Low-Bandwidth Web Access with Tandem Proxies

by

Samidh Chakrabarti

Submitted to the Department of Electrical Engineering and Computer Science
in partial fulfillment of the requirements for the degree of

Master of Engineering in Computer Science and Engineering

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

September 2002

Author .....
Department of Electrical Engineering and Computer Science
July 23, 2002

Certified by .....
Saman Amarasinghe
Associate Professor
Thesis Supervisor

Accepted by .....
Arthur C. Smith
Chairman, Department Committee on Graduate Theses

Low-Bandwidth Web Access with Tandem Proxies

by

Samidh Chakrabarti

Submitted to the Department of Electrical Engineering and Computer Science
on July 23, 2002, in partial fulfillment of the
requirements for the degree of
Master of Engineering in Computer Science and Engineering

Abstract

This thesis explores techniques for building special-purpose proxies to speed web access in the developing world. It presents a design for Tandem Proxies (TP), a system of two cooperating proxies that utilize a combination of caching, diffs, and compression to minimize data sent across poor quality bottleneck links. These three methods are then explored in greater depth: a procedure for aggressive caching is analyzed for its efficacy, a method of performing diffs relative to an entire database of files, rather than a single file, is detailed, and an algorithm for image compression with large dictionaries is specified and demonstrated. Taken together, the Tandem Proxy architecture can enable users in the developing world to access content that was previously beyond reach.

Thesis Supervisor: Saman Amarasinghe
Title: Associate Professor

Acknowledgments

I would like to express my gratitude to my thesis advisor, Saman Amarasinghe. Through his consistent encouragement, ability to reduce a problem to its essentials, and lucid communication of his fertile thoughts, he exemplified everything an advisor should be. The Immortal Lion has renewed my faith in that sacred scholarly connection that has for millenia bound guru and disciple.

Bill Thies, my graduate student colleague and friend, deserves an equal amount of thanks. Brilliantly tireless and tirelessly brilliant, he supported my project every step of the way. Bill will revolutionize the world in our lifetimes. I will look back on this time and realize what an honor it was to work with him.

Toliver Jue, Pavel Langer, and Ang-Chih Kao were my partners during an earlier incarnation of this project. The hours they spent scrutinizing protocol details at the coffeehouse and hacking code in the clusters is much appreciated.

I also owe a debt to my friend Robert Ragno. Rob is the Old Faithful of ideas; I could always count on him when my brainstorming required a reliable fountain of creative thought.

Finally, I thank my mother, my father, and my sister for *everything*. Their quiet confidence and unwavering support gave me the energy to prevail, even when the prognosis was bleak. The most valuable thing I learned at MIT over the past five years is the knowledge of how fortunate I am to be a member of my family.

Contents

1 Introduction **15**

1.1 Motivation 15

1.2 Overview 15

1.3 Project Goals 16

1.3.1 Design Criteria 16

1.4 Problem Analysis 17

1.4.1 Network Topology of the Developing World 17

1.5 Existing Techniques 19

1.5.1 Compression 19

1.5.2 Caching 20

1.5.3 Dynamic Tags 21

1.5.4 Diffs 21

2 Architecture **23**

2.1 Design Overview 23

2.2 Tandem Protocol 24

2.2.1 Cached Pages 24

2.2.2 Non-Cached Pages 24

2.2.3 Nearly-Cached Pages 25

2.2.4 Protocol Summary 26

2.3 Theoretical Expectations 27

2.3.1 Projected Strengths 27

2.3.2 Projected Weaknesses 28

2.4 Similar Projects 29
2.4.1 Rproxy 29
2.4.2 Spring and Wetherall 30
2.4.3 Low-bandwidth File System 30
2.5 Experimental Techniques 30
2.5.1 Content Type Selection 31
2.5.2 Diffs 32
2.5.3 Compression 32

3 Byte-Level *n*-Diff **33**
3.1 Task Description 33
3.2 Standard Approach 35
3.3 Source Statistics 36
3.3.1 Data Sets 36
3.3.2 File Size Distribution 36
3.3.3 Byte Distributions 37
3.3.4 Byte Repetition 38
3.4 *n*-Diff Algorithm 40
3.4.1 Hamming Distance 40
3.4.2 Block Encoding 40
3.4.3 Fast Similarity Search 42
3.5 Evaluation 43

4 *Mosaic* Compression **45**
4.1 Task Description 45
4.2 *Mosaic* Algorithm 47
4.2.1 Assumptions 47
4.2.2 Decomposing Images 47
4.2.3 Reconstructing Images 48
4.2.4 Perceptual Similarity Search 48
4.3 Results 50

4.3.1 Prototype 50
4.3.2 Performance 50
4.3.3 Quality 51
4.3.4 Compression 51
4.3.5 Informal Poll 53
4.3.6 Lessons 53
4.4 Popular Image Formats 55
4.4.1 GIF 55
4.4.2 JPEG 56

5 Conclusion **57**
5.1 Summary of Techniques 57
5.1.1 Aggressive Caching 57
5.1.2 Diffs 58
5.1.3 Compression 58
5.2 Future Research 59
5.2.1 Caching 59
5.2.2 Cache Updates 60
5.2.3 New Content 60
5.3 Impact 61

List of Figures

1-1 Strathmore College Network Topology 18

1-2 Ping statistics from MIT to Strathmore 19

2-1 Tandem Proxy Architecture Overview 24

2-2 Decoding Proxy Protocol Pseudocode 27

2-3 Encoding Proxy Protocol Pseudocode 28

3-1 The Standard 1-Diff Task 34

3-2 The Tandem Proxy n-Diff Task 35

3-3 Database Footprints as a Function of File Size 37

3-4 Distribution of Individual Bytes in Image Databases 38

3-5 Probability of Repeated Byte Patterns in Image Databases 39

3-6 Structure of Block-Level Diffs 41

4-1 Perceptual similarities of two skies? (Photo Credit: EPA) 46

4-2 Mosaic and Original of ZDNet Logo 51

4-3 Mosaic and Original of Young Man Eating Cotton Candy 52

4-4 Pictures from Informal Poll on Mosaic Quality 54

List of Tables

2.1 Breakdown of Web Traffic by Content Type 31

3.1 n-Diff Performance on 16 MByte Database 44

Chapter 1

Introduction

1.1 Motivation

During the summer of 2001, I had the unique opportunity to teach computer programming to students at Strathmore College in Nairobi, Kenya. One day during class I noticed that one of my students was attempting to read a web page on HIV/AIDS published by the Centers for Disease Control in Atlanta, Georgia. Internet connections in Kenya, and most of the developing world, are extremely slow. So it was no surprise to me when I observed that the HIV web page did not load in its entirety. Key images were missing, the text was cut off, and the menus were nonfunctional. The page, in short, was unreadable. I did not think of this episode again until three weeks later when the very same student of mine was hospitalized for Hepatitis C, a virus correlated with HIV infection. Sadness turned into anger and anger turned into resolve. Certainly, I thought, there must be techniques to make web access faster in developing countries. That experience was the genesis of this thesis project.

1.2 Overview

This thesis is composed of five chapters.

The remainder of this chapter describes the design challenges of this project and surveys existing techniques researchers have devised to address this problem.

Chapter two presents the architecture and communications protocol of the Tandem Proxy (TP) system.

The following two chapters detail the two experimental techniques the Tandem Proxies could utilize: chapter three explores byte-level diffs, and chapter four discusses compression with large dictionaries.

Finally, chapter five concludes the thesis with a summary of the relative merits of each technique and provides a roadmap of possible future research directions.

1.3 Project Goals

To state it simply, *the goal of this project is to explore a variety of compression and caching techniques that can potentially improve delivery speeds of web pages without sacrificing data freshness.* Whereas other projects have sought to deliver content using offline store-and-forward methods [1], our emphasis is on enhancing the online browsing experience.

1.3.1 Design Criteria

While dozens of known techniques could be applied to the problem of speeding web access to the developing world, the solutions we explore aim to satisfy the following design constraints:

- No modification of existing web browsers or web servers.
- Compliance with HTTP 1.0 and HTTP 1.1 protocols.
- Downloaded content must not be stale, unless allowed by HTTP RFCs.

We wanted to devise a solution that did not only function on custom-configured systems in the laboratory, but one that would also be easily deployable out in the field. Hence, we exclude from consideration designs that would require special purpose modification of existing web browsers or web servers. Similarly, few people would adopt a solution that was non-compliant with the existing HTTP standards [2] due

to a fear of introducing unforeseen incompatibilities into their browsing experiences. Finally, we exclude solutions that force users to accept stale content in exchange for improved delivery speed. Such solutions hint of geographic discrimination and further reinforce the digital divide.

1.4 Problem Analysis

Before we can consider potential solutions, it is necessary to analyze why internet connections in the developing world slow to a trickle in the first place. The reasons, it turns out, are often more political than they are technical. As a case study, we describe the network topology of Strathmore College in Nairobi, Kenya.

1.4.1 Network Topology of the Developing World

Strathmore College has approximately two hundred Pentium-class PCs linked together on a 10-BASE-T ethernet LAN. Due to the prohibitive cost of high-bandwidth connections, the entire school shares a single 128kbps link to Africa Online, a local ISP. Africa Online, in turn, is forced for political reasons to send all of their data out to the internet through a single 2Mbps satellite link managed by the Kenyan telecom monopoly.

Most of the websites commonly visited by students at Strathmore are located in the United States and Europe. In particular, portal sites such as Yahoo! and web-based email sites such as Hotmail are exceedingly popular. As a consequence, most of Strathmore's HTTP requests require transit through Kenya Telecom's 2Mbps satellite link, where their bits march side by side along with all of the data from the rest of the nation. This excessive multiplexing produces a bottleneck that nearly paralyzes Strathmore's ability to access popular websites. Figure 1-1 illustrates Strathmore's network topology and its connection to the external world.

My on-site experiences at Strathmore revealed that the quality of the link was poor not only in terms of bandwidth, but also reliability and latency. Web transfer speeds from portal sites based in the United States averaged 300 bytes per second,

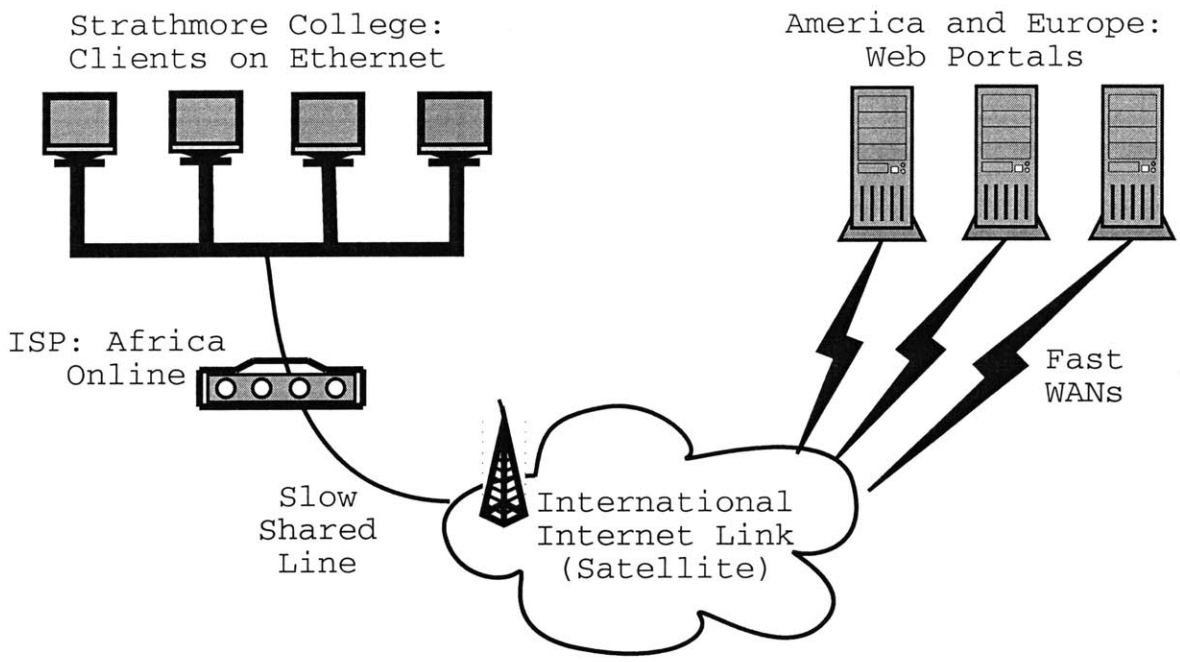

Figure 1-1: Strathmore College Network Topology

when not stalled. Typically, nearly thirty percent of these packets were dropped completely. Those that were not dropped required on the order of 800ms to make the one-way journey. Ping statistics for packets sent from MIT to Kenya are shown in Figure 1-2.

Presuming that Strathmore is representative of many institutions in the developing world, we extracted the following key observations about the developing world's network topologies and surfing habits:

- Each institution houses a fast LAN such as 10B-T ethernet.

- Users popularly access portal websites in the USA or Europe.

- Data exchange with the USA or Europe is high-latency, low-bandwidth, and unreliable.

```
athena% ping www.strathmore.edu
PING www.strathmore.edu (212.49.90.225) from 18.56.0.59 : 56(84) bytes of data.

64 bytes from 212.49.90.225: icmp_seq=1 ttl=236 time=1.763 sec
64 bytes from 212.49.90.225: icmp_seq=2 ttl=236 time=1.708 sec
64 bytes from 212.49.90.225: icmp_seq=3 ttl=236 time=1.758 sec
64 bytes from 212.49.90.225: icmp_seq=4 ttl=236 time=1.895 sec
64 bytes from 212.49.90.225: icmp_seq=5 ttl=236 time=1.856 sec
64 bytes from 212.49.90.225: icmp_seq=6 ttl=236 time=1.836 sec
64 bytes from 212.49.90.225: icmp_seq=7 ttl=236 time=1.671 sec
64 bytes from 212.49.90.225: icmp_seq=8 ttl=236 time=1.656 sec
64 bytes from 212.49.90.225: icmp_seq=10 ttl=236 time=1.714 sec

--- www.strathmore.edu ping statistics ---
13 packets transmitted, 9 packets received, 30% packet loss
round-trip min/avg/max/mdev = 1656.038/1762.360/1895.232/79.435 ms
```

Figure 1-2: Ping statistics from MIT to Strathmore

1.5 Existing Techniques

As is evident from the network topology, internet access in Kenya is painfully slow because most of the traffic is forced to travel along a transatlantic bottleneck link. The most effective way to speed content delivery is to reduce the amount of data transmitted across the bottleneck link. All of the existing techniques described below attempt to curtail such traffic.

1.5.1 Compression

Many existing systems utilize compression to curb network traffic. The originating web server compresses the content and sends it to the client, who then decompresses the data to recover the original content. Since most web content is an amalgam of text and images, it is worth discussing these data types separately.

Images, such as JPEGs and GIFs, are hardly compressible at all because the image formats are already encoded in a fashion to minimize file size. Consequently, these file formats are immune to standard lossless compression algorithms.

On the other hand, the best known text compression algorithms, such as PPM (Prediction by Partial Matching), typically achieve 2:1 savings on small files. This is a significant result. Though the HTTP standard [2] supports the transfer of compressed

web content (other than headers), webmasters rarely exploit this feature. Apache, for example, will send a compressed file in lieu of a raw file only when a compressed version of the file is pre-existing on disk. To conserve processor cycles, no online compression takes place. That means if webmasters want to send compressed content, they must manually duplicate all of their content periodically in compressed form. For most webmasters, it is not worth the trouble or the potential complexity.

We return to the topic of compression in the next chapter and show how Tandem Proxies can provide a transparent solution without burdening existing web servers or clients.

1.5.2 Caching

The most widely adopted technique used to relieve network congestion is caching. When a file is originally downloaded from a server, the client stores the file locally in memory. In the future if the file is requested again, the client fetches the file from memory rather than re-requesting it from the original server. Caches are frequently incorporated into HTTP proxies so that multiple clients can share the same pool of pages. The Squid caching web proxy (`http://www.squid-cache.org`) is one of the most popular.

Traditional caches and proxies can only achieve this savings if the cached file and the remote file are exactly the same. As the web becomes populated with increasingly dynamic content, the likelihood of a page being cachable is plummeting, particularly with portal sites. When loading the front page of CNN twice in a row, for example, the text is not cachable because CNN's webserver dynamically stamps the page with the current time. Hence, "all-or-nothing" caches are becoming ineffective.

In the next chapter, we show how Tandem Proxies can relax strict HTTP caching rules to achieve better performance.

1.5.3 Dynamic Tags

Website designers are gradually becoming aware of the non-cachable nature of dynamic pages. Some have proposed extensions to HTML that would allow a content creator to tag certain regions of a webpage as having dynamic content. To return to the CNN example, the area of the page with the time would be encased in special tags denoting dynamic content. A smart browser could then recognize the dynamic tags and use them to improve the efficiency of its cache. For the foreseeable future, however, dynamic tags are not a realistic method of reducing network traffic because they require massive adoption by website designers. We do not consider this technique to be viable for this project.

1.5.4 Diffs

One of the most promising methods of exploiting the redundancy of content within portal websites is to use diffs. Under this scheme, if the client has a page already cached, it can simply query the web server for a description of the changes that have since been made to the webpage. Because only the differences are transmitted over the network, the traffic is greatly reduced for dynamic pages that retain common elements. This technique, however, places a burden upon web servers by requiring them to retain a history of versions for each webpage and to compute diffs between pairs of versions.

Though the idea of a diff is straight forward on text sources, the concept of a diff between two images is more hazy. Is it simply the byte-level difference between two images? In that case, can we expect a pair of similar images to be similar on a bit by bit basis? Or is it good enough to only express the differences between two perceptually distinct pixels? These questions concerning image diffs and compression form the heart of chapters three and four.

Chapter 2

Architecture

This chapter presents our proposed Tandem Proxy architecture. Section 2.1 provides a brief overview of the design. Section 2.2 describes the communications protocol the proxies use to communicate with each other and Section 2.3 analyzes the theoretical expectations of such a scheme. We contrast our approach with similar projects in Section 2.4. The Tandem Proxy architecture allows the proxies to use a variety of cooperative methods, so we end the chapter in Section 2.5 with an introduction to the experimental techniques that we will explore in future chapters.

2.1 Design Overview

The solution we propose utilizes a pair of proxies that straddle the low-bandwidth low-reliability high-latency link. By using a combination of caching, diffs, and compression, the tandem proxies (TPs) are able to reduce the traffic across the slow link. This architecture can be realized at low cost given the voluminous, cheap hard disks available on the market. In essence, we attempt to trade away disk space in exchange for time.

Figure 2-1 depicts the core architecture of how the TPs are to be placed in the network topology. Adopting terminology introduced by the Rproxy [3] project, the proxy on the LAN in the developing country is referred to as the decoding proxy (DP). The proxy on a LAN in the USA or Europe is called the encoding proxy (EP).

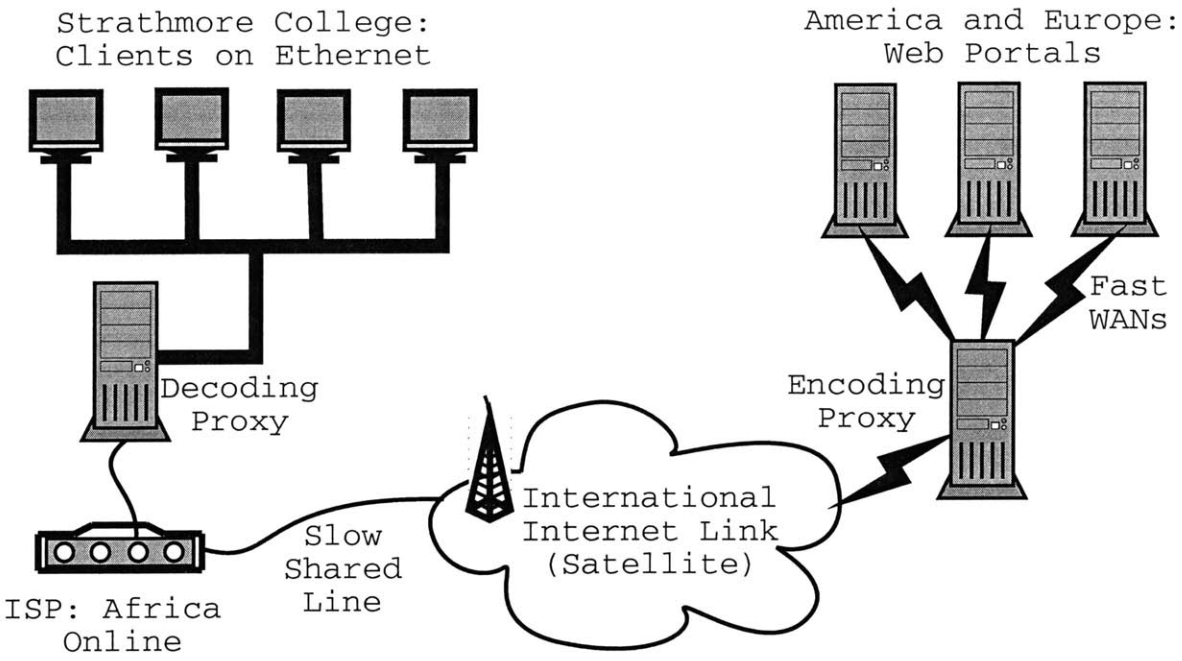

Figure 2-1: Tandem Proxy Architecture Overview

2.2 Tandem Protocol

To better understand the mechanics of the EP/DP tandem operations, it is necessary to examine what messages the proxies exchange with each other in the context of cached pages, non-cached pages, and "nearly-cached" pages.

2.2.1 Cached Pages

When a client requests a webpage from the DP, the DP first checks to see if the content is cached. If the file is cached and not expired, then the DP returns the page to the client as a normal proxy would do. Since the DP is on the client's LAN, this connection should be fast and will not require any traffic across the poor quality link.

2.2.2 Non-Cached Pages

If the page is not cached at all, then the DP queries the EP for the webpage. The EP fetches the page from the internet if it does not already have it cached. Before sending

the file to the DP in a compressed format, the EP caches the file itself. Once the DP receives the compressed content, it decompresses the file. The DP then caches it, even if it has a "Pragma-nocache" header, and finally sends the content to the original client. In this fashion, the DP caches every precious bit of data sent across the low bandwidth link. We refer to this technique as *aggressive caching*. Though it may seem a violation of the HTTP specification, it is not: the DP will never send it to the client unless it is guaranteed to be fresh (see Section 2.2.3).

When compared to a standard single proxy scheme where data travels directly from the server to the proxy, this method has the potential of introducing more latency because the data must travel a longer route first from the server to the EP and then from the EP to the DP. However, given that the EP has a much faster connection to the original web server than it has to the DP, this extra latency is not significant; the poor quality link between the EP and DP is still the bottleneck. Furthermore, since the content across the poor link is compressed, it reduces the traffic and potentially increases delivery speed.

2.2.3 Nearly-Cached Pages

The most interesting case is when a file is "nearly cached" on the DP. By nearly cached, we mean that the file is currently cached but for some reason the DP must verify the content of the file. This can happen, for example, if either the client explicitly reloads a page or if the cached file was not allowed to be cached by HTTP RFCs.

Now the DP's key task is to verify that the cached file is fresh. It sends a message to the EP asking for updates to the file. The EP fetches the file from the webserver and analyzes differences between the fresh file and the version that the DP is known to have. Since the EP and the DP have been maintaining consistent caches, the EP has knowledge of what version of the file the DP is currently storing. Once the differences are calculated, the EP compresses the description of the differences and sends it to the DP. If the diff is larger than the original file, then the EP simply sends the original file in compressed form. When the DP receives the response, it

decompresses the message and brings its version of the file up to date if necessary. Finally, it passes the data to the client.

If it turns out that the nearly-cached file is equivalent to the fresh file, then only a few packets of data are transferred across the poor link. The request essentially becomes a variation on the HTTP 1.1 "Conditional GET" request. The conditional GET is a type of message that a client can send to a server to request transmission of a file only if it has been modified after a specified time. Hence performance under this scheme can be no worse than the conditional GET.

If the nearly-cached file has changed slightly, as pages on portal sites often do, then this scheme offers a dramatic advantage over any other technique. Since only the differences are transmitted across the poor link, the overall bandwidth savings could be a significant fraction. Most dynamic content, which under any other scheme would be uncachable without the sacrifice of either transparency or consistency, will benefit from these savings.

Though rare, it is possible that the file has changed entirely. In terms of the amount of data transferred over the poor link, it will be as if the file had never been cached on the DP in the first place, so Section 2.2.2 applies.

No matter whether the file has changed or not, the DP is now guaranteed to have a fresh version of the file, so the HTTP specification is not violated even if the DP had originally cached the file despite seeing a "Pragma-nocache" header.

This protocol assumes that the partner proxies maintain consistent caches, which can often be a challenge. But in the event that the caches lose synchrony, the system would only lose efficiency and not accuracy. If the diff refers to a page that is no longer cached, the DP can report this back to the EP as a cache miss. The EP will then send the original version of the file and update its mirrored cache to note the miss.

2.2.4 Protocol Summary

To summarize the operations implemented by the tandem proxies, pseudocode for the communications protocols is provided below. Figure 2-2 specifies the protocol used

```
PROTOCOL FOR DECODING PROXY

[Input: HTTP Request from Client, Output: HTTP Response to Client]

if (Request is Cached) {
     Return Cached File
}
else if (Request is Aggressively Cached) {
     Create Update Request
     Compress Request
     Send Compressed Request to EP
     Wait for Response from EP
     Decompress Response
     Update Cached File by Merging Diff
     Return Updated File
}
else { // Brand New Request
     Create New Request
     Compress Request
     Send Compressed Request to EP
     Wait for Response from EP
     Decompress Response
     Aggressively Cache Response
     Return Received File
}
```

Figure 2-2: Decoding Proxy Protocol Pseudocode

by the decoding proxy and Figure 2-3 specifies the protocol used by the encoding proxy. Since clients and servers require no modification, they retain the protocols set forth in the HTTP RFCs.

2.3 Theoretical Expectations

2.3.1 Projected Strengths

In terms of theoretical performance, the tandem proxies either meet or far exceed the content delivery speeds of traditional proxies:

- For cached pages, the TPs behave like traditional caching proxies.

```
PROTOCOL FOR ENCODING PROXY

[Input: HTTP Request from DP, Output: HTTP Response to DP]

Decompress Request
if (Request is of type Update) {
     Lookup Old File in Mirrored Cache
     Query Original Server for New File
     Calculate Diff between Old and New File
     Store New File in Mirrored Cache
     Create Response from Diff
}
else { // Request is of type New
     Query Original Server for New File
     Store New File in Mirrored Cache
     Create Response from New File
}
Compress Response
Return Compressed Response
```

Figure 2-3: Encoding Proxy Protocol Pseudocode

- For non-cached pages, the TPs utilize compression for performance no worse than traditional proxies.

- For nearly-cached pages, the TPs utilize aggressive caching, compression, and diffs for dramatically superior performance.

The tandem proxies also implement a fully transparent solution:

- Clients see DP as a regular proxy. No special configuration is required.

- Servers see EP as a regular proxy. No special configuration is required.

- Aggressive cache verification ensures HTTP RFC compliance.

2.3.2 Projected Weaknesses

The chief obstacle to deploying this protocol is that it requires someone who wants to setup a DP in the developing world to find a partner in the USA or Europe willing to

setup an EP to complete the pair. While the EP would not consume much bandwidth for the host in the industrialized world, it would still require a hardware investment and occasional administration. Forging these partnerships may be a challenge and may limit scalability of the system.

From an operational standpoint, having a system so geographically separated could pose recovery problems as well. If the EP were to go offline, then the DP would be left incapacitated. Since the two proxies are likely to belong to different organizations, getting an offline proxy back online may not be a rapid process. Of course, the DP could simply revert to becoming a traditional proxy if it cannot find its partner EP, but then the system would lose any performance gains that the tandem proxies provide.

The last significant weakness of this design is that it is not sufficiently generic to serve as a solution to improving content delivery speeds to all low bandwidth clients. Only network topologies that have a well-defined bottleneck can benefit. This solution does not apply, for example, to speeding content delivery to clients on congested networks.

2.4 Similar Projects

In conducting this research, we encountered three projects similar in spirit to the Tandem Proxy design, but none of them were adequately suited for this application domain. We now briefly review them.

2.4.1 Rproxy

Rproxy [3] is an open source project intended to promote the integration of Rsync into the HTTP standard. Andrew Tridgell developed Rsync in his PhD thesis [4] as an efficient means of synchronizing two versions of a file. Instead of calculating an explicit diff, which would require the processor to have both versions of the file on hand, the Rsync method involves a client and server exchanging checksums as a means of identifying identical blocks. Rproxy consists of a pair of proxies utilizing Rsync

to reduce bandwidth consumption across bottleneck links when files need updating. We decided that the Rproxy approach was not adequate for unreliable links since the chatty exchange of checksums could handicap the system. In Tandem Proxy, we chose instead for the partnered proxies to maintain consistent caches, eliminating the need for checksum exchange.

2.4.2 Spring and Wetherall

In the same spirit as Tandem Proxy, Spring and Wetherall [5] presented a design where two caches on opposite ends of a bottleneck link cooperate to reduce traffic in a protocol-independent fashion. Each cache stores several megabytes of the most recent packets. When a new message is to be sent, the cache replaces any redundant data with references to previously cached packets. Inspired by Spring and Wetherall's approach, we attempted to extend their idea in a manner that takes advantage of cached blocks that are similar but not necessarily identical to the blocks of the new message (see Section 2.5.2).

2.4.3 Low-bandwidth File System

The Low-bandwidth Network File System (LBFS) [6] attempts to use smart caching to reduce file system transactions. A client caches open files and breaks them up into chunks. When the file is modified, only the edited chunks are transmitted. Though this innovative solution is well-tailored for file systems, it did not seem ideal for web browsing. The LBFS is optimized for the frequent editing of files, where content is inserted or deleted. This condition does not often hold for the majority of web content.

2.5 Experimental Techniques

Given the Tandem Proxy architecture and the specified communications protocol, the fundamental research question becomes which compression and diffing schemes

Table 2.1: Breakdown of Web Traffic by Content Type

Content-Type	References	MBytes
image/gif	53%	36%
text/html	23%	21%
image/jpeg	13%	20%
application/octet-stream	9%	13%
other	2%	10%

are most effective for this application? Notice that in the protocol pseudocode the mechanics of these operations are unspecified. In practice, any scheme could be implemented, but to get a sense of promising research directions we tried a novel experimental technique for each of these operations.

2.5.1 Content Type Selection

The best techniques will be dependent on what type of content is being delivered. For example, PPM has been shown to be much more effective for text compression than image compression. And the discrete cosine transform (DCT) has been shown to be much more effective for image compression than text compression. This should come as no surprise to compression researchers, who have long understood that the best algorithms are those that capitalize on the high-level structure and properties of the data source.

Consequently, to carry out our experiments, we needed to choose a specific data type to use in the tests. Our results would be most relevant, we felt, if we chose the data type that constituted the majority of web traffic in terms of bytes transferred. A 1997 study conducted jointly between Digital Research Laboratory and AT&T Labs examined the breakdown of web traffic by content type [7]. Table 2.1 is a short excerpt of their findings.

Images in the form of GIFs and JPGs compose the vast majority of web traffic both in terms of web page references (66%) and total bytes (56%). Hence, we chose images as the subject of our experiments. Although these results are from 1997

and the distribution of data types must have changed, it most likely has shifted even further in the direction of images as websites have become more graphically intensive.

2.5.2 Diffs

The first experiment concerned how much savings could be achieved by performing a byte-by-byte diff between an image and a large database of cached images. Whereas other binary diff schemes calculate differences based on sequences of matching bytes, the method we explore is more lenient. It is capable of expressing differences based on sequences of similar, though not necessarily identical, bytes. Chapter three details the methodology and results of this experiment.

2.5.3 Compression

Rather than transmitting a byte-level diff, another approach to compression is to try to transmit instructions to the DP that describe how to construct an image perceptually similar to the source image based on image blocks the DP already has cached. Under this scheme, the EP fragments the source image into a number of fixed-size tiles. Then the EP attempts to identify image tiles known to be cached on the DP that are perceptually similar to the source block. Once these blocks are identified, the EP simply sends references of each tile fragment to the DP. This technique takes center stage in chapter four.

Chapter 3

Byte-Level n-Diff

This chapter introduces an experimental byte-level diffing algorithm. This algorithm can quickly compute diffs between a source file and a database of cached files down to the granularity of individual bytes. First, in Section 3.1, we review the task the algorithm is meant to solve. Section 3.2 briefly discusses standard approaches to this problem. To predict the efficacy of the algorithm, Section 3.3 presents important statistics on byte distributions in images. Section 3.4 describes the proposed algorithm in detail, and Section 3.5 finally evaluates its performance.

3.1 Task Description

Stated simply, given a large database of cached data, how can we minimally encode a new file by referring to the cached data? This is relevant to the Tandem Proxy design because each proxy maintains a cache mirroring the contents of its partner proxy. When the two proxies wish to send messages to each other, they should take maximum advantage of the data their partner is known to have.

For clarity, this task is different from the standard diff task. In the standard task [8], the algorithm must compute the differences between a new source file and an old reference file. Where there is redundancy between the two files, the algorithm outputs a pointer to the repeated byte sequence in the reference file. This scenario is depicted in Figure 3-1.

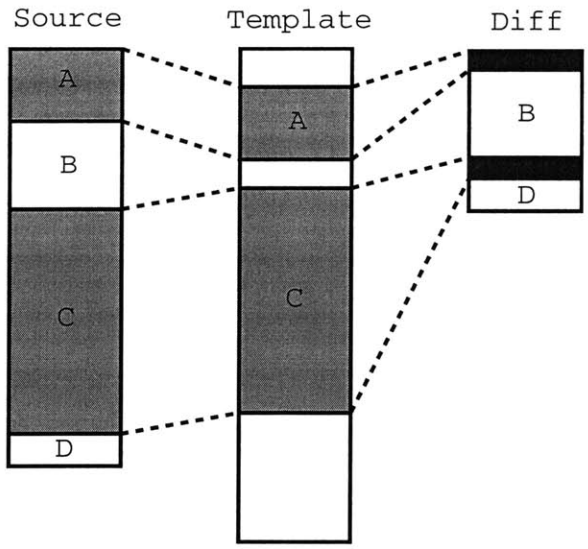

Figure 3-1: The Standard 1-Diff Task

In the Tandem Proxy task, however, the algorithm must compute the differences between a new source file and an entire database of reference files that could number in the millions. We refer to this as the n-diff task, which is illustrated in Figure 3-2. The algorithm should exploit redundancy between the source file and all other files in the database. Potentially, repeated byte sequences could be found throughout the database and the algorithm should output pointers to all of these reference files. Since the search space for this task scales with the size of the database, the complexity demands are far greater than in the standard diff task.

One condition we impose upon this problem is that the diffing scheme must be lossless, meaning that the partner proxy must be able to perfectly recreate the original data on a byte by byte basis. So although this chapter primarily discusses the application of these algorithms to images (for the reasons listed in Section 2.5.1), keep in mind that they are applicable to any type of data, whether they be text, audio, or even movies.

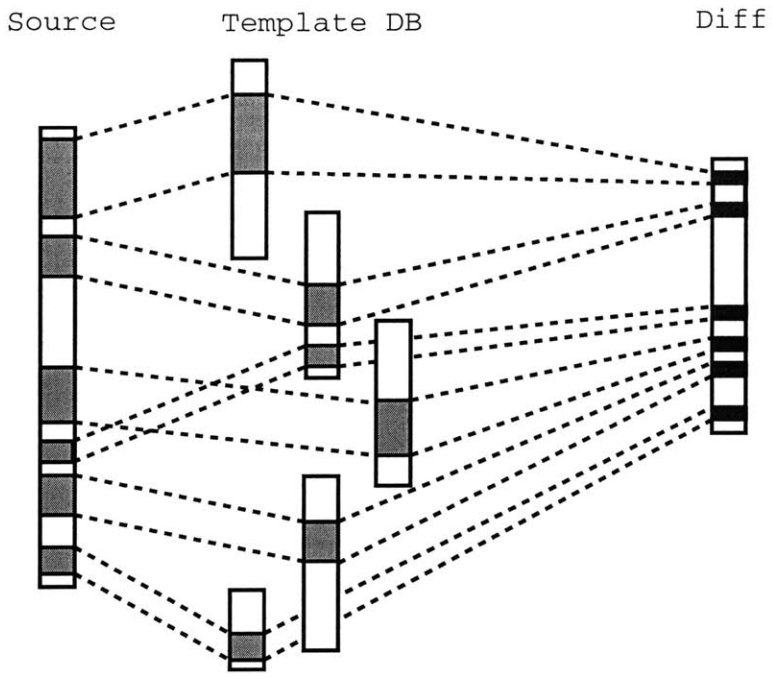

Figure 3-2: The Tandem Proxy n-Diff Task

3.2 Standard Approach

The standard approach to this problem is well known to cache designers. First, partition the database into fixed size blocks. Then, compute a hash code for each block and store this value in a table. When a new file is to be encoded, split it into fixed size blocks also. For each block of the new file, compute its hash code to see if it matches any blocks in the database. If there is a match, transmit the signature of the cached block. Otherwise, the raw data should be transmitted. Note that this is the same approach that Spring and Wetherall used in their system [5].

Observe, however, that this scheme does not necessarily produce the minimal-length diff. The system is limited by the granularity of the fixed size blocks. For a positive match, it requires all of the bytes within the cached block to be exactly identical to the target block in the new file. What if we relaxed that constraint and instead tried to take advantage of blocks that are only partially similar but not identical? That is the key idea behind the n-diff algorithm we present in Section 3.4. Though it is surely slower than the standard approach because of the expansion of

the search space, the ultra low-bandwidth of developing world internet connections affords us enough extra time to do the necessary processing.

3.3 Source Statistics

Before we describe our algorithm, it is worth examining some key statistics of the data source to make sure that our diff scheme will be worthwhile in the first place.

3.3.1 Data Sets

For these experiments, we created two separate databases of images. One database was composed of all of the JPG and GIF images on sixteen thousand educational webpages, as compiled by Google for their 2002 programming contest. This database, which we will call our "large" database, consisted of 42,684 images and had a total size of 638 MBytes.

The second database was composed of all of the JPG and GIF images on the one thousand most popular internet websites, as voted by ZD Net readers. This database, which we will call our "small" database, consisted of 5,540 images and had a total size of 16 MBytes.

3.3.2 File Size Distribution

One of the most striking things about the data sets was the distribution of file sizes. As the plot illustrates in Figure 3-3, files smaller than 10 kilobytes contribute to more than fifty percent of the total number of bytes in the small database. Even when each file in the database is compressed using a standard zip algorithm, this fact holds.

That means that the bulk of internet image traffic is not tied up by banners and photographs. Instead, tiny items like buttons, bullets, and spacers are the true bandwidth gluttons. Considering the mechanics of HTTP, where each file is requested in a separate transaction with separate headers, these small images consume an even greater proportion of traffic than shown in the diagram.

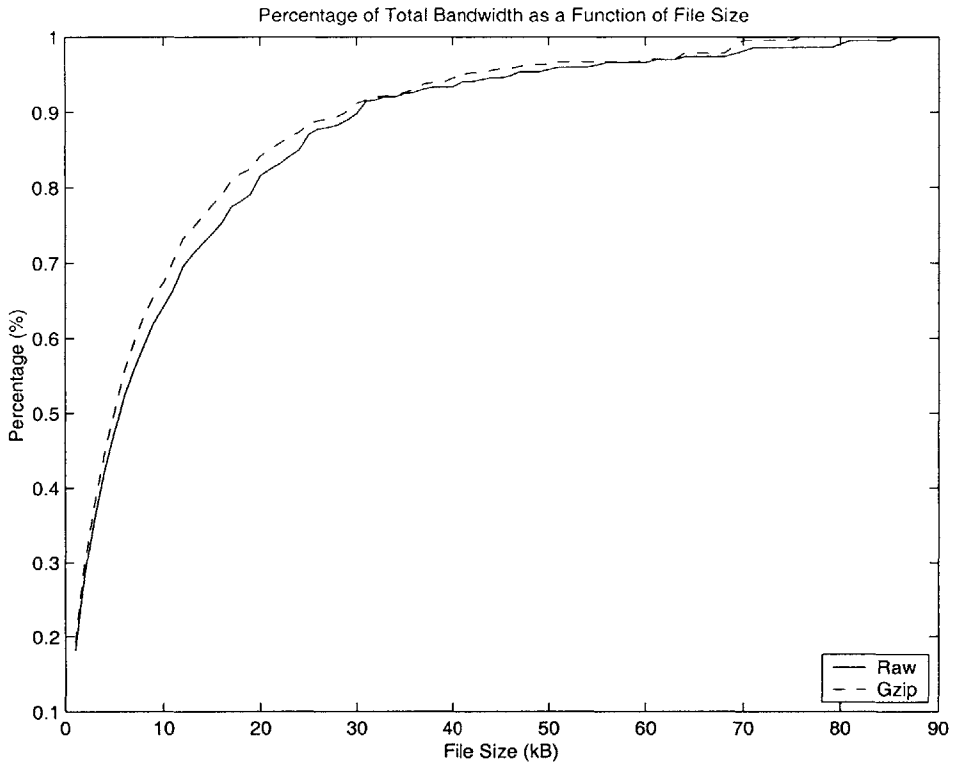

Figure 3-3: Database Footprints as a Function of File Size

The major lesson to draw from this statistic is that compression and diff techniques that operate effectively on small images will be particularly potent solutions.

3.3.3 Byte Distributions

We also wanted to characterize the distribution of individual bytes in the database. For each possible byte, we counted the percentage of time it appears in the database. Figure 3-4 is a histogram showing the frequency of occurrence of each possible byte. The foreground (black) data set is for the small database and the background (red) data set if for the large database.

Given that JPGs and GIFs are highly compressed formats, we expected the byte distribution to be nearly flat. That is, each byte should occur with equal likelihood. The distribution turned out not to be completely level. There were noticeable peaks at the zero byte and every multiple of thirty-two bytes. Thinking in binary, this just

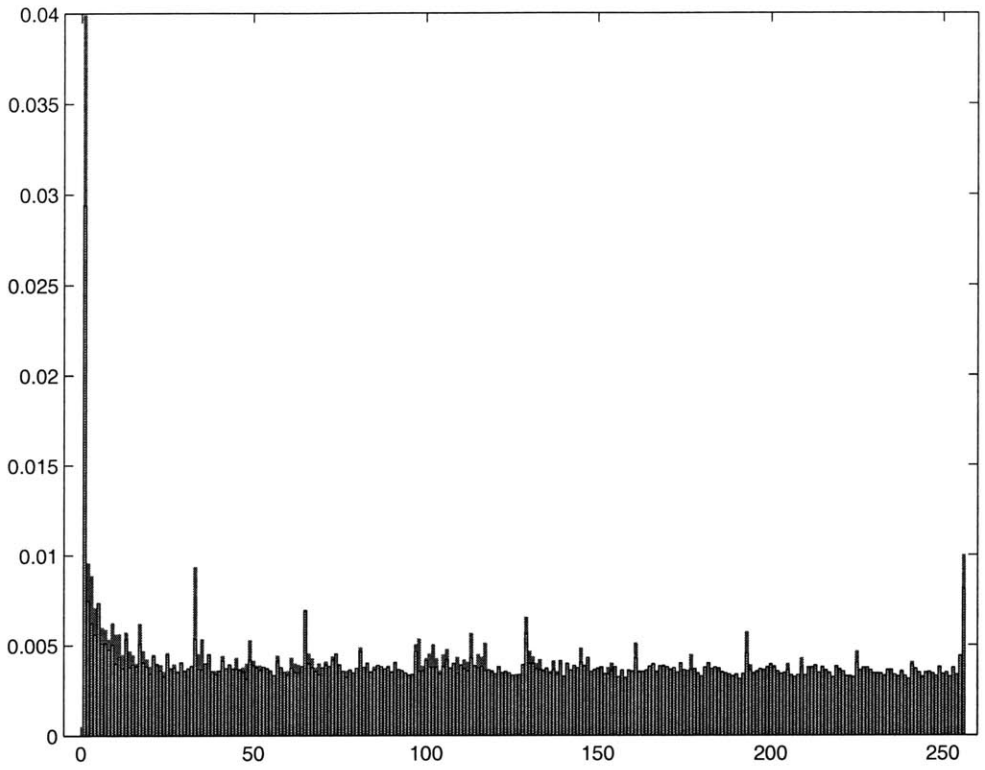

Figure 3-4: Distribution of Individual Bytes in Image Databases

means that strings of zeros come up more frequently than others.

From the look of Figure 3-4, it may be possible to design a special Huffman code for these data sets. But don't be misled. The most frequently occurring byte only has a four percent incidence rate, so although the distribution is not flat, any Huffman code would only achieve meager savings.

3.3.4 Byte Repetition

Knowing that it is not possible to take advantage of the distribution of bytes, the most important question pertaining to the diff problem is how often sequences of bytes repeat in the database. If there are no repetitions, then a diff scheme is futile. But if there are many sequences that repeat, then diffs can be promising.

For both databases, we calculated the probability of finding a repeated byte sequence. Within each database, we selected a random byte sequence s of length w and

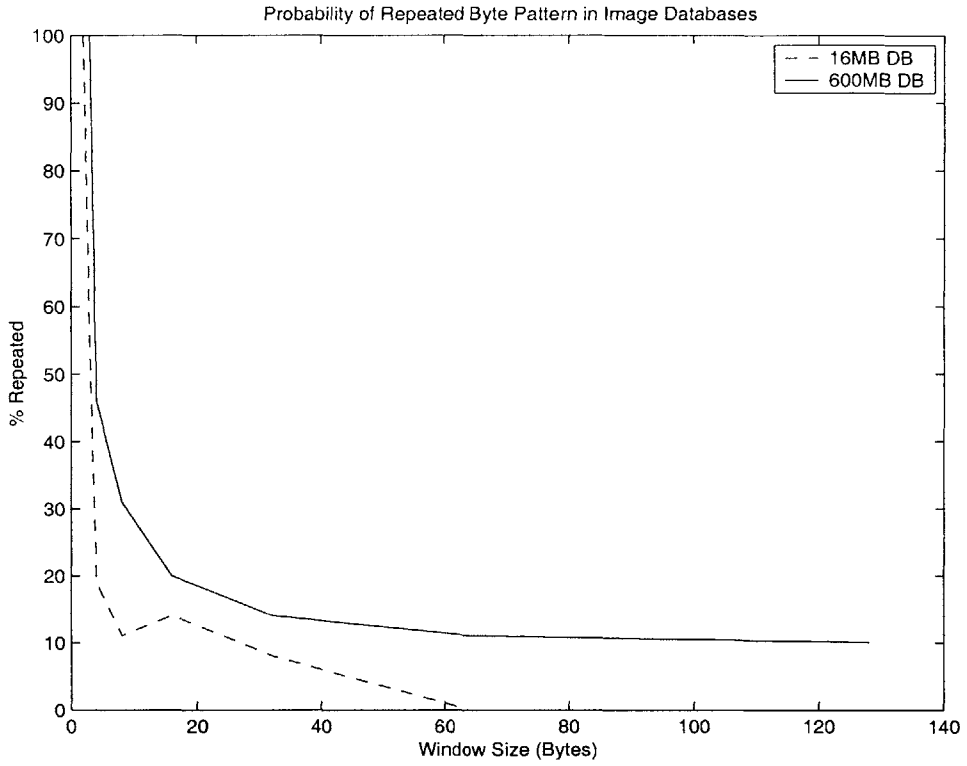

Figure 3-5: Probability of Repeated Byte Patterns in Image Databases

exhaustively searched the database for a repeated instance of s. We repeated this procedure for thousands of different s over a range of window lengths w.

Figure 3-5 is a plot of the gathered statistics. As intuition would suggest, the probability of a byte sequence repeating in the large database is greater than in the small database. And as w increases, the probability of a matching s of length w falls exponentially. This is a nice empirical demonstration of the law of large numbers.

But is there enough repetition to justify a diff scheme? If we were to build a cache of up to several gigabytes, we would only need 4 bytes to address any cached block. The plot shows that approximately 30% of 8 byte blocks will be repeated in a large database. Therefore, 30% of the time we can get 4 bytes of savings on each 8 byte block; even a naive diff scheme could get in the neighborhood of 15% savings.

3.4 n-Diff Algorithm

Like the standard approach, our algorithm also partitions the cached database into fixed size blocks. When a new file is to be sent, it is also split into fixed size blocks. Now comes the point where our algorithm differs from the standard method. Rather than trying to find a cached block that matches the source block identically, we try to find the cached block that is most similar to the source block. Once that block is found, the differences between the two blocks are encoded and transmitted. If no block in the cache has sufficient similarity, the raw source block is transmitted.

In essence, this can be looked upon as a two-level recursive diff. The initial block partition yields a coarse level of diff granularity, and then the block-level similarity search achieves a byte-level granularity.

Three major issues warrant special attention. First, what is the similarity metric to be used while conducting the block-level search? Second, how should the byte-level diffs be encoded? Finally, how can the block-level search be performed quickly on the database?

3.4.1 Hamming Distance

Though many reasonable similarity metrics exist, our n-Diff algorithm uses Hamming Distance. Two strings are said to be a hamming distance k apart if and only if they differ in exactly k symbols. For example, "abddf" and "bbdcf" have hamming distance of 2 since they require two symbols to be replaced in order to match each other.

3.4.2 Block Encoding

Once the cached block with minimum hamming distance to the source block has been found, the differences must be encoded in a non-ambiguous form.

Three pieces of information must be transmitted:

- The ID of the most similar cached block (the template block).

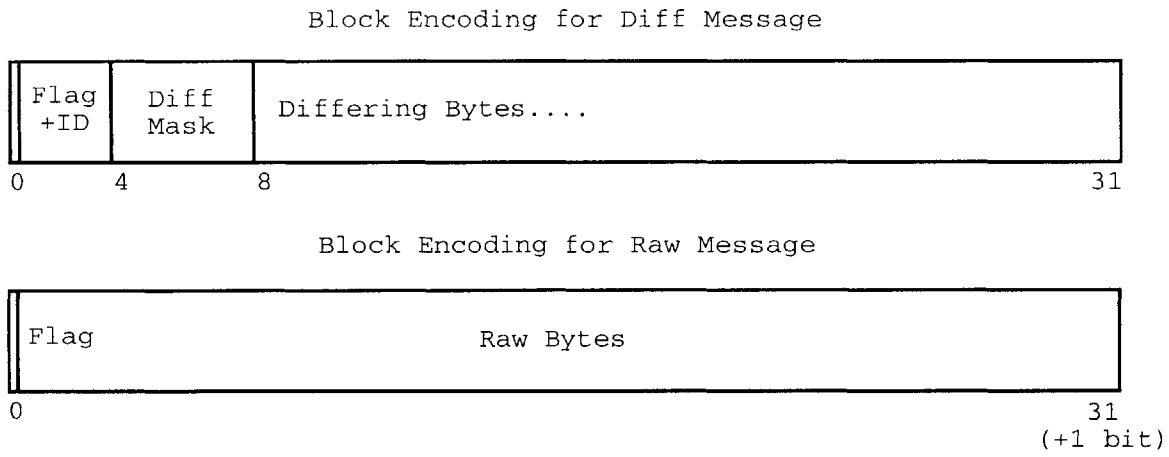

Figure 3-6: Structure of Block-Level Diffs

- The byte positions where the cached and source blocks differ.

- The differing symbols themselves.

In our implementation, we chose to have a block size of 32 bytes[1]. Figure 3-6 illustrates the block structure we chose. The first bit is a flag indicating whether the block is being transmitted in raw form or as a diff.

For raw blocks, the following 32 bytes are the raw source data.

For diff blocks, the following 31 bits are the cache ID of the template block. The next 4 bytes are a binary mask specifying the differing byte positions. And the final 24 bytes are used as needed to contain the k differing byte symbols, where k is the hamming distance. For example, if the template and source blocks have a hamming distance of 3, then the total diff block will only have $4 + 4 + 3 = 11$ bytes.

Because of this layout, the algorithm will shrink the size of a source block as long as a block in the database has a Hamming distance of 24 or fewer. This allows much more leniency than an all-or-nothing block caching scheme. If all the blocks in the database have a hamming distance greater than 24 from the source block, then this encoding scheme only incurs a penalty of one bit per block.

[1]We ran several experiments to find the best block size. See Section 3.5

3.4.3 Fast Similarity Search

The chief obstacle to implementing the proposed n-diff algorithm is that it may seem expensive to conduct the similarity search. Given that one search is required for each 32 byte sequence, a brute force search through millions of database entries would be unable to produce acceptable throughput. With the standard caching approach, the search for identical matches can be performed quickly using conventional hash functions. But in the case of carrying out a similarity search, we must use a more involved method.

Standard Hashing

It is helpful first to review why the standard approach uses hash functions. The idea is to associate a unique number (hash code) with a byte sequence. The most preferable functions are those that hash the space of distinct byte sequences to as many different values as possible (to minimize hash collisions). But since they are functions, two identical byte sequences will hash to the same value. This means that if we have a hash table indexing the database contents, it is possible to quickly find a match to a source sequence by simply searching whether the source sequence's hash code is present in the table. This search can be done in logarithmic time.

This technique breaks down for similarity searches. If the source sequence is not identical to any of the database entries, its hash code may land it near hash codes corresponding to sequences that are not at all close in hamming distance to the source sequence. The problem is that with conventional hash functions, byte sequences that are close in hamming distance do not map to hash codes that are close together.

Locality-Sensitive Hashing

The ideal hash function for the n-diff similarity search is one that maps sequences close in hamming distance to hash codes that are near each other. In the parlance of computational geometry, this is the problem of collapsing a high-dimensional Hamming cube into fewer dimensions while preserving neighborhoods. It is a difficult

problem. Consider an example: the hash code for `ab` must be close to that of `qb` but far from `ba`.

To address this problem, Indyk and Motwani [9] developed the Locality-Sensitive Hash (LSH). The core idea behind the LSH is to construct a randomized filter that attempts to create hash collisions between sequences that share corresponding elements. Since two similar sequences will have many corresponding elements, they will be more likely to hash to the same value. Each individual filter may not hash two similar sequences to colliding hash values, so an array of filters is used to reduce the uncertainty to an acceptable amount.

In applying Indyk and Motwani's method to bioinformatics sequence alignment problems, Buhler [10] provided a cogent explanation of the LSH: consider two strings s_1 and s_2 of length d over an alphabet Σ. Fix $r < d$; s_1 and s_2 *similar* if they are within Hamming distance r. Choose k indices $i_1, ..., i_k$ uniformly at random from the set $\{1, ..., d\}$. Define the hash function $f : \Sigma^d \rightarrow \Sigma^k$ by

$$f(s) = \langle s[i_1], s[i_2], ...s[i_k] \rangle$$

f is locality-sensitive because the probability that two strings hash to the same LSH value under f varies directly with their similarity. If s_1 and s_2 are similar,

$$\Pr[f(s_1) = f(s_2)] \geq (1 - r/d)^k$$

So now when we calculate the LSH of a source block, if there is a matching hash code in the cache database, then we know there is a template block that is likely to be within a small Hamming distance. By searching all candidate template blocks with matching hash codes, we can find the one that is most similar.

3.5 Evaluation

Initially, we implemented the n-diff algorithm using brute force search. In our tests, we randomly chose 1000 different 32-byte sequences from the large image database

Table 3.1: n-Diff Performance on 16 MByte Database

Trials	Block Size	% Compression	Total Time (sec)
1000	16	9.59%	8.12
1000	32	14.09%	10.60
1000	64	12.95%	12.51
1000	128	10.79%	15.18

and tried to compress them using the small database as our n-diff dictionary. The percentage of bytes reduced in the diffed representation was 15%, but the time required to compute these 32 KBytes of diffs was on the order of several days. Given that the machine we used was a 1.4 GHz Pentium IV and all tests were done exclusively in RAM, this was far too slow.

Later, we utilized locality-sensitive hashing to speed the similarity search. The percentage of compression continued to hover just below 15%, but the time required to do all one thousand searches shrunk to a few seconds. Locality-sensitive hashing demonstrated itself as a stunning tool; it accelerated the search without substantially degrading the quality of the search results.

We repeated this experiment over a range of block sizes to ascertain how much block size affected the running time and achievable byte savings. Results are shown in Table 3.1. For small block sizes, the overhead of the cache ID limits compression. And for large block sizes, the sequence becomes too expansive to find a cached sequence with small Hamming distance. The intermediate sweet spot for maximal compression of our database was a block size of 32 bytes.

Relative to our projection that standard caching schemes could achieve in the neighborhood of 15% compression (see Section 3.3.4), these results for n-diff are not groundbreaking. Keep in perspective, however, that we used the most challenging data set possible, that of intrinsically-compressed JPEGs and GIFs. On other data types, it might be more successful. In fact, preliminary tests on text sources indicated byte savings of over 35%. So although this technique may not be appropriate for the Tandem Proxies, it may be more useful for future applications in other domains.

Chapter 4

Mosaic Compression

Encouraged by the finding in chapter three that there is a fair amount of redundancy in images at the byte-level, this chapter explores whether it is possible to exploit even more redundancy at the human perceptual level. In particular, we present a method for encoding a source image in terms of perceptually similar fragments of previously transmitted images. In Section 4.1, we specify the task this method addresses. We detail our algorithm for encoding the image in Section 4.2. In Section 4.3, we present a sampling of results from our encoding experiments. And finally in Section 4.4 we compare our results with the popular JPEG and GIF algorithms.

4.1 Task Description

The major lesson we learned in the previous chapter is that by caching millions of images, the law of large numbers introduces a moderate level of redundancy at the byte level. This, in principle, should lead to an even greater amount of redundancy at the human perceptual level since two distinct byte sequences can look indistinguishable to a human observer. In an true color image, for example, can anyone identify all 65,536 different shades of blue? Entire classes of byte sequences map to perceptually identical images.

It is this key idea that we are trying to realize in the context of the Tandem Proxies. Since the encoding proxy knows all of the images the decoding proxy has

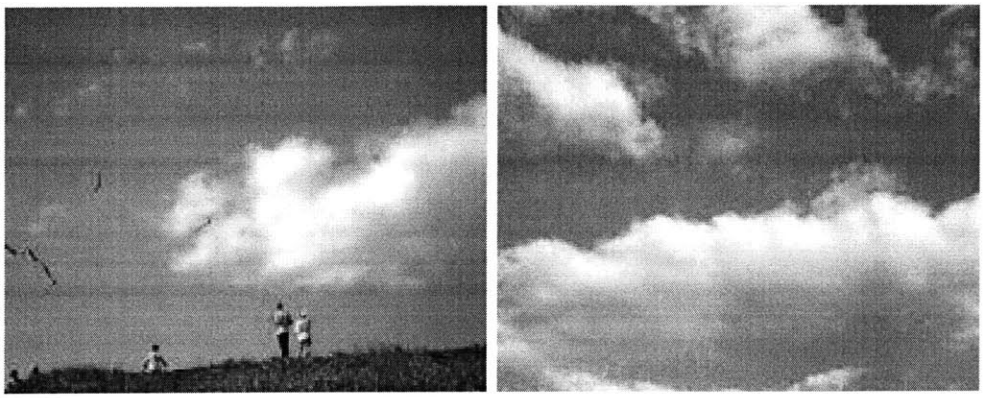

Figure 4-1: Perceptual similarities of two skies? (Photo Credit: EPA)

cached, it can be more efficient in sending a new image to the DP. The EP can compute which images in the DP's cache are most perceptually similar to the new new image. Instead of sending the new image in raw form, the EP can send instructions on how to assemble an image perceptually similar to the source image from the pictures the DP already has cached.

Let's discuss the intuitive foundation behind this approach with an example illustrated in Figure 4-1. Suppose there is a source image like the one on the left side of Figure 4-1 that contains a large region with a sky. By the law of large numbers, we are hoping that in the DP's database there is an image with a sky of similar character (perhaps a picture like the right side of Figure 4-1). If this is the case, then the EP can instruct the DP to substitute the cached sky in the appropriate regions. How will the resulting image appear to the eye? Will it look suspicious? That is the question this experiment aims to answer.

As compared to the task described in the previous chapter, this one has far less stringent requirements. The n-diff problem required that the transmitted message enable a perfect (lossless) reconstruction of the source message. But in this problem, the transmitted message should only enable the reconstruction of a perceptually similar (lossy) message; it almost surely will not be identical on the byte level.

4.2 *Mosaic* Algorithm

There are two complimentary sides to our algorithm, which we call *Mosaic*. One half of the task is how the encoding proxy should decompose the source image into instructions. The other half of the task is how the decoding proxy should use those instructions to reconstruct the image.

4.2.1 Assumptions

For both tasks, we assume that the decoding proxy maintains a cached database of all of the images previously received. In particular, the DP has fragmented each image into a number of finite size square regions, which we refer to as tiles, and associated a unique ID with each of them. Since the EP also has these tiles cached, it should know their identifiers as well. Although we do not explicitly discuss the mechanics of how this takes place, it should be a straight forward implementation.

4.2.2 Decomposing Images

To encode a source image in terms of the cached image tiles, the EP must:

- Fragment the source image into multiple fixed size square tiles Σ.
- For each source tile $s \in \Sigma$, identify which cached tile c is most perceptually similar to s.
- Transmit the ID of each cached tile c found in the previous step.

The first step can be implemented using most image manipulation libraries. In our experiments, we used Magick++ (`http://www.imagemagick.org`). The second step contains the bulk of the algorithm's complexity. Finding the most perceptually similar tile is just as much a demanding scientific problem as it is a creative art. We explore perceptual matching in detail in Section 4.2.4. The third step is trivial. Though intelligent encodings of the cache IDs are possible, in our experiments we just send them in raw form.

4.2.3 Reconstructing Images

The *Mosaic* reconstruction task is far easier computationally. Once the DP receives the reconstruction directions, it simply:

- Constructs a blank image canvas of appropriate dimension.
- For each cache ID c in the reconstruction directions, it looks up c in the database and paints it into the next empty tile in the canvas.
- Returns the synthesized image mosaic.

All of these steps can be implemented using off the shelf image manipulation libraries. In our implementation we again used the Magick++ library.

4.2.4 Perceptual Similarity Search

Of course, the major challenge is how to find the cached tile with the greatest perceptual similarity to the source tile. Not only must this be done accurately (that is, humans must find the difference between the cached tile and the source tile to be acceptable), but it must be done quickly. With a database of a million images, the chosen method must be amenable to extensive search pre-compilation.

Naive Approach: RMS

One naive method would be to calculate the root-mean-square (RMS) error between each pixel in the source block and the corresponding pixels in all of the cached blocks. We did not feel that this method was adequate either in terms of perceptual accuracy or speed. Although it accurately computes intensity differences between corresponding pixel values, it does not take advantage of the overall structure of the images, such as contours or patterns. Imagine two tiles containing the same picture, except one of them is more "zoomed" than the other. These two tiles would be very similar perceptually due to their common structure, but could have an immense RMS error. Furthermore, it is difficult to pre-compute RMS errors, so searching a database with this metric would be prohibitively expensive.

Wavelet Theory

Instead, we chose to implement our perceptual similarity search using wavelet as the basis functions. Stollnitz, DeRose, and Salesin provide an excellent primer [11] on the application of wavelet theory to computer graphics, which we only summarize here.

Wavelets are basis functions that facilitate hierarchical decomposition of signals. Any signal can be specified in terms of coefficients along a wavelet basis. What make these coefficients so appealing is that they represent instructions for hierarchically reconstructing the signal. The coefficients, in order, describe refinements to the signal from coarse to detailed. The first coefficient provides a very rough approximation of the signal (in fact, it is the overall average of the signal). Each subsequent coefficient refines the previous approximation with greater and greater detail.

Such hierarchical descriptions have been shown to accord well with the human perceptual system. Humans tend to first look at the overall structure of a picture and then examine it in closer and closer detail. At some point, the level of detail becomes so fine that we can no longer perceive it.

As a result of these properties, two images that have wavelet coefficients with close magnitudes are likely to be structurally similar to human observers. Hence, by using wavelets as our basis functions, we can satisfy our first goal of finding a technique that returns a perceptually similar match.

Fast Querying

The remaining problem is how to quickly query a large database and find the tile with the closest wavelet coefficients. Jacobs, Finkelstein, and Salesin designed a system that accomplishes precisely this kind of fast image querying [12].

In their system, they first pre-process the entire database of images. The pre-processing step is the most innovative part of their algorithm. It involves calculating the Haar wavelet coefficients for each image (for pseudocode, see [12]). Then, the n wavelet coefficients with maximum magnitude are found. For each of those n coefficients, if it is the kth wavelet coefficient, then a pointer to the image is inserted

into the kth position of a special search array. Therefore after the pre-processing has been done for the entire database, each entry i of the search array will contain a list of pointers to images in the database whose ith coefficient is of large magnitude. Note that all of this pre-processing is done offline and has no impact on the query time.

To actually conduct a query, the source image is also first decomposed into its Haar wavelet coefficients and its n maximum coefficients are found. For each of the n coefficients, if it is the kth wavelet coefficient, then all of the images listed in the kth index of the search array have their similarity score incremented. In the end, the image with the highest similarity score will be the one that had the greatest number of matching large magnitude wavelet coefficients. Since the contents of the search array were pre-computed offline, the online search, which only involved a single decomposition and n array lookups, is fast.

4.3 Results

4.3.1 Prototype

Using Salesin's fast image querying system [12] as our similarity search engine, we implemented a prototype utilizing the *Mosaic* algorithm. We built the cache database from approximately one million image tiles each of dimension 8x8 pixels. The images were drawn from the "large" database described in Section 3.3. For our test source images, we randomly chose images from the "small" database. All of our experiments ran on a 1.4 GHz Pentium IV with 384 MB of RAM.

4.3.2 Performance

Though the database took nearly five days to pre-process, online querying was extremely fast. It took less than ten seconds to determine the most similar image tile in the database for each source tile. Although this is still too slow to function as a viable proxy, we are confident that there is room to significantly optimize the querying process.

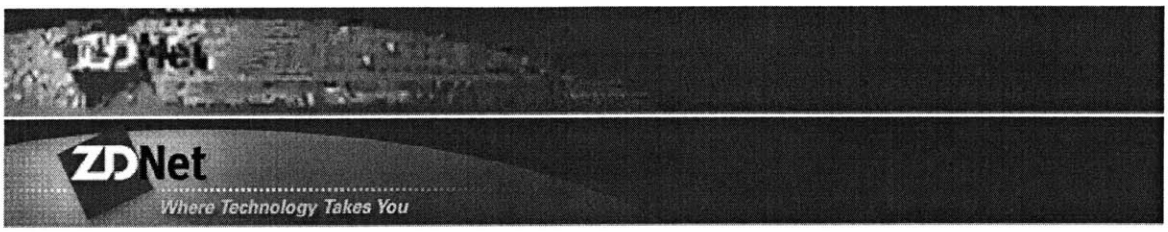

Figure 4-2: Mosaic and Original of ZDNet Logo

4.3.3 Quality

Deferring to the cliche that a picture is worth a thousand words, in Figures 4-2 and 4-3 we present two samples of images compressed using the *Mosaic* method.

Quality is in the eye of the beholder. For most people, the subject of the picture is easily recognizable, but the lack of accurate details makes the overall quality objectionable. On a tile by tile basis, the similarity metric successfully found a suitable match with the help of the law of large numbers. But even though each pixel block was similar, stark edge effects between adjacent blocks made the unified image chaotic, albeit artistic. The reader should now appreciate why the algorithm was named *Mosaic.*

4.3.4 Compression

How much compression was *Mosaic* able to achieve?

To get a sense of how much bandwidth savings this approach can claim, let's examine Figure 4-3 again. Being a 450x360 image, it was broken into approximately 2520 tiles. In our system, each tile requires three bytes of address space in order to identify up to 16 million image tiles. This gives Figure 4-3 a total encoded file size of 7.4 KBytes. Gzip encoding of the cache IDs might afford even more savings. For reference, the original file when encoded as a GIF was 109.6 KBytes and 19.8 KBytes when encoded as a medium quality JPEG. This equates to a 93.3% savings over GIF for this picture and a 62.7% savings over JPEG. So although the compression savings was significant, it came at a significant tradeoff in quality.

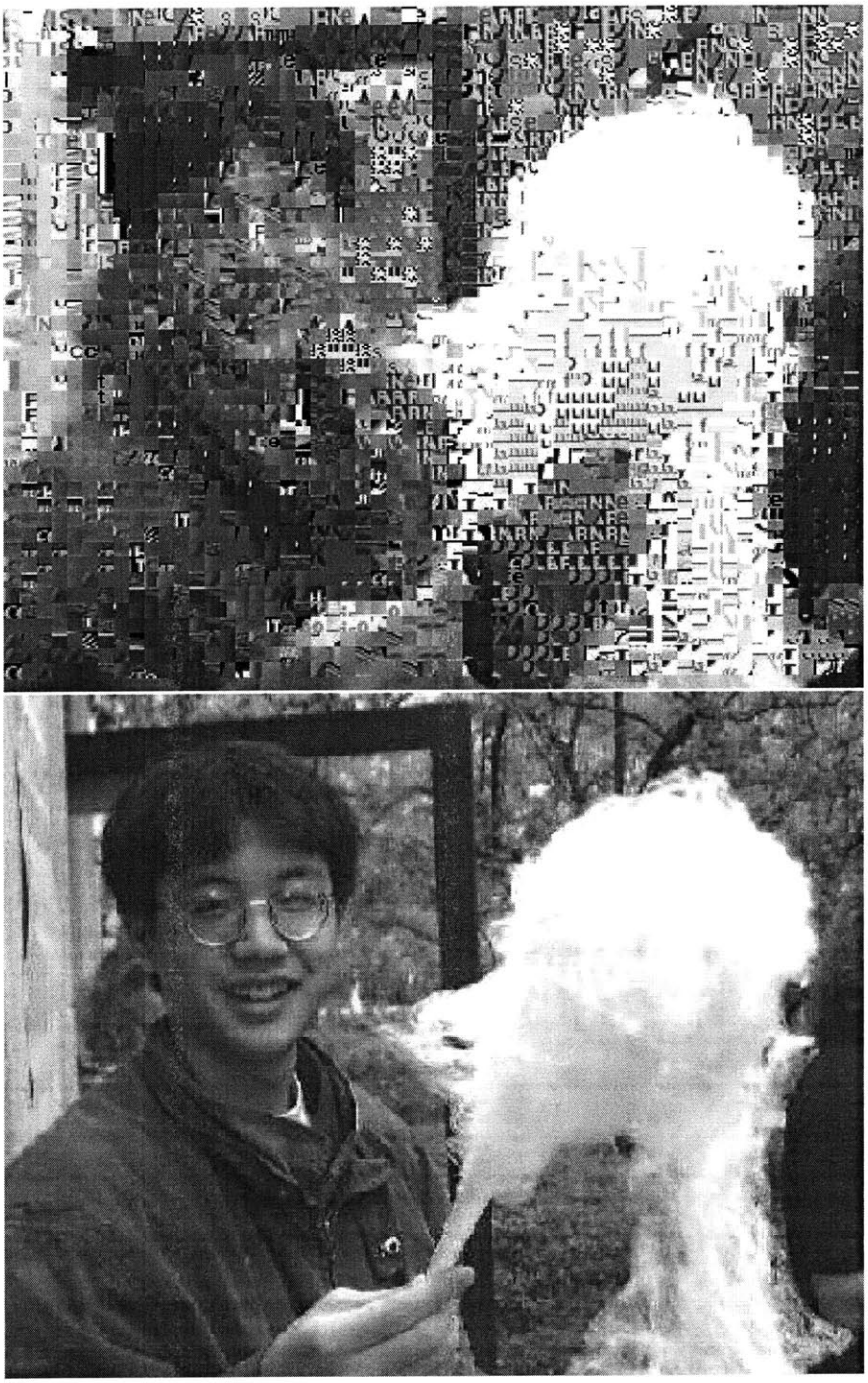

Figure 4-3: Mosaic and Original of Young Man Eating Cotton Candy

4.3.5 Informal Poll

To better understand what exactly people find most undesirable about the mosaics, we conducted an informal cognitive science study by polling twenty people. We showed each subject the four pictures shown in Figure 4-4. The top left picture is the original image, a 21st century Lena that would make Andy Warhol proud. The top right image is the *Mosaic*-encoded image, except this time using a tile dimension of 32x32 pixels to make the effect even more pronounced. The bottom right image was the same Mosaic image blurred slightly to eliminate boundary distinctions. The bottom left was a control image created by breaking the original image into 32x32 fragments, averaging the color value within each fragment[1], and then blurring the entire picture slightly.

We asked each subject two questions in a random order:

- Which of the bottom two pictures looks more similar to the original picture?
- Which of the bottom two pictures looks more different than the original picture?

The results were revealing. Everyone polled gave the same responses. All twenty people thought that the blurred mosaic looked more like the original picture. Yet all twenty people also thought that the blurred mosaic looked more different than the original picture. How can this be? The reason most people cited was that the mosaic image has more detail and higher order structure. Some of the details match the original image. But many other details are not common with the original, and therefore distract from it.

4.3.6 Lessons

Although the results of the *Mosaic* compression experiments were not stunning, they suggest a number of lessons that future incarnations should consider:

- Images tend to be far more correlated to themselves than to each other. Therefore, the compression scheme should not independently encode each tile. Rather,

[1] Using this method, the control image and the mosaic require the same number of bits to describe.

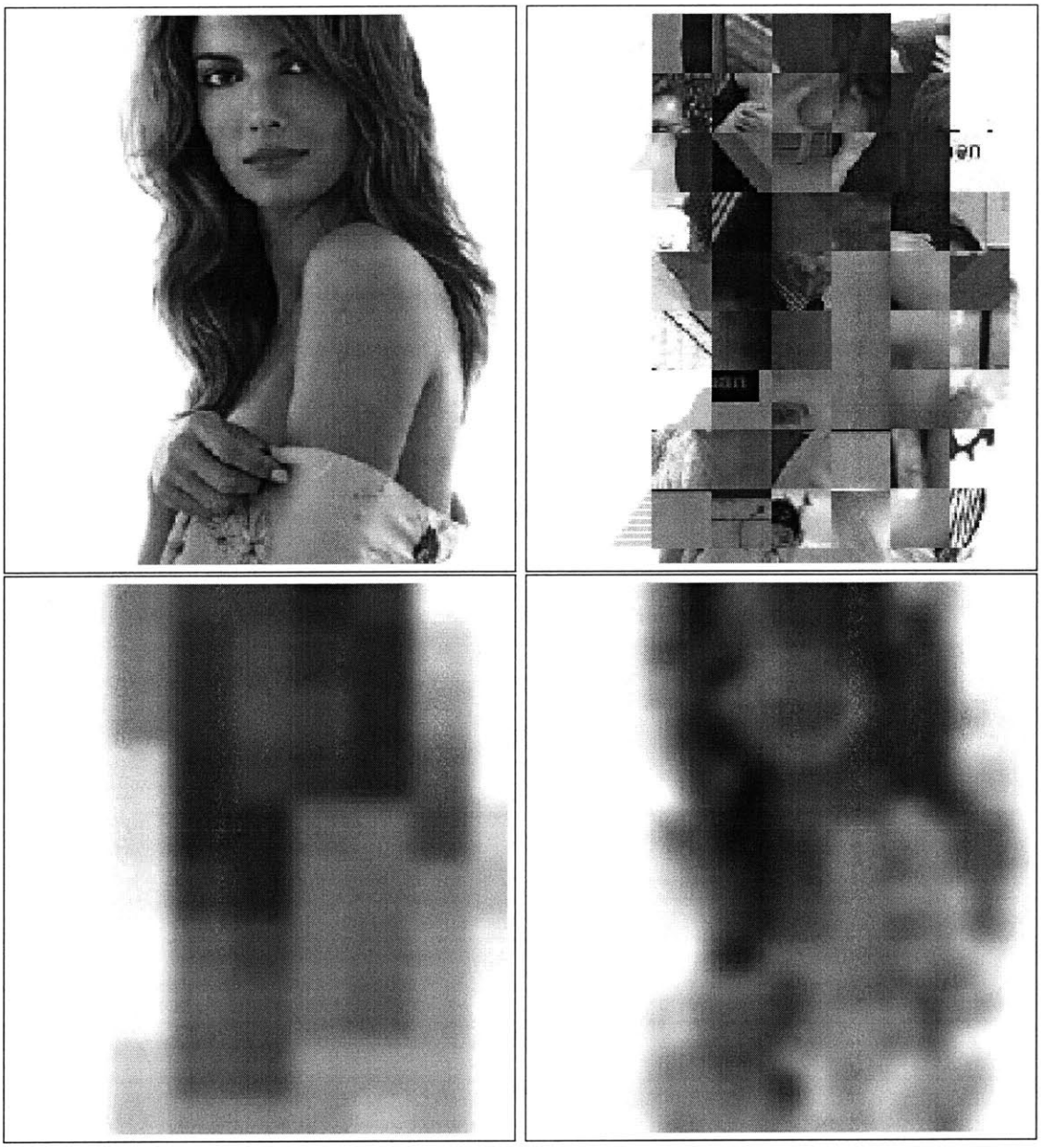

Figure 4-4: Pictures from Informal Poll on Mosaic Quality

it should leverage each tile's similarity to its neighbors.

- Edge effects significantly degrade perceptual quality. The vast edge smoothing literature should be scoured for potential solutions. One solution may be to fragment the image into non-fixed size blocks so that homogenous regions are not tiled.
- Although the cached tiles contain some details that make them similar to the source tiles, other details distract. Perhaps the most similar cached tile could be used merely as a template, and the algorithm could perform an intelligent diff based on this template.
- Wavelets are a good starting point for perceptual similarity metrics. But do better ones exist?

4.4 Popular Image Formats

In thinking about how to improve *Mosaic*, it is helpful to briefly review the basics of how popular image compression formats work. This is intended to be an overview. For a more thorough treatment, refer to Khalid Sayood's excellent textbook on data compression [13].

4.4.1 GIF

GIF (Graphics Interchange Format) is the most popular lossless image format used on the web. Portable Network Graphics (PNG) are gaining popularity, but GIFs remain the Goliath of web graphics.

GIFs are essentially an application of the LZW string matching algorithm to the image domain. LZW is a clever method of adaptively building a dictionary to encode a source with unknown characteristics. It has been proven to be asymptotically optimal in exploiting string redundancies. In the case of GIFs, the encoder runs LZW on the image's pixel values and computes the dictionary. The compressed image contains only indexes to the dictionary, which is sufficient for reconstructing the original image.

LZW was patented by Unisys and, unfortunately, Unisys demands royalties on any implementation of LZW. Hence, GIF falls under this umbrella and is not a "free" format to use.

4.4.2 JPEG

JPEG, the Joint Photographic Experts Group, is an international consortium that develops image standards. There are many different JPEG formats, but the one we touch on here is the most popular lossy image compression format in existence.

JPEG works by first fragmenting an image into 8x8 pixel blocks. For each block, the discrete cosine transform (DCT) is computed and the DCT coefficients are recorded. Recognizing that the human perceptual system is more sensitive to noise in certain DCT sub-bands than others, the DCT coefficients are quantized in a manner that allows less quantization noise on perceptually important sub-bands and more quantization noise on perceptually insignificant sub-bands. In the final step, the quantized coefficients are encoded with a two-level Huffman code. The first level exploits inter-block correlations and the second level takes advantage of intra-block patterns.

JPEG is an extremely polished and optimized standard. It can code grayscale images down to a bit rate of 0.5 bits per pixel with good fidelity. The new JPEG 2000 standard, which is based on wavelets instead of the DCT, intends to improve quality at bit rates below 0.5 bits per pixel.

JPEG's nuanced appreciation for the characteristics of the human visual system and cunning use of intra-image correlations make it an excellent model for future versions of *Mosaic*.

Chapter 5

Conclusion

This chapter summarizes the major findings of this thesis. First, Section 5.1 reviews the relative merits of each proposed technique. Section 5.2 presents a road map of interesting future research directions based on this evaluation. Section 5.3 concludes.

5.1 Summary of Techniques

Excessive multiplexing of international telecommunication lines makes internet access extremely slow in developing countries. To accelerate web content delivery, we proposed the Tandem Proxy architecture. In the Tandem Proxy system, two proxies straddle a poor quality bottleneck link and use a variety of cooperative techniques to minimize bandwidth consumption across the link. Specifically, the methods the partner proxies could leverage are aggressive caching, byte-level diffs, and differential compression. We explored each of these candidate techniques in detail.

5.1.1 Aggressive Caching

The HTTP standard is overly conservative in its caching rules out of paranoia that a client might receive stale content. This is understandable given that the HTTP model is meant to protect the integrity of critical operations like financial transactions within a traditional client and server architecture.

But the tandem proxy architecture allows more flexibility. The client's local proxy can aggressively cache every single file it receives. When there is uncertainty whether a file is fresh, the local proxy can query the remote proxy for cache validation.

This scheme can realize enormous bandwidth reduction. There are no foreseeable drawbacks to implementing this flavor of aggressive caching. Any future Tandem Proxy system would we unwise to overlook this feature.

5.1.2 Diffs

When a cached file is stale and needs to be updated, it is natural to consider transmitting only a diff between the cached file and the updated file. Text-based diffs should be an effective technique, but they were not explicitly considered in this thesis because text does not contribute to the majority of internet traffic.

Instead, we explored the notion of byte-level n-diffs in the context of images. When a new file needs to be transmitted, it is expressed in terms of byte blocks the client already has cached. Instead of requiring that the cached blocks be identical to the source blocks, our algorithm does a second-level recursive diff to exploit blocks that are merely close in Hamming distance. To speed the search for cached blocks with close Hamming distances, we used locality-sensitive hashing.

In our tests, n-diff was not able to achieve more bandwidth savings than a traditional block-based caching scheme. Most likely this was because we were using compressed images as our data set. If two blocks weren't exactly identical, they weren't likely to be similar either. Whether this method would be more successful on other data types is an open question.

5.1.3 Compression

Since there was not as much redundancy at the byte-level as we'd hoped, we moved our investigation to the perceptual level. We presented a technique called *Mosaic* compression that attempted to transmit instructions to the client on how to construct a perceptually similar image based on images that client already had cached.

To carry out this method, the encoder fragments the source image into a number of square tiles. For each tile, the encoder attempts to find the cached tile with the closest perceptual match. Search accuracy and speed is achieved through wavelet-space analysis and extensive search pre-compilation.

Our experiments revealed that images compressed using the *Mosaic* method were perhaps more artistically captivating than technically acceptable. Though each individual tile was similar in character to its corresponding source tile, the aggregate image did not perceptually appear to be unified. If *Mosaic* were to take better advantage of intra-image correlation and find a method of reducing edge effects, it has the potential of being a powerful technique since it dramatically shrinks file sizes.

5.2 Future Research

The Tandem Proxy protocol represents a viable infrastructure for implementing techniques to reduce bandwidth consumption. The space of possible techniques breaks down into three categories:

- Those that make caching more effective.
- Those that make updating a cached file more efficient.
- Those that accelerate the delivery of new content.

5.2.1 Caching

With the method of aggressive caching available, the first category of technique is largely a solved problem. Aggressive caching stores every bit of data sent across the network, so there is no method that could cache more data. One interesting line of research, however, might be on how to predict which pages a user will view and then pre-fetch those pages in advance.

A few security-related questions also remain. One of the reasons HTTP does not allow such aggressive caching is that it opens a possible security risk if a proxy

caches data the server considers confidential. Finding safe procedures for aggressively caching sensitive data would benefit users.

5.2.2 Cache Updates

In the case where a cached item needs to be updated, we can separate our discussion by data type.

Updating a cached text file is most commonly done with diff algorithms to synchronize two versions of the same file. In this thesis, we introduced n-diff to update a text file relative to an entire corpus of cached text. This is an exciting concept since the diff is not restricted to a single file; it can draw from several files in the corpus as appropriate. n-diff represents an early exploration of what is bound to be a theoretically rich research domain. We welcome other researchers to consider this problem.

Cached images are unlikely to undergo slight changes. When an image is modified on a webpage, most often the image is completely replaced. Even trying to build the source image in terms of previously cached image blocks is not likely to be a successful effort. Recall that in the previous chapter, we demonstrated how images are far more likely to be correlated with themselves than with any other image.

5.2.3 New Content

Since images compose the bulk of web content, and they are unlikely to benefit from diff methods, the open problem of how to speed web access to the developing world ultimately lies in better image compression algorithms optimized for bandwidth impoverished connections.

This is not to say that *Mosaic*-style differential compression on images can't be effective. On the contrary, we simply caution against encodings that exclusively look for local similarity between source and cached tiles. As explained in Sections 4.3.6 and 5.1.3, a *Mosaic* algorithm that took a more holistic approach and leveraged intra-image correlations could be wildly successful. The cached tiles should be used more

as templates for other image manipulation operations.

For example, one interesting scheme would be to find the most perceptually similar cached tile and then transmit a diff in the wavelet basis between the source tile and that tile. Then, when encoding the diffed wavelet coefficients, the algorithm could use a JPEG-style two-level Huffman code. An algorithm of this flavor would therefore exploit both inter-image and intra-image correlations.

Image compression research is an active area of pursuit and we hope that researchers in that field will invest energy in this problem.

5.3 Impact

Next week, I depart for Ghana to teach for the same program that took me to Kenya one year ago. I will be taking a prototype of the Tandem Proxy system along with me. But I hope this is just the beginning.

Many of us in the United States have become desensitized to how empowering a tool the internet can be. We see the web as a way of checking our stock portfolio or our bid status on Ebay. But for billions of people across the world, where there are no well-funded libraries nearby, the internet has the potential to transform their lives. Teachers can point curious students to university course material. Doctors can reference extensive medical case studies. And farmers can learn new agricultural methods.

I therefore call on systems researchers to have the courage to tackle this tough problem and make this field an active area of pursuit. Even modest improvements of ten percent can enable people in the developing world to access information that was previously beyond reach. Enough, perhaps, to make an HIV webpage readable and a young Kenyan hospitalized with Hepatitis C a little less scared.

Bibliography

[1] Libby Levison, William Thies, and Saman Amarasinghe. Searching the world wide web in low-connectivity communities. In *Proceedings of The 11th International World Wide Web Conference*, 2002.

[2] R. Fielding, J. Gettys, J. Mogul, H. Frystyk, and T. Berners-Lee. RFC 2068: Hypertext Transfer Protocol — HTTP/1.1, January 1997. Status: PROPOSED STANDARD.

[3] Andrew Tridgell, Peter Barker, and Paul Mackerras. Rsync in http. In *Proceedings of the 1999 Conference of Australian Linux Users*, 1999.

[4] Andrew Tridgell. *Efficient Algorithms for Sorting and Synchronization.* PhD thesis, Australian National University, 1999.

[5] Neil T. Spring and David Wetherall. A protocol-independent technique for eliminating redundant network traffic. In *Proceedings of ACM SIGCOMM*, August 2000.

[6] Athicha Muthitacharoen, Benjie Chen, and David Mazières. A Low-Bandwidth network file system. In Greg Ganger, editor, *Proceedings of the 18th ACM Symposium on Operating Systems Principles (SOSP-01)*, volume 35, 5 of *ACM SIGOPS Operating Systems Review*, pages 174–187, New York, October 21–24 2001. ACM Press.

[7] Jeffrey C. Mogul, Fred Douglis, Anja Feldmann, and Balachander Krishnamurthy. Potential benefits of delta encoding and data compression for HTTP.

Technical Report 97/4, Digital Western Research Laboratory, 250 University avenue, Palo Alto, California 94301, USA, July 1997.

[8] Walter F. Tichy. The string-to-string correction problem with block moves. *ACM Transactions on Computer Systems*, 2(4):309–321, November 1984.

[9] Piotr Indyk, Rajeev Motwani, Prabhakar Raghavan, and Santosh Vempala. Locality-preserving hashing in multidimensional spaces. In *Proceedings of the 29th Annual ACM Symposium on Theory of Computing*, pages 618–625, 1997.

[10] Buhler. Efficient large-scale sequence comparison by locality-sensitive hashing. *BIOINF: Bioinformatics*, 17, 2001.

[11] Eric J. Stollnitz, Tony D. DeRose, and David H. Salesin. Wavelets for computer graphics: A primer, part 1. *IEEE Computer Graphics and Applications*, 15(3):76–84, 1995.

[12] Charles E. Jacobs, Adam Finkelstein, and David H. Salesin. Fast multiresolution image querying. *Computer Graphics*, 29(Annual Conference Series):277–286, 1995.

[13] Khalid Sayood. *Introduction to Data Compression.* Morgan Kaufmann, San Francisco, California, second edition, 2000.