

2.1.2 High Bandwidth Variability 12

2.1.3 High Error Rate 13

2.1.4 Heterogeneity 13

2.1.5 Security Risks 14

2.1.6 Mobile Host Protocols 15

2.1.7 Frequent Disconnections 15

2.1.8 Mobility 16

2.1.9 Mobile Computer Capabilities 16

2.2 WWW Browsing Through a Proxy Server 17

2.2.1 Proxy Based Approach 18

2.2.2 End-to-end Approach 19

2.2.3 Application Partitioning Approach 20

CHAPTER 3: BACKGROUND AND RELATED WORK.....22
3.1 Wireless Networks 22
3.1.1 Wireless Technology Overview.....23
3.1.2 Cellular Network Topology.....26
3.1.3 Web Browsing (WWW).....28
3.2 Related Work 28
3.2.1 Hybrid Network- and Application-level Approaches.....29
3.2.2 Transcoding Proxies 32
3.2.3 Partitioning of Application Complexity.....37
3.3 Summary.....40

CHAPTER 4: EXPERIMENTAL ENVIRONMENT.....43
4.1 Objectives 43
4.2 Image Collection.....44
4.3 An Overview of GIF and JPEG File Formats 47
4.3.1 GIF File Format.....48
4.3.2 JPEG File Format.....51
4.4 Static Image Characteristics 53
4.4.1 Image File Size.....54
4.4.2 Number of Colors.....55
4.4.3 Spatial Geometry Size 57
4.5 Image Transcoding Characteristics.....58
4.5.1 Reducing the Spatial Geometry.....59
4.5.2 Reducing the Number of Unique Colors.....62
4.5.3 Changing the Image Format 64
4.5.4 Changing the JPEG Compression Metric 68
4.6 Key Observations 70
4.6.1 Effect of Imaging Software.....70
4.6.2 Sequence Order of Transcoding Operations.....72

4.7 Summary.....73

CHAPTER 5: N IMAGE TRANSCODING PROXY 75

5.1 Introduction.....75

5.2 Image Transcoding.....77

5.2.1 Transcoding Functions.....78

5.3 System Architecture 82

5.3.1 The Prime Components of the Transcoding Proxy Architecture.....83

5.3.2 Content Analyzer.....85

5.3.3 Input Parameters.....86

5.3.4 Transcoding Scenario.....88

5.3.5 Cache Scenario 90

5.4 The Algorithm of the Transcoding Proxy.....92

5.5 Summary.....101

CHAPTER 6: EXPERIMENTATION AND EVALUATION.....103

6.1 Experimental Setup and Results 103

6.2 Transcoding Performance.....113

6.3 Scalability Concerns.....116

CHAPTER 7: CONCLUSIONS AND FUTURE WORK.....119

7.1 Conclusions 120

7.2 Future Work 122

BIBLIOGRAPHY.....126

LIST OF FIGURES

Figure 1.1: A Classic Client-Server Model 4
Figure 1.2: A System Environment for Transcoding Proxy.....5
Figure 2.1: WWW Browsing Through a Proxy Server.....18
Figure 3.1: Schematic Arrangement of Cells in a Cellular Wireless Network.....26
Figure 3.2: Network Topology of a Cellular Wireless Network.....27
Figure 3.3: TCP Connections for WWW 28
Figure 4.1: File Type and File Size Distributions for Selected WWW Users 45
Figure 4.2: Background Transparency 50
Figure 4.3: GIF and JPEG Image File Size Distribution.....53
Figure 4.4: Distribution of Number of Unique Colors in GIF Images 55
Figure 4.5: Image Spatial Size Distribution – GIF Images 56
Figure 4.6: Image Spatial Size Distribution – JPEG Images 57
Figure 4.7: Reducing The Spatial Geometry by a Factor 2 Along Each Axis.....59
Figure 4.8: Reducing The Spatial Geometry by a Factor 4 Along Each Axis.....60
Figure 4.9: Reducing the Number of Unique Colors in GIF Images 62
Figure 4.10: Converting a TrueColor JPEG Image to a GrayScale JPEG Image.....63
Figure 4.11: Converting From GIF to JPEG Format.....65
Figure 4.12: Converting From GIF (#Colors > 250) to JPEG Format 66
Figure 4.13: Converting From JPEG to GIF Format.....67
Figure 4.14: Changing JPEG Image Quality Factor 68
Figure 4.15: Saving Identical Images with Different Image Processing Software.....71
Figure 5.1: Image Customization 77
Figure 5.2: The Image Transcoding Proxy Architecture.....82

Figure 5.3: Cache Scenario 91
Figure 5.4: The Transcoding Proxy Algorithm (Main Body) 94
Figure 5.5: GIF Transcoder Routine 95
Figure 5.6: GIF to JPEG Conversion Routine 96
Figure 5.7: Color Reduction Routine 97
Figure 5.8: Thumbnail Scaling Routine 98
Figure 5.9: Quality Reduction Routine 99
Figure 5.10: Image Scaling Routine 100
Figure 6.1: The General Experimental Testbed Setup 104
Figure 6.2: Response on a Resource Rich Client 106
Figure 6.3: Response on a Resource Poor Client 106
Figure 6.4: The Delivery Time for the Web Pages with and without Transcoding 108
Figure 6.5: Page Access Time 110
Figure 6.6: Transcoding Time Consumed on a Resource Poor, Medium, and High Client 111
Figure 6.7: The Achieved Compression Ratio 112
Figure 6.8: The Transcoded Image of Soda Hall 112
Figure 6.9: End-to-end Latency for Images with and without Transcoding 116
Figure 6.10: Image Transcoding Latency Versus the Number of Simultaneous Users on a Single Proxy 118

LIST OF TABLES

Table 1.1: Physical Variation Among Clients 2
Table 1.2: Typical Network Variation.....2
Table 2.1: Samples of Wireless and Wired Networks Bandwidths 11
Table 4.1: Image Category Distribution.....58
Table 4.2: Saving Identical Image Using Different Software 70
Table 4.3: Creating Identical Image with Different Image Processing Software 72
Table 4.4: Sequence Order of Transcoding Operations 73
Table 6.1: The Cost of Transcoding Popular Web Sites 107
Table 6.2: The Amount of Data Needs to Be Transmitted After Transcoding.....110
Table 6.3: Transcoding Latency and New Sizes (as Percent of Original) 114

Chapter 1

INTRODUCTION

1.1 Motivation

With the recent developmental activity in wireless networks, mobile devices, and web technologies, the number of people who use mobile computers to access their business information through the web has rapidly increased. Today, users want to have information at their fingertips wherever they go. These users need speedy access to data. It is frustrating for them to wait for hours to download web pages containing multimedia data over a wireless channel. The small portable computers such as palmtops, smart cell phones, or laptops cannot even handle most of the data. Moreover, Internet clients vary in many aspects, including screen size, color depth, processing power, and ability to handle specific data formats, e.g., GIF, PostScript, or MPEG. They often communicate over wireless links, which are characterized by lower bandwidths, higher error rates, higher costs, and more frequent disconnections [25]. As shown in Tables 1.1 and 1.2, each type of variation often spans orders of magnitude. Further, as was discussed in [17,19,21], the

web (more explicitly, the HTTP protocol) is designed to operate in wired, high bandwidth environments, and does not act particularly well when the access point has poor resources. HTTP implementations are usually layered on top of TCP. Both HTTP and TCP suffer from a number of performance problems [90] that render wireless access to the WWW painfully slow. The reasons behind these problems, for example low/variable bandwidth or disconnections [19,10], are quite well known in the mobile environment.

Client Device	**Bandwidth (bps)**	**Memory**	**Display Size**	**Display Color**	**Bits/Pixel**
Typical PDA	14.4K	low/2M	320 x 200	b/w	2
HHC	28.8K	4M	640 x 480	gray	8
Typ. Laptop	56K	16M	800 x 600	RGB	8
Midrange PC	56K	32M	1024 x 768	RGB	16
Workstation	10M	128M	1280 x 1024	RGB	24

Table 1.1: Physical Variation Among Clients

Network	**Bandwidth (bps)**	**Round-Trip Time**
Local Ethernet	10-100 M	0.5 – 2.0 ms
ISDN	128 K	10 – 20 ms
Wireline Modem	14.4 – 56 K	350 ms
Cellular/CDPD	9.6 – 19.2 K	0.1 – 0.5 s

Table 1.2: Typical Network Variation

These problems are further compounded by the current model of web information access, where the user has to navigate the information space to find the required information. Transmitting a lot of useless data across the wireless channel in the browsing process thus wastes valuable bandwidth. A continuous connection is also needed during the information retrieval process. Disconnections frequently occur due to the handoff process or shadowed areas and typically require that the information be fetched again. More generally, the problem is a lack of a continuous mechanism - i.e. not allowing the mobile user to disconnect during the information retrieval. Moreover, web servers have no knowledge about the client's resources and assume that it can handle any data sent by them. For example, a user may retrieve data from the server on a machine that does not have the capability of handling this type of format, and thus will not be able to use it. This results in useless consumption of available bandwidth and waste of time while transferring data across the wireless network. The mobile client also expends battery resources in receiving this data. With the ever-increasing use of different types of multimedia content on the web, this capability mismatch phenomenon represents a growing problem.

1.2 Classic Client-Server Model

In the traditional client-server information system as shown in Figure 1.1, a server is any machine that holds a complete copy of one or more database. The server usually has high bandwidth connectivity. A mobile client is connected to a base station and is able to access data residing on any server via a low bandwidth wireless link. Classic client-server systems assume that the location of client and server hosts is fixed and the bandwidth among them does not fluctuate. As a result, the functionality between client and server is statically partitioned. Under the highly variable computing environment conditions that characterize mobile computing, it is believed that the existing client-server model is not capable of providing adequate support for the mobile wireless computing environment.

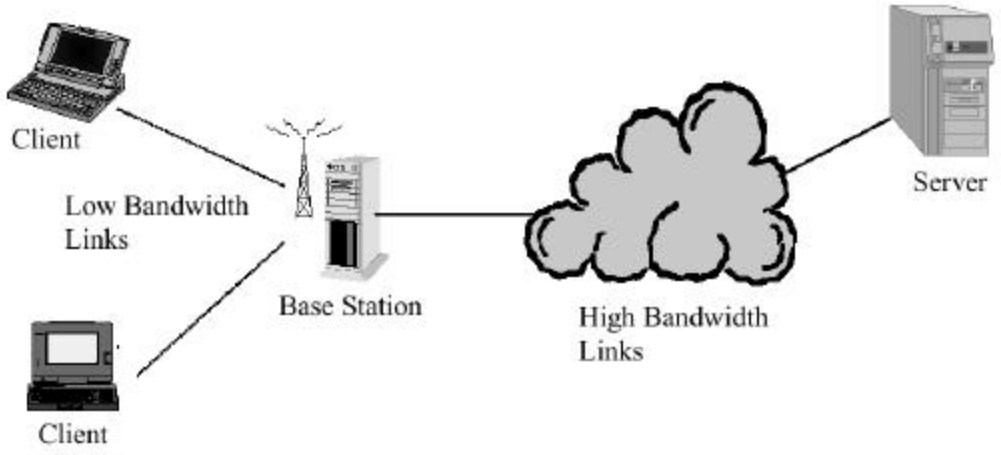

Figure 1.1: A Classic Client-Server Model

One way of bridging the gap between a resource-rich web server and a resource-poor mobile client is to interpose a system which transforms the contents on the web into

a suitable format for the mobile user. Hence, the need for middleware, which adapts to the mobile environment, leverages the best of what is available on the web and does not pose any additional work on the mobile client.

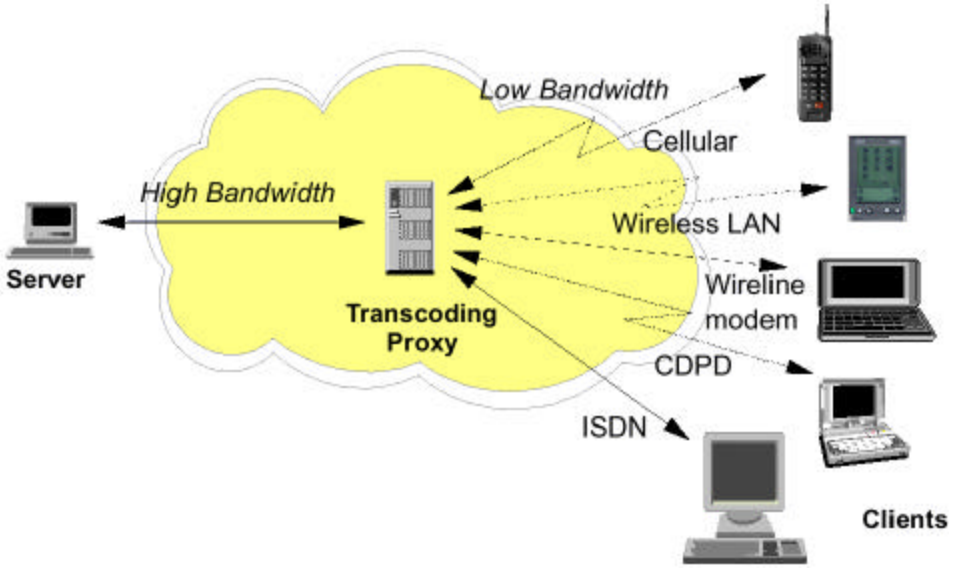

Figure 1.2: A System Environment for Transcoding Proxy

We have devised such middleware based on a solution that allows users of mobile computers to adjust the way in which their data from the web is retrieved. The middleware dynamically transcodes the data to suit the receiver equipment and available bandwidth. We achieve this by introducing a transcoding proxy system. As illustrated in Figure 1.2, transcoding proxy acts as an intermediary between World Wide Web servers and a variety of Web-enabled client devices that are connected over communication links with widely varying characteristics. The proxy is a powerful machine in the (wired side of the) wireless access network that filters and compresses the data stream originating from the server to suit the current wireless bandwidth and device capability. This processing of

the data stream (compression, filtering, conversion, etc.) is commonly referred to as *transcoding*. Most of the systems follow a "split-TCP" approach, where data is downloaded completely to the proxy, transcoded there, and then forwarded to a mobile host.

The idea of using proxy servers to improve Web browsing is not new [3], and some implementations do exist [2,7]. The contribution of this thesis is to propose an easy-to-use, portable and effective transcoding proxy architecture, supporting a dynamic adaptation model for image transcoding in a modular and extensible way.

1.3 Thesis Objectives

Several interesting ideas have been introduced to improve the performance and usability of World Wide-Web browsing in wireless environments. These ideas have considered both hardware and software. Our main interest was to find a dynamically adaptive approach that could reduce the download time at the client end. Surveys done on web traffic within the last two years have consistently confirmed that images present a large percentage of the webpage transmission load. Hence, reducing image file size would improve transmitting files across low speed wireless channels. In this thesis, our major aim was designing and developing an adaptive system for transcoding images in the Internet in order to improve their delivery speed to a variety of client devices with a wide range of communication and display capabilities. This system would be placed between clients and servers, performing several operations on images such as lossy compression,

scaling, color reduction, etc. on behalf of clients. We refer to this system as *transcoding proxy*. We believe that this approach will increase the performance of WWW browsing and reduce the data retrieval latency as well.

1.4 Thesis Contributions

Our work has the following contributions:

In order to improve the accessibility of images in the Internet, we have developed and implemented a transcoding proxy system that analyzes, manipulates and transcodes images on the fly. Without requiring modifications to Web servers and browsers, the transcoding proxy enables the following: (1) Dramatic reduction in Web download times over low bandwidth links via data transcoding. (2) Reduction of per-byte costs over tariffed links via data transcoding. (3) Tailoring of images for the multitude of small, weakly connected but Web-enabled mobile devices that are now available. The transcoding proxy is designed to adapt to dynamically changing bandwidth and display limitations, see Section 5.3.

One of the most common techniques for reducing data retrieval latency on WWW is caching. We have investigated several proxy caching techniques in order to discover the most suitable method for our transcoding proxy. Some caching techniques only cache the transformed images and discard the original images as the case in [28, 73]. This policy can save some time since the transformation operations will not be applied over and over. This technique, however, assumes that both of the client and server are stationary and the

bandwidth between them is fixed which is not the case with the mobile environment. In addition, the device characteristics, bandwidth, and user preference must all be considered in order to match the device and channel capabilities. As each client has a different recourse limitation, sending the same transcoded image to all clients will not supply each client with his requirements. Therefore, we have suggested a caching policy suitable for current web traffic. Caching both the original and the transcoded images at the proxy side is beneficial since the transcoding operations will be performed only when necessary. In Section 5.3.5, we describe the caching technique.

A paper entitled "Image Transcoding for Proxy Internet Wireless Access" was published [89].

1.5 Thesis Organization

This thesis is divided into seven chapters including this chapter:

Chapter 1: Introduction - This chapter introduces several motivations and briefly outlines our work.

Chapter 2: The challenges of mobile computing - Several challenges and problems associated with the wireless computing environment are described in more detail.

Chapter 3: Related work - It mainly illustrates related work. In particular, it reveals the work has been done to use client-proxy-server as an infrastructure for tackling mobile environment constrains.

Chapter 4: Experimental environment – This chapter describes image collection setup and characteristics of the images and the results of performing various transcoding operations on the images. Also, several key observations discovered during the experiments will be described.

Chapter 5: An image transcoding proxy - The transcoding proxy infrastructure is presented and transcoding functions and policies are explored. It also describes the various methods of transcoding data that we have used.

Chapter 6: Experimentation and evaluation - The results of our experiments are presented and discussed in this chapter.

Chapter 7: Conclusion and future work - It contains a brief summary, which is drawn from our work and some possible areas of future work are indicated.

Chapter 2

THE CHALLENGES OF MOBILE COMPUTING

2.1 Wireless Communications Network

Mobile computers may move through different areas that provide a wide variety of operating qualities. As a natural behavior, they may be physically attached to a high speed or better (wired) connection at one moment and to a low speed or restricted (wireless) connection at the next moment. Most applications and services have been designed under the assumption that the terminals are stationary and are connected to fixed networks. Consequently, these applications react poorly to sudden, drastic changes in network resources. Furthermore, it is much more difficult to achieve wireless communication than wired communication due to variations in propagation conditions and to excessive noise and interference. As a result, wireless connections are of lower quality than wired connections: low bandwidths, bandwidth variability, high error rates, disconnections, etc. Wireless connections can also be lost due to user mobility. Users may cross to different radio cellular areas or enter areas of high interference.

The first section of this chapter covers the major problems and design challenges in wireless communications: limited bandwidth, greater variation in available bandwidth, high error rate, heterogeneity, frequent signal fading, mobility, and mobile computers capabilities. The discussion reveals various approaches that have been proposed to handle some of the indicated issues. The second section introduces the well-known client-proxy-server model, which is anticipated to be a suitable solution for the mobile environment challenges.

Wireless Network	Bandwidth	Wired Network	Bandwidth
Infrared Communication	1 Mbps	Ethernet	10 Mbps
Radio Communication	2 Mbps	FDDI	100 Mbps
Cellular Phone	9 - 14 Kbps	ATM	155 Mbps

Table 2.1: Samples of Wireless and Wired Networks Bandwidths

2.1.1 Limited Bandwidth

Wireless networks deliver lower bandwidth than wired networks as shown in Table 2.1. There are no clear anticipations that these values will be increased significantly in the near future. In contrast, increasing traffic loads could even reduce these values. Unlike stationary computers, mobile computer designs should be more concerned about bandwidth consumption in order to improve their performance. As a rule of thumb, network bandwidth is split among the users sharing the same cell [11]. Subsequently, the bandwidth decreases whenever a new user joins the cell.

One way of improving network capacity is by installing more wireless cells to service a user population. It could be done by: overlap cells on different wavelengths, or reduce transmission ranges so that more cells fit in a given area. Also, certain software techniques such as compression and filtering techniques [26] can effectively conserve bandwidth. File pre-fetching is another technique [91], which can be used to smooth out the data flow to the mobile computer for applications having bursty traffic. Adaptive communication protocols have been proposed to compensate for the slow speed of some existing mobile communication links and to save the communications cost by reducing link usage. Schemes for aggregating network bandwidth have been suggested to combine several wireless link signals so as to provide higher bandwidth for a period of time to certain receivers.

2.1.2 High Bandwidth Variability

The wide variation in network bandwidth is another problem that mobile computing suffers from compared to stationary computers. Bandwidth can dynamically change from one to four orders of magnitude between being plugged in versus using wireless access. This can be caused by switching from wired to wireless access. Due to such variability, adaptation has become an essential factor in communication protocols and systems software for mobile computing. An application can approach this variability by adapting to the currently available resources, providing the user with a variable level of detail or quality. As an example of adaptation in systems software, a generalization of the Coda [39] architecture called Odyssey [35] has been suggested to support the capability of

"application-aware adaptation" of mobile clients.

2.1.3 High Error Rate

In contrast to wired links, the bit-error rate on wireless links is much higher. As a mobile computer migrates from one place to another, different levels of quality of service (QoS) will be provided by diversity of networks. Also, higher bit-error rates might result from a combination of factors such as multi-path fading, terrain and environmental factors, and interference from other transmissions [5]. Wireless links have a bit-error rate of 10^{-3}, while wired links enjoy bit-error rates between 10^{-9} and 10^{-6}. Error recovery schemes for data transmission over wireless channels are routinely used [20].

2.1.4 Heterogeneity

As a mobile unit moves to a different location, it may come across different network environments, qualities, and services. It may also need to use different access protocols. The majority of wireless network services use different modulation and transmission methods; therefore, users must have a different proprietary modem to tap into each service. Such problem of interoperability can affect the scale of mobility. Another problem with heterogeneous networks concerns access cost. Most wireless network services charge a flat fee for their service, which usually covers a fixed number of messages. Additional charges are levied on per packet or per message basis. In contrast, the cost for sending data over cellular is based on connection time instead [6]. Since different services have different access costs, the cost of a query to a centralized database

may depend on the user location. A query asked when a user is connected to a wireless LAN may incur a different cost from one posed when the user is connected to a wireless WAN. Therefore, new methods for dynamic and distributed query optimization will have to be developed to handle varying access costs. Mobile computers also may need to switch their air interface devices when moving from indoors to outdoors. For example, the infrared devices cannot be used outside because sunlight drowns out the signal. Even if only radio frequency transmission is used, the interface may still need to change access protocols for different networks.

2.1.5 Security Risks

Wireless communication is susceptible to high error rates and transmission interference or interception. Transmission interception may cause security risks. Therefore, there is a need for security measures, which can be achieved via encryption and authentication methods implemented in either software or specialized hardware. An example of such security software is MIT's Kerberos [9]. Kerberos is a trusted third-party authentication service. It can authenticate users without revealing their passwords on the network and create encryption keys that can be shared among mutually suspicious parties. It also allows a mobile unit to authenticate itself in a new domain. However, its security is not perfect and is susceptible to off-line password guessing attacks (since Kerberos still relies on well-chosen passwords and their secrecy) and replay attacks (an attacker retransmitting packets intercepted from the network) within a timeout period.

2.1.6 Mobile Host Protocols

In order to deal explicitly with the concept of computers that move, new communication protocols are needed. The current assumptions made in protocols for the fixed network may no longer be valid due to the effects of mobility. The developments of protocols for locating a mobile host are currently under way. There have been several proposals for mobile host protocols that are compatible with the TCP/IP protocol suite, of which the best features are incorporated into a proposed standardization document called Internet Draft [40]. These protocols attempt to make the operation and performance of a mobile host indistinguishable from that of a fixed host. This goal translates into two essential requirements: operational transparency and performance transparency above IP. Operational transparency means not having to reinitialize the system or individual applications after relocation has taken place. One of the factors required to ensure performance transparency is optimal routing of packets to and from mobile hosts. The assumptions made in traffic management for the fixed network need to be rethought and revised since, for instance, backing off or slowly adjusting the transmission rate when congestion is assumed is not the correct behavior in a mobile network.

2.1.7 Frequent Disconnections

In connecting a mobile computer to a network via radio or infrared links, the connection suffers from periods of disconnection, called fading. In order to ensure proper operation in spite of signal fading, two potential approaches have been explored to mask the

interruption intervals. One approach is to make the mobile computer more autonomous (i.e. less dependent on the network) by using such methods as file caching or pre-fetching. Another approach is to decouple the mobile computer and the network so that they can operate asynchronously. These techniques, therefore, have the potential to mask some network failures.

2.1.8 Mobility

As people move, their mobile computers will use different network access points, or 'addresses', which consequently increases the volatility of some information. Today's networking is not designed for dynamically changing addresses. Once an address for a host name is known to a system, it is typically cached with a long expiration time and with no way to invalidate out-of-date entries. In the Internet Protocol (IP), for example, a host IP name is inextricably bound with its network address – moving to a new location means acquiring a new IP name. Human intervention is often required to coordinate the use of addresses. There are four basic mechanisms [12-15] for determining the current address of a mobile computer, which are the building blocks of the current 'mobile-IP' schemes.

2.1.9 Mobile Computer Capabilities

Small size and weight of a mobile computer means limited memory size, small storage capacity, and a small user interface. Various methods have been suggested to cope with the problems of limited memory and storage capacity including compressing file systems,

compressing virtual memory pages, and accessing remote storage over the network. The small user interface affects both data entry (keyboard) and data display (screen size). The physical limitations of the mobile terminals require new types of interfaces, which do not rely on keyboard and screen size. Moreover, power is consumed mostly by the screen backlighting, the central processing unit (CPU), the memory, the hard disk, the display, and the keyboard. Since battery technology is fairly mature and the lifetime of a battery is not expected to increase very much over the next decade, low power consumption makes energy conservation a key issue in both hardware and software. The current memory technology for portable computers is dynamic RAM (DRAM) with power consumption of 0.5 watts for a bank of 4M. Flash EEPROM, a low power, non-volatile, dense storage technology, is considered as a potentially cheaper alternative. Flash memory has a read latency close to that of DRAM and a write latency close to that of disk, and can only withstand a limited number of writes over its lifetime. However, the power required for flash memory access is 20 times more than when it is idle.

2.2 WWW Browsing Through a Proxy Server

There is a large body of work that addresses the problem of capability mismatch for multimedia content in wireless web access. The client-proxy-server model [1,3], typically, has been used as general-purpose solution. The proxy transcodes multimedia formats, most often images, according to some predefined rules, usually in some manner that trades quality for bandwidth. This section will illustrate three approaches that have

been proposed to tackle some of the issues described in the previous section. Application partitioning, proxy based, and end-to-end approaches will be briefly described. Figure 2.1 captures the proxy server architecture for these approaches.

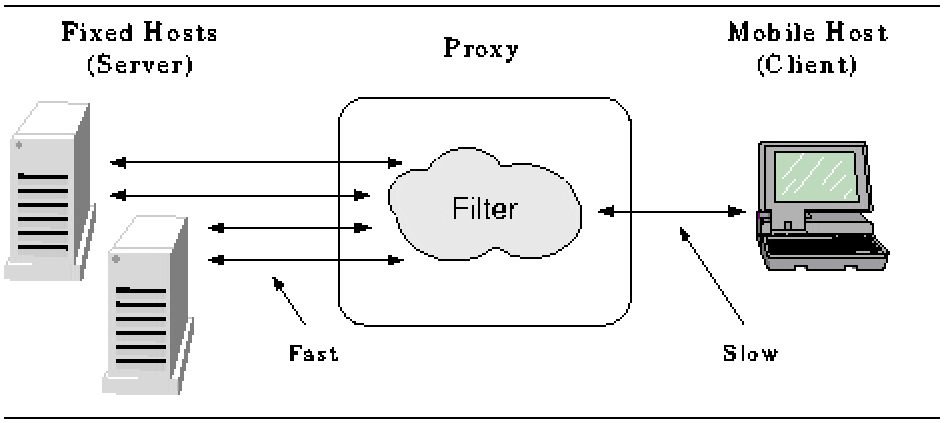

Figure 2.1: WWW Browsing Through a Proxy Server

2.2.1 Proxy Based Approach

Many proxy-based systems [1,26,30] have been developed to provide web access to mobile users. Typically a proxy server can be interposed between clients and servers to become the "workplace" in which different kinds of computation can be performed on behalf of clients. Figure 2.1 shows that the proxy has the capability of filtering and processing clients' requests and all the corresponding answers from the servers. In general, the software architecture of the proxy is modular and extensible, and this implies that it can integrate different and additional functions to meet specific user needs. Transformation Aggregation Caching Customization (TACC) model [8] takes a step towards a more general proxy architecture. This proxy architecture reduces the bandwidth

demands on the infrastructure through lossy compression and allows legacy and other nonstandard (including thin) clients to inter-operate with existing servers. The TACC model also supports application adaptation, which enables the proxy agents to react to the environment changes on behalf of mobile clients. This approach avoids inserting adaptation machinery at each origin server. From the client's perspective, the proxy is simply a server that gets the data from someplace else. The TACC model contains a variety of building blocks, or *workers*. Each worker is responsible for a particular task such as scaling/dithering images, format conversion, filtering, etc. Workers are built in such manner that they can easily chain or contact each other.

2.2.2 End-to-end Approach

With the growth of the wirelessly connected user community and the advent of mobile type devices, the transformation needed by the client, and hence the computational resources needed by the proxy to affect it, increase significantly. This has led to the argument that a purely proxy based approach will become gradually non-scalable. There is also a debate as to whether proxy based solutions are really needed to provide networking services to mobile client. The alternative solution, which was presented firstly by [16], is to make the server itself provide data in a format that suits mobile access. This typically represents an instance of end-to-end approach, which is well known in networking and system literature. In web context, multiple versions of web pages with different resolutions or formats kept at servers represent an end-to-end approach. The

end-to-end approach is predicated upon the client being able to express its preferences for particular resolutions to the server. Once the client sends the preferred representations as part of the request, the server can automatically send the right representation. For example, an image file may be made available in different resolutions by the content provider on the server. When the request is made the server sends the image file, which has the resolution appropriate to the present QoS and client parameters. This implies that the preferred format is included in the request.

Since it is not certain that every server will be able to provide data in the format and resolution that the client needs, it is obvious that we may still need some proxy functionalities such as disconnection management, multimedia transcoding, protocol translation, caching, etc. Thus, the theme of incorporating both end-to-end and proxy-based approaches, using each as appropriate, was proposed by [17,18,22] to support web access from mobile platforms. Some tasks, such as complex transcoding, which require significant computational resources, are best done in an end-to-end manner, while the proxy can handle other tasks. For example, if the server is not able to provide the multimedia content at a resolution appropriate for the client, the proxy monitors and transcodes the server response.

2.2.3 Application Partitioning Approach

The concept of application partitioning has been proposed by [23,24,27] to match the capabilities of mobile hosts and the wireless links. The core mechanism of this approach

is based on dividing the application's functionality statically between the mobile host and the wired network. This is similar to a client/server architecture, but with slightly different characteristics than are typical for that model. In the application partitioning approach, the application's objects are usually split between the client and proxy server, rather than between the client and the server. The guideline for this approach has been to allow the client to perform those tasks with lightweights, and move as much as possible of the heavy weight tasks to the proxy server. For example, sending HTML document to thin clients would be too cumbersome for them, but sending screen-sized bitmaps is not cumbersome. As shown in Figure 2.1, the proxy server acts as middleware between client and servers, performing aggressive computation such as data filtration on behalf of the client, and then sends the output result back to the client. This way enables thin clients to offload some of their work onto powerful machine and to access existing content with no server modifications.

Chapter 3

BACKGROUND AND RELATED WORK

This chapter starts with a section that briefly surveys today's state of wireless data technology. An overview about wireless technology, cellular network topology, and web browsing will be given in Section 3.1. The second section will mainly describe a variety of related work aimed at handling some of the problems identified previously. We have organized the related work into three categories: hybrid network- and application-level approaches, transcoding proxies, and partitioning of application complexity.

3.1 Wireless Networks

The use of radio waves to communicate information has been known for over a century now due to pioneering work by inventors like Marconi, De Forest, and Armstrong. Since Marconi's demonstration of Trans-Atlantic radio communication in 1901, the field has grown by leaps and bounds to where it is today.

The past three decades have seen numerous research and commercial efforts in

building and deploying systems to transmit digital data over wireless media. In the late 1960s and early 1970s, the ALOHA project led by Norm Abramson at the University of Hawaii investigated packet-switched networks over fixed-site radio link [52]. The contention resolution protocols developed in ALOHANET laid the framework for later Carrier Sense Multiple Access (CSMA) protocols for channel access. In 1972, the U.S. Defense Advanced Research Project Agency (DARPA) launched the Packet Radio Program and its successor program on Survivable Adaptive Networks (SURAN) to study issues in packet radio networks such as channel access, link protocols, and routing [55, 60]. However, despite this work, wireless data communication remained largely a laboratory curiosity for several years.

In the 1990s, this has changed dramatically. Advances in hardware technology have enabled cheap and portable wireless devices for data communication. This, coupled with the rapid expansion of the Internet and the World Wide Web, has resulted in a variety of wireless technologies becoming commercially available.

3.1.1 Wireless Technology Overview

Wireless data networks typically operate in the Radio Frequency (RF) (3 KHz to 300 GHz) or Infrared (IR) (1000 GHz to 30000 GHz) frequency range. IR offers high speeds over limited distances. Experimental research IR networks offer up to 50 Mbps over a few meters [53], while commercial, IR-based wireless LANs have maximum bandwidths between 1 and 10 Mbps today. The IR frequency range is very close to that of visible

light, which precludes transmission through walls or floors. Thus, while its range is more constrained than its RF counterparts, it offers a much higher degree of frequency reuse. The existence of impenetrable objects and reflections often causes IR dead spots in rooms, which in turn causes packet loss. The Infrared Data Association (IrDA) [63] has standardized the lower layer protocols and interfaces for IR devices, which is almost certain to be a part of every portable computer and handheld device in the future. Commercial IR-based LAN products available today include IBM's Infrared wireless LAN and InfraLAN.

RF can usually penetrate walls, which makes it an ideal technology for total seamless coverage. However, compared to IR, RF devices typically suffer from higher levels of interference and noise, which degrade the received signal and cause bit-errors. Because a large number of users in the same frequency band can degrade overall usability and performance, the use of different wireless frequencies is governed by strict spectrum management regulations. In the United States the FCC regulate the use of radio frequencies. Other countries have similar organizations. Typically, licenses are required to operate RF devices in most frequency bands.

For several years, the application of RF to data communication exploited *narrowband* technology. Here, the input signal is modulated over a constant frequency wave, called the *carrier wave*, and the resultant signal is transmitted through the sender's antenna. When it reaches the receiver (often tainted by interference or noise), the receiver picks up this signal through its antenna, demodulates it, filters out the carrier frequency,

and attempts to estimate the signal that was transmitted based on certain *decoding rules*. An error could occur in the decoding process if the level of noise or interference is too high-this manifests itself in the form of corrupted bits, which higher-level data protocols or applications must handle.

One of the major problems with narrowband systems that make it vulnerable to interference is its lack of *frequency diversity*, since the entire carrier is concentrated in a single frequency. *Spread spectrum* [64] is a technique to combat this problem. Here, the transmitted signal is spread over a broad frequency band, which makes it more robust to interference or jamming and more secure against eavesdropping. There are two common kinds of spread spectrum techniques: *direct-sequence* and *frequency-hopping*. In direct-sequence spread spectrum, the input signal is transmitted simultaneously over a broad range according to a pre-assigned code. In frequency-hopping spread spectrum, the transmitter sends data on one narrowband frequency for a short amount of time, then jumps to another narrowband frequency, using a pseudo-random hopping sequence. The receiver synchronizes with the sender's hopping sequence and tunes in to those frequencies to receive and decode the signal. While frequency-hopping is more robust to sources of interference than direct-sequence spread-spectrum, maximum data rates are typically lower. Thus, it is not as suitable for high-speed wireless access as for slower, wide-area links.

Examples of RF-based wireless LAN products include Solectek's AirLAN, Aironet's ARLAN [66], Motorola's Altair Plus-II, Proxim's RangeLAN products [65],

and Lucent Technologies' WaveLAN [70]. Most of these operate in the unlicensed Industrial, Scientific and Medical (ISM) bands at 915 MHz and 2.4 GHz that have been set aside by the FCC for experimental purposes.

Examples of wide-area RF networks include Metricom's Ricochet network [67] that operates in the 915 MHz band and uses frequency-hopping spread spectrum technology, the Cellular Digital Packet Data (CDPD) network [69], and the GSM data service [68]. Peak link bandwidth in these outdoor networks today are typically between 10 and 100 Kbps, while future systems are likely to be in the 100-500 Kbps range (e.g., Metricom's proposed "Autobahn" network [71]).

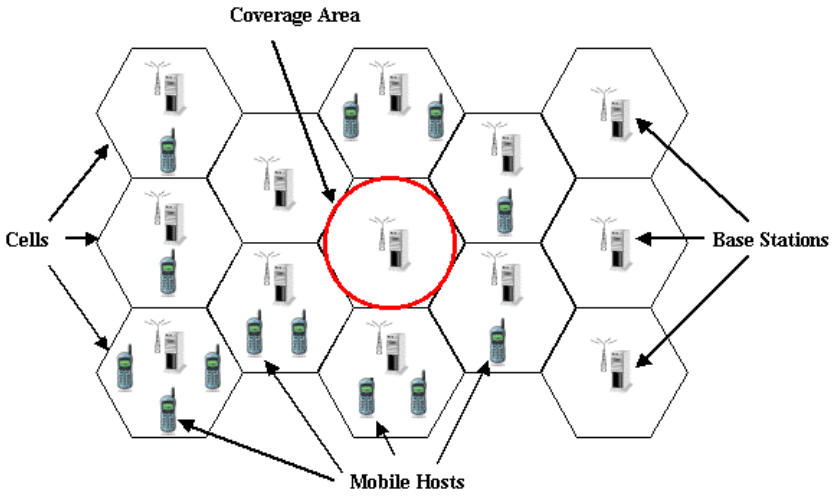

Figure 3.1: Schematic Arrangement of Cells in a Cellular Wireless Network

3.1.2 Cellular Network Topology

Many wireless and mobile networks are organized in a *cellular topology*. These topologies are composed of a wired, packet-switched, backbone network and a wireless

network. The wireless network is organized into geographically defined cells, with a control point called a *base station* (BS) in each of the cells, as shown schematically in Figure 3.1.

These base stations are also directly connected to the wired network, routing packets between the wireless and the backbone network as shown in Figure 3.2. A mobile host (MH) receives data from a fixed host (FH) in the Internet routed via the BS of the cell it is currently in. As the MH moves between wireless cells, the task of forwarding data between the wired network and the mobile host must be transferred to the new cell's BS. This involves updating routing information in the wired infrastructure to reflect movement, and is known as a *handoff* [56,57,59]. The IETF standard for mobility is the Mobile IP protocol [54]. It is important for several applications and higher-level protocols that handoffs be as seamless as possible, incurring low latencies and causing little or no packet loss [58,59]. Numerous schemes have been proposed in the literature and several systems have been deployed to achieve these goals [e.g., 58,61,62].

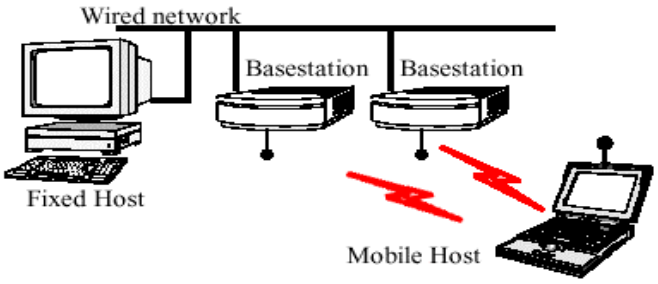

Figure 3.2: Network Topology of a Cellular Wireless Network

3.1.3 Web Browsing (WWW)

HTTP [72] is a network protocol used to deliver virtually all files and other data on the World Wide Web such as HTML and image files. The Web client communicates with a Web server using one or more TCP connections (Figure 3.3). HTTP normally establishes its first TCP connection and retrieves the HTML document identified by a URL. After the document is retrieved this TCP connection is terminated. What happens next depends on the client browser. Typically, the client browser will extract the embedded HTML document entities from this document and retrieve them based on extracted URL's.

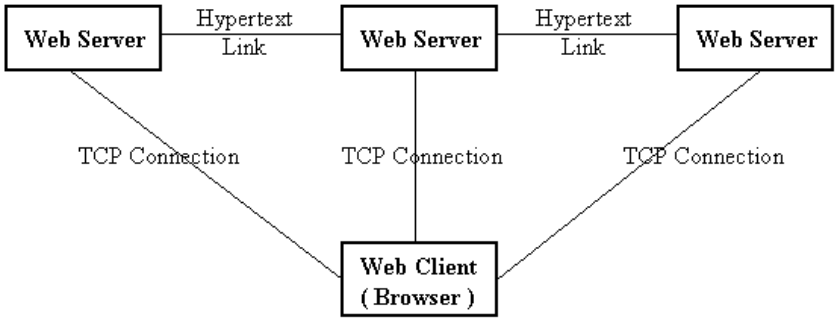

Figure 3.3: TCP Connections for WWW

3.2 Related Work

A considerable amount of recent efforts has focused on the area of information access from mobile platforms, using different methodologies. The client-proxy-server model that aims to overcome the challenges faced in the mobile computing scenario has been used by the most popular projects including our project. Several techniques at the network level have been used to shield clients from the effect of poor (especially wireless)

networks [4]. At the application level, data transcoding mostly using proxy server has become more common. This section describes some well-known projects for the purpose of improving wireless network access from mobile computers.

3.2.1 Hybrid Network- and Application-level Approaches

To the best of our knowledge, there are at least two projects that follow the policy of combining network-level optimizations with some application-level content filtering.

The ***Mowgli*** [33] system consists of two mediators, the Mowgli WWW Agent and Proxy located on the mobile host and the mobile-connection host respectively. They communicate with each other through the Mowgli HTTP protocol, which reduces the number of round-trips between client and server. A specialized transport service, the Mowgli Data Channel Service is used for reliable communication between the mobile-connection host and the mobile host. Mowgli WWW reduces the data transfer over the wireless link in three ways: data compression, caching, and intelligent filtering. Mowgli WWW supports content-type specific as well as generic data compression, using the splay-prefix algorithm [36] as the generic compression algorithm. Content-type specific compression means that each particular document type (e.g. text, image data, audio data) can be assigned a different compression algorithm that performs best on that type of data. Generic compression is applied to all document objects that have not been assigned a content-type specific compression method. Currently, the only supported content-type specific algorithm is GIF to JPEG conversion, which strictly is not a data compression algorithm. However, GIF to JPEG conversion often provide good results, because the

JPEG format uses a lossy compression algorithm that sacrifices color information in favor of smaller size. In Mowgli WWW the user is given the opportunity to specify a maximum size for retrieved text documents as well as embedded images. Since the size distribution of WWW documents is heavy-tailed [37], a small percentage of them are quite large. By specifying maximum sizes for document objects the user effectively "cuts the tail" of the distribution and gives the system an upper limit for how long a document retrieval may take. Large embedded images that exceed the size limit are not transferred to the mobile node. The Mowgli WWW Agent replaces them with a generic image to inform the user that the images were not received.

A drawback of this work is that Mowgli cannot dynamically adapt its behavior to changing network conditions. Also, their system replaces standard communication protocols with specialized communication protocols, which prevents the user from installing the same protocol software on fixed and mobile computers. In particular, this problem occurs when the user wants to operate a mobile computer as a fixed host as well. In addition, Mowgli's protocol-level lossless compression stands in contrast to our document model's semantic lossy compression.

Bruce Zenel's "***dual proxy***" architecture [26] can be realized as a proxy server situated between a client on the mobile host and a server on a fixed host, which are separated by an area of restricted network resource. This system also provides both low-level and high-level filters. The low-level filters operate at the socket API level and require modifications to the mobile device's network stack. The high-level filters can use

application-specific semantics to filter data before it is sent to a client. The action of filters may range from simple discard/forward to arbitrarily complex processing. Filters may be pre-existing or dynamically downloaded from an associated mobile host application. The mobile hosts are using IP to communicate with the fixed hosts along with Mobile IP [45] to handle IP routing. In Mobile IP, a Mobile Support Router (MSR) is an IP router with the extra capability of running a special protocol that keeps it and other MSRs informed of the current location of mobile hosts. Packets from fixed hosts, destined for a particular mobile host, are routed via standard IP to an MSR that advertises itself as the reachability point for the mobile host. The packets are then encapsulated, tunneled to the MSR that is serving as the default gateway for the mobile host, decapsulated, and finally sent to the mobile host using regular IP. The proxy is made up of three cooperative components: High Level Proxy, Low Level Proxy, and Event Manager. The High Level Proxy allows filters for application layer protocols (e.g. MPEG, SMTP, HTML) to be downloaded dynamically from mobile host applications. The Low Level Proxy is used to create and install filters for the transport and network layers (e.g. UDP, ICMP, RPC). The Event Manager provides a control interface for the filters running within the High and Low Level Proxies.

The disadvantage of [26] is that the high level filter is part of the application rather than a middleware component, which complicates its reuse by other applications and makes it awkward to support legacy applications. This approach also does not consider the problem of proxy handoff process.

3.2.2 Transcoding Proxies

A number of research efforts have focused on using transcoding proxies to compress and filter the Internet content in order to improve data transmission over wireless link.

The ***GloMop*** [1,29] project intends to ameliorate the resource inequality between fixed and mobile computers through the use of application layer proxies. A proxy residing on the wired side of a network uses application specific information to tailor the filtering of data destined for the mobile host. Filtering is in the form of distillation, which can be envisioned as a highly lossy, real-time, datatype-specific compression. This process preserves most of the semantic information to make a document useful while reducing the size in order to facilitate cheaper transmission and rendering on the mobile host. The proxy can convert each semantic type (only text and images so far) to a common intermediate representation (a subset of HTML for text, and PPM [31] for images), distill the intermediate representation, and convert it to a different target representation for the client if desired. GloMop implements an image distiller called *gifmunch*, which performs distillation and refinement for GIF [32] images. The image distiller is constructed largely from source code in the NetPBM Toolkit [31]. Currently the distiller picks a color palette based on the known capabilities of the client (which identifies itself when it first connects to the proxy and establishes a session), and optimizes for a particular target size in bytes of the distilled representation by predicting compression. Prediction is done by observing the expansion when converting the original image to the PPM intermediate format, and multiplying this by an encoding-specific "expansion ratio" based on the effective bits per

pixel achieved in past runs using the same target encoding. GloMop also implements a rich-text distiller, which performs lossy compression of PostScript-encoded text. The distiller replaces PostScript formatting information with HTML markup tags or with a custom rich-text format that preserves the position information of the words.

Although the model of this system provides a reasonable set of guidelines for thinking about partitioning it does not address the disconnection problem at all. Currently, there are some thoughts to integrate Rover into their model to provide a rich abstraction for dealing with disconnection operations.

The ***Mowser*** [34,38] project follows the client-proxy-server model, which is similar to our work. Mowser allows mobile users to specify their viewing preferences, based on the network connection and available resources, and performs active transcoding of HTTP streams accordingly. The viewing preferences stored for each MH include a starting point, color capability, video resolution, sound capability, etc. The preferences can be modified by the MH whenever the network connections or available resources are changed. When the proxy receives a request from the MH, it looks up the preferences stored with the IP address of the MH and processes the request accordingly. If no preferences had been assigned by the MH, default preferences are considered. For a MH like the PalmPilot, which can handle only text and images, the proxy greatly reduces the data transfer by selectively GETing the files. That is, when the proxy receives a GET request from a PDA, it sends a HEAD request to the WWW server to get information about the content type of the file, and then GETs the file only if the PDA can handle it.

For example, if the proxy finds an image tag in the HTTP stream received from the server, the proxy will read the URL of the image file to be fetched and first sends a HEAD request to the server. The proxy checks the content-type and content-length information received from the server to decide whether to transcode or not transcode the image. Typically, if the content-length is small enough to be handled on the MH, unmodified image will be sent to the MH. But if the image is larger than what can be handled by the MH, it is reduced in size or color as requested by the MH. The image files are scaled down in size, or the number of colors is reduced, or both without sacrificing semantics. The content-type information is used to decide the transformations that the image file has to go through. All images are converted to portable pixmap format for processing and then converted back to GIF format for displaying. Also, the original URL in the image tag is replaced with the URL of the modified image stored locally by the proxy and sent to the MH. This makes the MH GET the modified image file from the proxy. With PDAs, the proxy might have to reduce images to 2-bit gray scale and thumb size.

In this work, the proxy does not use all the preferences set by the user to limit or transform the data before serving the MH. Furthermore, the proxy adds a preference overhead due to two reasons. Firstly, it is written in Perl and uses netpbm for the processing of image files. The speed could be increased by writing optimize C code and image conversion routines. Secondly, messages go all the way up to the application layer in the proxy even if data just needs to be written from one socket to another.

The ***InfoPyramid*** [41] model presents a general framework for handling the Internet content. It typically allows specialized methods to be plugged-in for analyzing, filtering, translating, and manipulating the Internet content. InfoPyramid develops a conceptually redundant representation of the Internet content that aggregates multiple versions of the content along the dimensions of modality (video, image, text, and audio) and fidelity (which includes summarized and compressed versions) [42]. The translation and summarization methods generate the alternate versions of the content as needed. The translation methods convert the content between modalities, such as text to audio, or video to images. On the other hand, the summarization methods generate versions within the same modality, but with different fidelity, such as compressing the images, summarizing text, and extracting and re-animating the key-frames from video. The transcoding system retrieves, analyzes, and ingests the Internet content into an InfoPyramid representation. A policy engine gathers the capabilities of the client, the network conditions and the transcoding preferences of the user and publisher to define the transcoding options for the client. The system then generates and selects the output versions of the content to be delivered to the client device. The InfoPyramid system provides the mechanism for assigning content value scores to the alternate versions of the content. In some cases, the content value scores are derived automatically by measuring the loss in fidelity that results from translating or summarizing the content. Otherwise, the content value scores can be tied directly to the methods that manipulate the content. The content value scores comprise only part of the information that can be used in the content

selection process. Both the publisher and user may have preferences for how the content is transcoded.

The system load on proxies is not considered in their model. Decisions are mainly based on current link conditions and client device capabilities. Moreover, object mobility and proxy migration issues are not addressed in their design. Similar to the other approaches described, their system does not provide any further information about how the system supports the case of multi-user access.

The ***Class-based Proxy*** [73] system proposes to group the displays of popular mobile computers into seven classes according to the size and color. The class-based proxy server then distills and saves image file according to the class. The system provides two modes; the class mode in which the user can easily choose the display configuration, and the expert mode in which users can customize their own display configuration. For example, the device of 256 x 24 display is used in a special environment of automobile, and thus it is configured in the expert mode. In the class mode of HTTP 1.0, classes are implemented using the port without changing the protocol. The port number must be set up along with the IP address when the proxy server is configured in a web browser. The use of port number makes it possible to use the class-based proxy server without modifying the web browser in HTTP 1.0 environment and easily identify the class. For instance, the port number 8885 is used as the port number of proxy server and web browser of a mobile computer of class **5**. It can easily expand the proxy server by assigning a new class number when mobile computers of new display characteristics

appear. In class mode of HTTP 1.1, the information on the mobile computer can be transferred to the class-based proxy by using the CC/PP of HTTP 1.1 without using the port number. In expert mode, users define the display size and other options. The user first logins to the server using the login name, and then sets up the environment parameters used for distillation. Cache is also assigned to each user.

Their system suffers from the drawback that image distillation is typically based on the display characteristic. The system does not consider the network constrains or the user preference. For example, Image distillation would not take place when the display size is 800x600 or larger even if the bandwidth was low. The system uses the CC/PP, which is one of the HTTP 1.1 features, to transfer the user profile. Because the profile of the class-based proxy is written in XML/PDF, both the web server and web browser should be able to process XML.

3.2.3 Partitioning of Application Complexity

Another line of work has focused on the idea of portioning applications between a thin or poorly-connected client and more powerful server.

The ***Wit*** [43,44] project uses application partitioning to increase an application's utilization of dynamic resources. The Wit typically partitions mobile applications between a client running threaded Tcl on an HP palmtop, and a workstation-based proxy process. This partitioning is realized as the assignment of application data and functionality to both mobile and stationary machines. Applications running on Wit may: execute entirely on

the palmtop, be partitioned between the palmtop and a LAN workstation, or execute entirely in the wired network, using Wit only for simple terminal I/O operations. Partitioned elements are known as hyperobjects, which may be migrated or replicated across the wireless link. These objects are designed to be linked to each other in ways that expose application structure to the system in a uniform, manageable way. The system then manages these objects using caching, prefetching and data reduction. In general, the Wit system is divided into network-proxy and palmtop components. Palmtops are usually connected to the LAN via ParcTab transceivers and gateway processes. Each palmtop is represented in the wired network by a network-proxy process on a workstation. Messages are forwarded from a gateway to the Wit proxy for the sending palmtop. The Wit proxy may handle the message, or may pass it along to an application. Applications always communicate with the palmtop via the proxy, which insulates them from the complications of palmtop mobility.

The ***Rover*** [47] toolkit offers mobile communication support based on the idea of mobile objects. In the Rover Toolkit, relocatable dynamic objects and queued remote procedure calls are provided in order to better support application mobility. A relocatable dynamic object (RDO) is an object (code and data) with a well-defined interface that can dynamically be moved between a client and server to reduce client-server communication requirements. Queued remote procedure call (RPC) is a communication mechanism that permits RPCs to be queued for later transmission and execution, allowing the caller to continue processing even when a host is disconnected. For example, simple GUI code can

be migrated to the mobile, where it will use queued RPC to communicate with the rest of the application running on a server. Applications are designed to consist of a set of RDOs that move between client and server depending on the state of the mobile host. The Rover system provides an execution environment for code associated with RDOs and handles the transport of RDOs between the client and server (via queued RPC). The system also contains extra functionality such as an object cache in order to make mobile applications more efficient.

One of the drawbacks of such a system is that mobile applications must be designed from scratch using a new programming model. They take the approach that bandwidth can be conserved by intelligently partitioning the application, in contrast to our design in which conservation is achieved by filtering protocols. It is unclear whether this is a practical solution. Application partitioning may lead to a more optimal solution, while protocol filtering (if done properly) is guaranteed to be more optimal than the base case. In addition, application partitioning may lead to security and fault tolerance problems if code and data originally designed to reside at the client is moved to the server in order to conserve bandwidth.

In [46,48], software architecture for supporting mobile applications has been proposed. The mechanism of this system is based on two notions: service proxies and object graphs. In their design, an application is partitioned into two pieces, one piece runs on a mobile computer, and another piece runs on a stationary computer somewhere within the wired infrastructure. These sides are connected by wireless networks that may be

replaced on the fly. The piece on a stationary computer is called a service proxy. The service proxy filters or caches data from servers before transmitting the data to the piece on a mobile computer. These two pieces are constructed by composing small object graphs whose composition can be dynamically reconfigured by adding or removing replaceable devices.

Their work mainly focuses on constructing an infrastructure to allow these object graphs to be created and reconfigured in a dynamic way depending on state information obtained from the mobile host. Mobile host interference is required to trigger the handoff process. The running application usually suspends for long periods of time, from the mobile host triggering the handoff process to the end of the migration process. This work is also limited in that it does not address object graph management (i.e., initial setup of object graphs and their possible movement).

3.3 Summary

Mobile users have special requirements that should be taken into account when designing applications and communications architecture. We have discussed a number of performance problems with the mobile computing environment in general. We have also described several projects that have proposed a variety of solutions to improve the performance of mobile access information. As can be seen from the previous work, some proxy servers do not have any information on the hardware specification of mobile hosts; the image file is typically distilled without full consideration on the client device

capabilities. The main aim in their work is to adapt the web content to bandwidth variations by selecting a suitable compression scheme. On the other hand, the work has been done by [73] emphasize fully about the hardware specification of mobile computers. Their system adjusts the image file according to the size and color property of the mobile device display. Other transcoding proxies [1,29,74] typically consider a few client devices and employ static content adaptation strategies. A common policy is to distill all images by a fixed factor. Thus, these transcoding proxies fail to dynamically address the variation in the resource requirements of different web documents. The sole purpose of the service is to improve response times for clients connected over slow links such as modems. The set of client devices will also grow more diverse. Certain resources, such as effective network bandwidth, costs and patience of the users can be different for similar client devices. The static adaptation policies used by these systems do not handle well this variability in web content and client resources. Accordingly, there has yet to be a system that combines display catachrestic, network bandwidth, and user preference in order to provide a general dynamic filtering architecture. The contribution of my thesis is the design and realization of such a system, and an evaluation of its usefulness. In our proxy transcoding system, adaptation to the changes in the network resources, user preferences and device characteristics are left to the proxy server. The proxy intercepts client device's requests for web pages, fetches the requested content, adapts it and sends the adapted version to the client. This content adaptation is often termed "*transcoding*". In our system, images are customized based on three parameters: display characteristics,

available bandwidth, and user preferences. The system selects a number of different transcoding operations that provide the "best value" within the constraints of a client's resources. The mobile host subsequently gains the benefit of transcoding images in speeding image delivery as the customized images are often much smaller than the original images. The caching mechanism of our system can also increase the performance of image delivery. Caching the original and the transcoded versions of image files can reduce the process time of the proxy server and improve the response times.

Chapter 4

EXPERIMENTAL ENVIRONMENT

4.1 Objectives

Transcoding is a powerful technique employed by network proxies to dynamically customize multimedia objects for prevailing network conditions and individual client characteristics. Transcoding can be performed along a number of different dimensions and the specific transcoding technique used depends on the type of multimedia object. The scope of this thesis is limited to only one type of multimedia objects, which is image multimedia. Understanding the nature of typical Internet images and their transcoding characteristics was one of our research goals that would enable us to develop a powerful transcoding technique. We focused our attention on transcoding that customize an image for file size savings. Gaining such kind of knowledge allows the service to choose potential transcoding techniques that offer benefits for a wide variety of images. It also allows the service to avoid choosing transcoding techniques that might appear promising but are not effective for their target workload. We designed our experiments to answer the

following questions: Firstly, what are the characteristics of the images accessed on the Internet today? Secondly, how do various image transcoding techniques perform for these images? We are concerned mainly with transcoding that yield file size savings. Hence we define a transcoding that saves at least 50% of the file size for 50% of the images as a *productive transcoding*.

This chapter is organized as follows: Section 4.2 describes our image collection setup. Section 4.3 provides a brief overview of the GIF and JPEG file format. Section 4.4 and 4.5 describe the characteristics of the images and the results of performing various transcoding on the images, respectively. Section 4.6 describes several key observations discovered during our experiments. In Section 4.7, a brief summary will be provided.

4.2 Image Collection

A workload of typical images accessed on the web is crucial for drawing realistic conclusions about the effectiveness of a transcoding operation. However, most of standard access trace collections from server and network proxy web traces [80, 81] do not accurately reflect the data stream requested and experienced by users. For our study, we were much interested to trace the data stream that captures the user perspective to attain accurate access trace statistics. In other words, we need the actual images that a user is downloading. So we can measure the applicability of transcoding to the actual images and not to some synthetic images that were generated to be of the same size as the images available in an access trace. We therefore collected caches of nine users for a period of

five weeks to trace the most frequent file types and sizes used in web pages when accessing the Internet. Then we applied a very simple batch program on the collected caches for classifying files based on their type and size. Users had a widely different range of interest and browsed a very heterogeneous collection of web sites. For each user we confirmed that the most frequently used file types are GIF and JPEG images.

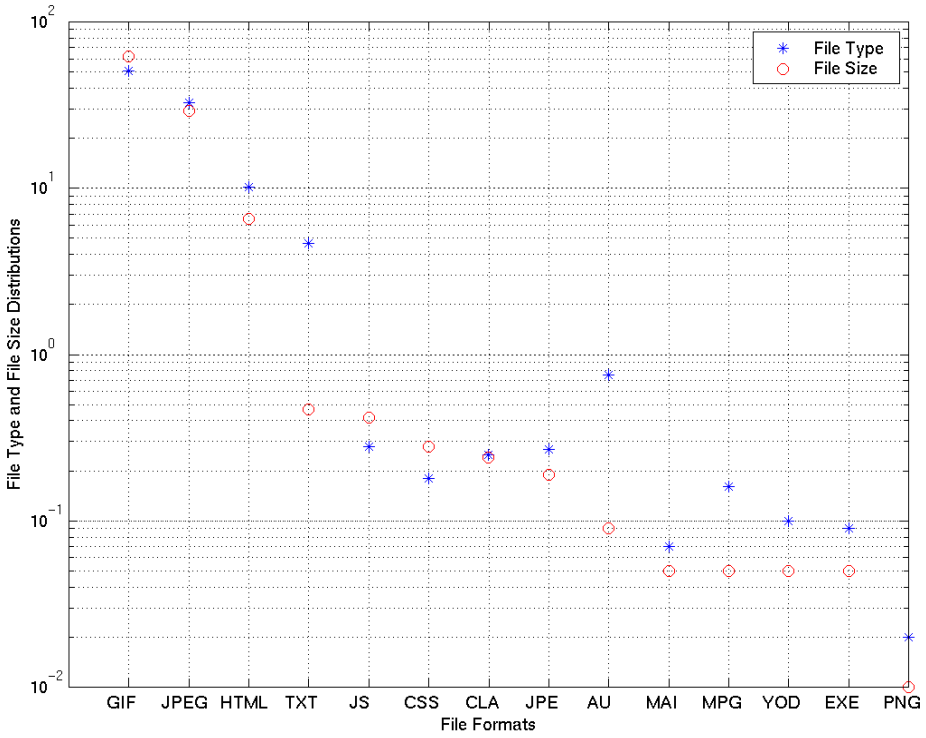

Figure 4.1: File Type and File Size Distributions for Selected WWW Users

Figure 4.1 demonstrates our findings across all users and shows that GIF and JPEG represent 62.24% and 29.24% respectively of the user traffic. Other file formats, including PNG, make up the rest of the (8.52%) requests. GIF and JPEG images comprised 50.43% and 32.56% of the image bytes transferred, respectively. Interestingly,

while most of the images are GIF's, bytes transferred are more evenly distributed between GIF's and JPEG's.

For our experiments, we downloaded a working set of "typical" inline images, using *Gozilla* [82]. In order to use *Gozilla* to collect images, one has to provide it with the URLs of the web sites of our interest. It then scans each web site and stores any GIF or JEPG images in a specific directory if that web site has any. We provided *Gozilla* with more than 200 URLs of different web sites such as education, sport, food, and stock. In a first step, we amassed about 1500 GIF and JPEG images. To ensure that our results are not biased, we explored whether the set of images corresponds to the distribution of images found on the WWW. We were looking for information that would give us some idea about the representative usage samples of the web, focusing on the average file size of images on the typical web page. After collecting such data, several interesting observations were drawn. First, the smaller files were requested with greater frequency than larger files. Second, it has been found that images are the most requested items and account for the most traffic, which we had confirmed as well. Third, images have an average size of 14KB, 19KB, and 21KB from the client, proxy, and server side respectively. The differences in these numbers confirmed our assumption that server-side characteristics do not necessarily reflect the client-side views. Fourth, the most popular domains accessed via the Internet were equally split between educational and commercial domains at 30% each. Based on these observations, we found that the average size of the collected images was greater than the one mentioned in the papers. Therefore, we

downloaded 2050 smaller images in order to better approximate these web page characterizations. We therefore ended up with a collection of 1555 GIF images and 1995 JPEG images, totaling 27 Mbytes and 85 Mbytes respectively, collected from a diversity of web sites for our experiments. Not counting repeated accesses to the same image, we have found that the majority of the images in our collection (65% of GIFs and 58% of JPEGs) are from the .com domain.

For our experiments, it was necessary to identify an appropriate image processing software to perform various transcoding operations on the collected images. Currently, we only consider two types of images, GIF and JPEG, which are the most commonly used over the Internet as shown by the collected statistical. We reviewed and tested a number of image processing software [77, 84, 85]. We selected ImageMagick [77], which is well documented and was relatively straightforward to install and execute. Furthermore, ImageMagick supports large number of functionalities and image formats. It also can work easily under different platforms (i.e., Unix, Linux, Windows, Vms, Macintosh).

4.3 An Overview of GIF and JPEG File Formats

Whenever you serve the web and an image pops up on your screen, it can be with very high probability, one of only two types: GIF or JPEG. Furthermore, most of the popular web browsers support these types of graphic file formats. In fact, even after many years of use on the WWW, GIF and JPEG images are poorly understood by many. There has been confusion about these formats in terms of their intended purpose. In this section, we will

see the differences between these formats, and why these two image file formats are the perfect complement to one another for publishing on the World Wide Web. For GIFs, we will discuss color depth and its effect on file size and image quality. Also, we will discuss some of the unique features of the GIF format. For JPEGs, we will see the relationship between the amount of JPEG compression applied to an image and the amount of file size reduction. Progressive JPEGs also will be discussed.

4.3.1 GIF File Format

GIF [32] stands for Graphic Interchange Format, and was originally developed by the Compuserve Information Service in 1987 as an efficient means to transmit images across data networks. GIF images can contain a maximum of 256 colors (8-bit), which are stored in a *color palette* or *color table* within the image file. Each color in the GIF color table is described in terms of Red, Green and Blue (RGB) values, with each value having a range of 0 to 255. To date, there are two versions of the GIF format, versions *87a* and *89a*, which were released in 1987 and 1989 respectively. Both versions contain support for LZW [49] file compression, interlacing, 256-color palettes and multiple image storage. Version 89a added background transparency and a few other additions such as delay times and image replacement parameters, which made the multiple image storage features more useful for animation. The version or format specified when saving a GIF image is critical. Loading a GIF89a file and saving it as a GIF87a may result in the loss of transparency and perhaps other important data as well. The GIF format includes some key

features, which makes it a unique and valuable format for the World Wide Web. These features include *file compression, color depth, transparency, interlacing* and storage of multiple images within a single file.

The GIF file format uses a variant of the Lempel-Ziv Welch compression algorithm (LZW [49]) that squeezes out inefficiencies in the data storage without causing a loss of any data (lossless compression) or distortion of the image. The LZW compression scheme is most efficient at compressing images with large fields of homogeneous color. It is not very good at compressing complex pictures with lots of grainy texture. LZW file compression merely compacts the image data by identifying and storing patterns found in the image. As these patterns are repeated elsewhere in the original image, only the *index number* of the pattern is stored in the compressed file, thus achieving the data compression. When the GIF image file is decompressed, the pattern index numbers are replaced with the original patterns stored in the translation table.

The variable GIF color depth is strongly associated with the image's file size and visual quality. It takes a certain number of bits to represent a specific color in a GIF image's color table. The more colors there are in the table, the greater the number of data bits which are required to represent each color, and therefore, the larger the file size. Reducing the color depth of any GIF image will reduce the file size, but the image quality may suffer. We cannot precisely predict the reduction in file size due to changes in compression efficiency and the fact that some of the file's header information is not affected by the reduction in color depth.

Transparency is the feature of the GIF89a format which allows for the specification of one of the colors in the palette to be ignored while processing the image for a display device. As shown in Figure 4.2, when the specified transparency index is encountered, the corresponding pixel of the display device is not modified and processing goes on to the next pixel. Using transparency, users can create images that seem to merge with or overlay the existing background, giving the illusion that the graphic is not rectangular: the parts of the rectangle that you do not need are simply made transparent.

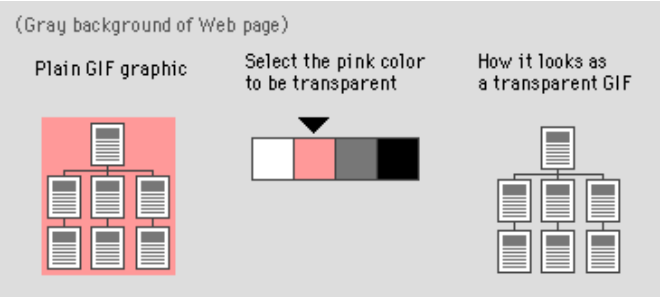

Figure 4.2: Background Transparency

The conventional (non-interlaced) GIF graphic downloads one line of pixels at a time. Web viewers, like Netscape, display each line of the image as it gradually builds on the screen. In interlaced GIF files the image data is stored in a format that allows Web viewers to begin to build a low-resolution version of the full-sized GIF picture on the screen while the file is still downloading. The "fuzzy-to-sharp" animated effect of interlacing is visually appealing, but the most important benefit of interlacing is that it gives the reader a quick preview of the full area of the picture. In fact, interlaced graphics are not faster-loading than non-interlaced graphics. They just look as if they download

faster because the rough preview comes up faster.

Although not intended for animation, the GIF89a specification did add a few enhancements to the file header, which allows browsers to display multiple GIF images in a timed and/or looped sequence. Although the GIF87a specification also allowed multiple data streams (images) to be contained within a single file, there was insufficient image control capability to do much of anything with it. GIF89a's addition of delay times between image displays and some control over the removal method for the previous image made for a more complete system for animation.

4.3.2 JPEG File Format

Joint Photographic Experts Group (JPEG [50]) is another graphics file format commonly used on the Web, which is designed for minimizing graphics file sizes. JPEG is actually just a compression algorithm, not a file format. The files commonly called JPEG on the Net are really stored in format called JFIF (JPEG File Interchange Format [51]). JPEGs work well on continuous tone images like photographs or natural artwork; not so well on sharp-edged or flat-color art like lettering, simple cartoons, or line drawings. JPEGs support 24-bits of color depth or 16.7 million colors. A new form of JPEG file called "progressive JPEG" gives JPEG graphics the same gradually-built display seen in interlaced GIFs. Progressive JPEGs are transmitted and displayed in a sequence of overlays, with each overlay becoming progressively higher in quality. This feature helps speed up the appearance of an image by sacrificing the initial quality. JPEG uses a very

sophisticated mathematical technique called a discrete cosine transform (DCT) to produce a sliding scale of graphics compression. Thus users can choose the degree of compression they wish to apply to an image in JPEG format, but in doing so they are also choosing the image quality. The more one squeezes a picture with JPEG compression, the more one degrades its image quality. A small amount of compression has a negligible effect on the image quality, yet a substantial effect on file size. As we increase the amount of compression, the reduction in file size is less pronounced, yet the deterioration in image quality becomes more and more noticeable. JPEG can achieve extremely high compression ratios, squeezing graphics down to as much as 100 times smaller than the original file. This is possible because the JPEG algorithm discards "unnecessary" data as it compresses the image, and is thus called a "lossy" image technique.

Most JPEG compressors allow the user to specify a range of values for the scaling factor, by specifying a compression metric called Quality Factor. This Quality Factor is an artifact of JPEG compression. Different software implementations use different values for Quality Factor. Quality Factors are not standardized across JPEG implementations. The IJG Library [88] and ImageMagick Library [77] use a 0-100 scale, while Apple formerly used a scale running from 0 to 4. Recent Apple software uses a 0-100 scale that is different from the IJG scale. Paint Shop Pro's scale uses a 100-0 scale, where lower numbers imply higher quality. Adobe Photoshop gives discrete maximum/high/medium/ low choices.

4.4 Static Image Characteristics

In order to understand the characteristics of images accessed on the Internet today, we first analyzed the static characteristics of the images. Static image characteristics such as image geometry, file size, color, etc. give a sense of the possible axes along which transcoding can take place. In the next section, we will analyze the transcoding characteristics for the images. We placed emphasis on the characteristics of transcoding operations that make sense for our image collection based on these static image characteristics.

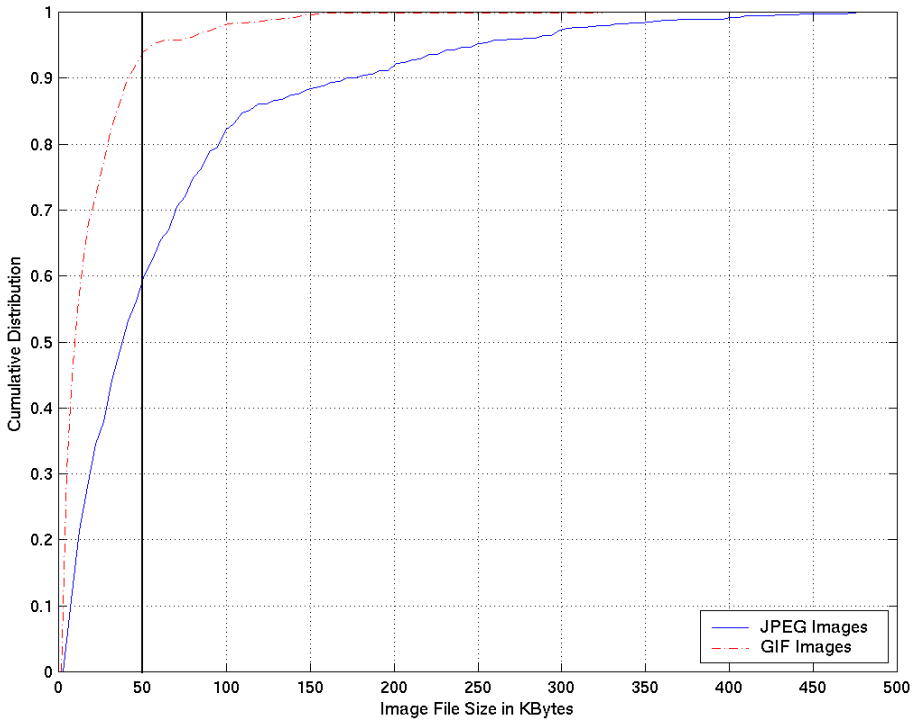

Figure 4.3: GIF and JPEG Image File Size Distribution

4.4.1 Image File Size

First we analyzed the file sizes of the images in our workload. We plot the image file size distribution as a cumulative distribution in Figure 4.3. From this Figure, we note that most (about 94%) of the GIF images are smaller than 50 Kbytes. By contrast, about 60% of the JPEG images are smaller than 50 Kbytes. Since GIF images may be animations where a number of individual images are packed into a single GIF image file, we analyzed the number of animations in GIF images. We found that about 8% of the GIF images contain more than one animation frame. 1% of the GIF images had more than 10 animation sequences per image. An obvious way to deal with large GIF files composed of numerous animations is to reduce or eliminate all but one of the individual animation frames.

Traditional human factors research [86] has shown that the response time for accessing a resource should be in the 1 to 10 second range for information to be useful. If the response time is longer than this range, the users tend to lose interest and go on to other things. For our work, we choose a response time of 5 seconds as our preferred latency. We estimate the file size limit for images to be served within 5 seconds in the Internet. Users at least use a 9600-baud modem to connect to the Internet. We compute the image size that can be served within a latency of 5 seconds over 9600-baud modem to be 5.9 KB. Images with file size equal or less than 5 KB take less than 5 seconds to download. For many Internet usage scenarios, it is therefore less interesting to transcode images less than 5 KB in size. We conclude that typically transcoding is not necessary for images that are less than 5 KB.

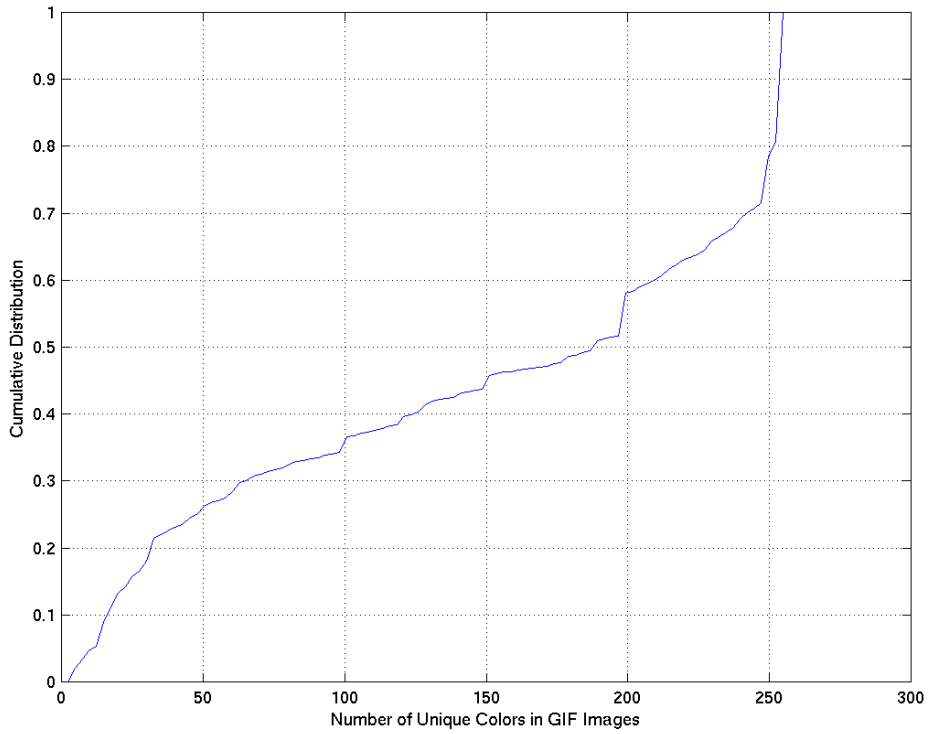

Figure 4.4: Distribution of Number of Unique Colors in GIF Images

4.4.2 Number of Colors

Next we analyzed the number of unique colors in an image. GIF compression is optimized for line drawings with a small number of colors. Hence, reducing the number of colors is a potential transcoding operation for GIF images. For the GIF images, we plot the number of unique colors in each image as a cumulative distribution in Figure 4.4. GIF specifies the number of bits required per pixel in an image in its screen descriptors. This restricts the number of unique color map choices to a power of 2, which is evidenced in the clustering of the number of unique colors (Figure 4.4). Also Figure 4.4 shows that about 20% of the GIF images have 256 unique colors, suggesting that these images may

be photographs (which are better suited to JPEG compression). On the other hand, the number of unique colors in an image compressed using a lossy compression technique such as JPEG not only depends on the image, but also on the precision of the decoder. Integer round-off errors introduced by the decoder add imprecision in the decoded color values. The exact number of colors does not provide much information about the original image and hence the number of unique colors for JPEG images is not explored here. JPEG is optimized for photographs and defines two color modes, TrueColor and GrayScale. For the images in our collection, about 2% of the JPEG images were GrayScale.

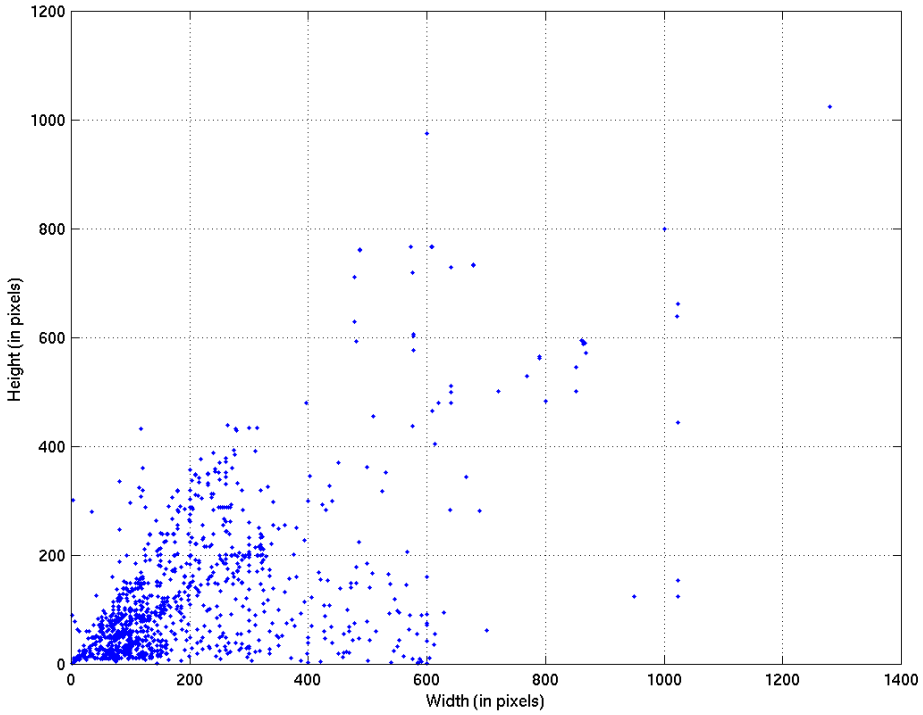

Figure 4.5: Image Spatial Size Distribution – GIF Images

4.4.3 Spatial Geometry Size

Scaling images is a popular transcoding operation. For scaling to be useful, the original images themselves should be originally above a threshold minimum dimension. Hence, we analyzed the spatial geometry characteristic of this set of Internet images. For the images in our collection, the spatial size of the images (width vs height) for GIF and JPEG images are plotted in Figure 4.5 and Figure 4.6 respectively. From Figure 4.5, we note that a significant proportion of GIF images are small (less than 180x180 pixels). Also, many GIF images are wider than taller.

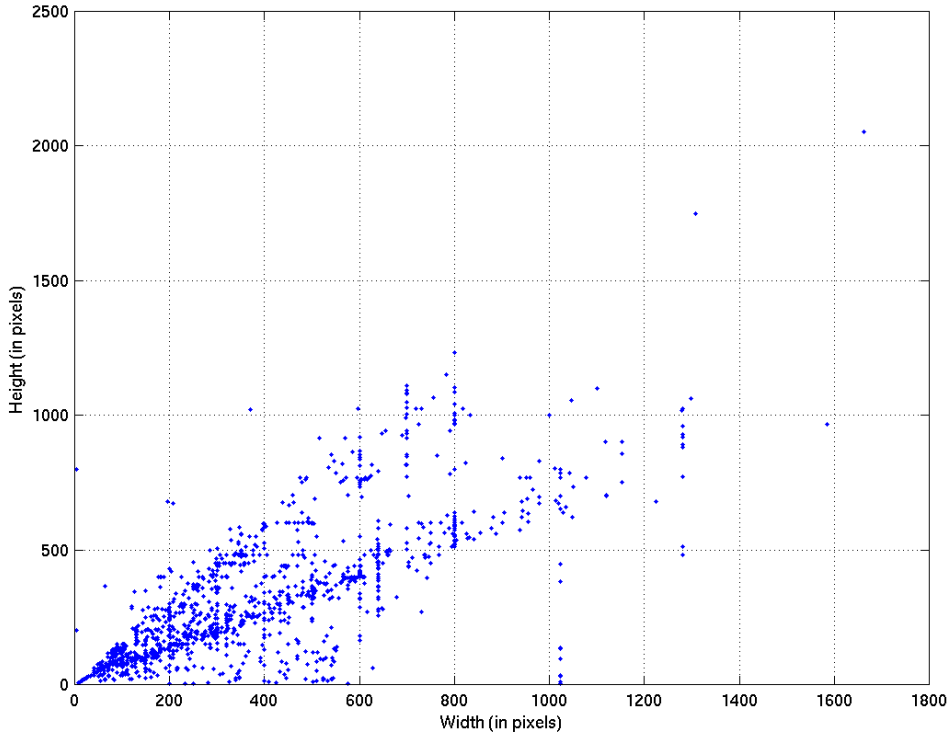

Figure 4.6: Image Spatial Size Distribution – JPEG Images

From Figure 4.6, we note that JPEG images dimensions are larger than GIF images dimensions. We used Bitmap Information Tool [87] to classify images types. For our collection, the distribution of images in the various categories is tabulated in Table 4.1. From Table 4.1, we note that about 37% of GIF images and 20% of the JPEG images are for categories other than true images.

Image Type	GIF Images (%)	JPEG Images (%)
Bullets	5.94	0.60
Lines	4.92	3.26
Icons	15.55	11.85
Banners	10.80	3.40
True Images	62.79	80.89

Table 4.1: Image Category Distribution

4.5 Image Transcoding Characteristics

In the previous section, we analyzed the static characteristics of images accessed on the Internet. In this section, we will analyze the transcoding characteristics of these images. For our study, we utilized ImageMagick [77] for transcoding the collected Images. We explored transcoding that reduce the spatial geometry (frequently referred to as scaling), the number of unique colors in an image, the JPEG compression metric as well as transcoding that change the image formats.

4.5.1 Reducing the Spatial Geometry

First we analyzed a transcoding operation that reduces the spatial size of an image, frequently referred to as scaling. For our experiments, we reduced the spatial geometry of all the images by a factor of 2 and 4 along each axis (which translates to 50% and 25% of the original image pixels). The resulting image file size as a percentage of the original image file size is plotted as a cumulative distribution in Figure 4.7 and Figure 4.8, respectively.

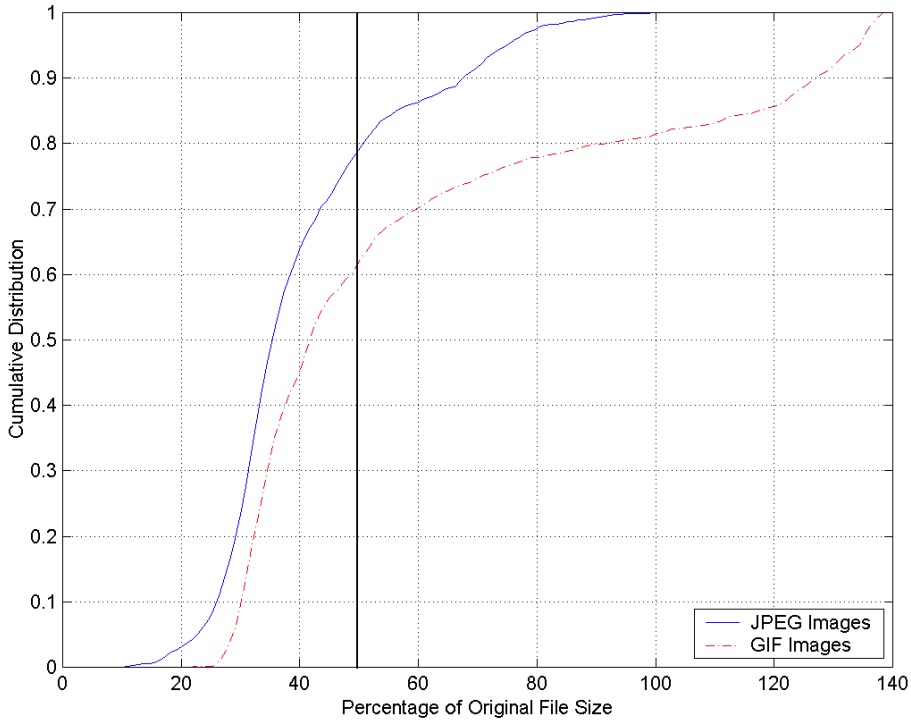

Figure 4.7: Reducing The Spatial Geometry by a Factor 2 Along Each Axis

From Figure 4.7 we note that about 79% of the JPEG images lost at least 50% of the image size for a transcoding that reduces the image geometry by a factor of 2 on each

axis. However, for the GIF images about 19% of the images actually increased in size compared to the original image (image transcodes to a size that is more than 100% of the original image size). About 60% of the GIF images saved at least 50% of the original image file size.

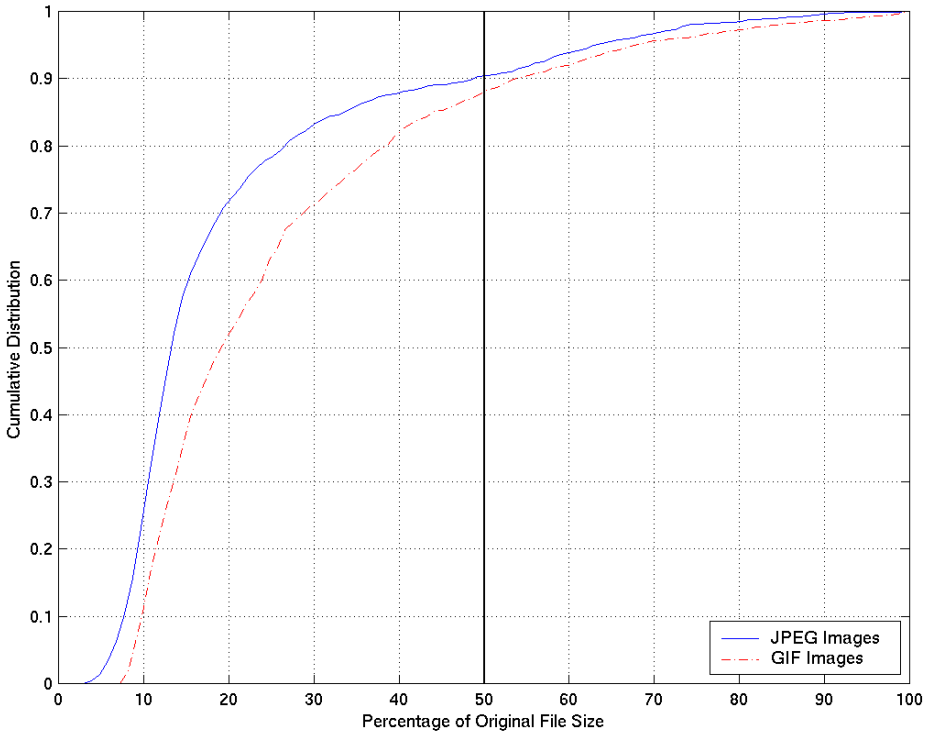

Figure 4.8: Reducing The Spatial Geometry by a Factor 4 Along Each Axis

Transcoding that reduce the spatial geometry by a factor of 4 along each axis are shown in Figure 4.8. We note that about 87% of the GIF images and 90% of the JPEG images lost at least 50% of the original image file size.

A few GIF images (19% of the GIF images for a transcoding that reduces the image by a factor of 2 along each axis) transcode to a size that is larger than the original image

file size. GIF [32] uses a variation of the LZW [49] compression algorithm to reduce the number of bits required to store frequently occurring color map values. In the GIF algorithm, pixels can be represented by 3 to 12 bits depending on the frequency of occurrence. A transcoding that reduces the spatial geometry tends to increase the number of unique colors in an image as original color values are replaced by a new average color value. Since it takes more bits to represent less frequent pixels, introducing less frequent color values with low occurrence frequency leads to an increase in the output image size; against our goal for transcoding an image to reduce its size. The following example will illustrate this problem clearly. One of the GIF images in our collection of geometry 360x150 (file size 3509 bytes and 12 unique colors) was transcoded to a GIF image of geometry 180x75. The new image was 4005 bytes and had 132 unique colors. The popular color (which will be represented by 3 bits) occurred 90.88% of the time in the original image, while it only occurred 85% of the time in the transcoded image. Subsequently, reducing the number of unique colors in the transcoded image to 12 produced an output image of size 2476 bytes. The images have to be transcoded to a sufficiently small spatial size to overcome this effect. This phenomenon can be observed by the increase in the number of images that show space saving between a transcoding that reduces the spatial geometry by a factor of 2 (60% in Figure 4.6) and a transcoding that reduces the spatial geometry by a factor of 4 (87% in Figure 4.7). Another way that overcomes this problem is to reduce the number of unique colors to the original number after scaling an image. Reducing the image spatial geometry (both by a factor of 2 and 4

along each axis) is a productive transcoding for JPEG and GIF images as at least 50% of the images lost at least 50% of the file size. However, in the case of GIF images, reducing the image spatial geometry by a factor of 2 along each axis need to be followed by a color reduction operation to return the number of unique colors in an image to the original value as explained previously.

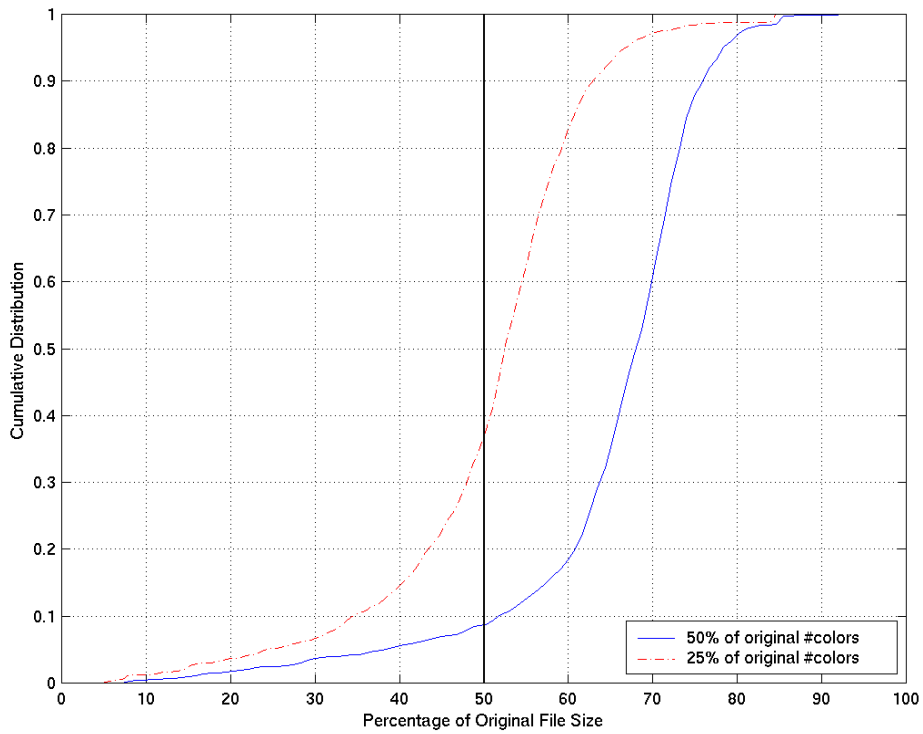

Figure 4.9: Reducing the Number of Unique Colors in GIF Images

4.5.2 Reducing the Number of Unique Colors

Next we analyzed transcoding operations that reduce the number of unique colors in an image. For the GIF images in our collection, we reduced the number of unique colors for

all the images to 50% and 25% of their original values. The resulting image file size as a percentage of the original image file size is plotted as a cumulative distribution in Figure 4.9.

From Figure 4.9 we note that for a transcoding that reduces the number of unique colors to 50% and 25% of the original unique colors, about 8% and 38% of the images lost 50% of the original image file size, respectively. Reducing the number of unique colors in a GIF image by 50% does not appear to save much in file size whereas reducing by 25% provides better results. However, this type of transcoding does not increase the output file size like the previous operation. This phenomenon of color value distribution change affecting the LZW compression was noted earlier in Section 4.5.1.

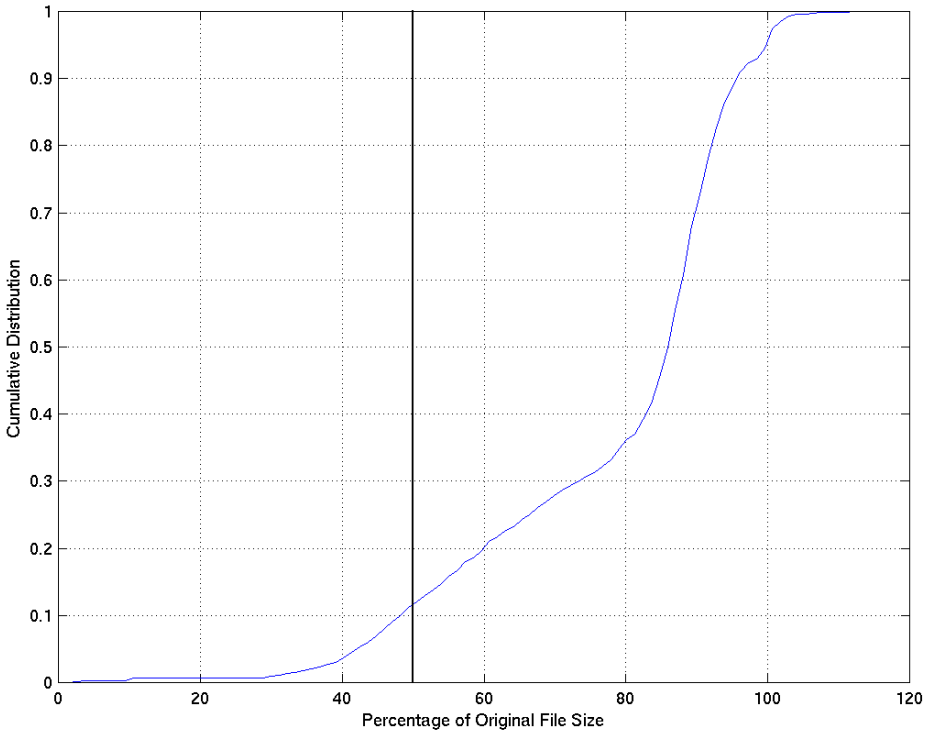

Figure 4.10: Converting a TrueColor JPEG Image to a GrayScale JPEG Image

From Figure 4.10, we note that about 12% of the JPEG images lost 50% of the image file size. Since less than 12% of images lost 50% in image file size, converting JPEG TrueColor images to GrayScale images does not appear to be a productive transcoding operation, especially since its cost in terms of time is high compared to other operations. In addition, a few of images transcoded to a size that is larger than the original image file size as shown in Figure 4.10. For JPEG images, it is clear that converting from TrueColor to GrayScle is not helpful; therefore, we decided to discard this type of transcoding from our system. As for GIF images, reducing the number of unique colors in an image by 25% or less may guarantee a significant reduction in file size.

4.5.3 Changing the Image Format

Next we explored a transcoding that changes the format of the images themselves, GIF to JPEG and JPEG to GIF format. Even though JPEG compression is better suited for full color photographs and GIF format is better suited for line drawings, transcoding among the formats is a popular image transcoding operation [1] and hence we explored its characteristics.

One variable in converting from a lossless compression technique such as GIF to a lossy compression algorithm such as JPEG is the choice of the compression metric for the JPEG images. For our experiments, we transcoded the GIF images to JPEG images of Quality Factor values of 25, 50 and 75. The resulting image file size as a percentage of the original image file size is plotted as a cumulative distribution in Figure 4.11. We note

that transcoding to JPEG images of Quality Factor values of 25, 50 and 75 provide at least 50% saving in size for 70%, 55% and 40% of the GIF images respectively. Hence transcoding GIF images to JPEG images is considered as a productive transcoding for GIF images as at least 50% of the images save at least 50% in file size.

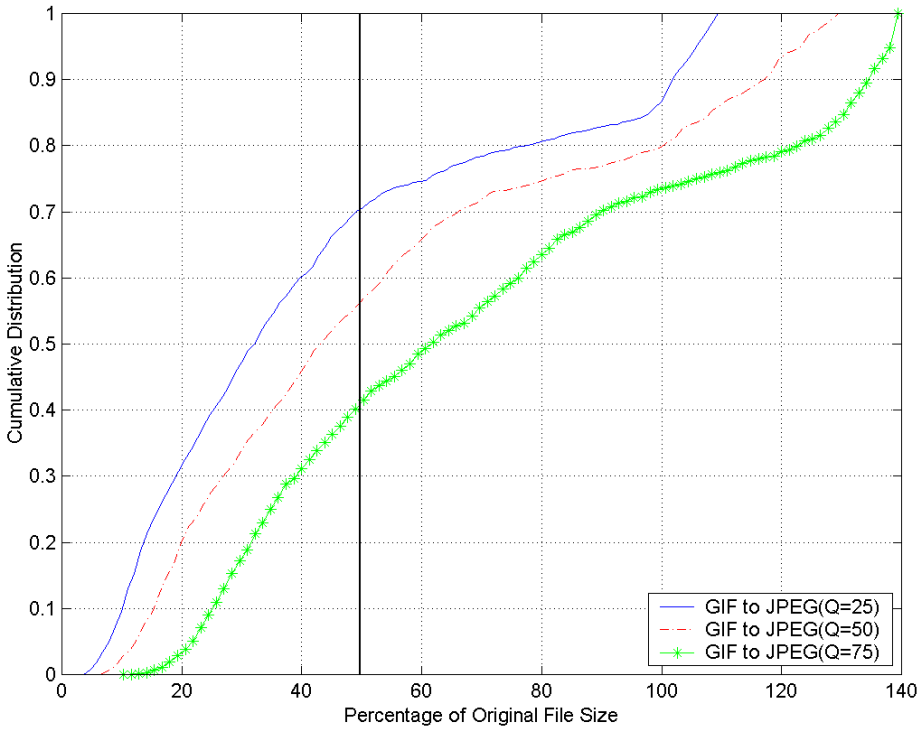

Figure 4.11: Converting From GIF to JPEG Format

From Figure 4.11, we also note that transcoding GIF images to JPEG images of Quality Factor values of 25, 50 and 75 can lead to an increase in the image file size for 13%, 20% and 27% of the GIF images respectively. Since an increase in file size is against our goal for performing a transcoding, such a transcoding needs to be checked whenever it is applied. So if this type of transcoding produces larger file size than the

original file size the format conversion should be discarded.

GIF images are better optimized for line drawings with few colors. On the other hand, JPEG is optimized for photographs. Hence a GIF image with many colors might be expected be better compressed as a JPEG image. To test this hypothesis, we analyzed the transcoding characteristics of transcoding GIF images with over 250 unique colors to JPEG images. The resulting image file size as a percentage of the original image file size is plotted as a cumulative distribution in Figure 4.12.

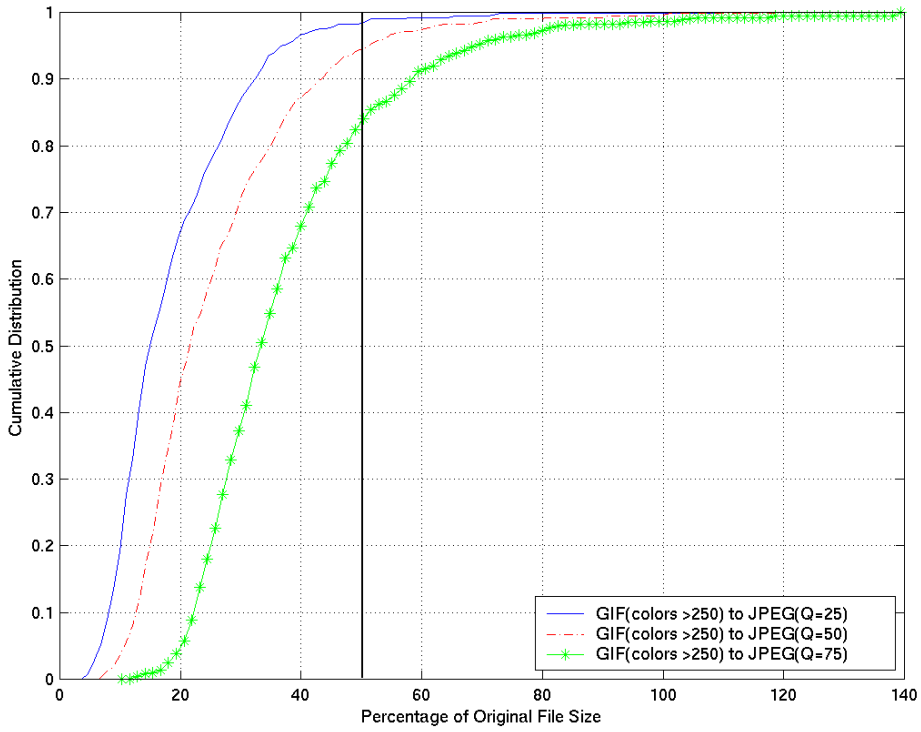

Figure 4.12: Converting From GIF (#Colors > 250) to JPEG Format

From Figure 4.12, we note that such a transcoding to JPEG images of Quality Factor values of 25, 50 and 75 provides at least 50% saving in size for 99%, 95% and

84% of the GIF images respectively. Hence transcoding GIF images with number of unique colors greater than 250 to JPEG images is considered as a productive transcoding for GIF images as at least 50% of the images save at least 50% in file size. Usually, the GIF images that contain a high number of unique colors are nature art views. This type of images is better compressed as a JPEG image. This might explain the effectiveness of such a transcoding operation. However, there are a few images which increased in their output file size. Even though the number of unique colors is over 250, the images are still overwhelmingly line drawings and hence are better encoded as GIF images.

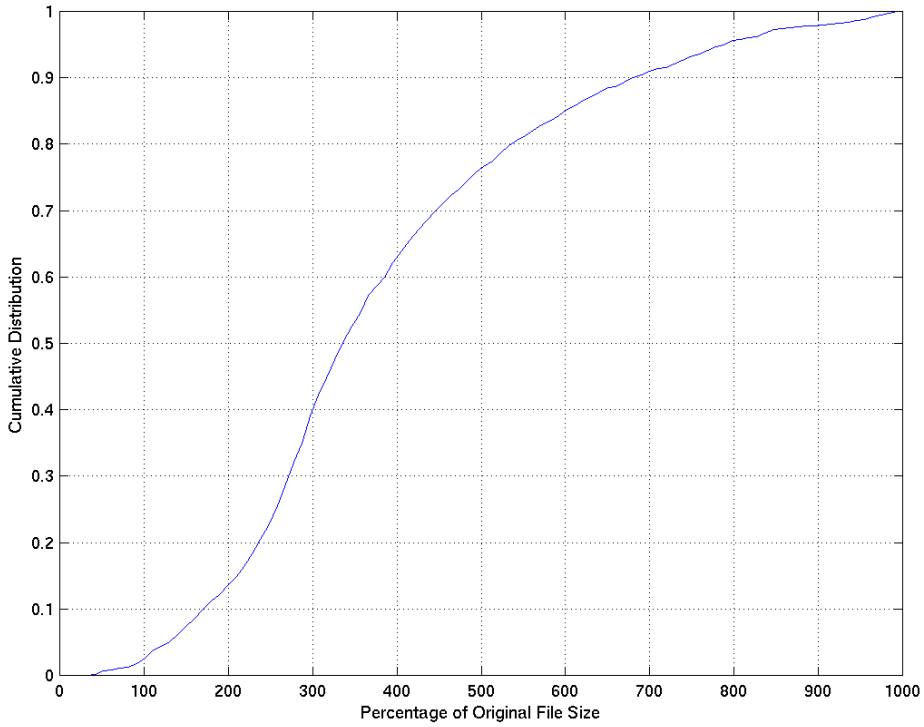

Figure 4.13: Converting From JPEG to GIF Format

Next we analyzed the transcoding from JPEG to GIF images. Conversion from

JPEG to GIF is straightforward. The resulting image file size as a percentage of the original image file size is plotted as a cumulative distribution in Figure 4.13. From Figure 4.13, we note that transcoding JPEG to GIF images is not a productive transcoding and is not recommended to be applied. Most images (98%) are larger than the original JPEG images.

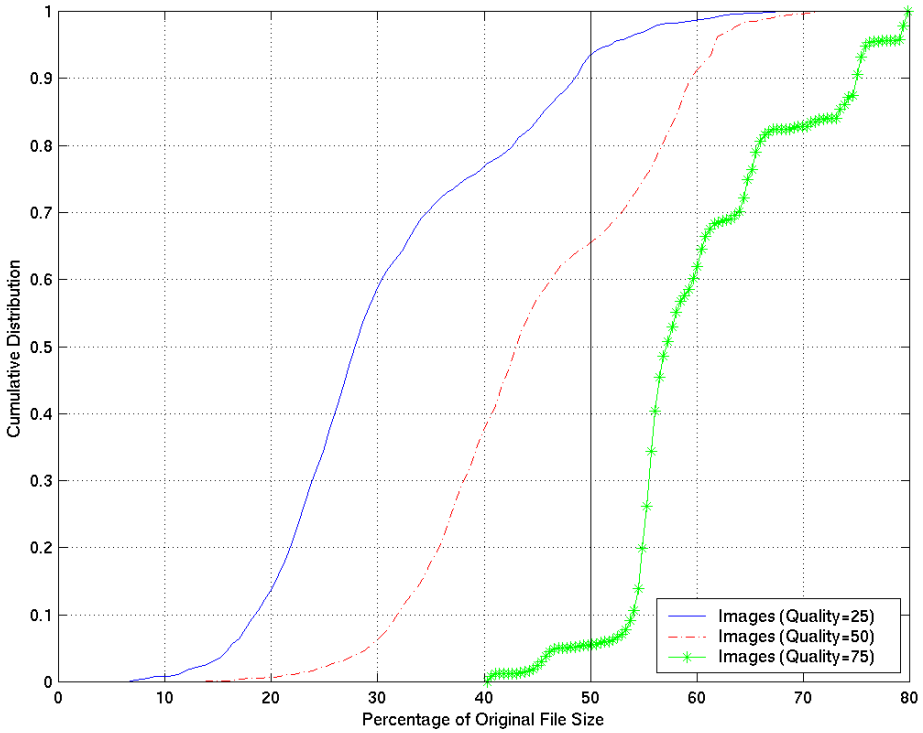

Figure 4.14: Changing JPEG Image Quality Factor

4.5.4 Changing the JPEG Compression Metric

Next we explored transcoding that change the level of "lossiness" in an image. Lossy compression techniques such as JPEG define a way to control the amount of lossiness using a compression metric. The compression metric utilizes quantization tables such that

images lose least perceptible artifacts first. Hence, changing the compression metric is intuitively a good transcoding metric for JPEG images.

For our experiments, we reduce the image Quality Factors to 25%, 50% and 75% of the original image Quality Factor values. The resulting image file size as a percentage of the original image file size is plotted as a cumulative distribution in Figure 4.14. From Figure 4.14, we note that for a transcoding that reduces the image Quality Factor to 25%, 93% of the JPEG images lost 50% of the file size. For a transcoding that reduces the image Quality Factor value to 50%, we note that 66% of the JPEG images lost 50% of file size. Finally, for a transcoding that reduces the image Quality Factor value to 75% we note that 5% of the JPEG images lost at least 50% of file size. Hence, we conclude that reducing the image Quality Factor to 25% and 50% are a productive image transcoding.

In summary, we note that the JPEG compression metric is a productive transcoding operation for JPEG images. In addition, for JPEG images, reducing the spatial geometry (scaling) is a productive transcoding. On the other hand, the only productive transcoding for GIF images are converting GIF images to JPEG images, a transcoding that reduces the spatial geometry by a factor of 4 along each axis and reducing the number of unique colors by 25%. None of the other image transcoding techniques explored provides significant savings for GIF images.

4.6 Key Observations

4.6.1 Effect of Imaging Software

After applying different operations on all images, we observed several things. First, with some images, the compression ratio value was very high. It sometimes exceeds 90%, even with effect operations such as *blur*, which was not expected to reduce the file size of images. At this point, some suspicions arose regarding the compression ratio rate for all images. We wondered if there is hidden factors that cause this huge reduction or is it the images characteristics that dictates the reduction size. Thus, we did several investigations to find out the following: Firstly, were we getting the right compression ratio? Secondly, if not, what is the factor that causes such huge reduction?

Out of curiosity, we used ImageMagick to open an image file and resaved it without performing any operation. The resulting compression ratio was the same as the one we got by applying *blur*. To explore this further, we used different image processing software just to open and save an image. The result is shown in Table 4.2.

The Original Image	ImageMagick	Paint Shop Pro	Microsoft Photo Editor
664 Bytes	132 Bytes	159 Bytes	911 Bytes

Table 4.2: Saving Identical Image Using Different Software

The result in Table 4.2 indicates that each image processing software uses different techniques for storing images. This is why we got three different file sizes. Subsequently,

our compression ratios may not be correct since most of the images over the Internet are created by different software products. For example, if we create the same image using the three programs listed above, we would get three different compression ratios. Since the compression ratio value is very important aspect in our experiment, it was mandatory for us to eliminate this factor in order to get more accurate results.

The same results hold for JPEG images. The way in which an image has been originally created or scanned is an important factor of the image file size i.e. the software used for creating or scanning the image could affect its file size. To verify this, we performed a simple test (Figure 4.15), saving the same image using different image processing software. As shown in Figure 4.15, the file size depends on the software used. This test has been applied on 4 different images. Another test (Table 4.3) creates the same image using different software, also resulting in different image file sizes.

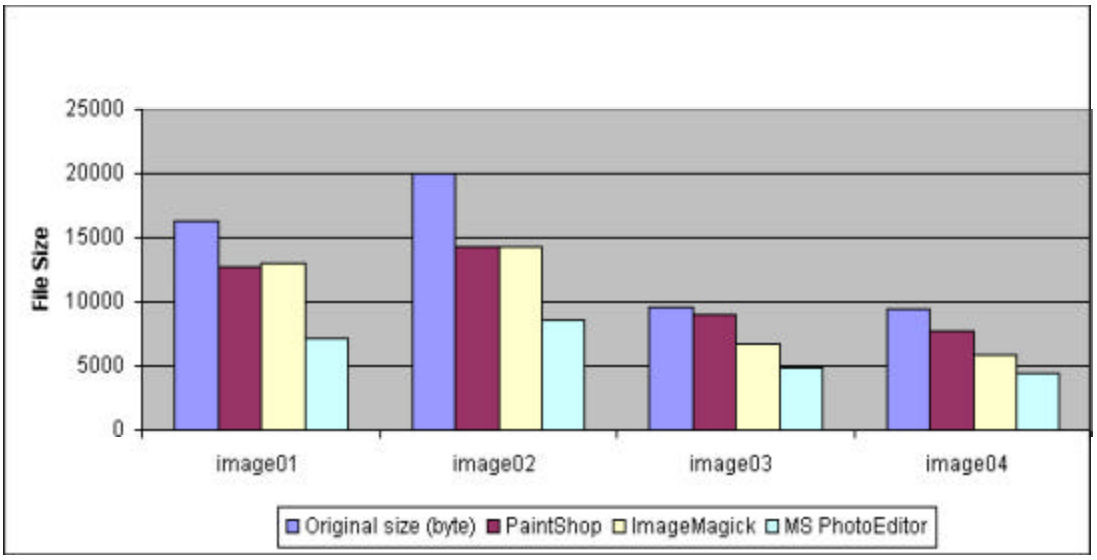

Figure 4.15: Saving Identical Images with Different Image Processing Software

Image Name	Paint Shop Pro	ImageMagick	MS Photo Editor
Test	7781 Bytes	7531 Bytes	9024 Bytes

Table 4.3: Creating Identical Image with Different Image Processing Software

Yet another test was performed to scan the same picture using different scanners. This also resulted in different file sizes. To eliminate the software factor and obtain a more predictable compression ratio, the same software should create all images under test. While this will not reflect the reality on the Internet, we choose to ignore the variability introduced by various software packets for the time being. We therefore resaved all experimental images by using ImageMagick. By doing this, we eliminate this factor and the resulting compression ratio will be more predictable. The results reported in the earlier sections are all based on these resaved images.

By sorting all images with their information in ascending order by the compression ratio parameter, we observed that images coming from the same web site have similar compression ratios. This clearly seems to indicate that the designer who designs the web site uses the same software for creating all images on that web site. Subsequently, when we applied the same operation on these images we got similar compression ratios. The software used for creating an image is clearly one of the factors that impact the compression ratio.

4.6.2 Sequence Order of Transcoding Operations

After determining the most effective transcoding operations, we performed them in

different sequences to see whether the reduction in file size would be the same or not. By doing so, we can decide in which order we should perform these operations in order to get the best reduction. For instance, we applied scaling, color reduction, and convert from GIF to JPEG operations in different order. Based on the results that we got after applying the operations in these order, it turns out that the reduction in file size is the same for each group. Thus, it does at first sight not matter in which order we perform the operations. Table 4.4 shows the results of applying these operations in different order on one image. The file size of the original image is **48.15** KB.

Sequence A-1	**Sequence A-2**	**Sequence B-1**	**Sequence B-2**	**Sequence C-1**	**Sequence C-2**
Scaling	Color Reduction	GIF to JPEG	Scaling	Color Reduction	Scaling
Color Reduction	Scaling	Scaling	GIF to JPEG	Scaling	Color Reduction
GIF to JPEG	GIF to JPEG	-	-	-	-
9.57 KB	9.57 KB	8.50 KB	8.50 KB	10.79KB	10.79 KB

Table 4.4: Sequence Order of Transcoding Operations

For transcoding JPEG images, we apply two operations “scaling and quality reduction”. Based on our tests, applying these two operations in different order produce similar result in file size reduction. Hence, we concluded that applying a set of operations in different order does not have an effect on file size reduction.

4.7 Summary

In order to answer the question whether transcoding operations will be worthwhile, we need to understand the nature of typical Internet images and their transcoding characteristics. We focused our attention on transcoding that customize an image for file size savings. We analyzed a large number of GIF and JPEG images collected from different Web sites such as education, sports, food, news, stock, etc. We analyzed the characteristics of the collected images and showed that most of the GIF images (about 94%) are smaller than 50 KBs. About 37% of these GIF images appear to be bullets, icons, lines, or banners. We found that about 8% of the GIF images contain more than one animation frame. 1% of the GIF images had more than 10 animation sequences per image. On average JPEG images are larger than GIF images, 40% of the JPEG images are larger than 50 KBs. About 20% of these JPEG images appear to be bullets, icons, lines, or banners. Our evaluation of image transcoding showed that there is significant opportunity for sophisticated transcoding of JPEG images. We showed that for JPEG images, the JPEG compression metric and a transcoding that reduces the spatial geometry are productive transcoding operations. Since the compression metric loses visually imperceptible information first, it is a good transcoding that reduces image file size, sacrificing as little visual information as possible. We also showed that transcoding techniques for GIF images such as color reduction, conversion from GIF to JPEG format, and scaling have the potential of actually decreasing the original image file size.

Chapter 5

AN IMAGE TRANSCODING PROXY

5.1 Introduction

There is a growing diversity of client devices that have access to the Internet. Small handheld computers are one example of Internet client devices that become more crucial in our daily lives. A handheld device equipped with a browser and a wireless connection can provide an opportunity to connect to the Internet at any time from anywhere. Such capabilities will increase the usability of PDAs tremendously by providing access to numerous information services; for example travel guides, entertainment advice, the latest news, flight schedules, even driving directions. Unfortunately, the devices that have limited communication, slow CPU, small storage, and small display size cannot handle much of the content on the Internet. Screen size and bandwidth limitations in particular require special attention, because they most directly affect the user's experience. For example, a recent study [75] on the effect of screen size and low bandwidth on completing browsing-related tasks shows that users with small display and low bandwidth

follow links less frequently than their counterparts who were furnished with larger displays and higher bandwidth, and that their success rate was lower. Accordingly, universal access to multimedia content has become increasingly important. Research on proxy servers for mobile computers has been conducted in recent years. These proxy servers distill image files aiming at only one hardware platform. Therefore, they cannot efficiently support various kinds of mobile computers of different display size, resolution, and network bandwidth. Another problem is that they do not have information about the resource of mobile computers. Consequently, the proxy server distills images into a fixed magnitude regardless of the size of the requesting PDA display and available bandwidth; and then transmits the same distilled image to every mobile device. Transmitting the same distilled image to a diversity of client devices that have different size screen, resolution, and network bandwidth is clearly suboptimal. Moreover, the current proxy servers for mobile computers assign a cache to each mobile user in order to distill images according to the specification predefined by the users. This mechanism prevents the users from sharing the image data saved in their caches, which leads to low efficiency. In order to improve the utility of and the image delivery to a wide range of client devices, we propose an image transcoding proxy for distilling inline images. The transcoding proxy is designed to take into account the bandwidth and display limitations while at the same time maintains the best possible quality of the delivered images. The system uses the mobile device characteristics as input parameters and automatically adapts to a dynamically changing bandwidth on the proxy-client link. Our implementation has

revealed that the proposed image transcoding proxy significantly improves the transmission speed of the existing proxy servers through customized distillation, and provides mobile computer users with a webpage at comparable speed and quality to desktop PCs. The efficiency and scalability can also be enhanced by caching both the original and the transcoded image files. This technique enables many client devices with the same characteristics and communication limitations to share the transcoded images if they are available in the cache without going through transformation operations. Otherwise the transformation would be necessary.

Workstation	Midrange PC	Laptop	HHC	PDA
Dimension: 150 x 190	113 x 143	75 x 95	60 x 76	39 x 49
File Size: 19 KB	11 KB	5.7 KB	2.3 KB	0.2 KB
Color: 24-bit RGB	24-bit RGB	256 colors	4-bit gray	B/W

Figure 5.1: Image Customization

5.2 Image Transcoding

Our experiments in chapter four and several surveys done on web traffic have consistently confirmed that GIF and JPEG images present the highest percentage of the Internet

content. Accordingly, it would be beneficial to process these image formats, on the fly, before transmitting them to mobile hosts. Image scaling, color reduction, type conversion, or lossy compression could greatly reduce the web page download time and match images to the device and network constraints. Some display or bandwidth situations may require the handheld device to receive a thumbnail of the desired image. As shown in Figure 5.1, the images are customized along the dimensions of image size, quality, and color in order to adapt them to the client devices. This kind of data stream customization is commonly known as "*image transcoding*".

5.2.1 Transcoding Functions

Our transcoding proxy provides a set of transcoding functions. It might be necessary to perform one or all of them to cope with a range of bandwidths and display size limitations. Actually, the system processes GIF and JPEG images with a different set of transcoding functions. For example, reducing the quality is not applicable to GIF images while it is one of the effective operations that reduce the JPEG image file size. In this section, we will introduce the transcoding functions that are mainly used by our transcoding proxy.

Image Scaling

Scaling is defined as reducing the size of the image by reducing the dimension parameters (width and height) of the image. There are two situations in which it makes sense to *scale* an image. A scaled image can easily be an order of magnitude smaller than the original

image. This will be helpful in a situation where the bandwidth constraints of the exchange demand a smaller file size for transmission. The transfer of the scaled image can take advantage of its smaller file size and, at the displaying end of mobile hosts there is no apparent degradation in image quality. Another situation where we might want to scale an image is where the mobile host's display size is smaller than the requested image. In this case, a scaling operation can fit the image on a small screen. Under some circumstances, more aggressive scaling could be achieved by scaling an image to a thumbnail.

Type Conversion

Transformation of an image from one format (such as GIF) to another (such as JPEG) may be useful in some circumstances. As we mentioned in chapter four, GIF and JPEG are designed to accommodate different type of graphics. GIF is usually best for images with sharp-edged areas of flat color. Examples include line drawings, simple icons, buttons, and cartoon pictures. JPEG is usually best for images that involve smooth color gradients; typical examples are photos and naturalistic artwork. Therefore, converting images to the correct format most often reduces file size without sacrificing image quality. Converting from GIF to JPEG is the type conversion we most often follow in our system. However, in some situations, where the mobile host does not support JPEG images, we convert any JPEG image to GIF image, and transcode it aggressively.

Color Reduction

As shown in the previous chapter, reducing color depth of any GIF image can significantly reduce the image file size. Most popular graphics programs can count and display the number of unique colors actually used in a GIF image. When the number of colors used is significantly less than the current color depth, then a color depth reduction can occur which will reduce the file size without having any effect on the quality of the image. In other words, it makes no sense whatsoever to have a GIF image that contains a 256-color palette (8-bit) if the number of colors used in the image is only 120. In such a case, the color depth should be reduced to 128 colors (7-bit). Reducing color depth and the number of colors in GIF images might be necessary when we deal with a mobile host device that supports very few numbers of colors. Consider the following example. A mobile host that supports only 16 colors (4-bit) requests a GIF image that contains 256 colors (8-bit). It is clearly inappropriate to send this image without reducing the color depth and the number of colors. Reducing color from 256 colors (8-bit) to 16 colors (4-bit) will reduce the file size to roughly half of the original. Otherwise, more data will be transferred over the wireless link and the color reduction will happen when the image is rendered, so we get the same image but transferred more data. But what happens when we actually reduce the number of colors used in the image? Well, the image quality may suffer.

Image Compression

There are basically two types of compression methods: lossy and lossless. Lossy

compression creates smaller files by discarding some information about the original image. It removes details and color changes it deems too small for the human eye to differentiate. Lossless compression, on the other hand, never discards any information about the original file. These two types of compression techniques are interchangeably used by the transcoding proxy depending on the available situation. An obvious *ideal* situation is one where the image can be transferred as the author had intended; the bandwidth and display constraints of the situation allow the image to be sent 'as is' or to be compressed using a lossless technique. If the bandwidth and display constraints demand a smaller image file size for transfer then it may be necessary to use a more efficient compression technique i.e., one that involves loss of information, such as lossy compression.

JPEG Compression Metric

With JPEG images, we can easily specify the amount of "lossiness" using metrics such as the compression metric (also referred to as the JPEG Quality Factor by Independent JPEG Group [50]). The Quality Factor value can be adjusted based on the current condition of the network bandwidth. If the available bandwidth is low then reducing the Quality Factor value will produce a small file size. As we mentioned in the previous chapter, reducing Quality Factor is a trade-of between image quality and file size. Aggressively reducing Quality Factor to reduce size can result in an image of such low quality that it becomes useless. Therefore, we should not go beyond the threshold that is obtained from [83].

5.3 System Architecture

The key to applications in a mobile, wireless, very heterogeneous environment is the proxy architecture, which uses a proxy as a smart intermediary between traditional servers and heterogeneous mobile clients. The fundamental driver for this architecture is the inability of (nearly all) servers to handle the incredible variation in software, hardware and networking capabilities of mobile clients. Through various forms of data transformation, the proxy can dynamically translate between the needs of clients and the formats and abilities of the servers. The basic architecture of the transcoding proxy is shown in Figure 5.2.

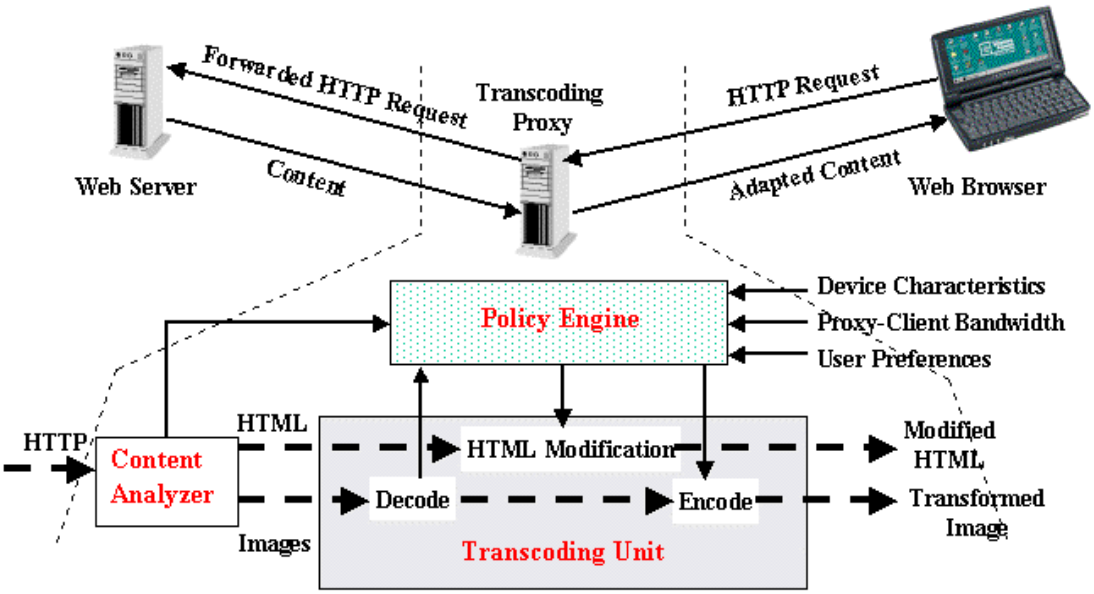

Figure 5.2: The Image Transcoding Proxy Architecture

A client communicates exclusively with the *proxy*, which is the main component of our system architecture. Our proxy is located logically between the client and the server.

As in any heterogeneous network environment, the proxy should be placed near the boundary between strong and weak connectivity, e.g., at the base station of a wireless mobile network. The proxy's role is to retrieve content from Internet servers on the client's behalf, determine the high-level types of the various components (e.g., images, text), and determine which distillation engines must be employed. When the proxy calls a distiller, it passes on information such as the client display characteristics, available network bandwidth, and user preferences. Based on these parameters, different transcoding modules are employed to generate versions of the content that best meet the client capability. Finally, the proxy sends the transformed versions of the content to the client. A cache that stores these client specific versions of content is used to enhance response times. In the following sections, we describe these processes in detail.

5.3.1 The Prime Components of the Transcoding Proxy Architecture

The transcoding proxy is constructed by integrating a transcoding subsystem into an HTTP proxy [28]. The transcoding subsystem consists of two primary components: the policy engine and the transcoding unit. The transcoding unit is responsible for modifying the data streams (i.e., HTML documents and GIF [32] images) that are being sent back as responses to the client Web browser. The decision concerning about what transcoding module (i.e., the transcoding function along with its parameters) to use and how much to transcode is made by the policy engine based on a number of situations, including: the current available bandwidth on the client-proxy link, the characteristics of the client

display capabilities, and the user preferences concerning the preferred rendering of the data. One example of the user preferences is the quality factor of images. The user can easily interact with the transcoding proxy to dynamically change the tradeoff between image qualities and download time. The user basically provides a constant value range from 1 to 100, which is then supplied to the transcoding proxy's policy engine. Eventually, the constant value is translated onto a transcoding parameter that is passed to the data transcoding unit. The policy engine typically generates a collection of transcoding vectors, which control the amount and forms of transformation performed by the transcoding unit. For instance, the scaling vector determines how much an image is scaled down. Also, the number of colors or the color depth in an image can be reduced, or a colored image may be converted to a grayscale image. Based on the policy engine's decision, the transcoding unit may perform one or a combination of transcoding operations. To make the transcoding mechanism more efficient, the policy engine should consider every situation mentioned previously to determine under what circumstances transcoding is able to improve the response time. Image transcoding normally introduces some delay, which must be balanced by the reduction in transmission time due to compression. However, as presented in [76], when the bandwidth of the proxy-client link increases, there comes a point at which it is no longer beneficial to transcode since the reduction in response time due to aggressive compression decreases as a function of the bottleneck link's bandwidth, while the transcoding time remains constant.

5.3.2 Content Analyzer

The authored content is analyzed to extract information that will be useful in transcoding and modification. Each content item is analyzed to determine the characteristics of data (e.g., byte size of the images, image dimensions, image types). From this content analysis the following information is determined:

- Content size in bytes. For images, this is simply the size of the file.
- Image dimension (height and width).
- The color depth and the number of colors in an image.
- Image type, purpose, and format.
- Detect particular content items such as JavaScript, Java applets, VBScripts.

The semantics of the content items are determined in the context of the entire document. We currently analyze images to determine their type, purpose, and format. Images can be of several types ranging from black and white graphics to color photos and a combination of color graphics and photos. So far, we did not come up with a technique that tells us whether the image is simple graphics or photos. The only information that the content analyzer can provide about the image type is monochrome, grayscale, or true color. We also detect the purpose of the image on the page. For example, we detect if an image is related to the content or if it is just advertising image. Furthermore, the image format (e.g., GIF, JPEG, PNG) is determined in order to select the right decoder and encoder. We detect the format and the type of an image by using functions of an image

processing toolkit [77]. As for image purpose, it is detected by parsing the HTML document tags such as (background, i.e., <body backgr=…> or advertising, i.e., <body adv=…>). This analysis allows us to improve image transcoding by selecting policies according to image type, purpose, and format.

5.3.3 Input Parameters

Our system considers only three types of input parameters: display characteristics, network bandwidth, and user preferences. These parameters are provided to the policy engine in order to generate a set of transcoding parameters that determine the amount and type of image transformation need to be done by the transcoding unit. Currently, the system considers the following input parameters:

1. *Display.*

   a) Size: width and height in pixels.

   b) Screen type: color/monochrome.

   c) Color depth: the number of colors supported by the display.

2. *Network bandwidth.*

Currently, the system is told the effective network bandwidth to the client. In the future, we plan to incorporate some mechanisms for sensing the actual bandwidth to the client.

3.*User preferences*

   a) Capabilities for displaying images: Yes/No.

   b) Image quality factor: (from 1 to 100).

c) Support animation: Yes/No.

d) Support JavaScript, Java applet, VBScript: Yes/No

e) Thumbnail image: Yes/No

f) Remove background and substitute advertising images: Yes/No

g) Compresses HTML documents: Yes/No.

There are a number of mechanisms that can determine the client device capabilities and resources. The User-Agent fields in the HTTP request header contain information about the browser and often the operating system. Windows-CE devices also specify the screen size, color depth and processor. Standardization efforts are under way to allow these request fields to contain more information about the client device. Many sites require users to login, or place cookies at the user allowing client capabilities to be retrieved from stored profiles. The client may also specify their capabilities explicitly through forms or applets. Our system follows a traditional approach for getting all these sources of information about a mobile host. Basically, the user provides all this information to the proxy server, which stores it according to the mobile host IP address. Since we have not reached the stage that enables us to sense the network bandwidth, the system is also provided externally with the available bandwidth parameter. In case the user does not provide any of this information, default information, which is resident within the proxy server, will be supplied to the policy engine.

5.3.4 Transcoding Scenario

Once the mobile host sets a specific proxy server as its proxy, all communication between the mobile host and the WWW servers is directed through this proxy. The setting can be made easily by assigning the IP address of the machine that runs the proxy to the mobile host browser. When the web browser of the mobile host is executed, the mobile host sends its profile to the proxy server, which is stored with the IP address of the mobile host. A default profile is used if the mobile host had set no values. When the proxy receives a request from the mobile host, it verifies first that the requested web page is in the cache. If not, the request is forwarded to the appropriate WWW server and the proxy waits for a response. When the proxy receives the response from the WWW server, it looks up the user profile stored with the IP address of the mobile host and feeds the policy engine with the user profile to generate the most suitable transcoding parameters. The transcoding unit then receives the transcoding parameters from the policy engine to perform the necessary transformation and ultimately delivers the transcoded images to the mobile computer. The transcoded image files and the original files are stored in the cache. If the image file exists in the cache, we verify first that the available transcoded version is appropriate for the mobile computer. If this is the case, then the image file will be transmitted directly to the mobile computer. Otherwise, the original image is retrieved from the cache and processed based on the user profile before being delivered to the mobile host.

Transcoding Image Files

While receiving an HTML page from a WWW server the HTML token parser parses it to extract references to inline images. The cache controller checks the images against its cache index and automatically starts prefetching those images that are not found in the cache. A cache entry is created for each image. The content analyzer checks the content-type and content-length information received from the server. If the content-length is small enough to be handled on the mobile host, the image file is sent to the mobile host unmodified. But if the image is larger than what can be handled by the mobile host, it is processed according to the transcoding vectors. As for the content-type information, it is used to determine the transformations that the image file has to go through. To display images on some thin devices, the proxy might need to reduce an images' color depth to 2-bit monochrome and thumbnail size. Also, advertisement and background images might be removed or substituted with text if the bandwidth is low. The original URL in the image tag is replaced with the URL of the modified image stored locally by the proxy and sent to the mobile host.

Transcoding HTML

We can reduce the transmission time by parsing HTML tags on the proxy side. We can eliminate all the tags that the mobile host does not support, or is not capable of handling. The user can easily decide not to receive any Java applets, JavaScripts, VBScripts, background images, or advertisement images. When such a preference is set, the HTML document to be transmitted is parsed and all the tags that reference any unwanted file is

eliminated before sending the document to the mobile host. This is specifically suitable for PDAs, which have very small disk space and low speed CPUs. For such severely resource constrained mobile hosts, the set of tags that it can handle may be so small that it is advantageous to strip of all unwanted tags at the proxy and encode the remaining tags using a few bits. Furthermore, the user can enable or disable an option for compressing HTML document to GZIP [78, 79] stream before being sent to the mobile host. This type of compression can reduce the HTML file size up to 70%.

5.3.5 Cache Scenario

When a mobile host requests a particular Web page, the proxy first checks the Web page's URL against the cache index. This check can produce one of two answers: missing or present. When the Web page is missing, the proxy forwards the request to the WWW server, waits for the response, creates a new entry in the cache index, and places the Web page in a directory. When a customized Web page is delivered to the client, the customized content is also stored in the cache. The cache associates the customized content with the parameters used for customization, thus effectively associating it with the specific client capabilities for which the content was customized. As we mentioned previously, the client capabilities are stored under a client IP address on the proxy side. Also, we have both the original and the customized content in the cache. When the Web page exists in the cache, the proxy first compares the capabilities of the client currently requesting the Web page with the parameters used for transcoding the images in the

requested Web page. If the current client capabilities match the parameters used for transformation, a customized copy for the requested Web page is delivered from the cache to the client. Otherwise a new customized version is generated from the original copy in the cache.

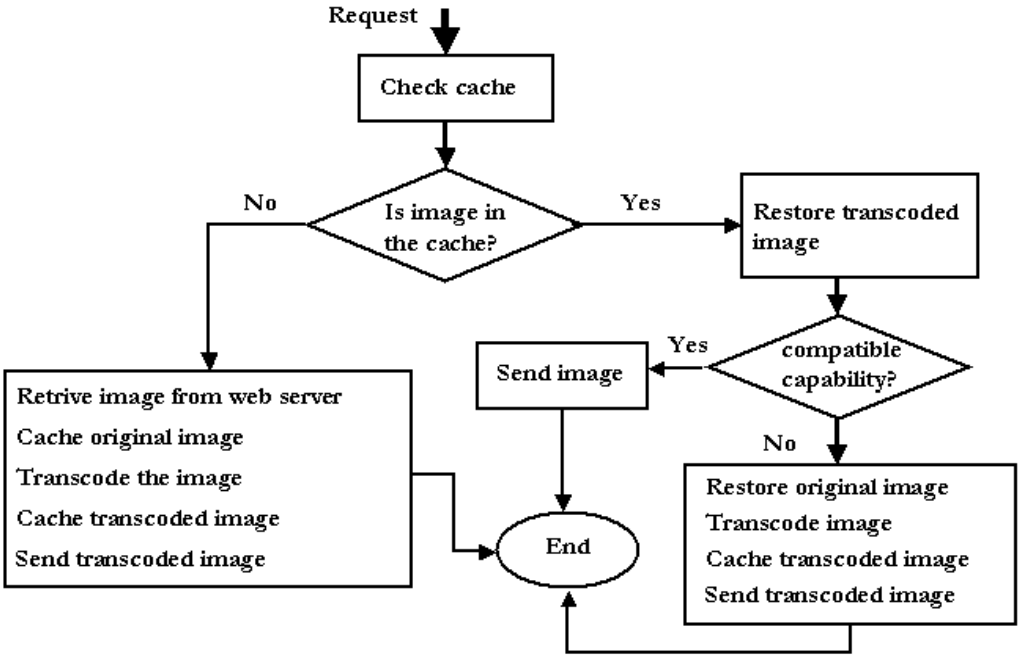

Figure 5.3: Cache Scenario

The cache implements the usual checks on the temporal validity of the documents. Since, for a busy site, the number of requests for a document is typically much larger than the number of different client devices, the cache can result in significant improvements in response times. Temporal variations in resources of the client, such as bandwidth, display capabilities, CPU resources, storage, etc., will reduce the cache-hit ratio. To effectively handle this, the cost of performing customization versus the variation in the resources will

need to be considered. As we shall see in the next chapter, the cost of performing customization is relatively low. Therefore, our system currently will perform customization if the resources for the requesting client differ from the cached versions. Figure 5.3 illustrates the cache scenario mentioned above.

5.4 The Algorithm of the Transcoding Proxy

In this section, we discuss the set of practical transcoding policies that we have implemented in our transcoding proxy system. These policies were developed incrementally by feeding back the lessons we learned throughout the development and evaluation of our transcoding proxy. The implemented policies adapt the transcoding to the client's device capability, user preferences, and current network bandwidth, but do not currently perform prediction of the image transcoding delay and output byte size.

Our current set of transcoding policies is summarized in several pseudocode figures, as we shall see. In Figure 5.4, our transcoding proxy first gets the IP address of the current client device and detects its profile as shown in line 1. Based on the IP address, the proxy will next check in line 2 to see if the user profile is available at the proxy. If this is the case the proxy will read the user profile (line 4); otherwise, a default profile will be read by the proxy (line 6). In line 7, the proxy checks whether the requested image is in the cache directory or not. If the image is in the cache, the proxy will retrieve the transcoded version and compare in line 9 the current and the previous client profile. If they are similar then the transcoded image will be sent to the client without transcoding.

Otherwise, the original image will be retrieved in line 12 by the proxy to go through the processing routines. If the requested image is not available in the cache then the image will be retrieved from the web server and stored in the cache (line 14 and 15). The proxy in line 16 checks the size of the requested image to see if the image is sufficiently large for transcoding. Images smaller than a threshold of 5KB are deemed to be not worth the savings in download time brought about by compression and hence are not transcoded. This threshold was obtained from our experiments in the previous chapter. Thus, if an image is less than the threshold the proxy will store the image in cache and send a copy to the client (line 25). Next, in line 17, our transcoding policy makes a distinction based on the input image format, such that input GIF images are transcoded differently than input JPEG images. If the input image is in GIF format, the proxy in line 18 invokes a special routine (see Figure 5.5) for transcoding this type of graphic file format. Similarly, if the input image format is in JPEG format, then in line 20 the transcoder for JPEG file format is invoked. In line 18 and 20, the image is transcoded according to the user profile received from the user. Stamping the transcoded image with the user ID (line 21) can guide the proxy to the user profile that has been used to customize this image. This can help the proxy to compare clients' capabilities and make its decision about whether to send the transcoded version. Ultimately, the proxy stores the transcoded image in the cache (line 22) and sends a copy to the user (line 23).

```
IP_Address = Get_User_IP_Address( );                                  Line 1
Status = Get_User_Profile( IP_Address );                                   2
if ( Status_Is_True )                                                      3
        Read_User_Profile( IP_Address );                                   4
else                                                                       5
        Read_Default_Profile( );                                           6
if ( Requested_Image_Is_In_Cache_Directory )                               7
        Retrieve_Transcoded_Image_From_Cache_Directory;                    8
        if ( Current_And_Previous_User_Profiles_Are_Equal )                9
                Send_Transcoded_Image;     exit;                           10
        else                                                              11
                Retrieve_Original_Image_From_Cache_Directory;             12
else                                                                      13
        Retrieve_Image_From_WWW_Server                                    14
        Store_Original_Image_In_Cache_Directory;                          15
if ( Input_Image_File_Size > 5_KBytes )                                   16
        if ( File_Format_Is_GIF )                                         17
            Output_Image = GIF_Transcoder ( Input_Image, User_Profile );  18
        else   /* input is JPEG */                                        19
            Output_Image = JPEG_Transcoder (Input_Image, User_Profile );  20
        Stamp_Transcoded_Image_With_User_ID;                              21
        Store_Transcoded_Image_In_Cache_Directory;                        22
        Send_Transcoded_Image_To_User;                                    23
else                                                                      24
        Send_Original_Image_To_User;  exit;                               25
```

Figure 5.4: The Transcoding Proxy Algorithm (Main Body)

```
Status = Is_Animated_GIF ( Image_File );                                  Line  1
if ( Status_Is_False )                                                          2
        Convert_From_GIF_To_JPEG_Format ( Image_File );                         3
        if ( Is_It_JPEG_Image )      /* the output file is JPEG */              4
                JPEG_Transcoder_Routine ( Image_File );   exit;                 5
if ( Status_Is_True AND User_Requested_to_Freeze_Animation )                    6
        Remove_All_Frames_Except_The_First_One;                                 7
if (User_Requested_Thumbnail_Image )                                            8
        Thumbnail_Scaling_Routine ( Image_File);                                9
else                                                                            10
        Image_Scaling_Routine (Image_File);                                     11
Color_Reduction_Routine (Image_File);                                           12
Return_Transcoded_Image_To_Main_Body;   exit;                                   13
```

Figure 5.5: GIF Transcoder Routine

Figure 5.5 illustrates the routine that is used to transcode GIF images. As shown in Figure 5.5, the routine first checks in line 2 to see if the image is animated or not. If the image is not animated, then the image in line 3 will be sent to a conversion routine (GIF to JPEG). Next, in line 4, we need to verify that the conversion to JPEG format is successful and the output file is JPEG. In some cases, the conversion may actually result in expansion of the image file size. In this case, the conversion routine returns the original image (GIF) rather than the converted image (JPEG). If the conversion is successful, then

the converted image (JPEG) in line 5 will be sent to the JPEG transcoder routine for further processing. If the image is animated or the conversion does not succeed, the routine will jump to line 6 directly. In line 6, if the image is animated and the user requests to freeze the animation then in line 7 all the image frames will be removed excluding the first one which will be distilled based on the user profile. The user also might ask for scaling the image to thumbnail; thus, if this is the case, the program will move to the thumbnail scaling routine in line 9 (see Figure 5.8). Otherwise, the image will be scaled based on the display dimension and current bandwidth (line 11). The color reduction function is called in line 12 (see Figure 5.7). Finally the transcoded image is returned to the main body which transfers the image to the client.

```
if ( Client_Device_Support_JPEG_Images )                    Line 1
        Convert_GIF_To_JPEG_Format;                              2
        if ( Output_Byte_Size > Input_Byte_Size)                 3
                Send_Original_Image;        /* return GIF image */   4
        else                                                     5
                Send_Transcoded_Image;    /* return JPEG image */   6
end                                                              7
```

Figure 5.6: GIF to JPEG Conversion Routine

In Figure 5.6, we check in line 1 if the client devices support JPEG images or not. If this is the case then the conversion will take place in line 2. Otherwise, the conversion

will not be applied and the control will be back to the Gif transcoder routine. If the conversion takes place then the output file will be checked to make sure it is smaller than the original file (line 3). If the output file size is smaller then the input file size the transformed image will be send (line 6); otherwise the original will be sent (line 4).

```
if ( Image_Is_Monochrome )   exit;                                   Line  1
if ( Client_Display_Is_Monochrome )                                        2
        Convert_Image_To_Monochrome;   exit;                               3
if ( Number_Of_Unique_Colors_In_Image < Min )   exit;                      4
Colors = Current_Bandwidth / Maximum_Bandwidth * ImageColors;              5
if ( Colors > DeviceColors)                                                6
        Colors = DeviceColors;                                             7
else   if ( Colors < Threshold )                                           8
              Colors = Threshold;                                          9
Reduce_Image_Colors( Colors );                                             10
```

Figure 5.7: Color Reduction Routine

Figure 5.7 shows the color reduction routine. First we make sure that the image is not monochrome (line 1). If the image is monochrome then the routine will exit quietly. In line 2, we check if the client display is monochrome. If it is true then the image will be converted to monochrome (line 3). If the number of unique colors in an image is less than the threshold, the color reduction routine will not be invoked, as shown in line 4. The formula in line 5 generates a new number of unique colors in an image based on the

available bandwidth. If the generated number is greater than the number of unique colors supported by client devices the number of colors will be reduced to the number of unique colors supported by client devices (line 6 and 7). If the previous condition is false then the generated number will be checked against the threshold (line 8). If it is less than the threshold then the color number in line 9 will be the threshold. Finally, reducing number of unique colors in an image is applied in line 10.

```
if ( Image_Width > 50 OR Image_Height  > 50 )                    Line  1
        Width_Factor = 50 / Image_Width;                                2
        Height_Factor = 50 / Image_Height;                              3
        if ( Width_Factor < Height_Factor)                              4
                Resize_Factor = Width_Factor;                           5
        else                                                            6
                Resize_Factor = Height_Factor;                          7
        X_Dim = Image_Width * Resize_Factor;                            8
        Y_Dim = Image_Height * Resize_Factor;                           9
        ScaleImage ( X_Dim, Y_Dim );                                    10
end                                                                     11
```

Figure 5.8: Thumbnail Scaling Routine

Figure 5.8 illustrates thumbnail scaling routine which is used to scale images with width or height above 50 pixels. In line 1, the width and height of the image will be checked. If one of them greater than 50 then the image will be scaled to a thumbnail size. Line 2 and 3 calculate the width and height factor respectively. Lines 4-7 compute the

resizing factor to use for resizing the image. In lines 8 and 9, the new dimension is generated based on the resizing factor. Finally, the image in line 10 will be resized.

```
if ( Is_Quality_Factor_Assigned_By_User )                                  Line 1
        Quality = QualityFactor;                                                  2
else                                                                              3
        Quality = Current_Bandwidth / Maximum_Bandwidth * InitialQuality;         4
if ( Quality < Threshold )                                                        5
        Quality = Threshold;                                                      6
Reduce_Image_Quality( Quality );                                                  7
```

Figure 5.9: Quality Reduction Routine

Figure 5.9 shows the quality factor reduction routine. The first line checks whether the user assigned a value to the desired quality variable. If this is true the image quality will be reduced to this value. If not, the formula in line 4 will generate a quality factor based on the available bandwidth. In line 5, the quality factor will be tested against the threshold. If the quality factor is less than the threshold then it will be reset to the threshold value. Finally, the quality factor of the image will be reduced (line 7).

```
if ( Image_Width > 4 OR Image_Height  > 4 )                                  Line  1
        Bandwidth_Factor = Current_Bandwidth / Maximum_Bandwith;                    2
        X_Display_Factor = Current_Display_Width / Max_Display_Width;               3
        Y_Display_Factor = Current_Display_Height / Max_Display_Height;             4
        if ( X_Display_Factor < Y_Display_Factor )                                  5
                Display_Factor = X_Display_Factor;                                  6
        else Display_Factor = Y_Display_Factor;                                     7
        if ( Bandwidth_Factor = 1 AND Display_Factor = 1 )   exit;                  8
        if ( Bandwidth_Factor < Display_Factor )                                    9
                Resize_Factor = Bandwidth_Factor;                                   10
        else Resize_Factor = Display_Factor;                                        11
        X_Dim = Image_Width * Resize_Factor;                                        12
        Y_Dim = Image_Height * Resize_Factor;                                       13
        ScaleImage ( X_Dim, Y_Dim );                                                14
end                                                                                 15
```

Figure 5.10: Image Scaling Routine

Figure 5.10 illustrates the image scaling routine. It just scales the images with width or height above 4 pixels. The first line checks the image dimensions. If one of the dimension is greater than 4 then the image will be scaled; otherwise, the resizing will not be applied. Line 2 calculates the bandwidth factor whereas lines 3-7 calculate the display factor. In line 8, if both factors equal 1 then resizing will not be necessary. Otherwise, the resizing factor will be the smaller factor between bandwidth and display factors as shown in lines 9-11. The new dimension of the image is generated in line 12 and 13. Finally, the image will be scaled based on the new dimension in line 14.

5.5 Summary

The vision of simple and ubiquitous information access requires solutions for an efficient integration of mobile devices within the WWW infrastructure. The main problems are the restricted resources of the mobile computing devices and the narrow bandwidth and unreliability of the wireless link. We have presented a system concept to support web browsing from mobile platforms with a wide range of communication, processing, and display capabilities. Our system follows the client-proxy-server model, which is the basis of most mobile applications and uses the proxy to provide an active transcoding mechanism. Proxies are mostly used for forwarding data between the mobile client and the stationary server. The idea of using transcoding at the proxy side is not new. Many proxy systems have been developed to provide web access to mobile users. However, these proxy servers are usually not aware of how much an image file needs to be distilled. They typically transmit the images distilled into a fixed factor regardless of the display capabilities, bandwidth limitations, or the user preferences. Moreover, since the cached data are kept separately for each user, they cannot be shared among the users of the same profile. In our system, we extend the notion of transcoding mechanism to be dynamically adaptive. We have proposed a system for adapting Internet content "images" called *image transcoding proxy*. This system adapts inline images to client devices with diverse capabilities. Therefore, the system enables universal access to the Internet by allowing different types of devices, and people with different abilities, to receive content "images" adapted to a form suitable for them. In the system framework, content adaptation is

analogous to compressing data streams to meet resource constraints imposed by the client device. However, unlike traditional compression, the user preferences are considered and the constraints are not limited to bits or bandwidth, but also include resources such as screen size, color and hardware and software capabilities.

The transcoding proxy system consists of three components: content analyzer, policy engine, and transcoding unit. The content analyzer is responsible for analyzing the images and classifying them into image type, purpose, and format categories. The policy engine then utilizes three sets of input parameters, device characteristics, bandwidth, and user preferences, to generate the transcoding vectors, which are passed to the transcoding unit to manipulate, transcode, and adapt the images. The transcoding unit can perform only one or all of the transcoding operations including: image scaling, color reduction, type conversion, lossless or lossy compression. The system also stores in the cache two version of each image: the original and the transcoded image. The proxy thus does not need to transcode the same image file requested previously by a client that has the same capabilities. In addition to transcoding images, the system also provides the options of removing active content such as background and advertising images, Java applet, JaveScript, VBScript. The mobile user might also ask for compressing the HTML document before transmitting it. The mobile client can select any or all of these options depending on the limitations of its device or the browser capabilities and network bandwidth. The users can change the options when the resources change. This makes our proxy very adaptable to serve the varying needs of the user.

Chapter 6

EXPERIMENTATION AND EVALUATION

In this chapter we will describe and evaluate different scenarios for image transcoding. The main goal of this chapter is to support our claim that in the majority of cases, image transformation is very effective in reducing the end-to-end latency. We achieve this by demonstrating that transcoding performance on today's desktop workstations is sufficiently fast. The time to produce a useful transcoded image is small enough to be more than made up for by the savings in transmission time for the transcoded image relative to the original. In Section 6.1, we describe the experimental equipment we are using for the test and the general experimental setup and results. In Section 6.2, the transcoding performance of different image transcoding scenarios is discussed and evaluated. In Section 6.3, the system scalability is evaluated.

6.1 Experimental Setup and Results

To test our transcoding proxy system, two different client machines were used. One was a

PDA device with poor resources and the other one was a desktop machine with rich resources. As for our transcoding proxy, it was running on a powerful PC machine with 550 MHz Pentium III, 128 MB memory, and 10 Mbps Ethernet card. The installed operating system was Linux 2.0.x kernel (RedHat 6.0). Server 1 to N could be any type of servers that our proxy communicates with to retrieve the requested data. The PDA used was a SHARP HC-4500, which has a display of 640x240 pixels with 256 colors and is connected to the proxy over a wireless network 'CDPD' with low bandwidth '9.6Kbps'. As for the desktop machine, it was a PC of 1024x768 pixels and linked to the proxy over Local Ethernet network with high bandwidth '10Mbps'. During our experiments, we simulated the two types of link between client devices and proxy to quantify the transmission time. The general experimental testbed setup is shown in Figure 6.1.

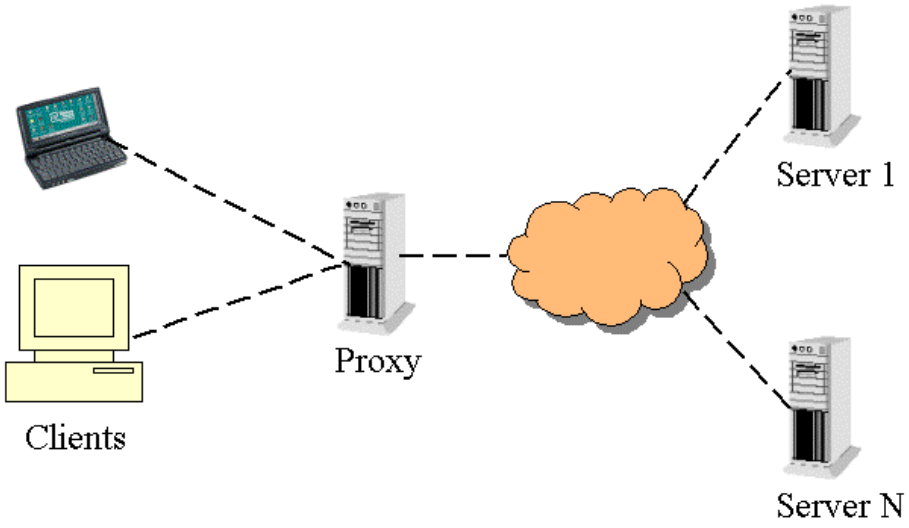

Figure 6.1: The General Experimental Testbed Setup

A test page was created with three images and some text. One image was animated GIF, another image was advertisement image and the other was non-animated GIF. The clients declared different preferences to the proxy, and accessed the same web page. For the wired host with riches resources, the proxy sent the web page with very slightly transformation. Therefore, the client received all components of the web page. For the wireless host with poor resources, the preferences set indicated that animated images are not supported and advertisement and background images are not desired due to the poor resources. Hence, the client received the web page without background image and the advertisement image was substituted with a text to save some bandwidth. As for animated and non-animated images, a small version of the first frame of the animated image and non-animated image were sent to the client. The transcoding proxy scales the images by the ratio of the horizontal width of the PDA display to that of a reference display of 1024x768 pixels. Measurements using the time command indicate that the process of transcoding the images and transferring the entire web page to the mobile client took about 3.5 seconds. As for the wired host, it took about 2 seconds to deal with many animated frames beside the other images. The different resulting pages are illustrated in Figures 6.2 and 6.3 respectively.

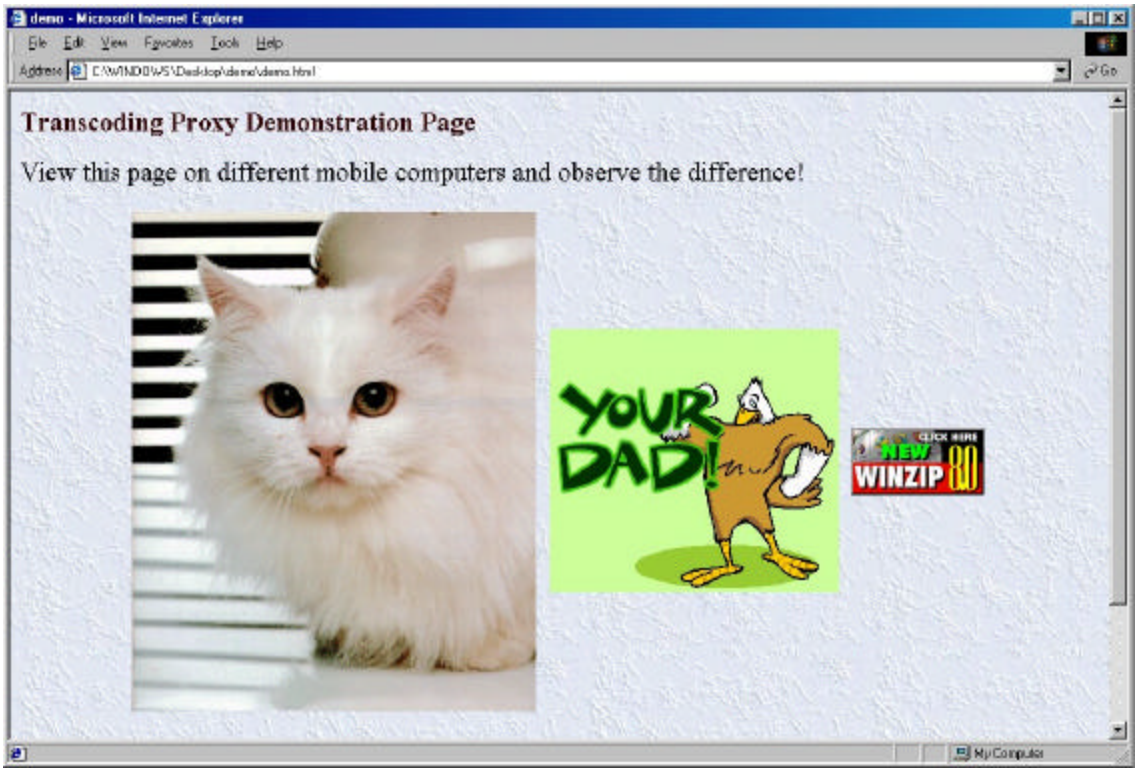

Figure 6.2: Response on a Resource Rich Client

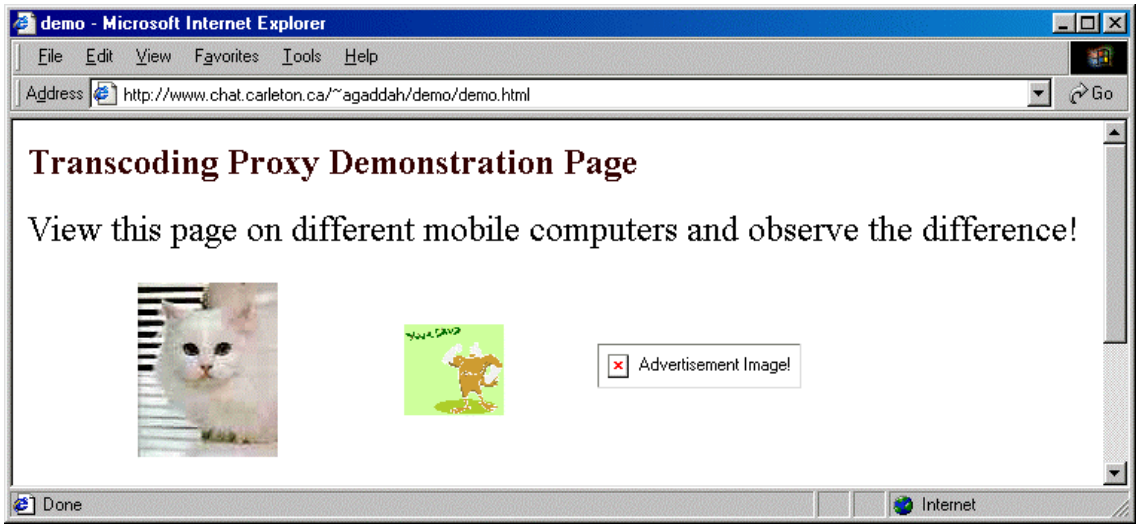

Figure 6.3: Response on a Resource Poor Client

Computational Cost

To get a clearer sense of the cost involved in doing image transcoding at the proxy, we measured the time required by the proxy to provide a client with a variety of web pages from different popular web sites such as CNN, OttawaCitizen, NewsWeek, Carleton U., and Ottawa U. Measurements were made for these web pages which had varying amounts of images. Table 6.1 shows these results.

Site Name	# of Transcoded Image	Transcoding Time (sec)	Average of Compression Ratio
Royal Bank	23	0.35	85.25%
OttawaCitizen	7	0.31	82.58%
CNN	5	0.08	87.18%
Carleton U.	21	0.57	86.99%
U. of Ottawa	14	0.49	89.96%
Yahoo	1	0.03	85.11%
NewsWeek	14	0.5	88.23%

Table 6.1: The Cost of Transcoding Popular Web Sites

Figure 6.4 shows the time taken for transmitting and processing the web pages in Table 6.1. Since our transcoding proxy is designed to have a relatively high bandwidth between the content server and the proxy we have ignored the delay between proxy and server. Here we measured only the delay between the proxy and client. The measurement in Figure 6.4 was taken for PDA client which connected to the proxy over CDPD network with low bandwidth '9.6Kbps'. The first bar of each category reveals the transmission times (including the processing time at the proxy) for different size web pages. The PDA

receives the web pages through the proxy which transcodes and caches the web pages, and sends the transcoded data to the PDA. The second bar is the transmission time without transcoding. As we can see from Table 6.1, the delay introduced by the transcoding proxy is too little to be significantly perceived by the user. In the case of "far" documents, which require a long download time, this additional delay represents only a very small fraction of the global download time. In the case of "near" documents, which can be downloaded efficiently, the delay introduced by the transcoding proxy can be as high as 80% but, in any case, is still small enough not to influence the user browsing activity.

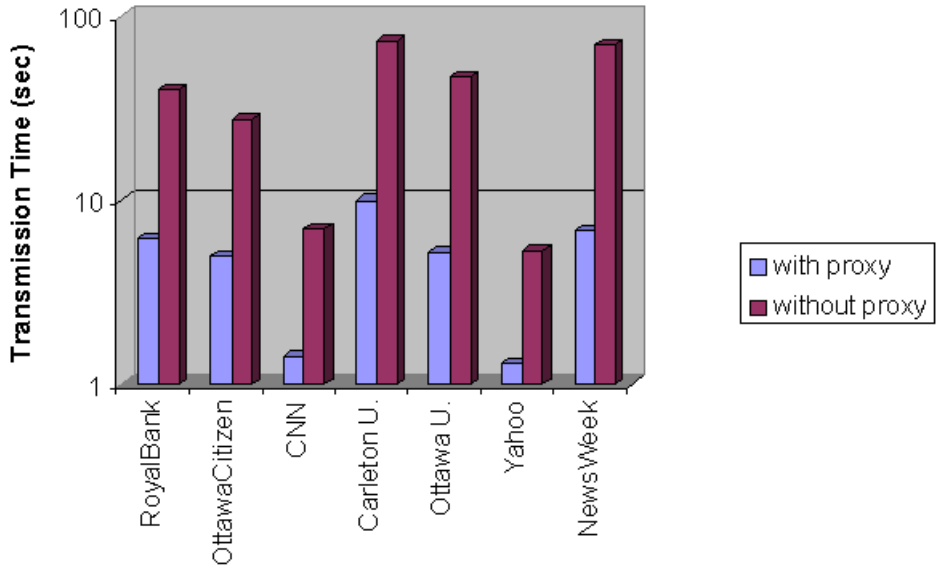

Figure 6.4: The Delivery Time for the Web Pages with and without Transcoding

The previous evaluation in Figure 6.4 does not take into account the presence of the proxy cache that can significantly speed up the browsing activity. In addition, it does not consider the location of the web server whether it is "far" or "near". The benefits in terms

of response time (from the click to the full visualization of the page) of the cache are shown in Figure 6.5, depending on the number of objects enclosed in the document. We recall that, in the HTTP protocol, each object within a page requires a separated TCP session. This factor, rather than the plain document size, influences the response time. Here the PDA client was connected to the proxy over wireline modem. The seven web pages were fetched using a 14.4Kb/s modem. In order to get clear view of the improvement in access time due to caching, we measured the download time for the same web pages when they fetched from a web server within our city and outside of our city (Italy). The first bar of each set in Figure 6.5 shows the download time for the pages from a web server in Italy without using the cache system. The second bar shows the download time for the same pages from a local web server (in Canada) without using the cache system. The third bar shows the pages access time using the proxy cache system. In proxy cache case, we assumed that the transcoded images were available in the cache. So we ignored the transcoding delay in our measurement. Thus, the proxy cache case illustrates the minimum time needed to access these web pages. Table 6.2 shows the amount of data that needs to be transmitted in the three cases. As can be seen, the proxy cache notably improves the access time to pages, especially in the case of access to far (i.e., out of our city) sites. Moreover, it achieves a significant improvement in the access times also in case of accesses to very local sites (i.e., within our city).

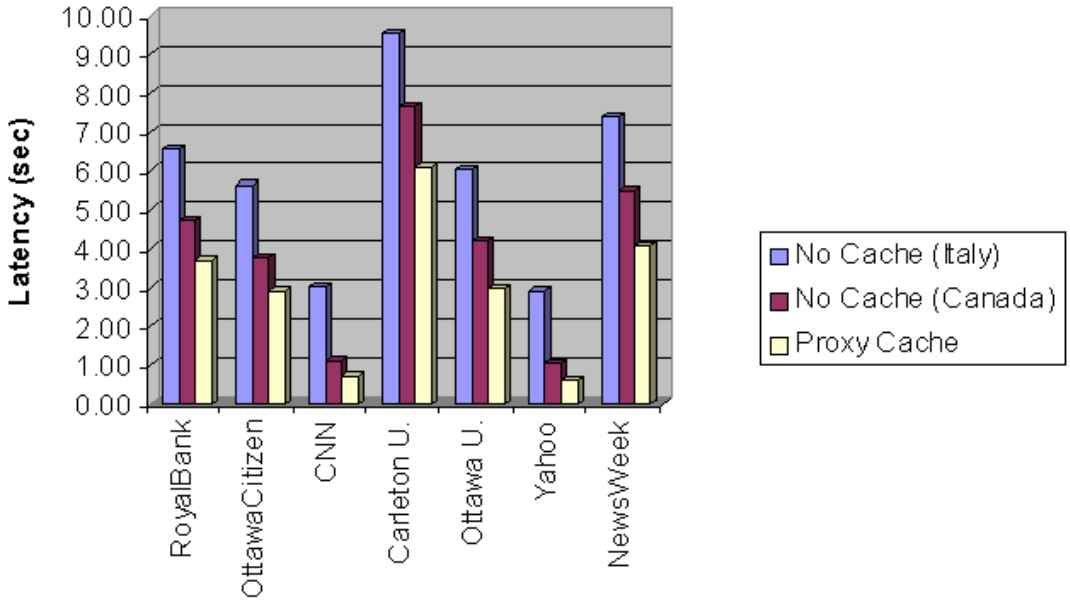

Figure 6.5: Page Access Time

Site Name	Total Size Before Transcoding (Bytes)	Total Size After Transcoding (Bytes)	Percentage of Original Size
Royal Bank	47120	**6375**	13.53%
OttawaCitizen	32638	**4985**	15.27%
CNN	7847	**1004**	12.79%
Carleton U.	88911	**10699**	12.03%
U. of Ottawa	55597	**5044**	9.07%
Yahoo	5776	**860**	14.89%
NewsWeek	84308	**7000**	8.30%

Table 6.2: The Amount of Data Needs to Be Transmitted After Transcoding

In order to evaluate the cost of the transcoding process accurately, we conducted experiments in three different network bandwidth situations using three different clients. For the test, we set up the configuration of the three mobile clients to poor, medium, and high resources. So the transcoding process can act differently in each case. Each client

accesses the same web page individually with different configurations. Transcoding measurements were made for 50 pages that had different numbers of images, some of which required no transcoding. The time taken to trancode the images in each web page is measured for each dient. Figure 6.6 shows the result of this experimental. About 90% of the images required less than 0.2 seconds to process the images. Recall that the transcoding process was performed on a Linux machine with 128 MB memory and 550 MHz Pentium III. Figure 67 illustrates the compression ratio achieved by transcoding the images. About 90% of the images achieved more than 70%. Comparing the transcoding time with the achieved compression ratio can give a clear idea about the cost of our transcoding process.

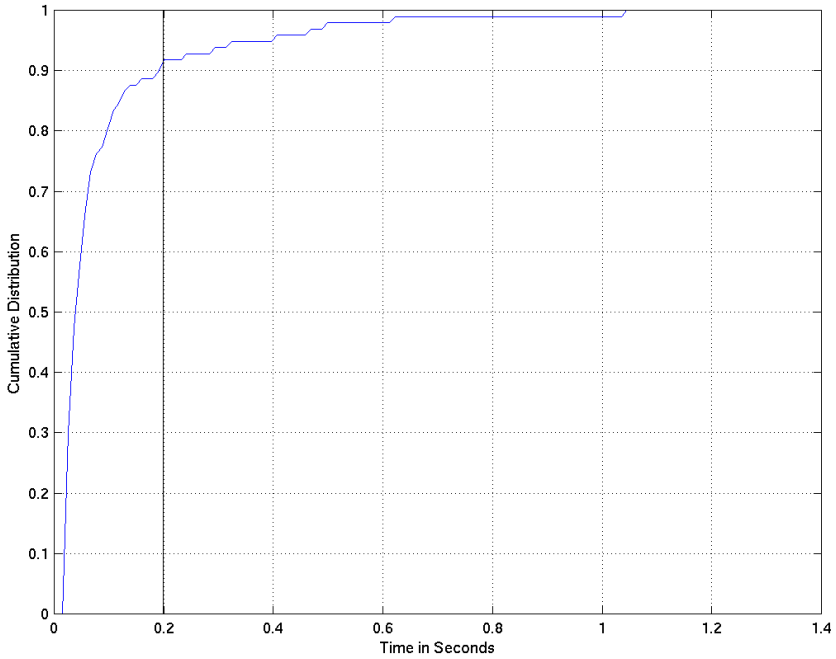

Figure 6.6: Transcoding Time Consumed on a Resource Poor, Medium, and High Client

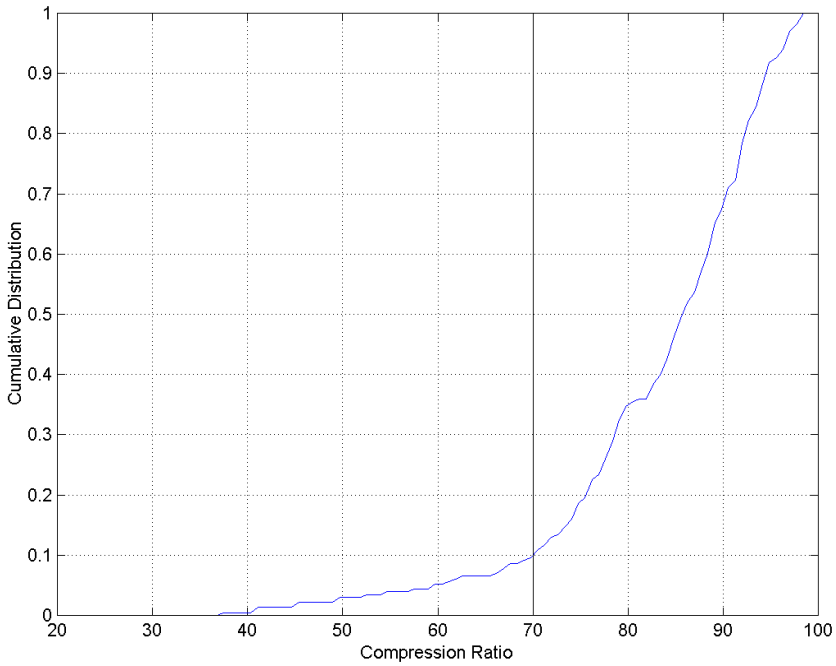

Figure 6.7: The Achieved Compression Ratio

Figure 6.8: The Transcoded Image of Soda Hall

6.2 Transcoding Performance

As mentioned in Chapter 5, we have implemented an image transcoder, which implements transformation routines for GIF and JPEG images. It consists largely of source code from the ImageMagick Toolkit [77]. Figure 6.8 shows the result of running our transcoder on a large color GIF image of the Berkeley Computer Science Division's home building, Soda Hall. The image in Figure 6.8 measures 320x200 pixels and uses only 16 colors, making it suitable for display on a low-end notebook computer. The image occupies 11 K bytes at 320x200 pixels in 16 colors, compared with 492 K bytes, 880x610 pixels and 249 colors in the original (not shown). Transcoding took less than one second on a lightly loaded PC 550 MHz Pentium III. We achieved this result by applying two operations, scaling and color reduction. We compared our result with the result achieved by the *gifmunch* distiller described in [8]. Distilling the same image using *gifmunch* produces an output file size that occupies 17K bytes at 320x200 pixels in 16 levels of gray. This distillation took about six seconds on a SPARCstation 20/71. It is clear that our transcoder achieved better reduction in less time. In addition, we can achieve more reduction in file size by converting the image to JPEG format and reducing its quality. In this case, the file size will be 8 K bytes instead of 11 K bytes and the transcoding latency is still less than one second. The quality of the image produced by our transcoder and the *gifmunch* distiller is similar.

Original Image (GIF)			Scaling		Scaling and Map to 16 Colors		Scaling, Convert to JPEG and Reduce Quality Factor to 25	
Size KB	Col-ors	Dim-ension	File Size (%)	Time (sec)	File Size (%)	Time (sec)	File Size (%)	Time (sec)
492	249	879x609	13.7	0.15	4.5	0.55	1.6	0.46
234	252	487x760	13.9	0.14	6.6	0.48	3.8	0.38
128	256	640x512	17	0.08	8	0.31	1.6	0.22
94	256	396x481	15.8	0.05	4.9	0.21	1.9	0.15

Table 6.3: Transcoding Latency and New Sizes (as Percent of Original)

Table 6.3 shows the latencies of transcoding a number of GIF images with three different sets of transformation parameters, and the resulting size reductions. Again the measurements were taken on a lightly loaded PC 550 MHz Pentium III running Linux 6.0. The three sets of transformation parameters were chosen as representative values for addressing the three categories of variation: image scaling to 35%, color reduction to 16 colors, and format conversion to JPEG plus quality reduction. The table data reveals three trends of interest:

- There appears to be a linear relationship between the file size and the transcoding latency time. The bigger the image file sizes, the longer it takes to transcode. This kind of relationship makes sense since the transcoder touches every pixel in the image. (compare rows 3-6)

- Performing two transcoding operations actually takes longer time than performing one operation. In the second column, the four images were scaled down to 35%. In the third column, the four images were scaled down and the number of colors in the images was fixed at 16. For all four images, the time required to scale and reduce the color to 16 was greater than the time required to scale down the images. However, in the fourth column, adding format conversion to the previous two operations did not introduce longer latency as we suspect. In other words, the additional work of transcoding to JPEG adds virtually no latency to the overall transcoding process (compare columns 2-4).
- There is consistency between file size reduction and the number of operations applied on the image. The more operations we apply the larger reduction we achieve. In other words, applying two operations effects file size more than applying one operation and applying three effects file size more than applying two and so on. This type of consistency does not exist in the *gifmunch* distiller.

The graph in Figure 6.9 depicts end-to-end client latency for retrieving the original and each of three transcoded versions of a selection of GIF images: the four sets of bars correspond to the four images in Table 6.3. The end-to-end latency was measured for the images with and without transcoding. Each group of bars represents one image with three levels of transcoding; the bottom bar represents no transcoding at all. The y-axis number is the transcoded size in kilobytes (so the bottom bar gives the original size). The images were fetched using a 14.4Kb/s modem through a PPP gateway, via a process that runs

each image through the ImageMagick toolkit. Each bar is segmented to show the transcoding latency and transmission latency separately. Clearly, even though transcoding adds latency at the proxy, it can result in greatly reduced end-to-end latency.

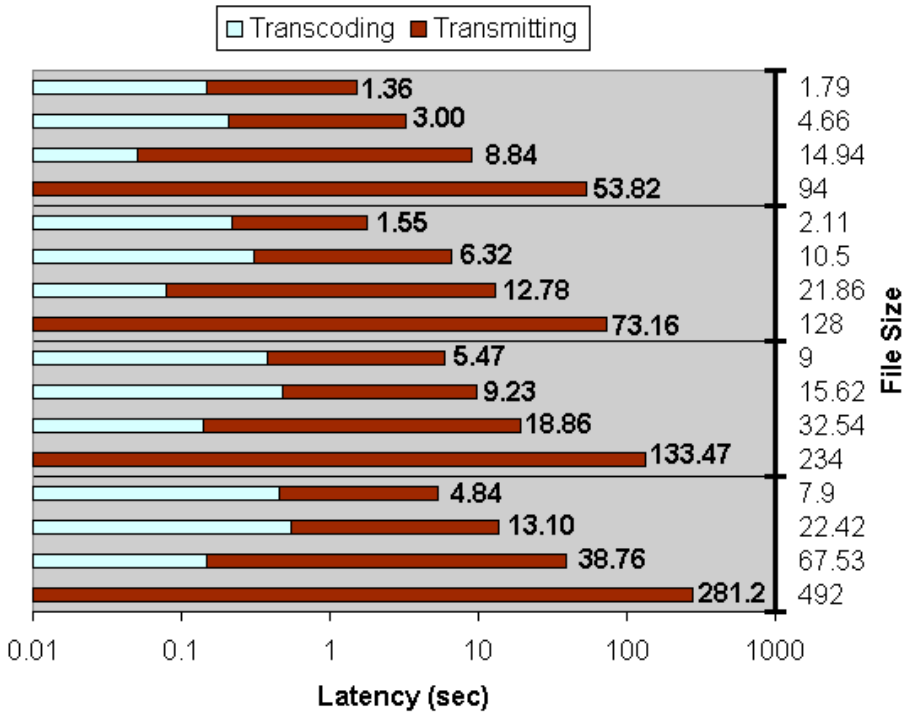

Figure 6.9: End-to-end Latency for Images with and without Transcoding

6.3 Scalability Concerns

The previous sections have demonstrated that modern workstations are sufficiently fast that transcoding time is often small compared to time saved in transmission of a transcoded image. Our transcoding proxy is implemented as a multi-threaded program. The *Connection* thread listens for client requests at an advertised port. For servicing a

client request, a *Proxy* object is created for each open TCP connection between the client and the proxy. Hence, the number of *Proxy* objects in existence at any time is equal to the number of open TCP connections between the client and the proxy. Multiple transcoding operations therefore can be serviced in parallel on behalf of a potentially large set of clients. To explore how well our system scales to large numbers of clients, we investigated the load placed on our proxy by a number of clients. For the purposes of this testing, our proxy was running on a single 550-MHz Pentium III workstation that serviced all image customization requests. We used the *gettimeofday* function to measure the transcoding latency perceived by the users and the number of simultaneous image customizations currently in progress on the same machine. Image customization is a CPU-bound task, since the process of image reduction and requantization requires a transcoder to "touch" all of the pixels in an image many times. We observed that the latency of transcoding was a linearly increasing function of the number of simultaneous operations, with a slope approximately proportional to the size of the original GIF (in bytes). Because N transcoders shared the workstation's CPU equally, each transcoding operation took N times as long to complete. Figure 6.10 shows the average latency of image transcoding perceived by a user as a function of the number of users supported by a single workstation. At approximately 20 users, requests arrive faster than they are serviced, and beyond this point, transcoding latency is unbounded for the single workstation. Nevertheless, this result suggests that even using today's desktop hardware, information access patterns (at least for the Web) allow multiple users to be served by one

compute server in a proxy installation. As we suggest in the future work, the actual work of transcoding can be off-loaded to other nodes in a network of workstations. By doing so, the scalability of the system will be improved.

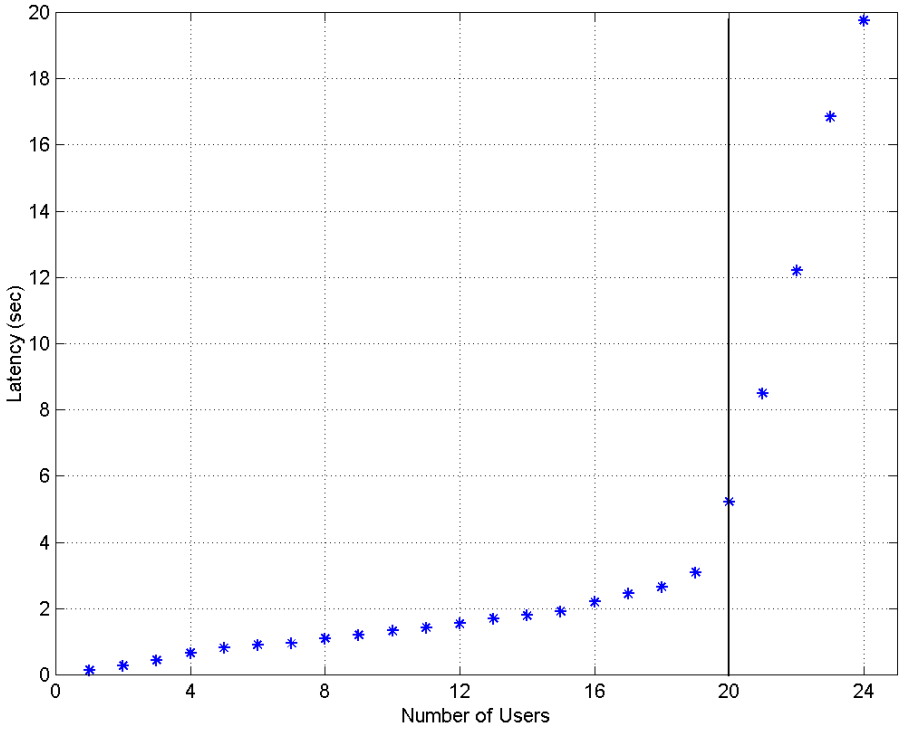

Figure 6.10: Image Transcoding Latency Versus the Number of Simultaneous Users on a Single Proxy

Chapter 7

CONCLUSIONS AND FUTURE WORK

Information access from different mobile platforms is becoming increasingly important. This trend is being driven by two factors: the increasing power and capability of mobile devices, and more readily available wireless networking technologies. The former is bringing low-cost, portable hardware with multimedia capability into broad use. The latter provides the means with which these small devices can access a broad wealth of shared data. This dissertation has shown that these mobile clients must adapt to changes in their environments, and that this adaptation is best provided through a transcoding proxy, which is placed between the client and the web server. The transcoding proxy is the vital implemented system to demonstrate the feasibility of multimedia image adaptation, and the evaluation of this prototype confirms the importance of this approach to reduce web page retrieval latency on WWW. This chapter begins in Section 7.1 with a brief summary of the contributions made in the thesis. This work has also uncovered many avenues of further inquiry; these are presented in Section 7.2.

7.1 Conclusions

The modern portable computers and wireless connections have created a new fundamental technology for a vision of information access for anyone, anytime, anywhere. However, the new platform for mobile information access faces several challenges that are mainly caused by the limited resources (e.g. display, battery, memory size, processing power) of mobile devices and low bandwidth of wireless environment. Due to these shortcomings, new concepts are needed to provide efficient access to World Wide Web information within a mobile environment. Therefore, technologies that can adapt Internet content to diversity of client devices and variability of network bandwidth will become critical in the coming pervasive computing era.

On many different network and application levels, a number of efforts have been done to enhance the performance and usability of Web browsing in wireless environments. The most common solutions that have been proposed for improving the accessibility of information from mobile platforms are based on the client-proxy-server model. In this model, a proxy is located between a highly resource-rich web server at one side and a highly resource-poor mobile client at the other side. The main role of the proxy is to filter and compress the multimedia content originating from the web server. Even though all these proxies provide solutions to enhance mobile WWW access, they do not address dynamic adaptation of WWW content for different mobile clients. Furthermore, they do not answer the question of how already existing WWW content can be adapted for a presentation on resource limited mobile computing devices. As a typical scenario,

the proxy does not have any knowledge about the capability of mobile computers or network bandwidth status. Consequently, the proxy server distills Internet content into fixed factor regardless to the requesting mobile host display capabilities or the available bandwidth, and delivers the same distilled content to all mobile hosts. With this type of mechanism, it would be difficult for proxies to support a diversity of mobile computers with different characteristics and channel constraints.

To achieve a general solution, we have proposed a transcoding proxy process that is placed on a proxy between client and server. This middleware provides several transcoding services, which can dynamically adapt inline images to the current context of the mobile host. This context includes the mobile device, the user's preferences, and the current available bandwidth of data transfer between the mobile host and proxy server. The proposed transcoding proxy is aiming to achieve the smallest image file size for quick transmission while at the same time maintaining the best possible quality of the delivered images. Therefore, image processing (e.g., image scaling, color reduction, type conversion) may be necessary to match the images with the devices and channels constraints. In order to carry out the adaptation, we built a collection of adaptation rules. The rules define the transformation or modification of images depending on the context parameters. These parameters have an influence on the action of a rule, which is defined by a Boolean expression. When the expression is evaluated and the condition is fulfilled the actions of a rule are executed. For example, the Boolean expression "is the image animated and is the freezing animation parameter equal true" is evaluated before the

action of removing all the image frames excluding the first one takes place as shown in Figure 5.5. An action correlates to one internal or external program that is responsible for a certain adaptation process. These kinds of programs are called *transcoding function*. The system can also reduce the download time by caching two versions of images: the original and the transcoded. By doing so, it is unnecessary to transcode the same image file requested previously by a client that has the same capabilities. In addition to image transcoding, the system provides options to eliminate active component (i.e. Java applet, JaveScript, VBScript) that the mobile host cannot support. In some critical situations, compressing HTML document and removing background and advertising images might be beneficial to reduce the download time. The achieved results have revealed that an adaptive image transcoding proxy is a good solution and a step in the right direction to adapt arbitrary inline images according to the properties of the end device, user preferences, and available bandwidth.

7.2 Future Work

This dissertation has opened many avenues to very interesting research topics, for which the transcoding proxy can serve as a starting point. Improvements can be made to several parts of the transcoding proxy system in order to facilitate information access from various mobile platforms. Some of these are modest in scope, while others are more ambitious. This section explores each of these missing parts in turn, and reveals some new investigation areas.

As we have mentioned previously, we do not have an appropriate mechanism to detect the available bandwidth during run-time. Bandwidth is one of the most important parameters that our system utilizes to determine data transcoding policy. Currently, we feed our system with the bandwidth information externally. This usually requires the user to interfere and change the bandwidth value from time to another. Therefore, we suggest automating this policy by building a monitor that can sense the bandwidth and provide it to the system without user interference.

Our system has considered only a single type of media (image), not composite multimedia documents. Other types of media such as video, sound, and text might need to be addressed for allowing universal access to various types of information. It would be very interesting to explore how several transcoding proxies, each responsible for transcoding one type of media, can be added to our system. Furthermore, reorganizing the complete elements of a webpage to fit in a small screen device is a new challenging area that needs to be investigated.

Estimating the cost of transcoding operations is a very important task that needs to be achieved before applying any operation. Similarly, estimating the output file size is worth consideration. We have not addressed these two tasks that could enhance the system performance. The estimation values of the cost of transcoding operations and the output file size should be considered as new parameters to the system in the future development. These parameters could greatly assist the system to make the right decision "to transcode or not to transcode".

In the current design, the transcoding proxy has been developed as a single proxy that can handle all requests from many users. We assumed that a few users might deal with the transcoding proxy at the same time. Thus, we have not investigated the case when the transcoding proxy is overloaded by many users. A study could be attempted to address how the workload of the current proxy can be shifted to a neighbor proxy in order to balance the current proxy load.

Since one proxy is not enough to serve all requests mobile users may attach to different proxies in the same or different cells. Therefore, a mechanism for transferring the mobile client information such as user preferences and device characteristics from one proxy to another is needed to ensure that service is not interrupted by reconfiguring operation when mobile client moves from one cell to another. Also, installing the proxy software on different proxy machines is another issue need to be addressed.

In the future work, we still need to consider the end-to-end security issue. One of the security issues is to establish trust between the proxy and the mobile host. It is necessary to have an authentication protocol that can be used to authenticate proxy service to the clients and to authenticate the client in the proxy to the servers. By doing so, we allow an *untrusted* proxy to interact with the clients, giving the clients full authenticated (and optionally encrypted) access to the proxy services but relieving them of a significant amount of authentication processing.

While our transcoding proxy makes a convincing case for data adaptation, it is only a starting point in addressing the general problem of mobile data access. Users should

become actively involved in adaptation decisions, and the notion of adaptation should be extended to computational processes as well as data structures. The very process of building adaptive systems is in its infancy. However, our system should serve well as a base from which to explore the previously mentioned problems.

BIBLIOGRAPHY

[1] A. Fox and E. A. Brewer, '*Reducing WWW Latency and Bandwidth Requirements by Real-time Distillations*', In Proc. Fifth International WWW Conference, May 1996.

[2] S. Jacobs, M. Gebhardt, S. Kethers and W. Rzasa, '*Filling HTML Forms Simultaneously: CoWeb – Architecture and Functionality*', Computer Networks & ISDN Systems, 28 (7-11), 1385-1395, 1996. http://www5conf.inria.fr/fich_html/papers/P43/Overview.html.

[3] C. Brooks, M. S. Mazer, S. Meeks and J. Miller, '*Application Specific Proxy Servers as HTTP Stream Transducers*', In Proc. WWW-4, Bosten, May 1996. http://www.w3.org/pub/Conferences/WWW4/Papers/56Application-Specific

[4] Hari Balakrishnan, Venkata N. Padmanabhan, Srinivasan Seshan, and Randy H. Katz, '*A Comparison of Mechanism for Improving TCP Performance over wireless links*', In Proc. of the 1996 ACM SIGCOMM Conference, Stanford, CA, USA, Aug 1996.

[5] R. Yavatkar and N. Bhagawat, '*Improving End-to-End Performance of TCP Over MobileIinternetworks*', in Proceedings of the Workshop on Mobile Computing Systems and Applications, Dec. 1994.

[6] Henning Koch, Lars Krombholz, and Oliver Theel, '*A Brief Introduction intoTthe World of Mobile Computing*', Technical report THD-BS-1993-03, May 21, 1993, Department of Computer Science, University of Darmstadt.

[7] U. Gall and F. J. Hauck, '*Promondia: A Java-based Framework for Real-time Group Communication in The Web*', Proc. 6th International WWW Conference, Santa Clara, CA, April 1997. http://decweb.ethz.ch/WWW6/Technical/Paper100/paper100.html.

[8] E. A. Brewer, R. H. Katz, Y. Chawathe, A. Fox, S.D. Gribble, T. Hodes, G. Nguyen, T. Henderson, E. Amir, H. Balakrishnan, V. Padmanabhan, and S. Seshan, '*A network Architecture for Heterogeneous Mobile Computing*', IEEE Personal Communications Magazine, 5(5): 8-24, 1998.

[9] B. Clifford Neuman, '*Protection and Security Issues for Future Systems*', In Workshop on Operating Systems of the 90s and Beyond, Springer-Verlag Lecture Notes in Computer Science #563, pages 184-201, July 1991.

[10] M. Satyanarayanan, '*Fundamental Challenges in Mobile Computing*', Fifteenth ACM Symposium on Principles of Distributed Computing, May 96, Philadelphia, PA

[11] Balakrishnan H., S. Seshan, and R. H. Katz, '*Improving Reliable Transport and Handoff Performance in Cellular Wireless Networks*', ACM Wireless Networks, December 1995, http://wind.lcs.mit.edu/~hari/papers

[12] T. Imielinski and B. R. Badrinath, '*Data Management for Mobile Computing*', SIGMOD Record, 22(1): 34-39, March 1993.

[13] John Ioannisdis, Dan Duchamp, and Gerald Q. Maguire Jr, '*IP-based Protocols for Mobile Internetworking*', In Proceedings of SIGCOMM '91 Symposium, pages 235-245, Sept 1991.

[14] Chaoying Ma, '*On Building Very Large Naming Systems*', In 5th SIGOPS Workshop on Models and Paradigms for Distributed Systems Structuring, 5 pages, Sept 1992.

[15] Fumio Teraoka and M. Tokoro, '*Host Migration Transparency in IP Networks: The VIP Approach*', Computer Communication Review, 23 (1): 45-65, Jan 1993.

[16] S. Seshan, M. Stemm, and R. Katz, '*Spand: Shared Passive Network Performance Discovery*', In Proc. 1st Usenix Symposium on Internet Technologies and Systems (USITS '97, 1997)

[17] A. Joshi, R. Weerasinghe, S. P. McDermott, B. K. Tan, G. Benhardt, and S.Weerawarna, '*Mowser: Mobile Platforms and Web Browsers*', Bulletin of the IEEE Technical Committee on Operating Systems and Application Environments, 8(1), 1996.

[18] A. Joshi, S. Weerawarana, and E. N. Houstis, '*Disconnected Browsing of Distributed Information*', In Proc. Seventh IEEE Intl. Workshop on Research Issues in Data Engineering, pages 101-108. IEEE, April 1997.

[19] A. Joshi, S. Weerawarna, and E. N. Houstis, 'On Disconnected Browsing of Distributed Information', In Proceedings of the seventh International workshop on Research Issues on Data Engineering, pages 101-107. IEEE Press, 1997.

[20] G. Benelli, '*Error Recovery for ATM Transmission Over Wireless Channels*', In Electronics Letters 1995 Aug 3 Vol. 31 No. 16 Pages 1325-1326

[21] R. Katz, '*Adaptation and Mobility in Wireless Information Systems*', IEEE Personal Communications, 1(1):6-17, 1994.

[22] R. Kavasseri, T. Keating, M. Wittman, A. Joshi, and S. Weerawarna, '*Web Intelligent Query – Disconnected Web Browsing using Cooperative Techniques*', In Proc. 1st. IFCIS Intl. Conf. On Cooperative Information Systems, pages 167-174. IEEE press, 1996.

[23] S. Gessler and A. Kotulla, '*PDAs as Mobile WWW Browsers*', In Third International WWW Conference, Darmstadt, Germany, 1995.

[24] J. F. Bartlett, '*Experience with A Wireless World Wide Web Client*', Tech. Rep., Western Research Laboratory, Palo Alto, CA, Mar. 1995.

[25] Friday A. J., G. S. Blair, K. W. J. Cheverst, and N. Davies, '*Extensions to ANSAware for Advanced Mobile Applications*', In Proc. International Conference on Distributed Platforms, Dresden, February 27 -- March 1, 1996, http://www.comp.lancs.ac.uk/computing/research/mpg/most/reports/icdp.adrian.ps

[26] Zenel B., '*A Proxy Based Filtering Mechanism for the Mobile Environment*', Thesis Proposal, Departement of Computer Science, Columbia University, June 1997 http://www.mcl.cs.columbia.edu/~baz

[27] T. Watson, '*Application Design for Wireless Computing*', In Proc. Workshop on Mobile Computing Systems and Applications, pages 91--94. ACM/IEEE, Dec 1994.

[28] Rabbit HTTP proxy: http://www.nada.kth.se/projects/prup98/web_proxy

[29] A. Fox, S.D. Gribble, Y. Chawathe and E. Breuer, '*Adapting to Network and Client Variation Using Infrastructure Proxies: Lessons and Perspectives*', IEEE Personal Communications, 5(4): 10--19, Aug. 1998.

[30] M. Liljeberg, M. Kojo, and K. Raatikainen, '*Enhanced Services for WWW in Mobile WAN Environment*', 1996. http://www.cs.Helsinki.FI/research/mowgli/mowgli-papers.html

[31] J. Poskanzer, '*NetPBM Release 7*', ftp://wuarchive.wustl.edu/graphics/graphics/packages/NetPBM/

[32] Graphics Interchange Format Version 89a (GIF). CompuServe Incorporated, Columbus, Ohio, July 1990.

[33] M. Liljeberg, H. Helin, M. Kojo, and K. Raatikainen, '*Enhanced Services for World-Wide Web in Mobile WAN Environment'*, Report C-1996-28 April 1996.

[34] A. Joshi, R. Weerasinghe, S. P. McDermott, B. K. Tan, G. Benhardt and S. Weerawarna, '*Mowser: Mobile Platforms and Web Browsers*', Bulletin of the IEEE Technical Committee onOperating Systems and Application Environments Vol 8, no. 1, 1996.

[35] B. Noble, M. Satyanarayanan, D. Narayanan, J.E. Tilton, J. Flinn and K. Walker, '*Agile Application-Aware Adaptation for Mobility*', Proceedings of the 16th ACM Symposium on Operating System Principles, October 1997, St. Malo, France

[36] D. W. Jones, '*Application of Splay Trees to Compressions*', Communications of the ACM, 31(8), August 1988.

[37] M. Crovella and A. Bestavros, '*Explaining World Wide Web Traffic Self-similarity*', Technical Report 95-15, Boston University, Computer Science Department, Aug 95.

[38] Mowser – A Web Browser for Mobile Platforms http://www.cs.purdue.edu/research/cse/mobile/mowser.html

[39] M. Satyanarayanan, '*Mobile Information Access*' http://www.cs.cmu.edu/afs/cs.cmu.edu/project/coda/Web/docdir/ieeepcs95.pdf

[40] Ubiquitous Computing: The home page of Ubicomp or the Ubiquitous Computing Project at Xerox PARC.

[41] J. R. Smith, R. Mohan and C. S. Li, '*Transcoding Internet Content for Heterogeneous Client Devices*', Proceedings of the IEEE International Symposium on Circuits and Systems (ISCAS), Special session on Next Generation Internet, June 1998.

[42] C. – S. Li, R Mohan, and J. R. Smith, '*Multimedia Content Description in The InfoPyramid*', In IEEE Proceedings of the International Conference on Acoustics, Speech, Signal Processing (ICASSP), Seattle, WA, May 1998. Special session on Signal Processing in Modern Multimedia Standards.

[43] T. Watson, '*Wit: An Infrastructure for Wireless Palmtop Computing*', Technical Report CSE-94-11-08, University of Washington, Nov 1994.

[44] T. Watson, '*Effective Wireless Communication Through Application Partitioning*', In Proc. Fifth Workshop on Hot Topics in Operating Systems, pages 24--27. IEEE, May 1995.

[45] J. Ioannidis, D. Duchamp, and G. Q. Maguire, Jr, ' *IP-based Protocols for Mobile Internetworking*', In Proc. SIGCOMM 91 Conf., pages 235--245. ACM, Sep 1991.

[46] A. Hokimoto, K. Kurihara, and T. Nakajima, '*An Approach for Constructing Mobile Applications using Service Proxies*', To Appear in Proc. of The 16th International Conference on Distributed Computing Systems, May 1996.

[47] A. D. Joseph, A. F. deLespinasse, J. A. Tauber, D. K. Gifford, and F. Kaashoek, '*Rover: A Toolkit for Mobile Information Access*', In Proc. Fifteenth ACM Symposium on Operating Systems Principles, pages 156--171. ACM, Dec 1995.

[48] Hokimoto A., and T. Nakajima, Robust Host Mobility Supports for Adaptive Mobile Applications, http://mmmc.jaist.ac.jp:8000/

[49] Welch, T. A., '*A Technique for High Performance Data Compression*', IEEE Computer, Volume 17, Number 6, June 1984. See also the Graphics FAQ - GIF legality: http://www.faqs.org/faqs/graphics/

[50] Extensive information about the JPEG graphic file format, including progressive JPEGs. JPEG FAQ: http://www.fags.org/faqs/jpeg-fag/part1

[51] Hamilton, E.,'*JPEG File Interchange Format'*, Milpital, CA: C-Cube Microsystems, 1992.

[52] N. Abramson, '*The ALOHA system – Another Alternative for Computer Communications*', In Fall joint Computer Conference, AFIPS Conference Proceedings, volume 37, pages 281-285, 1970.

[53] J. Kahn and J. R. Barry, '*Wireless Infrared Communications*', In Proc. of the IEEE, February 1997.

[54] C. Perkins, '*IP Mobility Support RFC*', Oct 1996. RFC-2002.

[55] J. Jubin and J. Tarnow, '*The DARPA Packet Radio Network Protocols*', In Proc. of the IEEE, 75(1):21-32, January 1987.

[56] S. Seshan, '*Low Latency Handoffs in Cellular Data Networks*', PhD thesis, University of California at Berkeley, December 1995.

[57] J. Ioannidis, D. Duchamp, and G. Q. Maguire, '*IP-based Protocols for Mobile Internetworking'*, In Proc. ACM SIGCOMM '91, pages 235-245, 1991.

[58] S. Seshan, H. Balakrishnan, and R. H. Katz, '*Handoffs in Cellular Wireless Networks: The Daedalus Implementation and Experience*', Kluwer Journal on Wireless Personal Communications, January 1997.

[59] R. Caceres and L. Iftode, '*Improving the Performance of Reliable Transport Protocols in Mobile Computing Environments*', IEEE Journal on Selected Areas in Communications, 13(5), Jun 1995.

[60] B. Leiner, D. Nielson, and F. Tobagi, '*Issues in Packet Radio Network Design*', In Proc. of the IEEE, 75(1):6-20, January 1987.

[61] R. Caceres and V. N. Padmanabhan, '*Fast and Scalable Handoffs in Wireless Internetworks*', In Proc. 1st ACM Conference on Mobile Computing and Networking, November 1996.

[62] R. Ghai and S. Singh, '*An Architecture and Communications Protocol for Picocellular Networks*', IEEE Personal Communications Magazine, 1(3): 36-46, 94.

[63] Infrared Data Association. IrDA Serial Infrared (SIR) Physical Layer Link Specification (IrPHY) 1.1. Infrared Data Association, 1995.

[64] R. Scholtz, M. Simon, J. Omura, and B. Levitt, '*Spread Spectrum Communications Handbook*', McGraw-Hill, 1994.

[65] Proxim–Wireless LAN Networking Products Based on Spread Spectrum Technology. http://www.proxim.com, 1998.

[66] Aironet Wireless Communications Home Page: http://www.aironet.com, 1998.

[67] Metricom, Inc. http://www.metricom.com and http://www.ricochet.net, 1998.

[68] M. Mouly and M. B. Pautet, '*The GSM System for Mobile Communications*', Cell & System, 1992.

[69] R. Quick and K. Balachandran, '*Overview of the Cellular Digital Packet Data (CDPD) System*', In Proc. of the PIMRC, pages 338-343, 1993.

[70] The WaveLAN Home page: http://www.wavelan.com, 1998.

[71] M. Ritter, '*The Metricom Autobahn Architecture*', Personal Communication, 1997.

[72] R. Fileding, J. Gettys, J. Mogul, H. Frystyk, T. Berners, '*RFC 2068: Hypertext Transfer Protocol*', January 1997 available at: http://sunsite.auc.dk/RFC/rfc/rfc2068.

[73] Jongkuk Lee, Myungchul Kim, Hee Yong Youn, Yusik Hahm, and Dongman Lee, '*Class-based Proxy Server for Mobile Computers*', Workshop on Wireless Networks and Mobile Computing, pages 559—566, IEEE, 2000.

[74] Intel: Quick Web. http://www.intel.com/quickweb/index.htm.

[75] Kawachiya, K. and Ishikawa, H., '*Improving Web Interaction on Small Displays*', in Proceedings of 8th International WWW Conference, 51-59, 1999.

[76] J. Smith, R. Mohan and C. Li, '*Content-based Transcoding of Images in the Internet*', Proc. Int'l. Conf. Image Processing, 1998.

[77] ImageMagick: a robust toolkit for reading, writing, and manipulating images in many image formats: http://www.wizards.dupont.com/cristy/ImageMagick.html

[78] J. A. Storer, '*Data Compression: Methods and Theory*', Computer Science Press, Rockville, Maryland, 1998.

[79] Network Working Group; RFC 1950, '*ZLIB Compressed Data Format Specification*', Version 3.3, 1996.

[80] Peter Danzig, Jeff Mogul, Vern Paxson, and Mike Schwartz, '*Internet Traffic Archive*', http://ita.ee.lbl.gov/

[81] Jeff Mogul and Tom M. Kroeger, '*Digital's Web Proxy Traces*', ftp.digital.com/pup/DEC/traces/proxy/webtraces.html, 1996.

[82] GOZILLA software: http://www.free-download.com/shareware-demo/gozilla.htm

[83] Mohmed El-Shantanawy, '*Accessing The WWW Over Low-BW Access Networks*', In Progress Thesis (M.C.S.) - Carleton University, 2000.

[84] X11 graphics programs and libraries: http://hpux.ee.ualberta.ca/hppd/hpux/X11/Graphics/alpha.html

[85] Pythia: ftp://daedalus.cs.berkeley.edu/pub/glomop/

[86] Jakob Nielsen, '*Usability Engineering*', Academic Press, Boston, MA, 1993. (hardcover), 0-12-518406-9 (paperback).

[87] Bitmap Information Tool: http://www.coli.uni-sb.de/~haase/pkg/

[88] T. Lane, P. Gladstone, L.Ortiz, J. Boucher, L. Crocker, J. Minguillon, G. Phillips, D. Rossi, and G. Weijers, '*The Independent JPEG Group's JPEG Software Release 6b*' ftp.uu.net/graphics/jpeg/jpegsrc.v6b.tar.gz

[89] M. El Shentenawy, A. Gaddah, Q. Guo, T. Kunz and R. Hafez, '*Image Transcoding for Proxy Internet Wireless Access*', Wireless 2000, Calgary, July 2000.

[90] H. Balakrishnan, S. Seshan, E. Emir, and R. H. Katz, '*Improving TCP/IP Performance Over Wireless Networks*', in First Annual Conference in Mobile Computing and Networking, ACM, 1995.

[91] H. Lei and D. Duchamp, '*Transparent File Prefetching*', Submitted for publication, March 1995.