Text / Graphics and Image Transmission over Bandlimited Lossy Links

by

Jeffrey Michael Gilbert

B.A. (Harvard University) 1993
M.Phil. (Cambridge University) 1995

A thesis submitted in partial satisfaction of the

requirements for the degree of

Doctor of Philosophy
in

Engineering Electrical Engineering
and Computer Sciences

in the GRADUATE DIVISION

of the

UNIVERSITY OF CALIFORNIA, BERKELEY

Committee in charge:

Professor Robert W. Brodersen, Chair
Professor Eric A. Brewer
Professor Paul K. Wright

Spring 2000

Text / Graphics and Image Transmission over Bandlimited Lossy Links

by

Jeffrey Michael Gilbert

Abstract

Text / Graphics and Image Transmission over Lossy Bandlimited Links

by Jeffrey Michael Gilbert

Doctor of Philosophy in Electrical Engineering
University of California at Berkeley

Professor Robert W. Brodersen, Chair

This thesis describes the application of image compression and networking techniques to the transmission of text / graphics and image data over bandlimitted and lossy links. While much research has focused on image and data compression, this thesis proposes that compression alone is not sufficient, and that transformations into progressive formats and explicit link scheduling can significantly improve performance over bandlimited and lossy links. Analyses and solutions are proposed for both application-independent and application-specific scenarios. Techniques including bitmap and drawing primitive-based approaches, as well as a novel hybrid scheme, are presented. Image compression techniques optimized for text / graphics bitmaps are presented. The application-independent techniques are then applied to the acceleration of the delivery of World Wide Web pages over modem and wireless links. Application-specific techniques are illustrated using the example of a web-based VLSI layout viewer. Various design points trading off bandwidth utilization, error tolerance, and client complexity and power consumption are presented. Architectures, algorithms, as well as prototyping techniques and development frameworks are presented for many of the approaches. Lastly, unifying themes and requirements are synthesized and their implications to network protocol design are discussed.

Robert W. Brodersen, Chairman of Committee

Table of Contents

List of Figures

List of Tables

Acknowledgments

Firstly, I would like to express my sincerest thanks to my thesis advisor, Prof. Bob Brodersen, for the direction, freedom, advice, and questioning that has helped to make the past six years a remarkable learning and growing experience. It is safe to say that I have learned many things from him that I could not have gleaned in a classroom or from a textbook. I would also like to thank him for often having more confidence in my abilities than I myself had. (And this is no easy task...) And finally thanks to him for objectively, and without compromise, wearing the many hats that were required in the final months of my education.

Thanks to Prof. Jan Rabaey for being my secondary, non-thesis, pseudo-advisor and serving as a role model for me to learn the art of teaching from. Being a Graduate Student Instructor (GSI) for EE141 was definitely one of the most challenging and demanding experiences of my graduate years, but receiving the Outstanding GSI award was one of my proudest achievements, and I don't think that it would have been possible without Jan. Most importantly I'd like to thank Jan (and Marlene, the Jansgroup social coordinator - see below for more information), for letting me be an honorary "Jan's Group" member for his group's recreational retreats! Some day I may take him up

the challenge to telemark as he snowboards, though it may be at a time that we can reminisce about the good old days of 0.001µm technology.

Thanks to Dr. Alice Chiang for her role as a mentor and for her support over the past 9 years at MIT Lincoln Laboratory and the Teratech Corporation. Her combination of creativity and technical excellence, and confidence in me, has in no small way helped my intellectual growth and curiosity. I have full confidence in the future of the Teratech corporation.

I would like to thank the other members of my thesis committee: Prof. Eric Brewer and Prof. Paul Wright. I'd like to thank Eric for the insight into my research he has shared with me. In particular I'd like to thank him for not letting me settle on the compressed framebuffer approach of Chapter 6 and to instead seek a more challenging problem, resulting in the hybrid approach of Chapter 7. I must thank Paul for the support and encouragement over the past years through InfoPad, quals, and the Management of Technology (MOT) program. I would also like to thank Prof. Randy Katz for chairing my qualifying examination - I just wanted him to have a practice run for his role as department chair!

Thanks to Prof. Nelson Morgan for his support of some of my earlier investigations into speech recognition as well as during my GSI of EE225d. It was a great experience from which I learned a lot. I would like to thank Prof. Anthony Joseph, Prof. Avideh Zakhor and Vivek Goyal for their helpful comments and suggestions on two papers I wrote based on some of my thesis research.

On the administrative side, I'd like to thank Ruth Gjerde, Heather Brown, Sheila Humphreys, and Tom Boot for keeping the EECS system running smoothly and making great strides to improve the quality of life for students. I have not met someone as willing to drop everything for a student at any time as Tom.

Funding for this research was supplied in part by DARPA and BWRC members Cadence, Ericsson, Hewlett Packard, Texas Instruments, ST Microelectronics, Lucent, and Intel. I was the recipient of an NSF Graduate fellowship for Fall 1994 through Spring 1997. And I must thank Herschel Smith again for funding my year at Cambridge just prior to Berkeley. That year has made a change in my life that will be with me for many, many years to come.

I am thankful to have had the privilege of interacting with some truly interesting, motivated, and bright students over my years at Berkeley. It would be impossible to name them all without having to publish this thesis in at least two volumes, but some of the BWRC folk that I will not soon forget follow. The dinner Analog RF crew: Dennis Yee, Chinh Doan, Brian Limketkai, and Sayf Alalusi. Thanks Chinh for putting up with my constant quest for sushi - I will think of you whenever I'm "not" having Mexican food! Thanks to Brian for making sure that someone out-ate me! Thanks to Sayf for keep the gym momentum going just when we started slacking. I'd really like to thank Dennis for the informal education in wireless communication at the RSF "class-room". Pumping iron while learning about out-of-band blockers and the benefits of DC notches in wide-band CDMA systems - it just doesn't get any better! Thanks for humoring my constant questioning and for some great advice over the years.

Thanks to the lunch / weekend crew: Marlene Wan, Josie Ammer, and Varghese "George" George (or was it George Varghese George? - I never could get it straight...) Thanks George for being a great friend and roommate. I can think of few people I would have rather lived with for the past 4 years. Thanks to Marlene for being a constant source of entertainment - even if on a rare occasion it was at her expense... :) I must thank Marlene and George for their undying encouragement over the past years as I decided what I wanted to do when I grew up. Thanks to Josie for having seen the error of her ways at MIT and becoming a bear! (Ok, I guess Misha had a small part in

this...) Thanks for teaching me wall climbing and the beginnings of hockey skating and just being a “super nice” person.

Some other BWRC folk that have made my time here much more enjoyable are David Sobel, Danny Patel, Johan Vanderhagen, Andy Klein, Vandana Prabhu, Chris Taylor, Trevor Pering, Ian O’Donnell, Tom Burd, and Chris Rudell. I must thank Dave for making sure that there was always someone more “Harvard” than me around. :) You’d better return to complete your Ph.D.!

Preparation for my qualifying exam was perhaps one of the most educational aspects of the past 6 years, and would not have been one tenth as meaningful if not for the active role that so many of the elder grads played in it. In particular David Lidsky, Lisa Guerra, Ole Benz, and Karl Petty jump to mind as having provided much time, guidance, and support.

Many thanks to Trina Chang for being a terrific friend and perhaps the most understanding person I will ever meet. Her support over the past year has been unwavering and I am deeply indebted to her for that. She will certainly make a fantastic M.D. (or travel agent / sushi chef if she chooses to go with her true passions!)

Thanks to (Prof.) Harry Gakidis for providing me with a constant source of business-venture diversions. When this thesis is valuable some day solely for mentioning his name in the acks because he becomes a multi-billionaire, he will have the last laugh! Seriously, prof, don’t give up the dream. Thanks for dropping everything at any time to give me advice or just cheer me up and prevent me from “digging my own grave” on many, many occasions.

Thanks to my brother Len and sister-in-law Tam for advice on many topics from work to life. And thanks to mom and dad, without whom this would not be possible :) I finally made it!!!

PART I *Introduction*

CHAPTER 1 *Introduction*

The explosive growth of the Internet as well as the increasing proliferation of wireless and broadband communication have caused significant shifts in the way people work, play, and communicate. Information access is the application of the new millennium. The days of computer screens equated with chunky green letters on a black 8" screen are over; vibrant images, full-motion video, and intuitive graphical user interfaces are a must for most applications. But supporting these rich multimedia displays remotely requires judicious selection of which information to send as well as how to send it. *Compression is not enough.*

However, interactively accessing this multimedia information, by its very nature, requires transmission of the text, graphics, images, and videos in real-time from a remote server to users' machines. This transmission is often over modem or wireless links of limited bandwidth and reliability. Transmission over the Internet introduces additional bandwidth constrictions and opportunities for data loss. Thus it is crucial to efficiently code and schedule the transmission of the multimedia data over the link. *Compression is not enough.*

Additionally, the push towards smaller, lighter, yet more powerful portable devices for everything from web browsing to stock portfolio management and teleconferencing is necessitating some fundamental paradigm shifts. Conventionally, applications used on these portable devices have to run locally on the devices. This constrains the size of the devices based on the computational and storage requirements of the applications. However, the InfoPad project [14] has shown that this constraint is not necessary if the applications are not run locally, but instead run remotely on a well-connected compute-server with client-server communication achieved via wireless links. This then shifts the burden to the design of networking and image compression protocols and algorithms to interactively send the multimedia data from the server to the client. Interactivity requires low latency, which in turn requires careful selection of the type of graphical updates to send and when to send them. *Compression is not enough.*

While Internet access used to be confined to the world of academia, today, thanks to the World Wide Web, it has become a significant consumer reality, and has tightly woven its way into almost every aspect of life. HTML, HTTP, and TCP/IP provide a flexible method of delivering multimedia content. However, many users connect to the Internet via slow modem links using Internet Service Providers, and an increasing number are connecting via lossy wireless links. Unfortunately, the original web protocols were designed for well-connected workstations and are not particularly network friendly. This leaves home and untethered surfers with a suboptimal setup. Yet, as shown in this thesis, if these protocols are designed to optimize interactive remote operation, through a combination of compression and networking techniques, the situation for surfers is greatly improved. *Compression is not enough.*

Most previous research attempts to improve upon text / graphics and image transmission through either lossy or lossless compression techniques. As this thesis will show, in most cases, simply compressing the information transmitted is not sufficient to obtain interactive operation

over bandlimited lossy links - link scheduling and image transformation are required to reduce latencies and improve end-user experience. *Compression is not enough.*

1.1. Thesis Overview

This thesis examines text / graphics and image transmission techniques for scenarios ranging from operating generic applications over wireless links to surfing the web over modems to application-specific methods of improving interactivity over bandlimited and lossy links. The thesis examines each of the scenarios, presents an analysis of the challenges and difficulties, and proposes and quantifies solutions combining image compression and networking techniques. Finally, the commonalities of the text / graphics and image transmission tasks are discussed.

For many of the areas, various design points trading off bandwidth utilization, error tolerance, and client complexity and power consumption are presented. Architectures, algorithms, as well as prototyping techniques and development frameworks are also presented.

This work found its origin in the InfoPad project [14] as an application-independent means to deliver the text / graphics and video data to a remote wireless black and white terminal, as described in Chapter 4. Its scope has grown to include more sophisticated, larger color image-enabled remote terminals, as well as other areas, such as the acceleration of web transmission over bandlimited lossy links and application-specific acceleration methods.

1.2. Common Themes

There are several themes that pervade the many variations of text / graphics and image delivery discussed in this thesis:

- Optimization from end-user perspective
- Global ordering / reordering
- Local progressive image transformations

1.2.1. Optimization from End-User Perspective

Interactive transmission of text / graphics and image data is a user-based activity, i.e. the text, graphics, and images are transmitted because the user wants to see them. It is critical to keep the goal of the user in mind when analyzing and designing transmission systems. Too often total transfer times are reduced through compression techniques alone while a far better user-experience can be delivered by determining which limited information is of use to the end-user right away, and which information is either not needed or is not needed initially.

Thus the key is to determine which information is critical, and this often requires determining user-intentions. Often only a small part of a large object is required since the amount of information that a person can scrutinize at any given time is limited, despite the fact that the amount of information that can be scanned is large.

1.2.2. Global Ordering / Reordering

The order of the data delivery can significantly effect the end-user experience. Often global reordering techniques can quickly deliver the most time-critical information at the expense of delaying non-critical information which may only affect final viewing.

1.2.3. Local Progressive Image Transformations

While reordering the delivery of the various text / graphics objects or images can yield significant improvements, often finer granularity manipulation yields further improvements. Thus parts of the objects need to be reordered or interleaved. However, it is required that the “more important” parts are sent first, followed by the “less important” parts. Progressive codings can be used to separate the more important from the less important parts. Conventional progressive cod-

ings are used on images to separate the components by spatial resolution or frequency. Application-specific progressive coding can separate global properties from fine details.

1.3. Existing Techniques

There is almost always a trade-off between reduced time-to-market and optimized performance. Reduced time-to-market favors modularity and optimized performance favors cross-boundary application-specific optimization. However, it is arguable that current solutions favor the former at the expense of the latter. One of the goals of this dissertation is to derive a new level of modularity which could be reused to exploit commonalities across multimedia transmission applications while obtaining the high performance required by these applications.

Current solutions to remote text / graphics and image transmission typically consist of determining all of the data that needs to be presented, and sending it through a reliable mechanism such as TCP/IP. When the user generates more updates via interaction, the results of these are queued up, never to be discarded until successfully received by the remote terminal. This results in a sub-optimal solution for several reasons.

1.3.1. Problems of Stale Data

By using a reliable mechanism such as TCP/IP for transport and passing all data to be displayed through it, data that is not useful will still be transmitted, and thus consume valuable bandwidth. While in a bulk file transfer, all data is useful, with text / graphics and image transmission, often data becomes stale if new data to display in the same region is generated.

1.3.2. Problems of Not Using User Intent to Govern Ordering

Additionally, not all text / graphics and image data is created equally. Transmission of text / graphics and image data is typically in an interactive setting where the user has particular goals

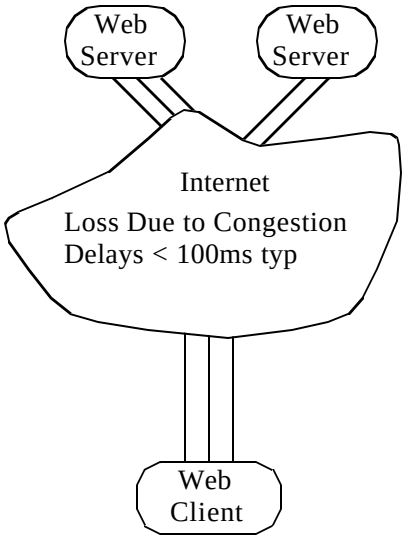


FIGURE 1.1. **Browsing in a well-connected TCP-friendly environment**

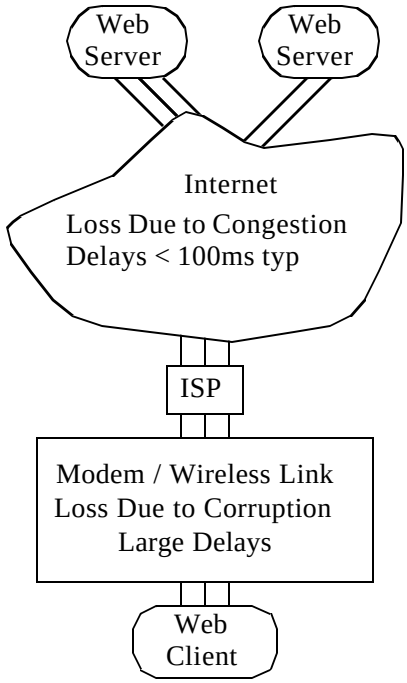


FIGURE 1.2. **Browsing over a TCP-averse modem or wireless link**

and intentions. Using these intentions requires intelligent data reordering and often recoding. Simply queuing all data in the order that it was generated by applications can cause undue delays.

1.3.3. Problems of TCP/IP over Wireless

The Internet Transmission Control Protocol (TCP/IP) is quite effective at sustaining multiple connections across the heterogeneous and time-varying Internet, as well as across local area networks. It is adaptive and scalable due to its congestion control mechanisms and end-to-end implementation. It is designed for networks where loss is primarily due to congestion and centralized management is not possible or practical. While this describes the Internet and many LANs, it does not aptly describe the situation presented by modem or wireless links at the last hop. Figure 1.1 and Figure1.2 depict the difference.

This difference in topology has significant consequences in terms of TCP end-to-end performance while web browsing or performing other operations using standard TCP/IP based connections. In the well-connected case, TCP/IP works as designed to allow hosts to establish links that are as high capacity as is fair in some global sense. Connections quickly "learn" what their fair share of bandwidth is. However, in the case of browsing over a modem or wireless link, many problems occur due to the interaction of the last hop with TCP's congestion and flow control mechanisms.

While research has addressed some of the issues involved with using TCP/IP over wireless links [8], substantial further benefits can be achieved in high-loss environments through optimization at the application-level. For instances of text / graphics and video transmission, eliminating false-dependencies in the data streams can significantly reduce latencies.

1.4. Thesis Organization

The rest of this thesis is organized as follows: Chapter 2 introduces application-independent text / graphics and image transmission, presenting uses and previous work. Chapter 3 describes conventional primitive-based approaches which communicate graphical information using drawing primitives. Chapter 4 then contrasts this with bitmap-based approaches which transmit the screen updates using rendered bitmaps. Chapter 5 extends the bitmap-based approach from monochrome implementations to color implementations including full-motion video support, considering bandwidth and reliability limitations. Chapter 6 then seeks to further reduce client power and cost through a compressed framebuffer approach. Chapter 7 proposes a final approach to application-independent text / graphics and image transmission which is a hybrid containing the best of the primitive-based and bitmap-based approaches. Chapter 8 details the image compression requirements for images of text / graphics and presents a novel compression technique designed

for that class of images. Chapter 9 gives a view of the development and analysis environment used for the research described in the previous chapters. Chapter 10 begins the part of this thesis dedicated to application-specific transmission by discussing optimization of web protocols for band-limited links. Application-level link management is then discussed in Chapter 11 in the context of an interactive Java-based VLSI layout viewer. Chapter 12 gives a view of the development environment used to support the application-specific transmission research. Finally Chapter 13 concludes the thesis by distilling the networking requirements of text / graphics and image transmission, summarizing the findings of the thesis research, and presenting future directions.

PART II *Application-Independent Transmission*

CHAPTER 2 *Overview*

2.1. Overview

Application independent text / graphics and image transmission is used for remote rendering in a generic manner that is not tailored to a particular application. Text, graphics, and images are specified in the most general terms such as *"Draw the string 'Hello' at (100,230) in Helvetica 11pt. font."* Applications describe the graphics they desire to present in this generic manner, leaving it to a centralized text / graphics system to effect the rendering. One advantage of this approach is that its flexibility allows almost any application to be supported. Additionally, it is highly modular in that applications need not know the details of how rendering occurs, or whether the display is local or remote. Thus improvements to the rendering system will improve the performance of all applications. However, due to this flexibility and modularity, application-independent transmission is the most difficult to implement efficiently.

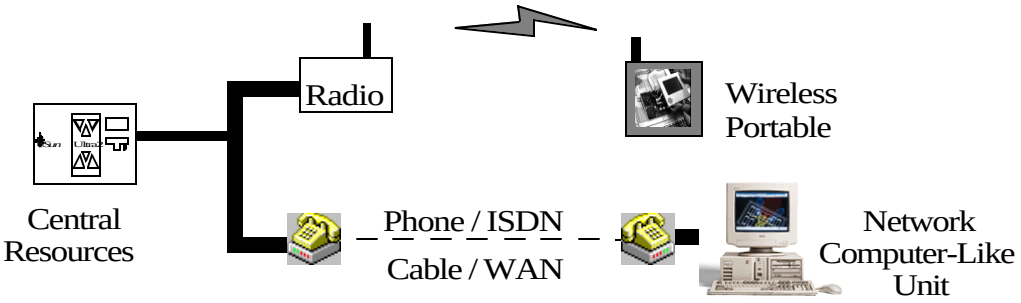


FIGURE 2.1. **Remote computation model**

Two principal uses of application independent text / graphics and image transmission are *Remote Interactive Computation* and *Multimedia Collaboration*.

2.1.1. Remote Interactive Computation

Remote interactive computation refers to the technique whereby an interactive application runs on a remote server while its display information is sent to a local client and keyboard or pen input is sent back from the client to the server as shown in Figure 2.1. Thus it extends the conventional client-server model of only sharing applications and data one step further. To the end-user, if the display data can be delivered rapidly enough, it appears as if the applications are running locally. However, remote operation has several advantages:

1. Computational economies scale with bursty usage
2. Lightweight / inexpensive clients
3. Ubiquity of access
4. Facilitates portable operation (ala InfoPad)
5. Protects sensitive equipment and storage
6. Centralization of administration
7. Allows “leasing / renting” of computation

Remote computation allows computational support of many users to be centralized in a single server or set of servers. In this way, the capabilities required to execute compute-intensive application need not be replicated at each client. If the peak demands of the clients are high, yet usage of resources is bursty and independent, centralization allows the resources to scale with the

average client demands, and not the peak client demands. Similar advantages of scale have been obtained in UC Berkeley's Network of Workstations (NOW) Project [2,21].

Remote computation allows complex tasks to be accomplished using only low-performance, low-cost clients. This can result in a reduction in total system cost and size. Additionally, this facilitates ubiquity of access since only simple clients need to be replicated. Client simplicity, in turn, enables portable client operation.

By reducing the compute and storage requirements of the clients, portable operation is facilitated. The InfoPad project [14], as described below, extends the remote computation concept by reducing the clients to little more than framebuffers with radios. In this way, ultra-low power, lightweight operation is possible since component count and battery requirements are dramatically reduced.

Remote computation also allows centralization of sensitive equipment to increase security and robustness. Since execution is remote, all storage is also moved away from the clients. Thus hard-disk failures, a constant threat to laptop computers, can be dramatically reduced since the disks are no longer moved. Additionally, sensitive information can be more readily safeguarded if it is kept in a single stationary location.

The inherent centralization of resources can greatly simplify system administration. No longer do all changes need be propagated to all clients, but rather the servers can be updated. As the complexity and capability of the clients is reduced, the configuration requirements are also reduced. Thus the "total cost of ownership" decreases.

Remote computation enables new economic / pricing structures. Using the remote computation model, computation is transformed from a product into a service. In this way, users can pay

Challenges w/Desktop or Wireless Portable:

Desired Aspect	Reality to Contend With
Fast Drawing	Limited Bandwidth
Rapid Response	Non-Zero Latency

Additional Challenges w/Wireless Portable only:

Reliability	Lossy & Error-Prone
Long Usage Time	Limited Battery Capacity

FIGURE 2.2. **Challenges posed by text / graphics and image transmission problem**

for computation on an as-needed basis and adaptation to varying needs is more agile. Additionally, this could reduce the recurring need to upgrade user equipment.

2.1.1.1. Challenges Posed by Text / Graphics and Image Transmission Problem

Some of the challenges associated with text / graphics and image transmission are listed in Figure 2.2. While the user requires fast drawing to display complex screens, this is difficult using a limited bandwidth connection. Similarly, the user's desire for rapid response to retain interactivity is thwarted by non-negligible link latencies.

The wireless environment presents additional challenges. While users demand reliable operation, the wireless link is often not privy to these demands. Similarly, the desire to operate portably for hours or days is often hindered by the limited capacity to weight ratio of existing batteries.

2.1.1.2. Connectivity Requirements Changed, not Created

It is important to note that although remote computation does require connectivity to send the display updates, many of today's applications involve information access. Thus connectivity is

required at some level anyhow. Remote computation is simply moving the connectivity partition but not adding new requirements *per se*. In many cases, using remote computation allows existing applications to be reused in new and varying environments.

2.1.2. Multimedia Collaboration

Multimedia collaboration is the process whereby several geographically separated parties can participate in electronic meetings sharing audio, video, and text / graphics information. While the audio and video primarily communicate the images and sounds of the participants, the text / graphics content can be shared whiteboards, pre-prepared slide presentations, or even shared jointly-controlled application executions. This text / graphics content is similar in nature to the text / graphics content produced by remote computation. In fact it can often be effected using the remote computation model.

2.2. Previous Work

2.2.1. X Window System

The X Window system [62,63,64] was developed at MIT as part of project Athena beginning in 1984, and gained significant popularity due to its free distribution. The X Window system allows distributed graphical computing in UNIX environments. It has been ported to many varieties of UNIX, including Linux and Solaris, and supports a range of graphics display hardware of varying capability, bit-depth, and acceleration. The X Window System operates in a client-sever model where the X server is run on a machine physically connected to a display monitor. Client applications can be run either remotely via TCP/IP or on the same machine. Keyboard and mouse input is sent to the clients from the server and text / graphics commands are sent back from the clients to the server.

The X Window system is designed with a highly layered architecture. The X protocol is at the lowest layer and describes the actual primitive commands such as draw line, draw text, clear area, etc. There are a set of *toolkits* that are layered on the basic X protocol to provide higher-level abstractions such as menus and other look-and-feel widgets. Some programming languages such as Tcl have been designed with toolkit extensions to X, such as Tk. Lastly, the basic X architecture decouples the look-and-feel of the system from the base architecture by introducing *window managers* whose sole purpose is to define the way that the user interacts with the system. These window managers are separate processes which run independently from the main system and can be freely interchanged.

The X protocol is drawing-primitive based which is described in the next chapter. This helps bandwidth efficiency but the encodings used are not very compact since X is not designed for bandwidth-limited environments. This typically leads to inadequate performance over band-limited links such as modems. The encoding is also quite error-sensitive, requiring a reliable protocol such as TCP/IP for proper operation.

A couple of architectural features improve interactivity of the X protocol. The first is the use of *graphics contexts* which store state which persists across multiple drawing primitives. The graphics contexts include current foreground and background drawing colors, font information, etc. This avoids respecifying the information in each drawing primitive request. Another feature to improve interactivity over higher latency links is the use of asynchronous operations. Drawing commands sent to the server are pipelined and identified using sequence numbers. Responses and error codes are returned to the clients asynchronously and matched up using the sequence numbers. This reduces the total latency experienced by a sequence of commands to one round-trip time.

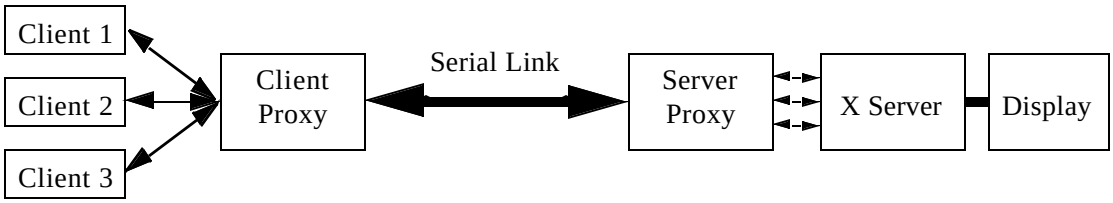


FIGURE 2.3. Xremote architecture

2.2.2. Xremote

Xremote is transformation of the X protocol designed to efficiently send X over serial lines [20,22]. The Xremote protocols works by running two proxies - one on each side of the slow serial line as shown in Figure2.3. Clients then connect to the client proxy using individual TCP/IP connections as they would connect to a conventional X server. The client proxy communicates with the server proxy via a managed, compressed protocol over a single serial connection. The server proxy then forms multiple connections to the X server just as the clients typically would.

The Xremote proxies are useful over serial lines because they both aggregate multiple X connections into a single stream, and additionally perform compression over the link. The compression entails several steps, the most important of which are delta-encoding and LZW compression. First, the X messages are delta-encoded whereby each message that is 64 bytes or shorter is compared to the previous 16 messages which were also 64 bytes or shorter. If a new message can be represented more compactly as a modification of a previous message, this representation is used. This is useful as many messages, such as mouse movements, are used many times with very similar contents. LZW dictionary compression[71] is then performed to exploit further redundancy in the delta-compressed stream. LZW compression finds repeated byte patterns and represents them more compactly. Xremote uses a reliable datagram transport protocol for transmission over the serial connection.

Xremote's overall compression is typically about 2.4:1 by typically achieving 3:1 compression on text-based messages and 1.6:1 on geometric messages [22].

2.2.3. Low-Bandwidth X (LBX)

Low-Bandwidth X [19,27,73] extends upon Xremote by using techniques to further compress some of the data stream, as well as techniques to avoid transmission of some data entirely.

Some of the compression techniques used by LBX include the use of CCITT Group 4 FAX compression for monochrome bitmaps [61]. This lossless compression technique exploits 2-dimensional redundancies in images to reduce the number of bits required to code them. Additionally many graphics primitives are recoded using 1-byte operands instead of 2-byte operands whenever possible.

The amount of data sent from the X server back to the client is reduced by caching of large data queries, such as keyboard maps, and just sending tags used to identify the data items. The type and size of the caches are negotiated upon connection of the LBX client to the server. Additionally, some constants which are typically queried from the server are handled locally or cached by the LBX proxy. Lastly, the number of motion events used to report mouse cursor movement can be limited to prevent excessive latency and uplink bandwidth utilization.

Like Xremote, LBX uses delta encoding and stream compression, but LBX uses the Zlib compression library [28] based on LZ77 coding instead of LZW compression [71]. The Zlib library typically compresses more effectively than LZW and also has patent-free status.

2.2.4. Higher Bandwidth X (HBX)

Danskin [22] further analyzed the work of Xremote and proposed improvements using statistical compression techniques on the X traffic in his HBX (Higher Bandwidth X) protocol. This

protocol improves upon Xremote's compression by about a factor of 3 to achieve roughly 7.5:1 overall compression relative to the standard X protocol on typical traces.

HBX uses arithmetic coding coupled with predictive models to compactly represent the X traffic. Different models are used for the various drawing primitive parameters. For instance, a polygon drawing primitive would be recoded by first converting all vertex coordinates to be relative and then statistically predicting later coordinates based on earlier ones. Text is predicted using the PPMC' method using hierarchical predictive models [47,10]. Bitmap images are compressed using context pixels to determine statistical predictions for the current pixel, in a manner similar to JBIG [4,43,61]. Small images, which are often reused, are cached at the server to avoid retransmission whenever possible.

2.2.5. Microsoft Terminal Server

The Terminal Server edition of Microsoft Windows NT 4.0 Server supports application independent text / graphics and image transmission, allowing multiple independent remote sessions on a single server [46]. The client machines display the remote data using a thin-client application. The protocol used for data communication is the Remote Desktop Protocol (RDP), also used in the Microsoft NetMeeting multimedia conferencing tool [45].

The Citrix Corporation has developed some low-bandwidth extensions to the terminal server protocol which are used in its Independent Computing Architecture (ICA). [16]

2.2.6. GraphOn Bridges

The Graphon corporation has developed a Bridges technology to replace its previous Go Global offering [35]. Go Global losslessly compressed X traffic from Unix workstations to thin PC clients.

2.2.7. Virtual Network Computing

AT&T's Virtual Network Computing (VNC) is a freeware application that allows remote operation of X Windows and Microsoft Windows [5]. VNC uses bitmap updates, as described in Chapter 4, including a copy-block update to transmit the screen changes. VNC also uses various image compression techniques to reduce the amount of data required for the bitmap updates.

CHAPTER 3 *Primitive-Based Approach*

3.1. Introduction

The primitive-based approach to remote text / graphics transmission involves sending drawing primitives which describe symbolically what to draw. These primitives are often the same drawing primitives used by the applications to describe their content.

Figure 3.1 depicts the operation of a typical primitive-based text / graphics system. Each application connects to the text / graphics server individually and sends its content as graphics primitives requests. The text / graphics server is responsible for combining these requests into a single stream which is sent to the remote client. The text / graphics server is also responsible for decoding user input, such as mouse or pen movements, and forwarding it to the correct application, as well as providing session and access control.

The primitive approach places two important requirements on the transport system:

1. Lossless transmission
2. Order and integrity must be preserved

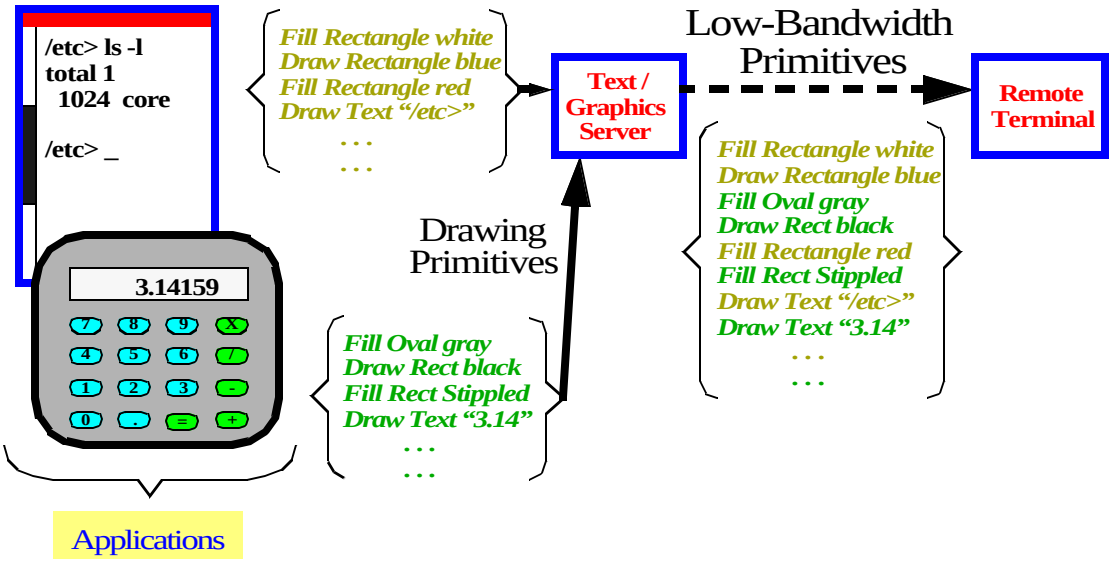


FIGURE 3.1. **Conventional primitive approach**

The transport system must be lossless because if any primitives are lost, this can have significant ramifications for the entire image. For instance, if a "`clear screen`" is dropped, the entire display would be incorrect. Typically, primitive-based systems also use notions of graphics contexts comprised of "current pen" and "current font" information. If commands to change these graphics contexts are dropped, then all subsequent primitives which use these contexts will be effected.

The transport system must preserve order and integrity. If primitives are reordered then their meanings can be dramatically altered. Often there are direct dependencies between two primitives; for instance, if a "`set current color red`" and "`draw line from 0,0 to 10,10`" are swapped, the color of the line drawn would not necessarily be red. If a "`copy rectangle`" is swapped with commands used to initialize the area being copied, this too would yield an incorrect result.

Thus transport-level protocols must ensure that the data stream is correct once it makes it to the text / graphics subsystem. Standard protocols such as TCP/IP[42] can satisfy the above requirements, even when using lossy links, and are thus typically used. However, this can result in significant increases in latency as shown in the next section.

3.2. Bandwidth and Latency Characteristics of Primitive-Based Systems

Primitive-based systems have the advantage that their bandwidth requirements are typically low since primitives can usually be specified compactly. However, the use of primitive-based systems can result in significant latencies to the end-user when used over lossy and/or bandlimited links.

3.2.1. Latency Due to Queuing Delays

In a primitive-based system, the applications specify the primitives used to render their graphical display. The text / graphics server must send each of these primitives, in order, to the remote terminal to assure proper display. Since all primitives must be sent, transmission over a bandlimited link can result in queuing delays as shown in Figure 3.2. The figure depicts the result of a remote user scrolling through a list of foods over a slow link which can only transmit two text items per time step. The positions the user is scrolling to are shown on the left side of the figure while the data received by the remote terminal are shown on the right side. At time T=0, the user has selected the top of the list and thus desires for the first four entries, Apple, Banana, Chicken, and Dessert, to be displayed. However, due to bandwidth limitations, only Apple and Banana can be sent over and Chicken and Dessert are queued to be sent over. At time T=1, Chicken and Dessert are received by the remote terminal but now the user desires to see Fudge, Gum, Ham and Ice,

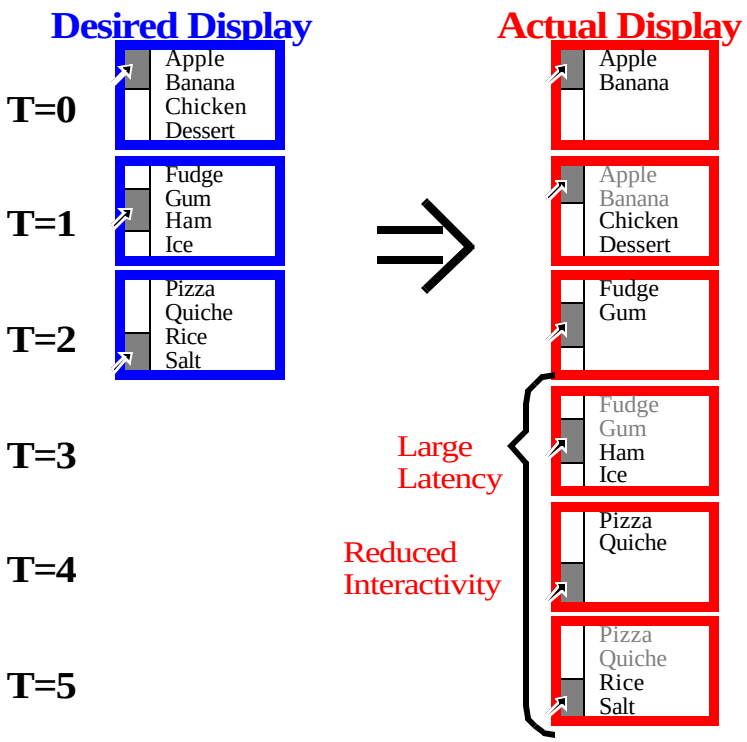


FIGURE 3.2. Latency due to queuing delays

and thus they must be queued up. At time T=2, the user has selected the final position viewing Pizza, Quiche, Rice, and Salt, but it is not until time T=5 that this appears. Thus queuing delays result in significant latencies for the user.

3.2.2. Latency Penalty Due to Loss

In order to understand the effect of data loss on latency, it is necessary to first review the operation of reliable protocols.

3.2.2.1. Reliable Protocols

As previously presented, the primitive-based system relies on an end-to-end guarantee that order and integrity of data will be preserved. If the link is lossy, a reliable protocol can be used to assure that data is not lost or reordered. These reliable protocols work by detecting packet losses at the receiver and requesting retransmissions from the sender. If a given packet is detected as

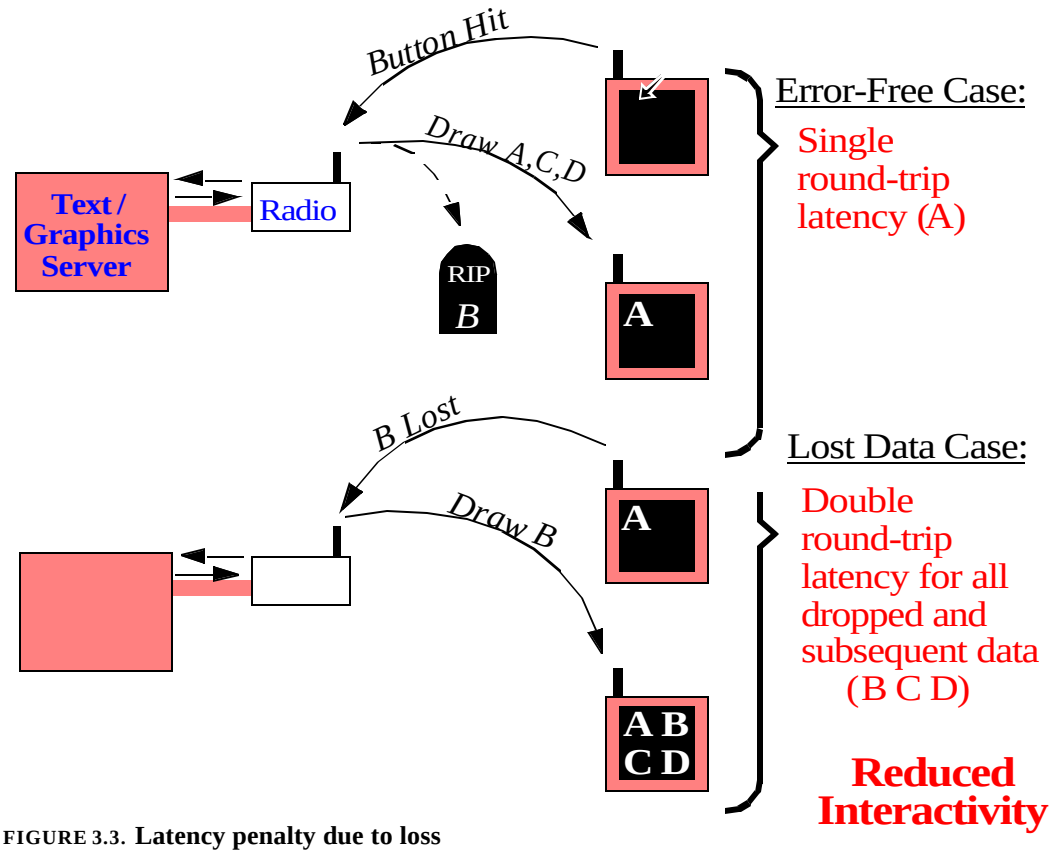


FIGURE 3.3. Latency penalty due to loss

having been dropped, it is re-requested by the receiver. The reliable protocol then holds all data it receives corresponding to packets which should follow the lost packet, until the lost packet is successfully received. Thus the application receives all data in order and without loss, but with increased latency if packets are lost.

3.2.2.2. Latency of Loss

The effect of reliable protocols on the transmission of text / graphics data is shown in Figure 3.3. The figure depicts the transmission of four drawing primitive packets - labeled A, B, C, and D in response to a button push on the remote client. In the figure, the four packets are transmitted in order, but packet B is lost in transmission, either due to signal degradation or congestion. The drawing of packet A proceeds without delay, incurring only the single round-trip latency necessary for the remote client to request an action, and the text / graphics server to effect that action. How-

ever, packets B, C, and D all incur at least a two-round trip delay since the loss of B needs to be detected and sent to the text / graphics server for retransmission[1]. This results in reduced interactivity.

1. Note that protocols such as SNOOP [8] can reduce this to a single traversal over the wired network, but an additional round-trip up to the basestation is still required.

CHAPTER 4

Bitmap-Based Approaches

4.1. Conventional Bitmap Approach

The conventional bitmap-based approach is depicted in Figure 4.1. In the conventional bitmap-based approach, drawing primitives from all applications are combined and rendered by the text / graphics server. The communication between the applications and the text / graphics server is performed using primitives as before, but the communication between the text / graphics server and remote terminal uses rendered bitmaps.

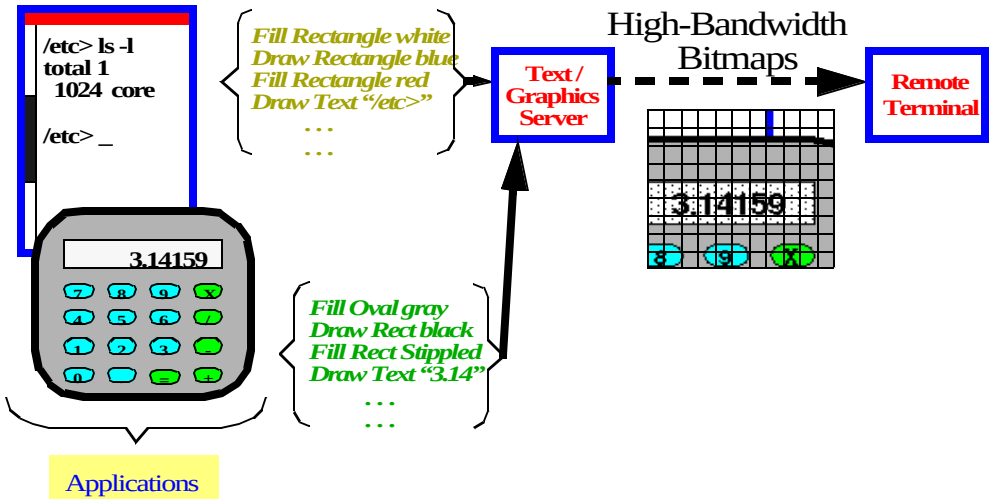


FIGURE 4.1. Conventional bitmap approach.

The bitmap-based text / graphics server operates similarly to a conventional text / graphics server attached to a framebuffer. The text / graphics server maintains its own framebuffer in memory since the information in a given primitive often does not contain enough information to determine all pixels in an update. For example, the "`copy block`" primitive requires the current contents of the screen for proper operation. Additionally, since bitmap updates are typically sent as rectangular blocks or unions of such blocks, primitives which would not completely modify all pixels in a block - such as "`draw circle`" or "`draw text`" - require the old pixel values to be known for use in the update packets. The connection between the text / graphics server and remote terminal is established using either a reliable link or a reliable transfer protocol layered on top of an unreliable link.

4.1.1. Assessment of Conventional Uncompressed Bitmap Approach

The conventional bitmap approach, as described, has one primary advantage, which is reduced complexity requirements in the remote terminal. Since the text / graphics server performs all rendering, the remote terminal needs only know how to display bitmap updates. Thus all drawing algorithms, intermediate state, and font information is confined to the text / graphics server. As described below, the InfoPad project exploited this to develop low-power, lightweight portable clients.

However, the conventional bitmap approach suffers from lower bandwidth efficiency which translate into greater latency using a given bandwidth link as compared to the primitive approach. This is because typically the bitmaps are less compact than the primitives used to generate them. Both the latency due to queueing delays and the latency due to loss previously described would still apply to the conventional bitmap system as described.

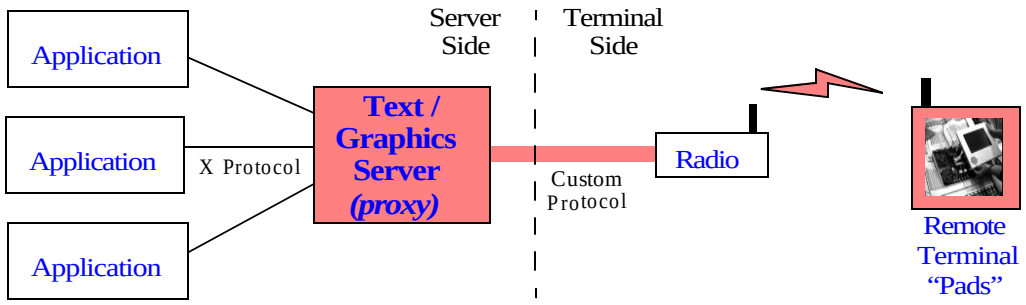


FIGURE 4.2. InfoPad text / graphics server context

4.2. InfoPad B/W Bitmap System

The InfoPad project [49] delivers ubiquitous portable computing using the remote interactive computation model described in Section 2.1.1. All applications are executed on a central compute cluster while display updates are sent to wireless portable "pads" as shown in Figure 4.2. Pen and audio information is sent back to the compute servers to allow user interaction. In order to obtain hours of operation using lightweight batteries, the InfoPad hardware is kept as simple and efficient as possible. Through careful system design, the pad hardware is reduced to a multimedia terminal with highly optimized data paths for heavily used functions such as the text / graphics and video. The text / graphics server forms the bridge between the custom low-power terminal hardware and the generic applications.

The text / graphics hardware subsystem is little more than a monochrome framebuffer that also decodes data packets specifying bitmap screen updates. The screen updates are sent as data packets with headers specifying the x, y, width, and height of a rectangular update region, as well as a data portion specifying the uncompressed bitmap. One bit is required for each pixel of data. The header and data are independently encoded, optionally using an error protection coding and checksum mechanism to protect against and detect bit errors.

Typically the graphics packet headers use error protection while the graphics packet data do not. (The exception is the implementation of asymtotic reliability described below.) This means that if the packet header is in error then the packet will not be processed. If the packet header is in error, the location or size of the update will not be correct, and thus the data will not be useful. However if a localized error occurs in the data, individual pixels might be incorrect but the bulk of the data will be correct and thus useful.

4.2.1. Pros and Cons of Uncompressed Bitmap System

While this simple bitmap update scheme does require significant bandwidth to support interactive applications, it offers some advantages in system performance and simplicity.

First, as previously stated, it allows for low-power hardware decoding. The entire datapath for decoding the packets and placing them into the framebuffer is readily implemented in custom logic resulting in very low power consumption. All protocol decoding consumed 1.9mW and the text / graphics framebuffer consumed 0.5mW [15].

Secondly, it reduces channel robustness requirements since data can be corrupted or lost without significant implications to system performance. The bitmap packets are independent, in that the loss or corruption of a given packet will effect the region of the screen it is targeting, but not subsequent updates. This means that if packets are dropped, it is not necessary to have them retransmitted and received before processing subsequent update packets. Thus a reliable transport protocol is not required and the latency due to loss of Section 3.2.2. is avoided. This is to be contrasted with the primitive approach of Chapter 3 where an error in one primitive might effect many later primitives. Additionally, if individual data bits in the packet are incorrect, the display errors will be small and localized. While the errors might be perceptible, they will rarely effect the overall intelligibility of the screen.

4.2.2. Asymtotic Reliability

Although it is acceptable for some temporary pixel errors to occur in order to greatly reduce latency, it is desirable if the "long-term" display is correct. Long-term error-free transmission must be designed not to prevent low-latency operation and should be possible if excess link bandwidth is available. In this way the user obtains the best of both worlds - low-latency and error-free display. The process by which slightly incorrect data will be displayed initially yet eventually the display will be error-free is called *asymtotic reliability* [36].

Asymtotic reliability, as described in [36], is achieved by using a low-latency unreliably primary display of data as previously described, combined with a background higher-latency reliable transport. In this way, a fast "best effort" is made which may result in some scattered bit-errors, followed shortly by one or more "refresh" updates which will be higher latency but will not introduce errors. The asymtotic reliability system reduces complexity requirements in the remote terminal since no uplink acknowledgments are necessary. Additionally, it can be used as a scalable information dissemination mechanism since only downlink traffic is used, any number of receivers can participate.

The refresh packets are sent at a low rate, in the background, using a higher level of error correction as well as error detection. The higher level of error correction reduces the probability that an error will occur. Error detection is used to suppress the display of packets if any error does occur. This error detection is critical to assure asymtotic reliability. Since only error-free packets are processed, asymtotically all errors on the screen will be corrected. Smaller packet sizes are used for the refresh packets since the packets must be error-free to be useful and the probability of one or more errors in a packet is exponential in the packet size.

4.3. Improved Bitmap using Virtual Framebuffer

While the system previously described allows remote operation of a wide variety of applications on a lightweight portable terminal, it does require a high-bandwidth communications link for interactive operation. Intuitively, since each drawing request results in the transmission of one or more screen update packets, actions that cause many updates to the screen in a short period of time can easily result in a backlog of the communications channel.

The key to efficient text / graphics transmission is to determine which information the user wants to see and how to send this information. In the case of remote text / graphics rendering, **the user only wants to see current information**. Thus if a user scrolls through a long list, they typically only want to see where they end up. If a user is viewing a progress bar, they only want to see the current value of it. If the user is participating in a video conference, they typically only want to see the most current image. The task is then to determine how to send only current information in such a way that a limited bandwidth link does not cause backlogs, and errors do not result in increased latency. Another way to view the problem is that the applications are typically designed for a high-bandwidth environments but the communications link is low-bandwidth.

One critical observation is that transmission of text / graphics information over bandlimited, lossy links is a form of remote-rendering just as transmission of video is. Two mechanisms which facilitate operation over bandlimited lossy links are data compression and data reordering. While video transmission techniques have used both of these aspects, text / graphics compression has so far been typically restricted to data compression only[1]. Thus in order to provide better performance, intelligent data reordering must occur.

1. One exception, in particular, is [37].

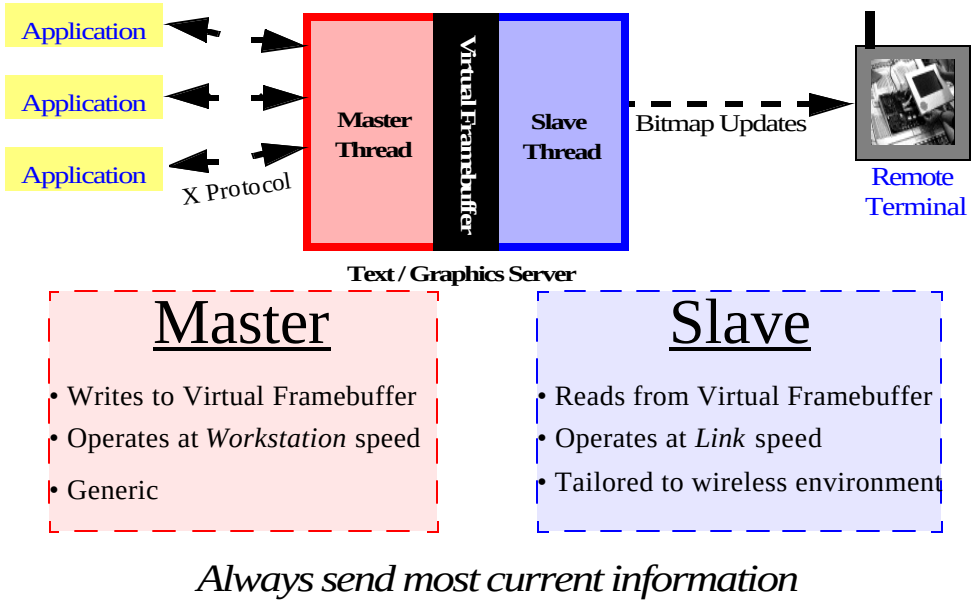


FIGURE 4.3. Improved bitmap approach using virtual framebuffer architecture

The virtual framebuffer architecture, shown in Figure 4.3 achieves this goal. In this architecture, the text / graphics server is split into two halves - the master and slave - which are coupled through an auxiliary buffer called a virtual framebuffer. The master communicates at full speeds with the applications and tracks the current contents of the screen on the virtual framebuffer. The slave then watches the virtual framebuffer and sends on any changes to the remote terminal.

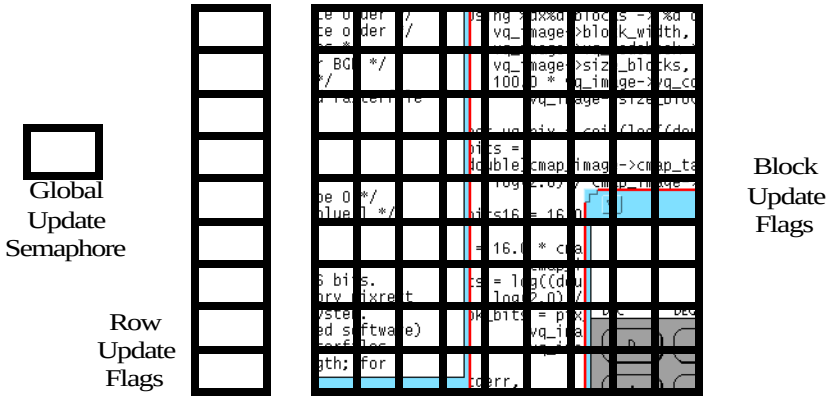


FIGURE 4.4. Virtual framebuffer

The virtual framebuffer, shown in Figure 4.4, consists of two arrays whose size matches the size of remote terminal's display. The first array contains the actual pixel data being displayed and is continually updated by the master and read by the slave. The screen is then divided into a set of blocks with one flag in the second array assigned to each block. These flags are used to indicate if the blocks have been updated by the master since the last read by the slave. The block size is chosen to be small enough such that granularity of updates is not too coarse, and large enough such that there are not so many blocks that the overhead of checking the blocks is noticeable. For efficiency reasons, row update flags are used to indicate if any blocks in a given row have been updated while a global update semaphore is used to block the slave until the master has updated something.

The master communicates with the applications at full workstation speeds. It responds to primitive drawing requests by rendering to the virtual framebuffer pixel buffer. When the master draws on part of the virtual framebuffer, it also sets the updated regions' "updated" flags. If the flags were already set, they remain set. In this way multiple updates are combined, reducing the amount of data sent to the remote terminal.

The slave runs in its own thread and asynchronously scans the virtual frame buffer from top to bottom in raster scan order. When it encounters blocks on the screen whose updated flags are set, it clears the flags and sends the data to the remote terminal. The slave uses *region growing* to form larger rectangular blocks from sets of contiguous blocks. This reduces the per-block overhead in transmission to the remote terminal.

The slave can also scan the virtual framebuffer in non-raster scan order to prioritize the display of certain parts of the screen. For instance it can scan the region surrounding the cursor more often than the rest of the screen since that area is typically of greater interest to the user. The spatial independence of the bitmap representation allows this.

4.3.1. Rate and Flow Control

The output of the slave can then be subjected to rate or flow control to match the link characteristics. Since it is decoupled from the master and applications, the slave's execution can be blocked without impacting application performance. The initial implementation in the InfoPad system used a rate-control system to limit the text / graphics traffic to be under a given rate. This rate is less than the capacity of the radio channel and can be dynamically changed. It was then expanded to include negative acknowledgments (NACK) described in Section 4.3.3.1., and can be extended to acknowledgment (ACK) based flow control as described in Section 4.3.3.2. Initially asymtotic reliability was used to reduce client complexity and protocol requirements.

Asymtotic reliability is readily integrated into the virtual framebuffer architecture by having the slave send refresh packets at a given rate while also sending normal updates. Adaptive bandwidth control is performed by setting the refresh rate as a number of bytes per complete slave pass through the virtual framebuffer and setting a minimum interval between complete passes through the framebuffer. Establishing the refresh rate in terms of bytes per complete slave pass causes the

amount of bandwidth dedicated to refresh to automatically reduce as the amount of foreground traffic increases. It also scales with the size of the "updated" area such that if a small area of the framebuffer is rapidly updated, refresh of the rest of the screen will still proceed rapidly, but if a large area of the framebuffer is continually modified, more bandwidth will be dedicated to its display, at the expense of slower refresh. Setting a minimum interval between complete passes establishes a maximum frame rate and can be used to reduce bandwidth utilized if a small region of the screen is updated very rapidly. At a minimum, it makes sense to set the transmitted frame rate to be no higher than the frame rate / refresh rate supported by the remote display device.

4.3.2. Analysis of Virtual Framebuffer Performance

In this section, the benefits of the virtual framebuffer technique are explored by analyzing the reduction in latencies due to queuing delays and loss.

4.3.2.1. Reduced Latency Due to Queuing Delays

Using the virtual framebuffer approach, latency due to queuing delays is bounded and dramatically reduced by the virtual framebuffer architecture through a process called *adaptive bandwidth compression* (ABC). ABC is a direct result of the virtual framebuffer architecture's ability to combine multiple writes to the same region of the screen. Recall that if the master writes to the same region of the virtual framebuffer before the slave has had a chance to send on the contents, the earlier updates are overwritten by the latest update. Thus effectively the bandwidth going into the virtual framebuffer is the high bandwidth of the application and coming out is the lower bandwidth that the link can support.

The example depicting latency due to queuing delays is revisited in Figure 4.5. Again the link capacity is set to two lines per time step. At time T=0, the user has selected the first four entries in the list. The slave starts sending from the top, only having time to send the first two lines

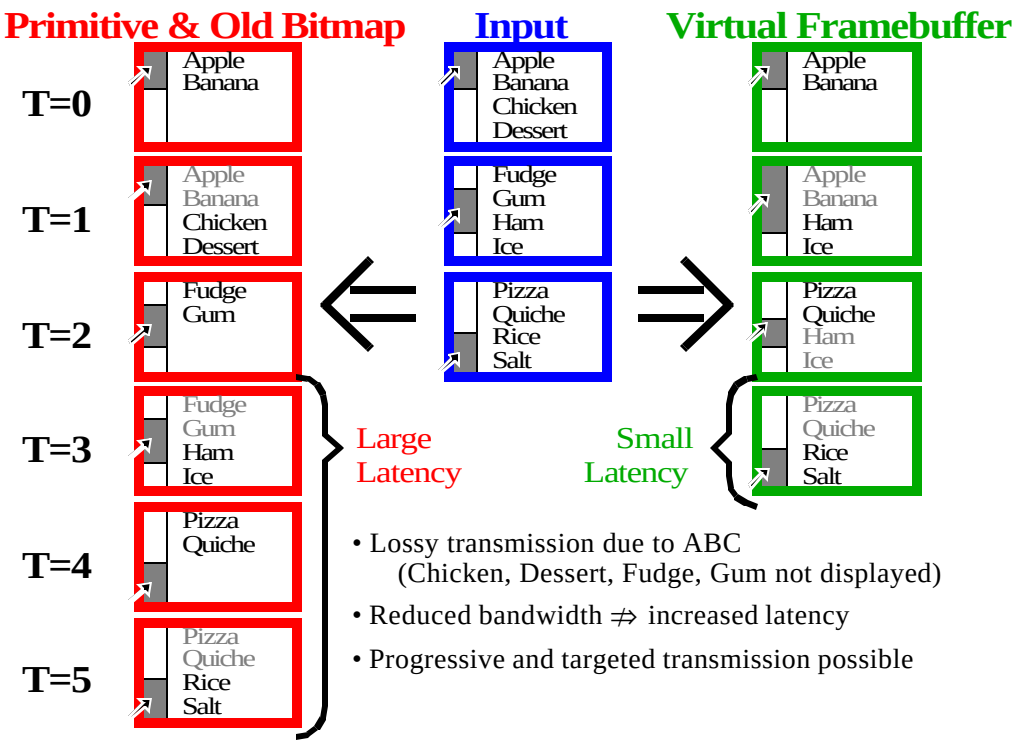


FIGURE 4.5. Reduced latency due to adaptive bandwidth compression (ABC)

- Apple and Banana. At time T=1 the user has selected to see Fudge through Ice. The slave is now reading the bottom half of the screen and would send over Ham and Ice. Chicken and Dessert would never be sent. At time T=2, the user has selected to see Pizza through Salt. The slave is now at the top of the screen and Pizza and Quiche would be sent. Fudge and Gum were thus overwritten before they could be sent. Finally at T=3, the user has not caused any further updates and the slave is at the second half of the screen and can send over Rice and Salt. At this point, all regions of the screen have been communicated and all updated flags are cleared. Thus using the virtual framebuffer technique, the user has a complete, correct picture of the screen only one time-step after they cease input activity, while the conventional primitive and old bitmap techniques require three additional time steps.

As a numerical example of the improved interactivity, consider user scrolling through a document in a 500x500 pixel monochrome window over a 500 kbps link. Each frame would require 250 kbits of data or about a half-second to send. Thus if the user scrolls five times in a sec-

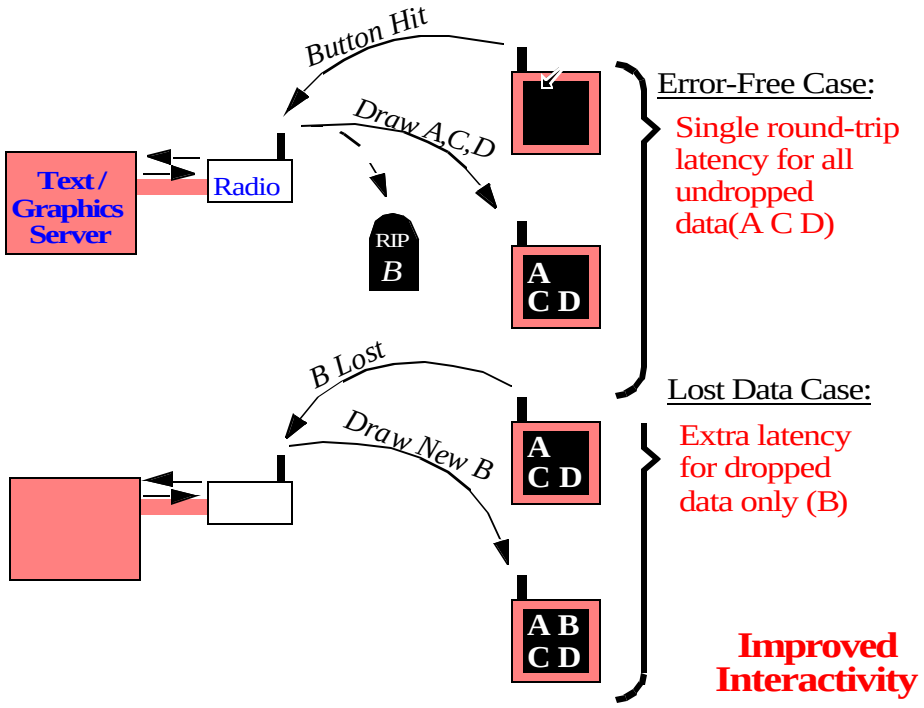


FIGURE 4.6. Reduced latency due to loss

ond, it will take an additional 1.5 seconds for all of the data to be delivered. If they continue at this rate for 5 seconds, it will take an additional 7.5 seconds for the final data to be delivered. Using the virtual framebuffer system the lag would always be at most the time to update the screen or 0.5 seconds in this case.

4.3.2.2. Reduced Latency Due to Loss

Figure 4.6 shows how the virtual framebuffer approach using asymtotic reliability results in reduced latencies due to packet loss. As in the example of Figure 3.3, the figure depicts what will happen if the second (B) of four packets is lost. The three packets which were successfully communicated (A, C, D) are displayed with a single round-trip latency while only the dropped B packet is delayed. Using asymtotic reliability, it would be delayed until a refresh packet could deliver the data. Using more sophisticated methods described below, the latency can be further reduced.

4.3.3. Integration of Virtual Framebuffer into Transport Control Protocol

The virtual framebuffer can be integrated into the transport protocol for further increases in throughput and reliability. Conventional transport protocols, such as TCP/IP, order individual packets in a stream with a sequence number. They use these sequence numbers to assure that every packet in the stream, and thus every byte in the stream, is successfully communicated in order. Since conventional transport protocols have no knowledge of the underlying data, their goal has to be to successfully transmit the entire stream of data. This leads to two problems:

1. Once data enters the transport layer, it will consume bandwidth.
2. Since no dependency information is known, it is assumed that all data is dependent on all other data and thus all ordering must be preserved.

However, in the case of the virtual framebuffer, the location on the screen that an update corresponds to contains valuable information that can be used to remedy the above problems. The integration of the virtual framebuffer with the transport layer works as follows: Instead of having a separate buffer to store data that has been committed but not yet acknowledged, the virtual framebuffer serves as the holding buffer. The update flags indicate which data must be sent and thus the actual data need not be copied. Thus actual packetization of the data does not occur until just before the packet is going to be sent over the network. In this way, old data cannot be queued up since old data is superseded as previously described. The virtual framebuffer can be used to implement a negative acknowledgment (NACK) or positive acknowledgment (ACK) based system as described next.

4.3.3.1. Negative Acknowledgments (NACK)

The InfoPad downlink traffic consists of text / graphics, video, and audio data. While the total available bandwidth to the pads is fixed at approximately 500kbps, the portion dedicated to each type of data varies based on the amount of traffic dedicated to the others. For instance, if no

other streams were present, text / graphics could use the full 500kbps link, but if a 300kbps VQ video clip (see Section 5.2.) is playing, the amount available to text / graphics drops to 200kbps.

The separate multimedia streams are not combined until a gateway which follows the text / graphics, audio, and video "type servers". This makes it more difficult for the type servers to measure the amount of traffic generated by other sources. In addition to long-term rate adaptation, short term management of the traffic is necessary. Since the gateway combines the various traffic streams, it is able to determine when the net rate exceeds the link capacity. The gateway buffers data, and thus if the total incoming traffic is greater than the outgoing rate limit, packets in the buffer will be aged. These "old" packets can be dropped to assure that the backlog of data is bounded. The gateway then sends negative acknowledgments (NACKs) back to the sender. These NACKs can then be integrated quite easily into the virtual framebuffer architecture by simply having the slave set the "updated" flags of the region corresponding to the packet that was NACKed. Thus the NACK indicates that the data specified in the packets is still outstanding and thus must be sent again. Note that if, in the interim, part or all of the region specified by the packet was modified again, the updated flag would already be set and thus no extra bandwidth will be consumed by the retransmission.

Note that with the simple scheme above, superfluous retransmission could occasionally occur that would waste bandwidth, but not produce an incorrect result. An example is as follows:

1. Region of screen is updated and transmitted as update A.
2. Same region is updated and transmitted as update B.
3. Update A is removed by gateway and NACK is returned.
4. Region is invalidated, thus causing resend of region.
5. Region is retransmitted as update C which is identical to update B.
6. Update B received correctly.
7. Update C received correctly.

Thus an additional update packet is sent because a region is updated between being sent and NACK'ed. However, since current data is always sent, extra update packets will never cause the incorrect results to be shown. An extension to the basic NACK algorithm could include keeping a sequence number, as described below in the ACK scheme, to avoid extra retransmissions in the above case: If a region is updated after a packet is sent, all retransmissions due to that packet are aborted.

4.3.3.2. Positive Acknowledgments (ACK)

While negative acknowledgments allow for rapid notification of congestion, they are not well suited for packet loss or error notification for the following three reasons:

1. It is often difficult to detect the absence of a packet.
2. NACKs can increase congestion if sent over the bandlimited medium.
3. If the NACKs are sent via a lossy medium, they too can be lost.

For these reasons, positive acknowledgments (ACK) are preferred. This was not implemented in the InfoPad system but could be used in similar systems[1].

The ACK-based system works by tracking the update packets sent to the remote terminal and having the remote terminal send back acknowledgments of each graphical update packet or set of such packets. In this way, the text / graphics server can track which updates have been successfully communicated to the remote terminal and retry any that have not. As before, no intermediate storage buffers are used since the only data that is useful is the most current data, which can be found in the virtual framebuffer. Also as before, multiple updates to the same region are combined whenever possible, discarding old updates.

1. Note that the ideas in this section, unlike those in the previous section, have not been implemented but are provided as an extension of implemented work.

In order to track the reception of each packet, a sequence ID is used. The sequence ID is incremented for each transmitted packet such that it will be unique to all packets that could be in flight. The virtual framebuffer includes a sequence ID and transmit time field for each block as described below. Each block in the virtual framebuffer also has a status field that can indicate one of three states:

1. Not updated
2. Update required
3. Update in flight

The "not updated" state is used when the contents on the remote terminal are current and thus the local contents in the virtual framebuffer have not been updated recently. When an update does occur, via the master, the status of the block is changed to "update required". No sequence ID is associated with the block in either the not updated or update required states. Once the slave detects that the block has been updated, it generates a graphical update packet, assigns it a sequence ID and transmits the packet. The sequence ID and time of transmission are recorded in the virtual framebuffer and the block's state is changed to "update in flight". When an acknowledgment of the update is received from the remote terminal, all blocks covered by the acknowledgment whose sequence ID still matches the ID of the acknowledgment are changed to the "not updated" state.

If any of the blocks are updated between the time that the update packet is sent and the time that acknowledgment was received, the master will then revert their state back to the "update required" state and their sequence ID field is no longer relevant. When the slave detects that they have to be sent, it will generate a new update packet with a new sequence ID. All links to the old update packet in flight will be forgotten since this would be stale data. Thus when the old acknowledgment packet is received, its sequence ID will not match the sequence ID of the blocks

and their state will not be changed to “not updated” until an acknowledgment of the most recent update is received.

If data is lost or corrupted, it must be resent. This can be detected by the absence of an acknowledgment of the packet. The absence is detected with via a time-out - i.e. if the acknowledgment is not received within a certain amount of time from the transmission of the packet, the packet is assumed to have been lost. Additionally, if packets transmitted after the packet in question are acknowledged, but the packet in question has not been, then it may be safe to assume that the packet has been lost. If the network can cause out-of-order delivery to occur then this must be considered before assuming that a packet has been lost. Much research based on TCP/IP has addressed these issues.

The packet loss detection can be incorporated into the slave’s scanning process. As the slave scans to see if any blocks have been updated, it can also check if any blocks are in the “update in flight” state and should be treated as lost. In this case, they are implicitly switched to the “update required” state and a new update packet is generated.

The acknowledgment protocol must differ from byte-stream reliable protocols such as TCP/IP. TCP/IP uses cumulative acknowledgments; a TCP/IP receiver sends back the sequence ID of the latest packet which has been successfully received and had all previous packets in the sequence also successfully received. In this way, each acknowledgment of a given packet also acknowledges all prior packets. This can be useful if an acknowledgment is dropped as later acknowledgments may accomplish the acknowledgment. However, in our case, this would create false-dependencies. Thus each packet must be individually acknowledged. A bit-vector representation can be used to acknowledge multiple packets in a single acknowledgment. I.e. an acknowledgment packet could contain the sequence ID of the first and last packet to be acknowledged and then a bit vector specifying which of the intermediate packets should also be acknowledged. The acknowl-

edgment of a given packet could be contained in multiple acknowledgment packets to protect against loss of acknowledgment packets.

Graphical updates whose data is partially corrupt but still usable could be displayed but not acknowledged. In this way, the user could obtain a mostly-correct display very rapidly and the fully correct display would follow as soon as the retransmission is successful.

CHAPTER 5 *Color Text / Graphics, and Video Support*

5.1. Bandwidth Requirements of Uncompressed Color

The bitmap approach previously described, as demonstrated in the InfoPad project, yielded an interactive display supporting an effective graphical user interface. However, the 640x480 monochrome display requires several hundred kilobits per second of bandwidth for interactive operation. While a color display is preferable from user and application perspectives, it does significantly increase the demands on the communications link. Without using compression, the monochrome display requires one bit per updated pixel. Thus a 200x200 window, updated at 10 frames per second (fps), would require 400kbps - which is feasible using high bandwidth indoor radios. However, using a true-color display, each pixel requires 24 bits - 8 for red, 8 for green, and 8 for blue. Thus the same 200x200 window updated at 10 fps would require almost 10Mbps, or given the same 400kbps link, an update rate of less than one half of a frame per second would be possible. Using a conventional 8-bit per pixel paletized display, about 3.2Mbps is required for a 10fps update rate and 400kbps will allow slightly more than one frame per second - neither of which yields a good system solution.

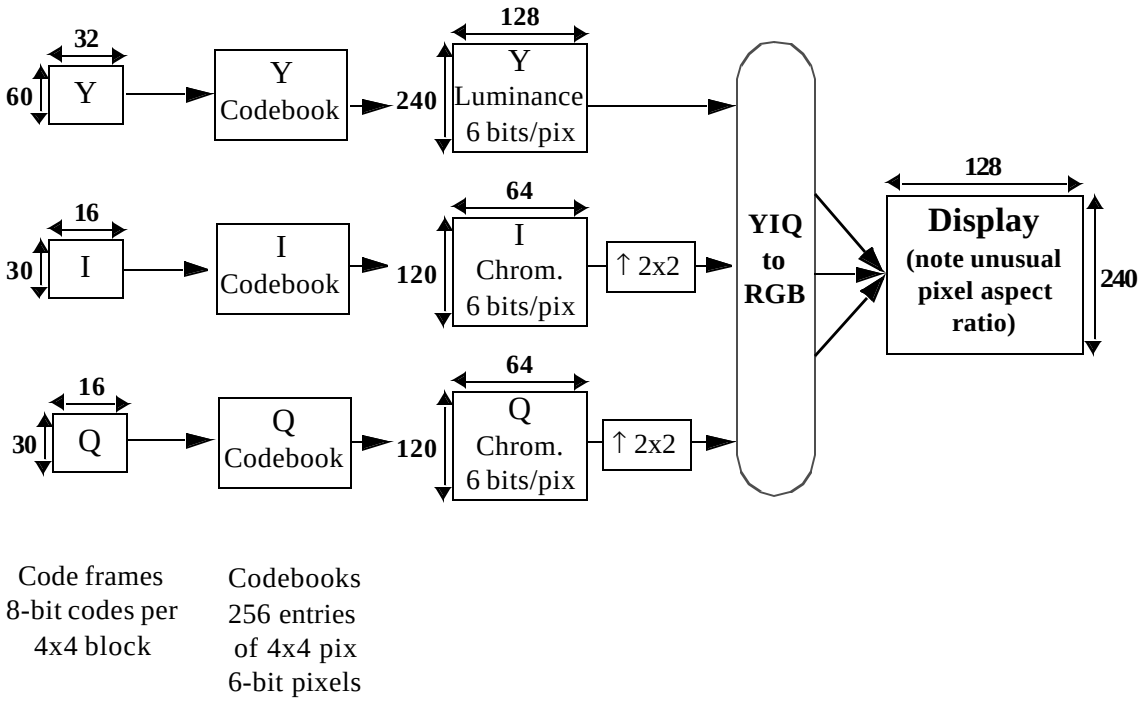


FIGURE 5.1. InfoPad full-motion VQ video support. Detailed in [15]

5.2. Full-Motion Color Video Support via a Separate Display

One way to enable full-motion color video, while not impacting the display of text / graphics used in applications, is to retain the monochrome display for text / graphics and use a separate display for full-motion color video. The InfoPad project used this approach to allow independent research into text / graphics and video delivery. From the perspective of wireless link research, the text / graphics traffic necessitated low latency delivery of bursty traffic, while the video traffic required high, but more or less uniform, bandwidth.

5.2.1. Lossy Vector Quantization for Image and Video Compression

The InfoPad project used lossy Vector Quantization (VQ) to deliver full-motion color video with minimal power consumption and hardware costs[1] [15]. Vector quantization entails repre-

senting groups of pixels, such as 4x4 pixel blocks, within the image using a single index into a "codebook". The codebook contains sets of the groups of pixels. Since the number of bits required to specify the codebook index is much smaller than the number of bits to specify the colors of the pixels in the group, compression is achieved. However, since not all possible combinations of pixels can be represented in the codebook, the compression is lossy. (If all combinations were represented then the codebook index would have to be as large as all of the pixels in a group combined.) Typically each group of pixels in the input image is assigned the index of the codebook entry which has the group of pixels that is most similar to it, as determined by a minimum mean squared coding error.

While vector quantization does not yield the highest quality video for a given bit rate, it does perform significant compression with low-complexity decompression. The coding is asymmetric in that compression is computationally intensive but decoding is not. Since the decoder is typically a portable device, and coding of movies needs only be performed once, it is well suited for a remote wireless portable device. Additionally, vector quantized video does not cause error propagation within each frame or across frames. Bit errors are localized to a particular region and will not persist into the next frame. However, bit errors in codebook updates will persist across all frames that use the codebook entries that are in error.

Decoding of the vector quantized bit stream can be performed in hardware as a set of memory lookups and an optimized coordinate space transformation. The InfoPad low-power hardware decoding operates with a power consumption of less than 2mW. VQ encoding can be computationally intensive since a codebook search has to be performed for each group of pixels in the image. However, techniques described below show how trade-offs between coded image quality

1. While this section presents my work in real-time VQ video transcoding, it should be clear that the choice of VQ display, the hardware, coding, and format were determined by Chandrakasan and Brodersen before my joining the group.

and encoding time can be achieved through the choice of the codebook and codebook search algorithm. Both high complexity (low-speed), high-quality coding, as well as low-complexity (high-speed), lower quality compression techniques are described.

The details of the InfoPad VQ video decoder implementation are shown in Figure 5.1. A display of 128x240 pixels is generated from a modified luminance (Y) / chrominance (IQ) color space where the I and Q have been decimated by 2 in both the horizontal and vertical directions. The decimation is to exploit the reduced sensitivity to chrominance information of the human eye to reduce system bandwidth requirements. The exact coefficients used to convert the YIQ into RGB were determined as a compromise between hardware power savings and the benefits of decoupling and subsampling the chrominance components with respect to the luminance components. The 128x240 Y image and 64x120 I and Q images are each generated through vector quantization decoding via a table lookup. All vectors in the system are 4x4 sample blocks where this corresponds to 4x4 pixels in the case of the Y component or 8x8 pixels in the case of the I and Q components, due to upsampling. The vectors for the Y, I, and Q images are selected from three 256-entry codebooks. Thus the images are specified by 32x60 8-bit Y codes and 16x30 8-bit I and Q codes. Typically the codebooks are updated infrequently so that only the Y, I, and Q codes are updated on a frame by frame basis. This requires 2880 bytes total, allowing 30 fps operation given a 690kbps downlink. If the vector quantization is not used, and the video is specified by 24-bit true-color pixels, each frame would consume 92160 bytes, requiring more than 22 Mbps for a 30 fps video stream. Since only the codebook and indices are retained in memory, and decompression is performed on the fly, memory requirements are significantly reduced[1].

1. Note that this approach, while implemented in hardware as a compressed framebuffer, is treated in this chapter and not Chapter 6 because the video is sent in complete frames, and thus none of the issues related to independent manipulation of subregions of the display are relevant.

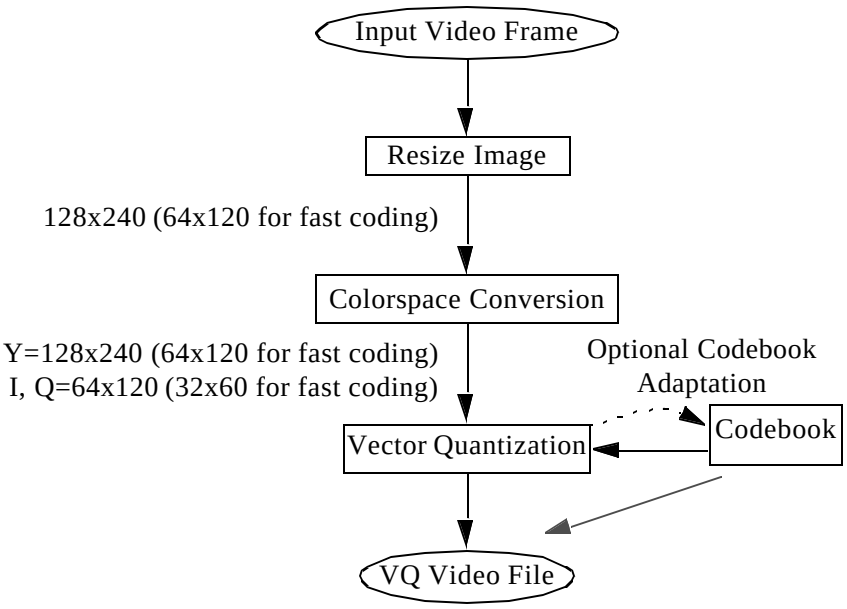


FIGURE 5.2. Vector quantized (VQ) video encoding

5.2.2. VQ Video Encoding

The steps required to encode a VQ video stream are shown in Figure 5.2. Since the VQ video compression format does not use inter-frame compression, such as motion compensation, the input frames are encoded independently. The input video frame is first resized to the size of the VQ display, 128x240. For fast coding, described below, a half-sized image is produced. Next the image is converted into the YIQ color space via matrix multiplication or a table-lookup equivalent. Finally the image is quantized by considering each of the 4x4 sample blocks in the Y, I, and Q image planes separately, and selecting the entry in the appropriate codebook that exhibits the least mean-squared error. The codebook can be chosen adaptively from the video clip or statically, yielding higher image quality and reduced coding time respectively.

5.2.2.1. Adaptive VQ Encoding

Adaptive VQ encoding entails generating the codebook based on the video sequence to be coded. In this way the codebook will most effectively represent the images in the video. A single

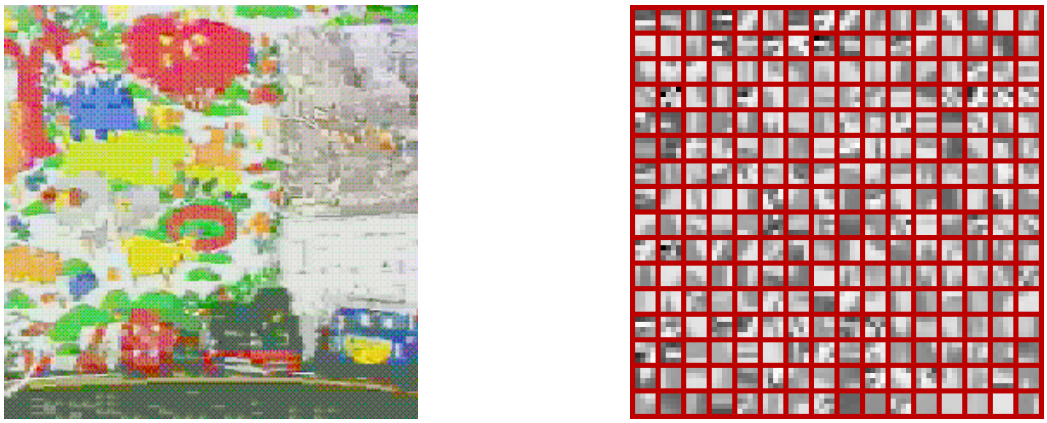


FIGURE 5.3. **Single frame from the video clip and luminance (Y) codebook adapted to it.**

codebook can be used for the entire video or else the codebook can be periodically updated on a scene-by-scene basis, or whenever the coding error exceeds a given threshold. In either case, the number of frames used to determine the codebook is typically limited to reduce the time to generate the codebook.

The K-means clustering algorithm is used to generate a representative codebook from a sequence of input frames. All 4x4 blocks in the frames of the input video are considered as training vectors. The Y, I, and Q codebooks are generated separately.

The K-means clustering algorithm adapts the codebook as follows: An initial codebook is used to code the input vectors. Then each codebook entry is recomputed as the average of all image vectors for which it is the best match. Thus each codebook entry is modified to better represent the vectors that match it. All vectors are then recoded and the codebook is updated until the total coding error stops decreasing. The initial “seed” codebook can be specified externally or defaults to the static codebook used for the fast coding described in the next section.

Some extra steps are used to ensure that the codebook best represents the diversity in the image. The 256 codebook entries are compared to each other, and if two are too similar then one is

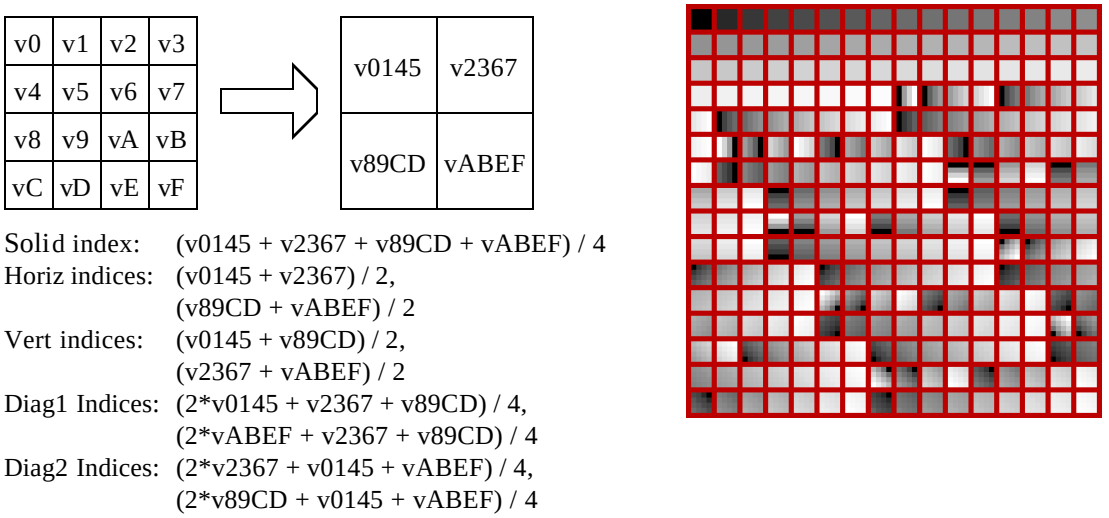


FIGURE 5.4. Gain / shape codebook used for fast VQ encoding. It consists of 56 solid gradients and 50 each of vertical, horizontal, and both diagonal gradients.

"freed" up for use by some other vector. The vectors which matched the freed codebook entry are then assigned to the one that it was similar to. Next, the unused codebook entries are filled with the input image vectors which had the greatest coding error, to ensure codebook diversity and reduce the worst-case coding error. Figure 5.3 shows the luminance (Y) codebook adapted from a video sequence.

5.2.3. Fast Fixed-Codebook VQ Transcoding

As presented, the VQ video encoding time is dominated by the time to search the codebook. In excess of 10 million pixel differencing operations are required per frame for the 128x240 video format. This results in a coding rate of only a few frames per second using optimized C code on a Sun UltraSparc 2 workstation. However, the search can be accelerated by tailoring the codebook design for fast search.

Gain / shape codebooks orthogonalize the "shape" of the codebook entries from the extent or "gain" of the entries. In this way if there are a few basic shapes, and the best gain for each

FIGURE 5.5. **Comparison of adaptive and fast codebooks. Left uses fixed, fast codebook, while center uses another video's codebook, and right uses a codebook adapted for the video in question**

shape can be quickly determined, then matching can be greatly accelerated. The gain / shape codebook used is shown in Figure 5.4. It consists of 56 solid gradients and 50 each of vertical, horizontal, and both diagonal gradients. The best match is determined by subsampling the 4x4 pixel blocks by two and using each of the four values to select the best gain for each of the five shapes in a single lookup. The indices used for fast match lookup are shown in the figure. Tables map the index value or indices values to the best codebook for each shape. The error is then computed for the best candidates of each of the five shapes and the codebook entry giving the least error is chosen. Because the gradients in the fixed codebook are smooth, half-resolution comparison is possible. The fast coding method can achieve 30fps coding for real-time compression

Figure 5.5 shows a comparison of the image quality delivered by coding frames from two video clips using three methods: fast coding, coding to a codebook adapted to another video clip,

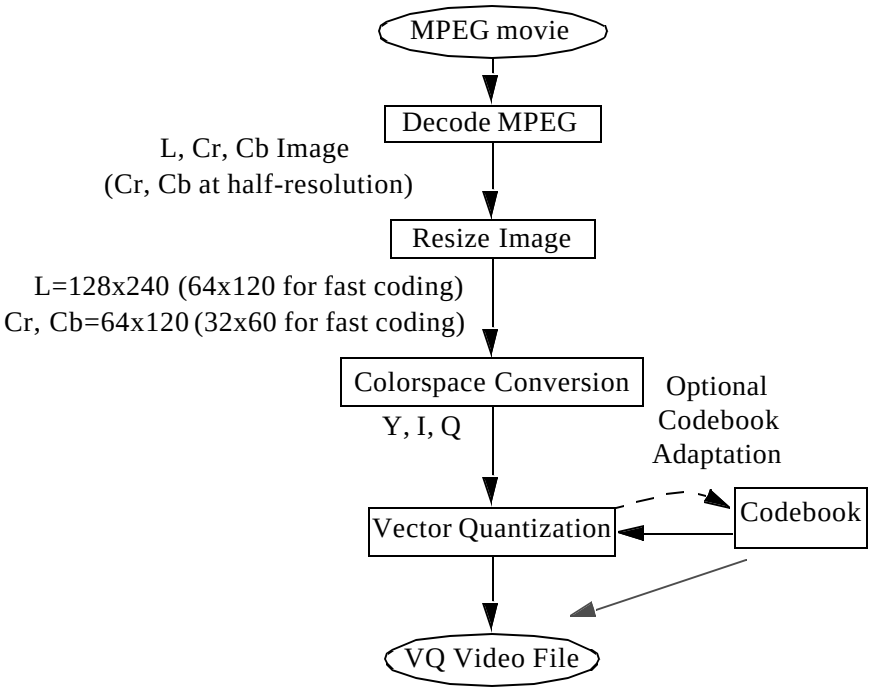


FIGURE 5.6. MPEG to vector quantized (VQ) video transcoding

and coding to a codebook adapted to the video clip in question. As can be seen, the fast coding method yields a coarser looking video, coding to another video can result in some artifacts, while adapting to a particular video results in the most aesthetically pleasing image.

5.2.4. MPEG to VQ Video Transcoding

Due to the abundance of MPEG video clips, an MPEG to VQ Video Transcoder was designed both as a source of VQ videos to demonstrate the pad as well as to serve as a vehicle to explore issues in VQ coding techniques. The data flow used to transcode MPEG to VQ videos is shown in Figure 5.6. MPEG natively generates separate luminance and chrominance images where the chrominance images have been subsampled by two in each direction. The MPEG and VQ color spaces differ, however, as the MPEG color space was optimized solely for human visual perception while the VQ color space also considers hardware color space conversion complexity. The utilities used to generate VQ video clips are detailed in Section 9.3.3. and Appendix A.

5.2.5. Live VQ Video Display of MBONE Transmissions

VQ video encoding was also integrated into the MBONE video gateway (VGW) [1] to allow live viewing of MBONE transmissions. The fast coding method allows real-time display. The GUI of the MBONE applications can be displayed in monochrome on the main text / graphics screen.

5.3. Motivation for Unified Text, Graphics, & Video Display

In order to minimize power consumption and complexity while supporting both general purpose user-interface based applications, as well as streaming video, the InfoPad system, as previously described, employs separate text / graphics and streaming video displays. The text / graphics display is a 640x480 monochrome, conventional framebuffer-based display. This allows individual pixel addressability to enable most user-interface based tasks, such as a shared whiteboard, handwriting recognition, and web browsing. The monochrome nature allows timely delivery of even uncompressed bitmap updates given a moderately high bandwidth wireless link. The monochrome display is not suitable for full-motion video, and thus an auxiliary display supporting color and compression is required. For this purpose, a separate vector quantized full-motion color display was used.

While using separate displays is quite effective to demonstrate the individual components, the low-power consumption achievable, as well as the capability of remote operation of both user-interface and streaming video tasks, it requires special applications to display the streaming video, only one video can be displayed at a given time, and the size and quality of the video is constrained. Additionally, the user-interface based applications cannot enjoy the benefits of color. Thus it is advantageous to have a single display which can seamlessly display both user-interface based graphics as well as full-motion video.

5.4. Uncompressed Framebuffer, Compressed Sends

One approach to improve this situation is to compress the bitmap updates before transmitting them. Thus the same basic infrastructure is used, though the updates are compressed before transmission and uncompressed at the receiver end. This can significantly reduce bandwidth requirements. Though it should be noted that this often renders corrupt packets useless. Different degrees of compression and channel coding allow a trade-off between bit rate requirements and error tolerance. Much research has focused on compression of continuous-tone images such as photographs. Some techniques include JPEG [40] and Wavelets [60]. Chapter 8 will present background and a new technique for compression of discrete-tone images such graphs, text, and most graphical user interfaces.

CHAPTER 6 *Compressed Framebuffer Approach*

6.1. Minimizing Client Hardware and Power Consumption

While the compressing bitmaps for transmission reduces the bandwidth requirements imposed on the communications link, it does not reduce the amount of storage required on the remote client. This then impacts the power consumption and cost of the portable terminal. As presented previously, the storage requirements of a color screen are 8 to 24 times that of a monochrome screen.

As Chandrakasan [15] demonstrated with the compressed VQ video display, further reduction in portable client power consumption and complexity can be achieved through the use of a compressed framebuffer.

A compressed framebuffer stores the data to be displayed in compressed form, and decompresses the data “on the fly” during the monitor refresh readout. In this way, the storage requirements are reduced. Additionally, since the amount of memory per frame is reduced, the bandwidth requirements from the memory can be reduced, and this can result in lower power consumption if the decompression technique is also “low-power”.

However, retaining only a compressed framebuffer means that all possible display configurations cannot be realized and thus the compression technique must be carefully designed to avoid excessive visual distortion. This chapter discusses the application of the compressed framebuffer approach to text / graphics display.

6.2. Requirements

A compressed framebuffer imposes constraints upon the choice of compression algorithm that do not exist if the compression algorithm is used only for transmission of the images. This section describes some of those additional constraints. (Note that additional transmission-only compression can be applied beyond the compressed-framebuffer compression.)

6.2.1. In-Place Modification of Compressed Data

Compression algorithms, particularly lossless data compression algorithms, often yield reductions in data requirements by exploiting inter-symbol correlation. Thus, the fact that one symbol can be predicted, at least in part, from a previous symbol, means that it can be stored more compactly if this prediction is incorporated into the coding. Only the information that cannot be predicted needs to be stored. However, this means that the later symbols depend on the earlier ones, and cannot be individually decoded.

However, the data in a framebuffer is modified in a random-access manner when a particular part of the screen is modified. It is typically not acceptable to send the entire contents of the screen to update a small region. Additionally, since the uncompressed framebuffer is not present, the modifications cannot be performed in the uncompressed domain.

The combination of these two factors requires that the modification of a region of the screen does not result in the insertion or deletion of data in the framebuffer, but rather simply the modification.

6.2.2. Update-Independence for Error Tolerance

In Section 3.2.2. it was demonstrated that a large latency penalty is incurred if the update of one block is dependent on that of preceding blocks. While the updates of blocks in an uncompressed framebuffer are independent, most compression algorithms exploit spatial redundancy between regions of the image, and thus would be subject to the latency due to loss. Thus the compression algorithm must be designed such that the interdependence between updates is minimized or trade-offs between interdependence level and bandwidth utilization can be controlled.

6.2.3. Must Work for Text / Graphics and Image / Video

As previously noted, the unified framebuffer contains both discrete-tone text / graphics regions as well as continuous-tone image / video regions. Thus either the same compression algorithm must work for all regions of the screen or else several compression schemes must be implemented with an automated way to select the best one for a given region.

6.2.4. Must Work for all Possible Screen Configurations

One of the advantages of the compressed framebuffer is reduced storage requirements. This requires that all possible screen configurations, when compressed, will fit into the framebuffer memory. Since no lossless compression will always result in data reduction, lossy compression or a lossy mode must be used. Furthermore, the minimum worst-case compression ratio is dictated by the ratio of the uncompressed framebuffer size to the compressed framebuffer size.

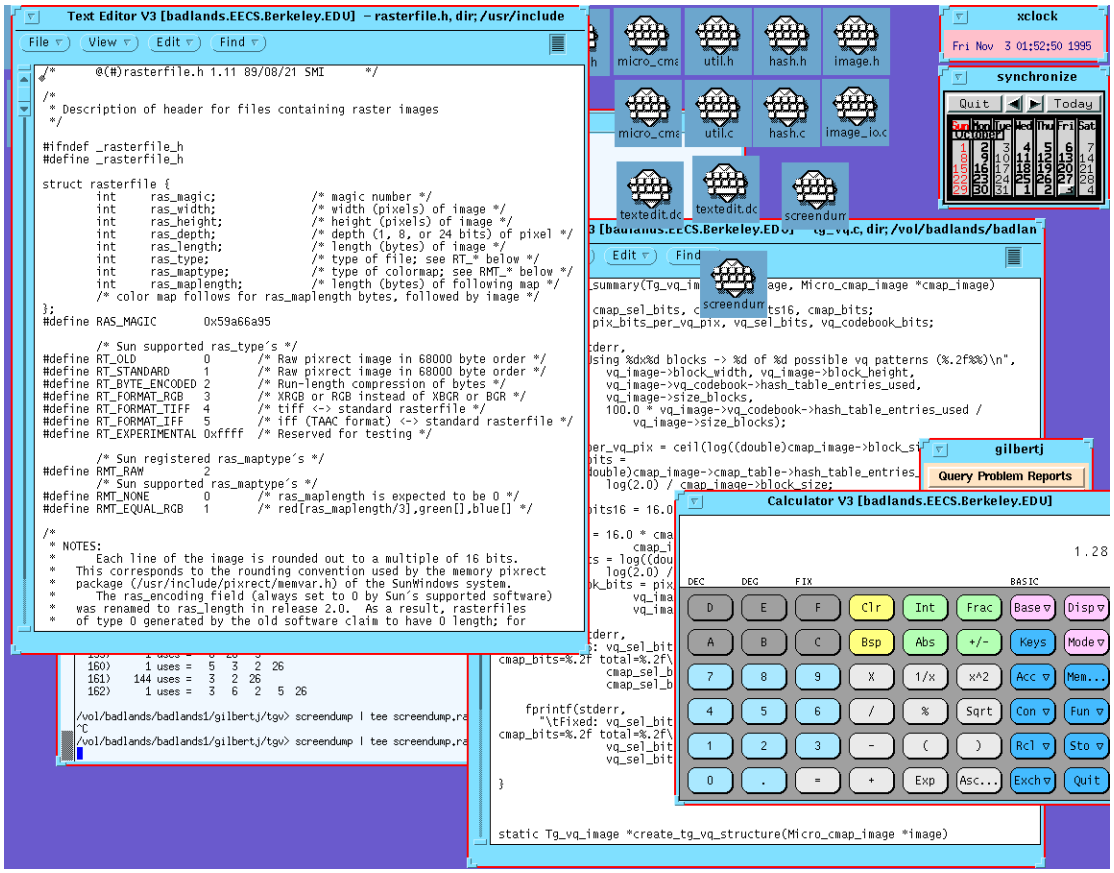


FIGURE 6.1. Typical screen image consisting of multiple graphical applications

6.2.5. Must be Tailored to Typical Screen Contents

As with all image compression techniques, the compression algorithm must be tailored to the types of updates encountered using remote rendering. While the VQ video coding used for the full-motion video display allows for a significant reduction in framebuffer size and bandwidth requirements, and does satisfy the above requirements, its lossy nature and design for continuous-tone images would be inadequate for most applications, as most user-interface components would be rendered unintelligible. A typical screen image is shown in Figure 6.1. This image contains text and graphics as found in many GUI applications.

6.2.6. Decompression Must be Low Complexity / Cost

The second advantage of the compressed framebuffer approach presented was a decrease in power consumption and complexity. For this to be the case, the on-the-fly decompression method must be low-power and low complexity, or else all gains achieved through the compression will be negated.

6.3. Pseudo-Color or Colormapped Display as Compressed Framebuffer

One type of compressed framebuffer that is often used in computer displays is called a pseudo-color or colormapped framebuffer. The term “8-bit color” display typically refers to this. Most personal computer or workstation framebuffers are either colormapped or support a color-mapped display mode as it significantly reduces memory requirements as compared to uncompressed or true-color modes.

Colormapped displays use both a framebuffer and a colormap (or color LookUp Table - LUT). Instead of storing the red, green, and blue values for each pixel, the framebuffer stores an index into the colormap. The colormap is a small (typically 256-entry) array of color descriptors which contain the red, green, and blue value of the colors. In this way, each pixel in the framebuffer only requires 8 bits. Since the colormap contains only 256 entries, it is small as well. If 8 bits are required for the red, green, and blue intensity index then the colormap would be 256*3 = 768 bytes and the main framebuffer would be 1*Width*Height bytes. This is to be contrasted with a display where each pixel has its red, green, and blue values specified, which would require 3*Width*Height bytes. Both storage and bandwidth requirements for the framebuffer are reduced. Since graphics memory often does not reside directly on the processor bus, access to it can be quite costly. The use of colormapped displays can reduce this cost.

The main drawback of the colormapped display method is that the number of colors that can be displayed simultaneously is limited. Applications must then compete for allocation of the colormap entries since it is a shared global resource. A centralized text / graphics server performs this function. Applications must be able to operate even if they cannot reserve all colors that they request, and thus operate with whichever colormap entries are active. Often image display applications dither between colors in the colormap to emulate colors that are not in the colormap.

Colormapped displays have another advantage in addition to reduced memory size and bandwidth requirements. Using colormapped displays, a technique called palette animation can be used to perform smooth animation without the use of auxiliary buffers. Palette animation exploits the fact that multiple pixels on the screen can be changed simultaneously by simply changing one or a few colormap entries. Thus new frames of an animation sequence are written to the screen in such a way that they map to the same set of colors as the old frame until a rapid colormap update occurs and the colors of the new frame are made visible.

6.4. A Compressed Framebuffer Compression Method - TGVQ

This section describes a compression technique which can be applied to text / graphics data, typically yielding less than a bit per pixel storage requirement, and satisfies all of the requirements outlined in Section 6.2. The text / graphics compression technique is also well suited for integration with a lossy continuous tone image compression technique as shown in Section 6.4.5.3. The text / graphics compression algorithm is based on hierarchical vector quantization and is thus designated TGVQ.

6.4.1. Local vs. Global Color Diversity

Typical text / graphics images can contain high *global color diversity* but almost always contain low *local color diversity.* Color diversity is the number of colors present in a region. Thus

global color diversity is the total number of colors present in the entire image while local color diversity is the number of colors present in a small local region of the image. Globally an image might use many colors, particularly if there are continuous-tone regions in the image. However, in any small discrete-tone region in the image, only a few colors are used repeatedly.

The conventional colormapped framebuffer technique described in Section 6.3. relies on the assumption that the global color diversity of an image can be limited to 256 colors without causing significant visual degradation. However, limiting globally to 256 colors does impose restrictions on the images - which can result in some degradation particularly for the display of continuous-tone images. Additionally, limiting to 256 colors still results in 8 bits per pixel which is too high for many applications as described in Section 5.1. Exploiting limited local diversity, however, can result in even greater savings, as will be demonstrated.

6.4.2. Micro-Colormaps

A new technique called *micro-colormaps* exploits limited local diversity by assigning individual colormaps to small blocks within the framebuffer as shown in Figure 6.2. In this way, since the number of unique colors in each block is typically quite small, only a few bits per pixel are required. For instance, if only four unique colors are used in an 8x8 block, only two bits per pixel are required for the pixels in the block. The colors used in the block are listed in the block's micro-colormap while the arrangement of the colors, called the *pattern*, is stored in the main framebuffer portion. Each block thus needs an indication of which micro-colormap it uses, as well as the number of bits per pixel (though these two are related).

6.4.3. Vector Quantization of Micro-Colormaps and Patterns

While the above exploits limited local color diversity to effect some compression, by exploiting spatial locality and redundancy via vector quantization, further coding gains can be

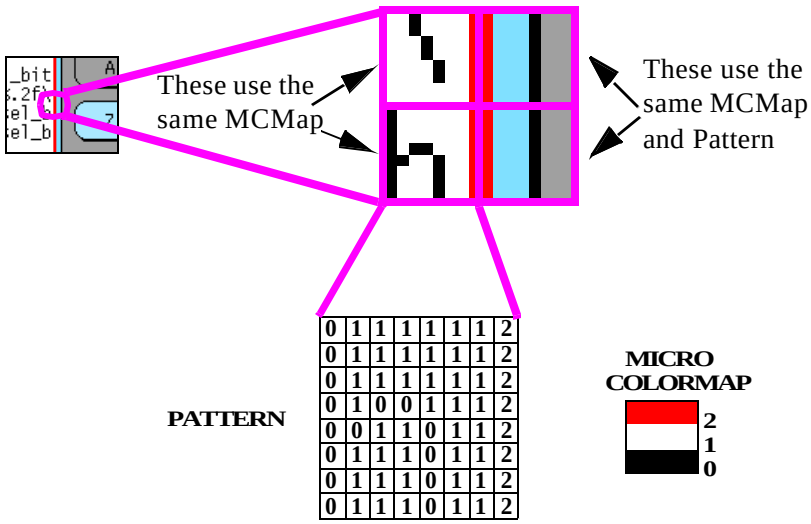
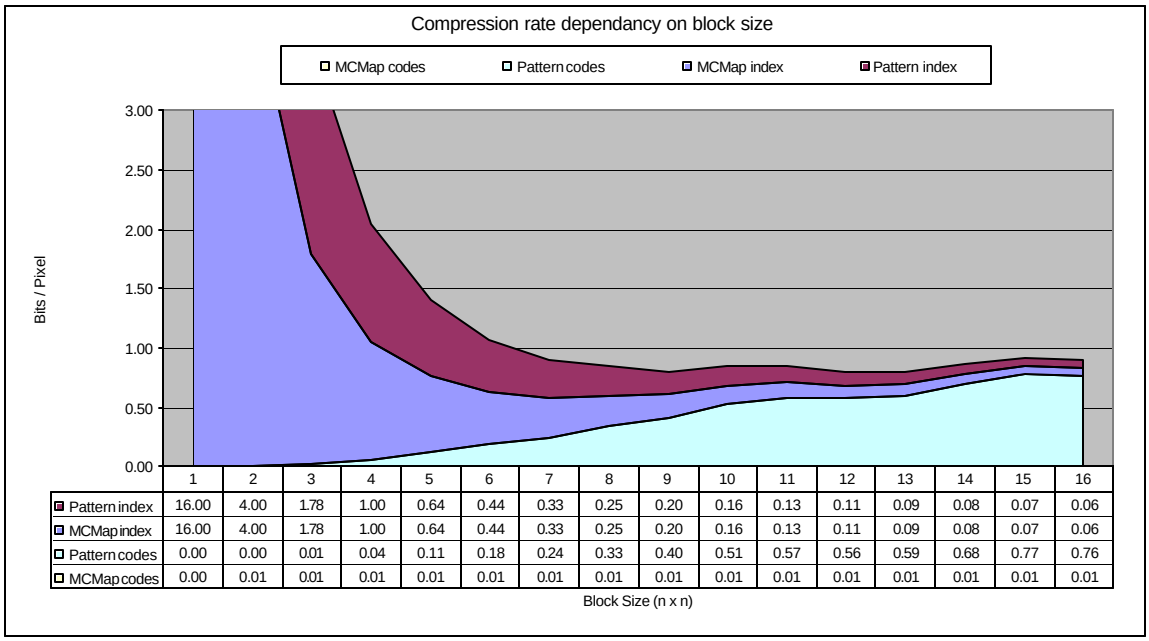


FIGURE 6.2. Block decomposition into pattern and micro-colormap (MCMap). The two blocks on the right share the same micro-colormap and pattern while the two on the left share only the same micro-colormap. The pattern and micro-colormap for the lower left block is shown.

achieved. The key observation is that both the same micro-colormaps and patterns are typically used across the image multiple times; the same sets of colors typically find themselves used together in multiple blocks, and often in the same configuration.

Vector quantization entails storing micro-colormaps and pattern blocks in two tables or codebooks, and having the blocks in the image contain indices that refer to the entries in the two codebooks. In this way, multiple blocks which use the same micro-colormap can share the memory required to store the colors, and blocks which use the same pattern entry can share that memory. This vector quantization is a lossless process as the micro-colormap and patterns are stored exactly. Thus conceptually, the framebuffer consists of an array of W/BlockSize by H/BlockSize pattern and micro-colormap pointers as well as micro-colormap and pattern codebooks of fixed total sizes. The amount of memory required for each entry will depend on the number of colors in the micro-colormap the bit depth used in the pattern. The amount of memory dedicated for the codebooks is determined by examining the amount of memory required to render typical images and then adding some “slack” factor. In this way, typically some part of the codebook will be “in



	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
Pattern index	16.00	4.00	1.78	1.00	0.64	0.44	0.33	0.25	0.20	0.16	0.13	0.11	0.09	0.08	0.07	0.06
MCMap index	16.00	4.00	1.78	1.00	0.64	0.44	0.33	0.25	0.20	0.16	0.13	0.11	0.09	0.08	0.07	0.06
Pattern codes	0.00	0.00	0.01	0.04	0.11	0.18	0.24	0.33	0.40	0.51	0.57	0.56	0.59	0.68	0.77	0.76
MCMap codes	0.00	0.01	0.01	0.01	0.01	0.01	0.01	0.01	0.01	0.01	0.01	0.01	0.01	0.01	0.01	0.01

FIGURE 6.3. Compression rate dependence on block size.
This graph shows the dependence of the various components of the coding rate on the chosen block size. 16-bit indices and 18-bit colors are assumed.

use" and the rest will be "free" - as is required for in-place modification as described next. The micro-colormap and pattern codebooks can be kept in the same memory or different memory banks. The former allows for reduced total memory requirements while the later can slightly reduce the size of indexes required to specify codebook entries. The rest of this section assumes that a single unified codebook is used.

6.4.4. Determining Block Size

The choice of block size will clearly effect the compression rate. The stacked chart in Figure 6.3 shows the effect of varying the block size on the compression rate for the sample image in Figure 6.1. The chart shows the contributions of the four components: the pattern index, the micro-colormap index, the pattern codes, and the micro-colormap codes. This assumes 16-bit indices and 18-bit color. The pattern and micro-colormap indices each take 16 bits per block in order to be able to address a large enough codebook. Thus with a block size of 1x1 pixel, the overhead is 16 bits per pixel but as the block size increases, the contribution of the indices decreases

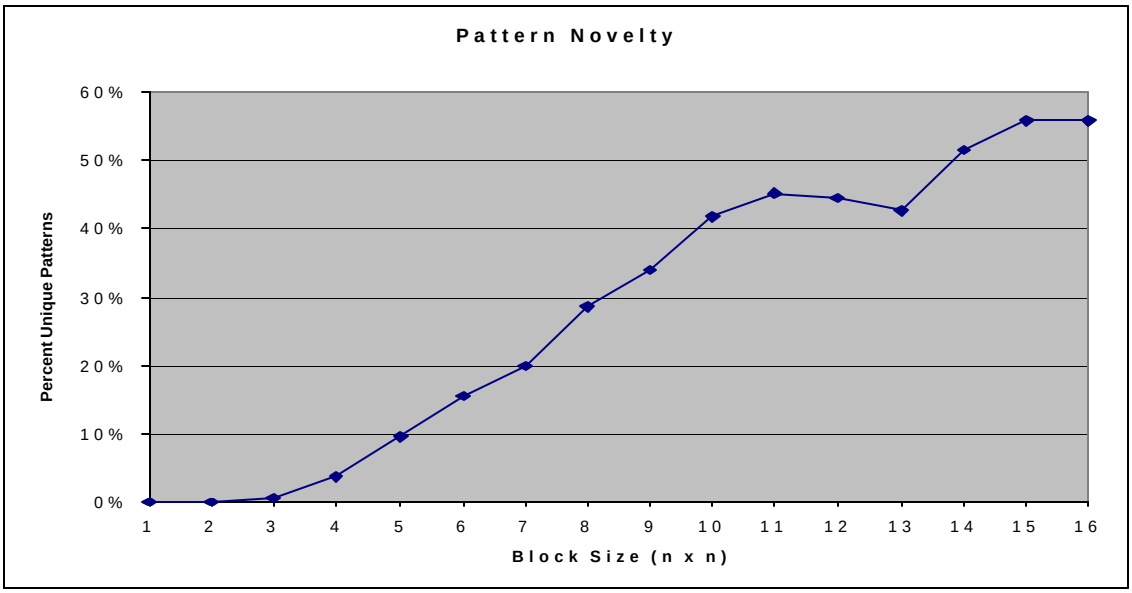


FIGURE 6.4. Pattern code novelty and reuse versus block size. The graph shows the ratio of unique patterns to total number of blocks.

until it is only 1/16 bit per pixel per index at a block size of 16x16 pixels. The micro-colormap data requires very little storage since a few micro-colormaps are reused many times. The storage requirements are typically less than 1/100 bit per pixel.

The storage requirements of the pattern bits increase with increased block size. This is because as the blocks grow in size, the average number of times that a particular pattern is seen in the image decreases since the block "uniqueness" increases. Thus the gains of the vector quantization decrease. Figure 6.4 shows the reduction in pattern code reuse, or the increase in code novelty as the block size is increased for the image in Figure 6.1. The net combination of the four components levels off at a block size of about 7x7 or 8x8 and stays fairly constant at a rate of 0.8 to 0.9 bits per pixel for the sample image.

6.4.5. Requirement Satisfaction

This section describes how the requirements described in Section 6.2. are all satisfied by the compression scheme described above.

6.4.5.1. In-Place Modification of Compressed Data

The compression system allows in-place modification as described in Section 6.2.1. In-place modification works as follows: Local copies of the codebook and index framebuffers are kept by the text / graphics server. While the index framebuffers are spatially mapped such that the locations always correspond to a particular region on the screen, the codebook is only mapped based on which indices point to it. Thus entries in the codebook are assigned dynamically in a manner similar to dynamic heap memory allocation. Since multiple indices can point to the same locations in the codebook, reference counts are required to determine when particular regions of memory are no longer in use. The reference counts are incremented when an entry is referred to by a new block, and decremented when that block no longer refers to it. Thus the sum of all reference counts is the number of blocks in the image.

When a section of the screen is modified, the colors in each block are examined to determine the unordered sets of colors constituting the micro-colormaps. If a micro-colormap with the required colors for a given block is already present, it can be reused. In this case, the new block's micro-colormap index is set to the index of the existing micro-colormap. Otherwise, a new micro-colormap entry is "allocated" in the codebook, its contents are sent to the remote terminal and then the new block's micro-colormap index is set to the index of the new micro-colormap entry. In either case, the reference counts are updated as described above.

6.4.5.2. Update Independence for Error Tolerance

If a transport protocol incorporating explicit dependencies is used as described in Section 13.1.2., the dependencies between an index update and the preceding codebook update used to set up the codebook entry should be noted. Thus an index update will not be effected by the terminal until the codebook update it required has taken place. Additionally, a codebook update is dependent upon the index that was pointing to it being updated. Thus if this is not explicitly coded, it is

advantageous to reuse codebook entries in a LRU (least recently used) manner. This also promotes reuse of codebook entries if an entry is briefly not used but then later reused.

6.4.5.3. Must Work for Text / Graphics and Image / Video

While the algorithm, as described, is lossless, and thus could not result in guaranteed compression rates, with the addition of a video coding, this can be achieved. The text / graphics vs. image / video decision can be made in a manner closely related to the text / graphics coding method in a way such that those regions that would not compress well with the text / graphics method would be selected as image / video.

Each block can classified as text / graphics vs. image / video by using the local color diversity used to determine the micro-colormaps. By simply counting the number of colors in a given block, and applying some continuity constraints, an effective text / graphics vs. image / video decision can be made. Figure 6.5 plots the local color diversity of a typical image with both text / graphics and image / video regions using 8x8 pixel blocks. Blocks with more than 4 colors are classified as “image / video” while those with less than or equal to 4 unique colors are classified as “text / graphics”. In this way, those blocks with high color diversity, which would require large and often unique micro-colormaps and patterns, will be coded using a lossy coding. Continuity constraints prevent small high-diversity patches from being interpreted as image / video and small low-diversity patches from being interpreted as text / graphics.

Figure 6.6 shows an example of automatic text / graphics and image / video merging based on color diversity. The top image is a section of the original image from Figure 6.5 while the bottom image has the blocks with color diversity greater than 4 replaced with a crude image coded version. The image coding entailed conversion to YUV space, decimating Y by 3x3 and U and V by 6x6. The three intensities were then coded at 6 bits per sample for a net coding rate of 1 bit per

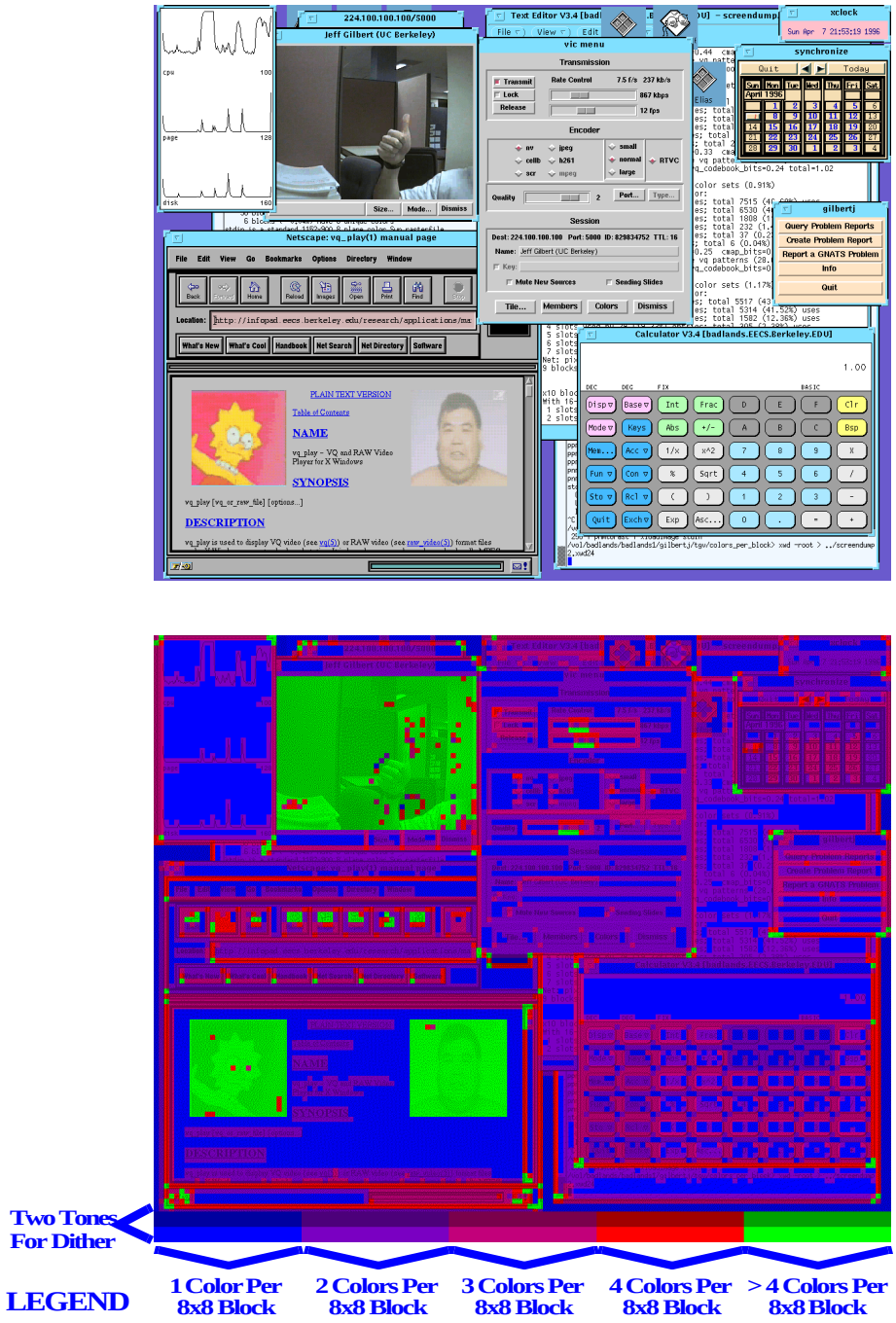


FIGURE 6.5. Using local color diversity to make text / graphics vs. video decision. The original image, shown above, is primarily text / graphics with three image/video regions. The local color diversity using 8x8 pixel blocks is shown below. Blocks with more than 4 colors would be classified as image while those with less than or equal to 4 colors would be classified as text / graphics. Continuity constraints would prevent small high-diversity patches from being interpreted as image/video and small low-diversity patches from being interpreted as text / graphics.

pixel. As can be seen, even this crude coding does not introduce excessive image degradation since it is only occurring in continuous-tone regions. Note that continuity constraints are not imposed so some isolated text / graphics regions, such as on the web browser buttons, are detected as being image regions and are coded using the lossy coding. Using a more sophisticated coding such as a DCT-based approach used in JPEG [40] would result in higher image quality and/or greater compression rate.

6.4.5.4. Must Work for All Possible Screen Configurations

While the lossless text / graphics coding described above cannot bound compression, when coupled with the lossy image / video coding, the overall system memory requirements can be bounded. Thus if the memory usage of the lossless portions ever exceeds the allowable amounts, blocks can be converted to the lossy format which will always fit. For example, using 8x8 blocks on typical text / graphics screens as described next, the lossless coding codes at less than 1 bit per pixel. The crude lossy image coding previously described codes at 1 bit per pixel. Thus if 2 bits per pixel of memory were available then all possible screen configurations could be stored.

The flexibility of TGVQ method allows detail to particular regions to be tailored as desired. In particular if small regions of the screen require high fidelity, they can be losslessly coded regardless of whether they are text / graphics or image / video.

6.4.5.5. Must be Tailored to Typical Screen Contents

Figures 6.7 and 6.8 show typical images and their compression rates. In the case of both screendumps in Figure 6.7 as well as the screendump without the continuous-tone images in Figure 6.8, the compression algorithm results in coding less than one bit per pixel. Even with some continuous-tone regions, as in the top image of Figure 6.8. The compression rate is still under 2 bits per pixel. If a separate lossy coding for continuous-tone regions is used, as described in

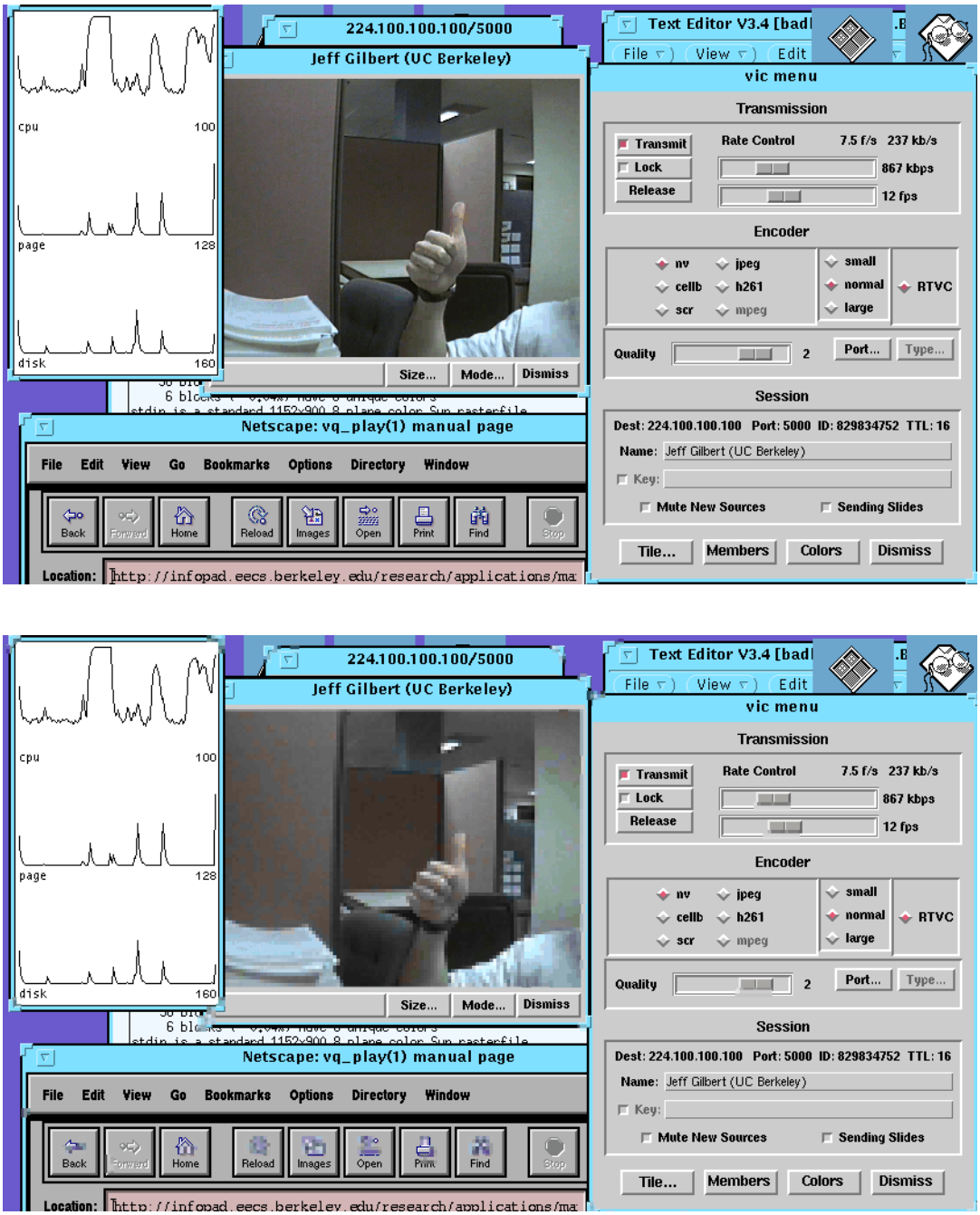


FIGURE 6.6. Automatic text / graphics and image / video merging using color diversity. The top image is a section of the original image from Figure 6.5. The bottom image has the blocks with color diversity greater than 4 replaced with a crude image coded version. The image coding entailed conversion to YUV space, decimating Y by 3x3 and U and V by 6x6. The three intensities are then coded at 6 bits per sample for a net coding rate of 1 bit per pixel.

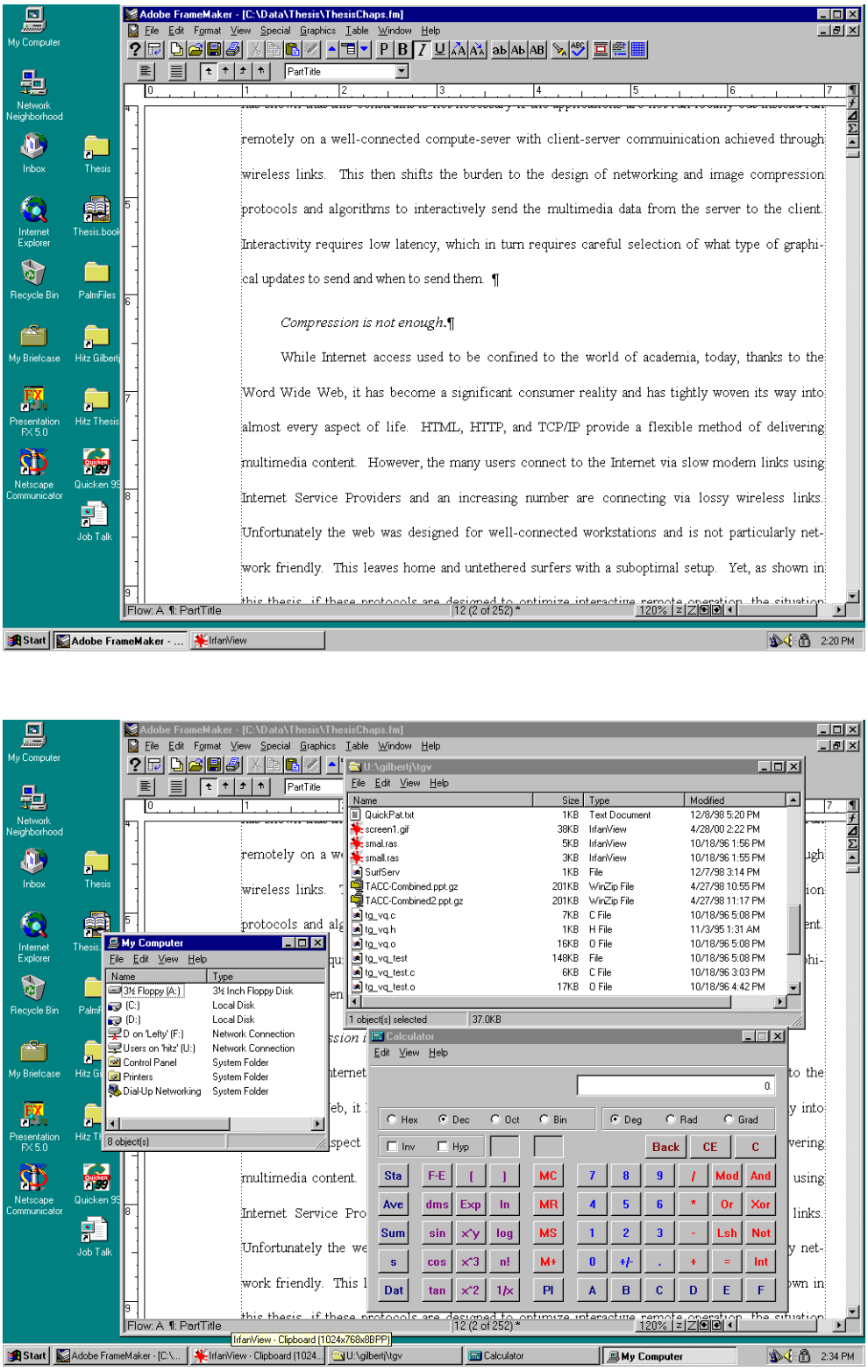


FIGURE 6.7. Two typical images compressed with TGVQ. The top image requires 0.84 bits/pixel while the bottom requires 0.94 bits/pixel.

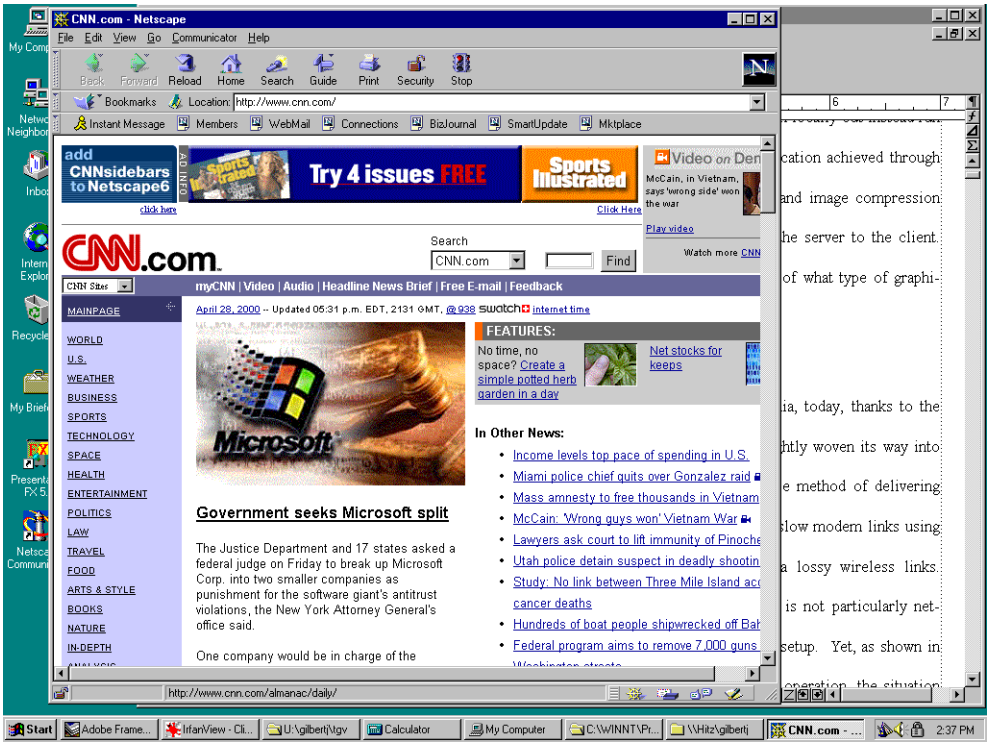


FIGURE 6.8. **A typical screendump with and without some continuous-tone regions. The screendump with continuous-tone regions (top) requires 1.7 bits / pixel while the one without (bottom) requires only 0.93 bits/pixel. In a complete implementation, the continuous-tone regions would be coded using a lossy method.**

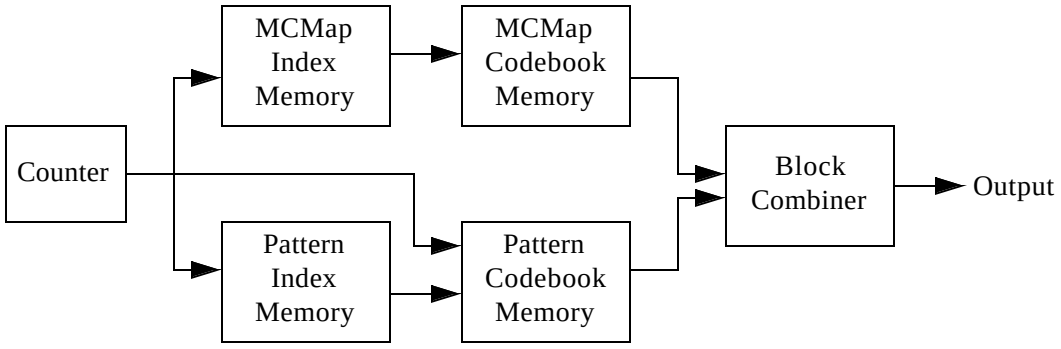


FIGURE 6.9. **Rough architecture of compressed framebuffer TGVQ decoder. Note that any number of the memories shown could be combined into single unified memory.**

Section6.4.5.3., the compression rate would be further reduced and bounded. Thus if the framebuffer is allocated 2 bits/pixel then typical images can be fully losslessly encoded.

6.4.5.6. Decompression must be Low Complexity / Cost

The TGVQ method is readily implementable in hardware. Its implementation would bear many similarities to the VQ Video described in [15] and referred to previously in Section 5.2.. Figure 6.9 shows a rough architecture of the compressed framebuffer system and some of the principles of operation are briefly described here. Also note that while multiple separate memories are show, they could be combined into a single unified memory to promote reuse between the two codebook arrays, but at the possible expense of power consumption due to tighter access time requirements.

A counter cycles through the two index memories in order to retrieve the indices for the blocks as the image is scanned. These indices are used to select codebook entries from the two codebooks. The counter output is also distributed to the pattern codebook memory so that it can produce the correct line within the pattern. A block combiner indexes the pixels from the pattern memory into the MCMap to produce the rendered pixels. It could also incorporate a lossy image

coding technique for video as previously described. Since the system works “open-loop”, latency is not problematic, and this can be used to reduce power consumption though pipelining.

CHAPTER 7 *Hybrid Approach*

7.1. Motivation

In Chapter 3, the primitive-based approach is presented which allows for good bandwidth utilization, but can result in high latency due to loss or queuing delays. In Chapter 4, Chapter 5, and Chapter 6, bitmap-based approaches are discussed which significantly reduce latencies, but at the expense of less efficient bandwidth utilization. This then leads to the question of whether a hybrid approach can yield the benefits of both the primitive and bitmap approaches as shown in the table below:

	Bandwidth Utilization	**Perceived Latency**	**Client Complexity**
Conventional Primitive	Good	Bad	Ok
Conventional Bitmap	Bad	Bad	Good
Improved Bitmap	Ok	Good	Good
Hybrid Approach	? Good ?	? Good ?	? Ok ?

This chapter presents some ideas and directions on the implementation of such a hybrid approach.

7.2. Approach

In order to obtain the best of both the bitmap and primitive approaches, a hybrid scheme is employed. The primitive approach obtains its bandwidth efficiency by retaining and transmitting the drawing requests in the compact primitive form, while the bitmap approaches deliver reduced latencies by reducing false-dependencies and eliminating updates that are superseded before they can be transmitted. The two key concepts required to combine these are *primitive dependency tracking* and *primitive squashing*.

The architecture proposed is the virtual framebuffer architecture presented in Section 4.3. with the modification that the virtual framebuffer is not a bitmap-based buffer but rather a primitive-based framebuffer. While the primitive approach simply queues the drawing requests in a single linear list, the hybrid approach explicitly notes dependencies by arranging the queued primitives in a set of directed acyclic graph (DAG) structures called the *pending primitive graph*. The primitive framebuffer stores the primitives that have been requested by an application but have not yet been sent and acknowledged from the remote terminal. Each primitive also has a flag indicating whether it has been yet transmitted to, but not acknowledged by, the remote terminal. Other information such as the time of transmission and graphics context information may be recorded. A secondary bitmap-based buffer is used to satisfy application image queries. The bitmap buffer contains the current rendered contents of the screen as with the bitmap-based virtual framebuffer architecture.

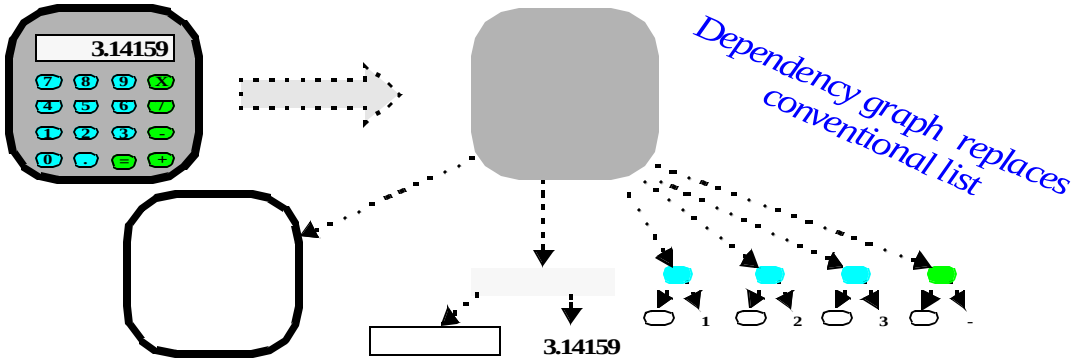


FIGURE 7.1. **The hybrid approach: pending primitive graph. Primitives that have been queued for transmission to the remote client, but not transmitted and acknowledged, are arranged in a set of directed acyclic graphs with links explicitly denoting dependencies and overlap.**

7.3. Master Operation

When a new drawing primitive is received from an application, the master places it as a child of all drawing primitives that must be rendered before it. This would include any previous primitive which geometrically overlapped the primitive in question. Figure 7.1 shows an example of a calculator image and its dependency graph. The calculator is comprised of a solid background, a surrounding border, a number area and a set of keys. Each key consists of a solid background, a border, and a label. The keys do not overlap with each other or with the number area.

The master also renders the primitives to the bitmap buffer so that later application queries for the current contents of the screen can be satisfied locally. The existence of a fully rendered bitmap buffer also allows sessions to be suspended and resumed.

7.3.1. Primitive Squashing

While the bitmap-based approach automatically replaces old updates with new ones via Adaptive Bandwidth Compression as described in Section 4.3.2.1., the hybrid approach, via the master, must do this manually. When a new primitive is added to the pending primitive graph, it is placed as a child of all previous primitives which overlap it. Before placing the new primitive, the

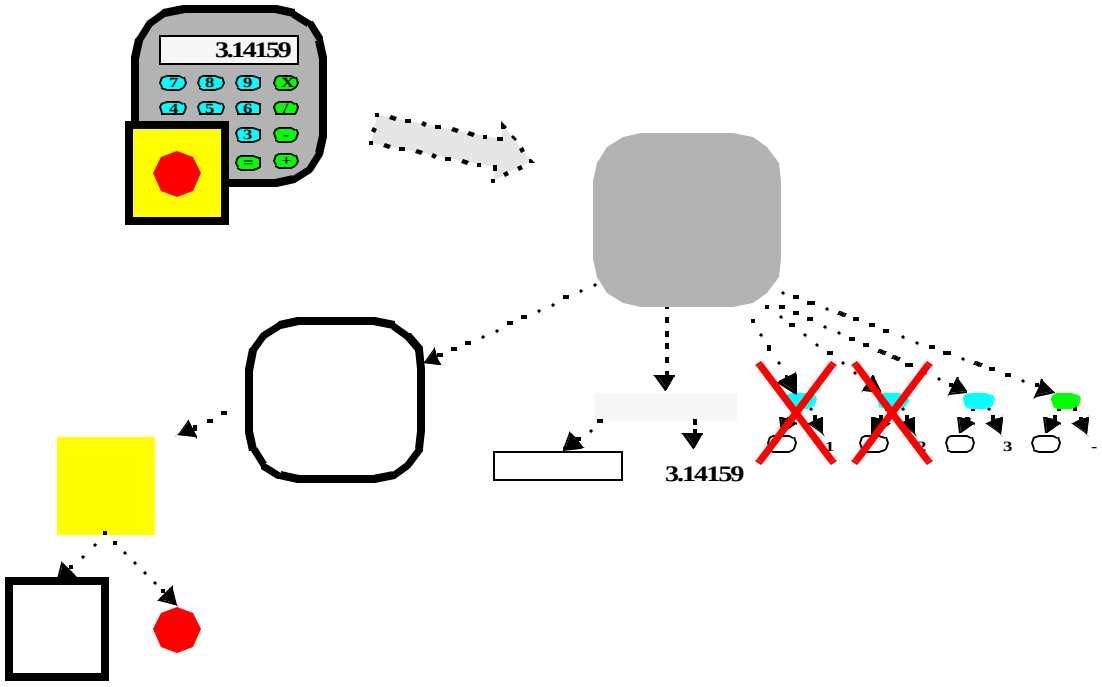


FIGURE 7.2. **Primitive *squashing*: removal of unneeded primitives. When a new primitive obscures previous primitives that have not been sent to the client, the previous primitives are removed since their effect has been nullified. The above shows what happens when a new object (the square/circle combo) is drawn, obscuring some old primitives.**

graph is examined and if the new primitive completely obscures any existing primitives, they are removed. If they have been sent already, but not acknowledged, they are forgotten. If they have not been sent, they will not be sent. In this way, redundant primitives will not consume valuable link bandwidth. The process of removing stale primitives is called *primitive squashing*.

Only primitives whose removal would not alter the final display rendering can be removed. In particular, if a non-opaque primitive such as XOR area completely covers an earlier pending primitive such as a line, box, or text, that earlier pending primitive cannot be removed as it would change the final display. However, if this is then completely covered with a later opaque primitive, the earlier opaque and non-opaque primitives can be removed since they will no longer effect the final display.

7.3.2. Dense Primitive Rendering

The virtual framebuffer architecture decouples the application from the remote terminal and protocols used to communicate with the remote terminal. Thus both operations on single primitives and multiple primitives can be effected. Since the primitives are queued in the pending primitive graph until being sent, during heavy usage, the pending primitive graph could contain many pending primitives.

While primitives are typically a more compact representation than bitmaps, if an application draws very fine detail, or uses many spatially small primitives, a bitmap representation of a given high-detail area may be more compact than the pending primitives specified by the application. This is particularly possible if bitmap compression as previously described is used. To this end, the hybrid approach can dynamically convert pending primitives to a pending bitmap update. Since the master knows the exact screen contents, it can create a bitmap write primitive that specifies the current contents of a particular region, and use it to squash all underlying primitive updates specified by the application. If later primitives are queued before the bitmap update is delivered, the new primitives are rendered into the virtual bitmap framebuffer and the updated image supersedes the previous one. This can be used to bound bandwidth requirements for high-detail images. Furthermore, progressive image delivery techniques as described below can be used.

7.3.3. Representing Region Copies

Bit Block Transfers (called BitBlt or Bit Blits) which cause a source region of the screen to be copied to a destination region of the screen, need to be noted both at their source and destination. For the destination region, they appear like any other primitive and can be opaque if they are a direct copy, or non-opaque if they are to be combined with a logical operation such as AND or XOR. BitBlts must also be noted at the source as a dependency since the block copy must be executed after all queued primitives in the region it references but before any subsequent primitives

that may further modify the region. This is because the application issuing the copy request assumes that its primitives will be executed in order and thus all primitives requested before the copy will have completed but none of the primitives requested after the copy will have begun. Thus a special copy link is needed which notes both source and destination.

BitBlts prevent squashing of the source region across the source reference in order to assure that the copied state referenced is rendered. However, the destination region can cause the copy to be squashed just as other primitives are squashed. If the destination region warrants squashing then the BitBlt primitive is removed and its reference to the source is removed. This could then allow further squashing if it was prevented by the existence of the source reference.

7.4. Slave Operation

The slave scans the primitive framebuffer as before, and sends any primitives that are not children of (dependent on) any other queued, but not sent, primitives. The ordering of this scan is flexible as will be discussed below. Once the primitives are sent, they are marked as such and then any of their children can be sent. The dependencies are also transmitted with the primitives such that if any primitive is lost, only those later primitives which depend on the lost primitive are not rendered until the lost primitive is retransmitted successfully. However, the rendering of independent primitives need not be delayed.

The primitives are coded in a format that is appropriate for transmission to the remote client. Compression and error-corrective coding can be used to trade off error-tolerance, bandwidth usage, and client computational requirements.

A particular primitive is removed as soon as it and all primitives it dependends on are acknowledged. When an acknowledgment for a particular primitive is received from the remote

terminal, that primitive is marked as acknowledged, and if it is not the dependent of any unacknowledged primitives, it is removed from the pending primitive graph. Additionally, if it had any dependents which were acknowledged but not removed because the parent was not yet acknowledged, they are removed as well.

7.4.1. Progressive Image Transmission

While the image display requests issued by the application are queued as single primitives, they do not have to be sent to the remote terminal as such. Multiple-pass hierarchical transmission is often quite useful for bandlimited or lossy links since it allows the user to quickly get a coarse idea of what is on the screen. If transcoding occurs in the text / graphics server, applications need not be designed for operation over a slow link. Often applications, such as Adobe FrameMaker ®, use image primitives for text rendering in order to retain full control over typesetting and font style. These applications are difficult to operate remotely over bandlimited links if image transmission is not efficiently handled. Additionally, standard video player or video conferencing applications can be used remotely if the image transmission to the remote terminal is performed in a bandwidth-conscious manner.

Progressive image transmission can be effected by having the slave, or some independent lower-priority thread, transcode and compress pending images into progressive formats. For continuous-tone images, spatial-frequency decomposition such as that used in progressive JPEG and wavelets could be used. For discrete-tone images, interlacing similar to interlaced GIF and PNG would be more appropriate and yield superior image quality at a given bit-rate. The choice of whether to use progressive images and which type to use could be made based on the server CPU load, client capabilities, link bandwidth availability, and type of image.

Once the images have been transcoded, each pending primitive has associated with it a set of flags indicating which layers have been sent. The layers are sent independently and the remote terminal acknowledges the receipt of particular image layers, and not just the entire image primitive. The images can be divided further spatially such that particular parts of particular layers could be independently sent and acknowledged. This is particularly useful in conjunction with the cursor-targeted updates via primitive reordering described in the next section.

The various layers are prioritized differently with respect to other images and other drawing primitives. Typically higher detail-level layers would only be sent after all other primitives have been transmitted, as they are only needed for final image quality. In this way, if an animation or movie is playing, other regions of the screen such as the player's GUI or other applications will not experience excessive delay in updating. The high-detail layers typically consume the greatest number of bytes yet deliver the smallest delta in image utility. Lottery scheduling [69] could be used to assure that high resolution images are not delayed indefinitely in the presence of other continuous activity.

Care must be taken if other primitives depend on the transcoded image since the other primitives cannot be rendered until all layers in the image are rendered. Alternatively, the dependent primitives can be re-rendered after each successive layer of the image is rendered. Also, if the image is later the source of a BitBlt, the image must be fully rendered before the BitBlt can proceed.

7.4.2. Primitive Reordering

As previously mentioned, using the virtual framebuffer architecture, the order in which primitives are sent from the text / graphics server to the remote terminal is independent of the order

that the applications send the primitives to the text / graphics server. This can help to reduce the net latency due to loss as well as the delay for the user to see the data they are interested in.

The latency due to loss can be reduced by sending primitives in an ordering that as few primitives as possible are dependent on other in-flight primitives. If a long stream of dependent primitives are transmitted and one of the earlier primitives is lost, the subsequent primitives must be delayed until the lost primitive is successfully retransmitted. However, if multiple independent streams are transmitted then losses will only delay the update of smaller regions of the screen. While dependencies cannot be removed, by sending primitives in a "breadth-first" manner as opposed to a "depth-first" manner, the number of outstanding dependencies can be reduced and improved performance in a lossy environment will result.

Additionally, the updates can be targeted such that regions of user-interest receive greater bandwidth. One way to infer user-interest is to assume that the cursor area is of higher priority and prioritize updates to that region before updates to other regions. In this way, even a low-bandwidth link supporting a complex display can retain interactive operation of a graphical user interface since typically the most responsiveness is required around the location of the cursor. A similar concept is applied to World Wide Web transmission described in Section 10.5.4.

7.5. Benefits / Conclusions

Thus it has been demonstrated that the hybrid approach combines the best of both the bitmap and primitive approaches by simultaneously reducing bandwidth requirements as well as reducing latency due to queuing and loss. By further separating the applications from the communications link, generic applications can be used in a bandwidth and loss aware manner. Progressive image techniques and information reordering allow limited resources to be directed at the goals of the user.

CHAPTER 8 *Text / Graphics Image Compression*

8.1. Introduction

In the past chapters, application-independent text / graphics and image transmission architectures have been proposed. While these architectures varied in bandwidth, latency, and client complexity requirements, all of the techniques transmitted images at least some of the time and could thus benefit from image compression techniques. The primitive-based approach of Chapter 3 used image transmission whenever the application chose to send images. The bitmap-based approaches of Chapter 4 and Chapter 5 transmitted images for all updates. The compressed-framebuffer approach of Chapter 6 could use additional image compression to reduce bandwidth requirements beyond the compression afforded by the compressed framebuffer algorithm. Finally, the hybrid approach of Chapter 7 used image transmission whenever the application chose to send images, as well as when many dense primitives were drawn.

This chapter gives some background on image compression techniques suited for transmission of text / graphics images and proposes a new algorithm specifically tailored to this class of images. It will be shown that dictionary-based image compression techniques determine and

exploit redundancy in images by decomposing the input image into repeated sequences and coding them as such. Conventional approaches such as Graphical Interchange Format (GIF) and Portable Network Graphics (PNG) are restricted to 1-dimensional repeating patterns. The technique described in this chapter, Flexible Automatic Block Decomposition (FABD), performs two-dimensional block decomposition to exploit arbitrarily-sized rectangular repeating blocks. Several optimizations are used to reduce the computation required for the block matching to approximately the same as traditional one-dimensional techniques. Employing simple entropy coding techniques to the compression of typical text / graphics images, a coding rate of 0.03 - 0.20 bpp can be achieved. This is 1.5 to 5.5 times more compact than GIF and up to 3.8 times more compact than PNG. Decompression is fast and simple, as is required in a web browsing or remote portable terminal environment [32].

8.2. Image Coding Overview

The basis of all lossless image compression techniques is the detection and exploitation of redundancy in the image to be compressed. The detection typically involves predicting parts of the image yet to be coded from those that have been previously coded and general knowledge about the class of images being compressed. The exploitation of the redundancy is effected by sending only the novel aspects of the data so that the more “predictable” the image by a given algorithm, the greater the achievable compression.

For instance, if it was known that a synthetic input image always consisted of a discrete set of squares of varying size, location, and color, then this could be exploited by coding the image as a few parameters, namely the number, size, location, and color of the squares and the color of background. Thus the size of the compressed image would be independent of the number of pixels in the image and the compression could be quite substantial.

However, if the image coder only expects the input images to consist of solid horizontal lines, the “square” images would have to be coded as many horizontal lines. While this would typically be more efficient than coding each pixel individually, it would not be as efficient as coding as squares since many horizontally lines would be required for each square. Thus as compression algorithms contain more information about an image class, they can compress the images more effectively.

8.2.1. Discrete-Tone Images

Discrete-tone images are those in which the pixel intensities do not vary smoothly, as in a photograph, but rather assume a small discrete set of values. Discrete-tone color images are computer generated and include screendumps, diagrams, and renderings of text - the types of images used in the transmissions systems described in this thesis.[1] These synthetic images typically exhibit significant redundancy in that large areas of the image are solid or consist of lines or shapes which can be predicted from other places in the image. Solid regions can be specified compactly as in the square example above, and text and symbols from one area in the image can be predicted from those in another area since identical patterns of pixels will appear for the same letters and words.

8.3. Previous Research / Existing Standards

Two existing approaches to the image compression problem are one-dimensional dictionary-based techniques (used in algorithms such as JBIG) and two-dimensional statistical techniques (used in algorithms such as GIF and PNG). Each exploits certain aspects of the input images and has strengths and weaknesses.

1. Continuous-tone images can be quantized or dithered to a discrete set of tones but will still exhibit characteristics of continuous-tone images. Scanned bi-level images are similar to dithered continuous tone images in many ways.

8.3.1. One-Dimensional Dictionary-Based Techniques

Dictionary-based image compression techniques find repeating sequences in images by creating a “dictionary” of common strings and then coding the sequences by their index into the dictionary. The image is considered a single, albeit long, sequence.

The Lempel Ziv Welch (LZW) data compression algorithm maintains an explicit dictionary of recently used strings. The dictionary initially contains only the single symbol sequences for all symbols. The coding proceeds by finding the longest sequence in the dictionary that matches the next symbols to be coded. The dictionary is grown by adding a new sequence consisting of the old sequence with the addition of the symbol that follows it (as determined by the decoder once the next sequence is received). In this way, the dictionary has a 'prefix closed' property whereby the prefix of every sequence in the dictionary is also in the dictionary. Thus the encoder and decoder can both build the same dictionary automatically without it being explicitly sent. The dictionary can be reset or frozen by the compressor. Compuserve’s Graphical Interchange Format (GIF) uses the LZW compression algorithm on the pixel values in the image in standard left to right, top to bottom raster-scan order.

Since GIF uses a 1-dimensional coding, horizontal patterns are effectively compressed but vertical patterns are not; while horizontally adjacent pixels appear consecutively in the scan order, vertically adjacent pixels are separated by large gaps consisting of the rest of the pixels in the line. For instance, a solid blue horizontal line appears as several consecutive blue pixels while a solid blue vertical line would cause several isolated instances of blue pixels, separated by the scan line width. Additionally, a text character would have to represented as many horizontal patterns instead of a single two-dimensional pattern. The method described in this chapter overcomes both of these shortcomings.

The Lempel Ziv 77 (LZ77) data compression algorithm works by considering the data to be encoded as the dictionary. Instead of specifying sequences as indexes into a dictionary, the sequences are specified as parts of the data stream which have already been coded by sending their "length" and "distance". For example, a sequence could be specified as "17 symbols starting 34 symbols from the last symbol coded" where its length would be 17 and its distance would be 34. Additionally, single symbols can be coded in case they are not present in the recent history. Portable Network Graphics (PNG) [59] combines LZ77 with Huffman coding [39] of the length and distance parameters to more compactly code common values. Additionally PNG performs sub-byte pixel packing so that for images with 1, 2, and 4-bit pixels, multiple pixels are joined into one byte before compression. As with GIF / LZW, horizontal patterns are effectively compressed but vertical patterns may not be and would thus suffer from the same problems. PNG performance typically outperforms GIF by 10-30% and additionally has improved progressive display capabilities and patent-free status.

8.3.2. Two-Dimensional Statistical Techniques

Statistical prediction has been effectively used to compress images by using a context around a given pixel to predict its value. When the value is often the same as the predicted value, little additional information must be sent and low coding rates can be achieved.

The Joint Bi-Level Image Processing Group's JBIG codes pixels in a bi-level image using a 10 or 12 bit context and arithmetic coding [4,43,61]. The neighboring pixels are used to estimate a probability distribution for the current pixel. This distribution dictates the codes to be used for 0 and 1 pixel values. Using arithmetic coding, codes can be fractions of a bit. If a particular pixel is predicted as being more likely to be a 0, the 0 code will be shorter than one bit in length while the 1 code will, by necessity be greater than 1 bit. In this way, if the prediction comes true, only a fraction of a bit will be required for the pixel. When the prediction is not correct, a pixel will

require more than one bit. The more skewed the probability, the shorter the “likely” code is and the longer the “unlikely” code is. Additionally, the “likely” code will occur more often and the “unlikely” code will occur less frequently. The combination of these two effects results in fewer total bits being required for coding. JBIG can be applied to grayscale or pseudo-color images using bit-plane decomposition. As will be shown in Section8.7. and Figure 8.12, this can lead to redundancy and poor coding when a similar structure appears across several bit-planes. However, despite this problem, JBIG performs well on both bi-level and color images. Its performance is particularly impressive on scanned and dithered images.

While JBIG uses statistics adapted over the entire image, each pixel must be coded individually. This is to be contrasted with the dictionary based techniques which can code entire sequences of pixels using a single code word. For instance, for a single character to be coded by JBIG, a code for each pixel has to be specified. While each code could be a fraction of a bit in size, this is to be contrasted to 1-dimensional dictionary-based techniques where roughly one code would be required per scanline of the character or a 2-dimensional dictionary-based technique, such as that in this chapter, where one code would be required per entire character or for several characters.

The Piecewise-Constant (PWC) image model [6] extends statistical coding beyond bi-level images. In this model, arithmetic coding is used to code the pixel colors of a palette image by prediction based on neighboring pixel colors. It assumes a model that images consist of small regions of pixels of the same color. The statistical framework is constructed through the use of four per-pixel questions:

- Q1: Is the current pixel’s color identical to that of a specified rectilinear connected neighbor?
- Q2: Is the current pixel’s color identical to that of a specified diagonally connected neighbor?
- Q3: Is the current pixel’s color identical to a guessed value?
- Q4: What is the current pixel’s color?

Using context-based arithmetic coding, the answers to these questions can be coded efficiently. As soon as one question is answered affirmatively for a given pixel, the other questions need not be answered. Q1 and Q2 exploit the fact that pixels are often the same color as neighboring pixels while Q3 exploits the fact that often in a region a small set of pixels are used. In this way a "guess" pool is kept of recently seen pixel colors[1]. Finally if Q1 through Q3 are all answered negatively, another method, such as linear prediction, must be used to answer Q4. Since statistical methods are employed, dithered images can be coded efficiently. However, as with JBIG, global statistics are used but each pixel must be coded individually.

Other approaches to bi-level image coding have focused on the subset of images consisting of primarily typed or printed text [18,38]. Image segmentation into "marks" is used to locate and individually code the characters. The residual is then coded in a lossy or lossless manner. While this allows full two-dimensional matching, this explicit segmentation limits the class of applicable images to those similar printed text. (Non-segmentable regions can be coded using other techniques.) Additionally the size of the segments is typically limited to single characters, reducing potential coding gains compared to using larger regions. The segmentation is required to make the matching computationally feasible by restricting the pattern matching to occur at fixed "mark" boundaries.

8.4. Flexible Automated Block Decomposition

In order to obtain high lossless compression rates, it is necessary to determine and exploit the redundancy found in the input image. While GIF and PNG implicitly assume that the redundancy is one-dimensional, and JBIG and PWC code pixel-by-pixel, assuming some local two-dimensional redundancy as well as global statistical redundancy, it is apparent from looking at

1. This is similar but developed independently from the color age notion introduced in Section8.6.2.1..

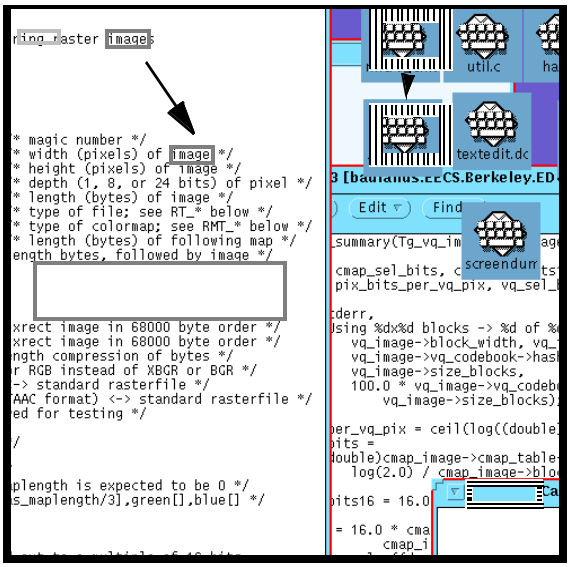


FIGURE 8.1. Typical image and its redundancies.
The vertical lined blocks are copy blocks, the horizontal lined blocks are fill blocks, and the hashed block is a punt.

synthetic images that much block-level global two-dimensional redundancy is present - whole blocks are repeated throughout the image. These blocks may be in the form of text or shapes. Additionally, large solid blocks are a form of redundancy and their efficient coding can aid compression.

The FABD algorithm, described in this chapter, decomposes an input image into two-dimensional blocks by scanning the image from left to right, top to bottom and dividing the image into a set of three types of blocks:

1. Copied blocks
2. Solid fill blocks
3. Punts

Figure 8.1 shows a typical image with the three types of blocks. Copied blocks (shown in with vertical lines) are regions of the image which appear verbatim before the current location. No restrictions are placed on the size or location of the source and destination blocks except that the start of the source block must appear before (above or on the same line and to the left of) the start

of the destination block. Solid blocks (shown with horizontal lines) are regions in the image which consist of a single color. Finally, punts (show with hashed box) are the areas in the image which do not fall into either of the first two categories. This decomposition can lead to efficient coding if the blocks are large such that a small number of blocks are required to represent an image. By parameterizing the image in terms of these three types of blocks, efficient entropy coding is possible.

Decomposition proceeds in a greedy manner from the top-left of the image to the bottom-right until all pixels have been accounted for. The area currently being classified is called the *destination region*. For copies, the location being copied from is called the *source region*. The destination region is increased in size until it no longer is consistent with a solid fill or block copy. All width and height combinations are tried to maximize the number of *uncoded pixels* covered by the block. Since the regions are arbitrarily sized rectangles, overlap is possible. Once a pixel has been covered by one block, it is neither advantageous nor detrimental to recode it.

Figure 8.2 shows the result of automatic block decomposition. Pixels in a block denote pixels from the same copy or fill block though the particular colors are not important. For copies, the source of the copies is not shown and for fills, the color is not shown. Punted pixels are shown in white. The decomposition leads to average block sizes of 200-400 pixels so that a compression rate of 0.1 bpp can be achieved if the blocks can be coded at 20-40 bits per block. Typical images result in only about 1% pixels punted which, even at a coding of 3 bits/pixel, only accounts for 0.03 bits/pixel.

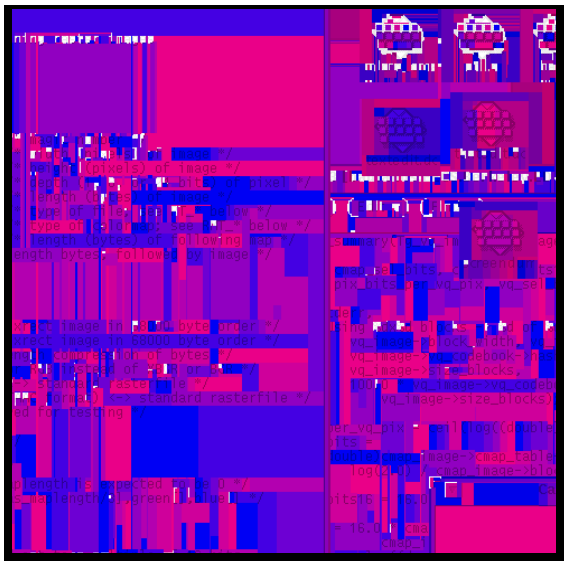


FIGURE 8.2. **Automatic block decomposition.**
Solids blocks denote regions of same copy or fill. White pixels are punts.

8.5. Accelerating the Search

While it may be evident that the previously described decomposition could lead to effective compression, it is not immediately clear that it can be done in a timely manner. As stated, the algorithm requires a full block search over the entire image for each pixel coded, requiring W^2*H^2*BlockSize comparisons and supporting instructions. For a 1000x1000 pixel image with an average block size of 200 pixels, $2*10^{14}$ comparisons would be required, which on a 100MIPS machine would take several weeks. However, the BlockSize factor drops out since search is only performed on pixels not yet coded, reducing the time to several hours. This is still not acceptable for many applications. The optimizations described in this section reduce the time to a matter of seconds.

Typically implementations of a GIF encoder [57] compress at rates of approximately 200-400 kilo-pixels per second (kps) on a Sun UltraSparc 2. For the 1000x1000 pixel image, this results in compression in 2.5 to 5.0 seconds. Without optimization, the FABD algorithm would code at

approximately 0.1kps. However, using the four optimizations presented here, the rate can be increased to that of GIF, 200-400kps.

8.5.1. Big Fill, No Copy Search

For large solid regions, the search for a copy match is not necessary since the solid regions can be coded quite efficiently as fills, and thus coding them as copies yields little benefit. Additionally, the search is likely to be lengthy since the solid regions are likely to match a large portion of the destination region. In regions covered by fill blocks of at least 20 pixels in width or height or 50 pixels in area, the copy search can be suppressed with negligible loss in compression. This typically increases the compression speed by a factor of 5 to a rate of 0.5kps.

8.5.2. Fast Match Lists

Although blocks can be any size, most useful blocks are larger than some minimum size such as 4x4. Otherwise stated, a source location is only worth investigating if a 4x4 region anchored by the source matches a 4x4 region surrounding the destination location. A minimum size of 4x4 was empirically determined to have a negligible effect on compression performance while increasing speed. Larger minimum sizes can result in faster compression, but lower efficiency.

To exploit this observation, the following optimization is performed: First, all overlapping 4x4 blocks in the image are categorized by pattern by placing all 4x4 blocks of the same pattern in *fast match lists*. The lists are sorted in reverse raster-scan order (bottom to top, right to left). Next, block decomposition is performed. However, to perform the copy search for each destination, it is no longer necessary to search over all possible source locations. Instead it is sufficient to search

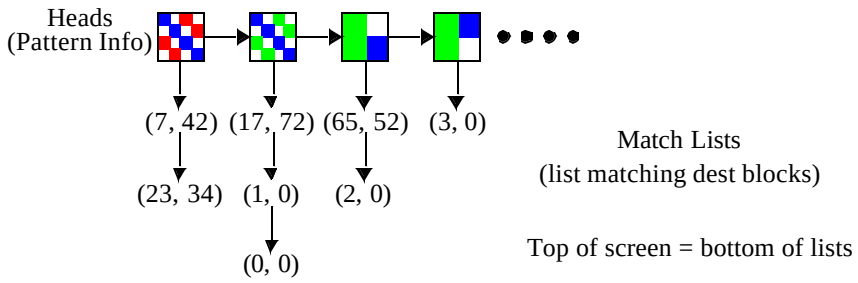


FIGURE 8.3. Match lists used for fast match.
Four patterns are shown horizontally and the locations they are found in the image are shown vertically.

over those with the same initial 4x4 block, saving much time[1]. Using the fast search lists reduces the search time from 30 minutes to between 10 seconds and 10 minutes depending on the image.

The fast match lists must be generated quickly. An efficient way to represent the match lists is to create a list of all used 4x4 patterns and associate with each member of this list another list of pointers to where the patterns are used in the image. (See Figure 8.3.) The list of patterns is called a *head list* while the patterns themselves are called heads. These heads form the beginning of match lists which link the destination locations together.

To generate the list of lists, the image is scanned from left to right, top to bottom. The 4x4 pattern at each pixel is compared to each of the heads, and if it matches, the pixel's location is prepended to the beginning of the head's match list. If it does not match any, a new head is created with the pattern.

The heads are kept sorted in order of last match so that if a pattern repeats, it will be found quickly. Hashing functions allow the head list to be split into multiple shorter head lists using a 14-bit hash on the value of the 4x4 pixel patterns (see Figure 8.4). These optimizations allow the fast match list to be generated in a second or two for typical images.

1. Note that the degenerate case of large solid regions is handled by the optimization of Section8.5.1.

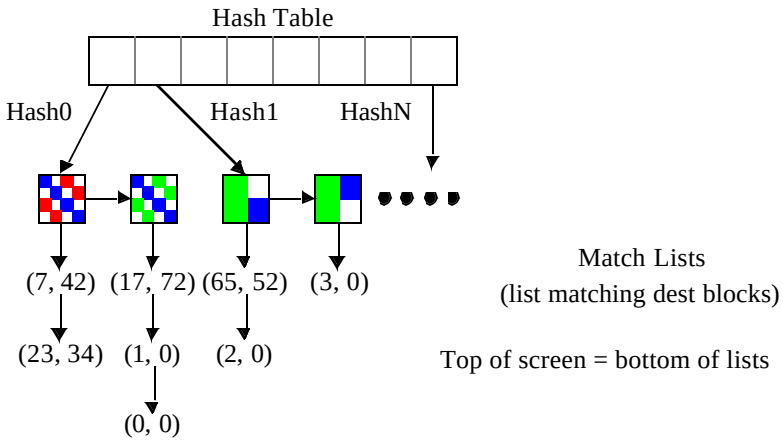


FIGURE 8.4. Hashed match lists - the two pairs of patterns each hash to the same

Max Search Depth	*Screendump* bits/pixel	*Screendump* compress time	*Paper5* bits/pixel	*Paper5* compress time
10	0.144 bpp	2.6 sec	0.108 bpp	3.9 sec
50	0.128 bpp	2.9 sec	0.086 bpp	4.6 sec
100	0.126 bpp	3.2 sec	0.083 bpp	5.4 sec
200	0.125 bpp	3.6 sec	0.080 bpp	6.2 sec
500	0.124 bpp	4.4 sec	0.078 bpp	7.9 sec
1000	0.124 bpp	5.2 sec	0.078 bpp	9.4 sec
100000	0.123 bpp	34.4 sec	0.077 bpp	82.6 sec

TABLE 8.1. Effect of search depth limits on compression time and rate

The net result of the fast-match lists is a speedup of 3-200x to yield a typical coding rate of 1.5 kps to 100 kps. The wide variation is due to the dependence of the search time on the input image. In particular, if there are many blocks with the same 4x4 patterns, the block matching can take significantly longer.

8.5.3. Bounded Search Depth

Every candidate location in the list has a high probability of yielding a block copy match. Since the fast match lists are ordered from bottom to top, physically closer candidates will be

searched first. As the compression time is largely determined by copy block search time, it allows a trade-off between compression time and resultant bit rate. Limiting search depth proves to be very effective in reducing the compression time while not noticeably impacting the compression rate. Limiting also prevents troublesome regions in the image from taking too long. A limit of 1000 does not sacrifice compression while keeping the compression time usually well under a minute. Limits as low as 50 dramatically reduce the compression time while only minimally impacting the compression rate. Table8.1 shows the effect in terms of compression rate and time of different limits on two typical images. The results in Section8.7. are given for limits of 100 and 1000. As compressed data is generated on the fly, the limit could be dynamically varied to match compute time and compressed data rate on an outgoing channel for interactive applications. The bounded search depth results in a speedup by a factor of 1.5 to 50 yielding a coding rate of 75kps to 150 kps. Thus the coding rate becomes much less image dependent.

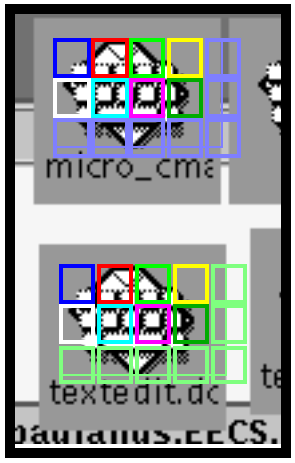


FIGURE 8.5. Coarse / fine matching.
The solid boxes represent 4x4 tiles that match exactly via coarse match while the striped boxes represent regions of partial matches which would only be found through refinement. The coarse match reveals 16x8 pixel match while refinement grows it to 18x11 pixels

8.5.4. Coarse / Fine Matching

The time to perform the block search can be further reduced by using fast-match lists of Section8.5.2. to perform the actual block comparisons (as well as specifying which blocks should be compared as just described.) Recall that the fast-match list membership of a given block uniquely determines the colors of the 4x4 pixels at a given area. Thus a comparison of the per-block fast-match list pointer can be used to quickly compare the entire 4x4 blocks. A two-stage process is used whereby a first coarse pass grows all candidate regions 4x4 pixels at a time in order to get the best match size (modulo 4) in a fraction of the time. (See Figure 8.5.) Then once the best coarse match is found, a refinement pass determines the exact size of the largest match. In this way the pixel-level comparison needs only be performed for one block instead of many, resulting in a speed increase by a factor of 1.5 to 3 to yield a net coding rate of 200-400kps, which is similar to one-dimensional decompositions such as LZW and LZ77 used in GIF and PNG. Although the final block size accuracy is not compromised due to the refinement pass, the best match determined by the coarse phase could in some cases not be the true best match which would

result in a non-optimal choice of blocks. However, results show that the impact on compression performance is typically less than 1%.

8.6. Entropy Coding Techniques

The block decomposition stage generates a parameterization of the input image in terms of block copy, fill, and punt primitives. Although there are many fewer blocks than pixels, these parameters still have to be coded efficiently. This is the job of the entropy coder. A simple entropy coder based on Huffman codes was used to verify the potential utility of the two-dimensional flexible block decomposition. More sophisticated entropy coders could result in further coding gains.

8.6.1. What to Code

The block decomposition generates the following three types of primitives:

CopyBlock(*dest_x*, *dest_y*, width, height, src_x, src_y)
SolidBlock(*dest_x*, *dest_y*, width, height, color)
PuntBlock(*dest_x*, *dest_y*, numPixels, pixel1, pixel2, ..., pixelN)

The dest_x and dest_y fields are not transmitted as they are implicit in the order that the data is transmitted. For copy blocks, the dimensions and source location have to be transmitted, for fill blocks the dimensions and color have to be transmitted, and for punt blocks the punted pixel values have to be sent.

8.6.2. Transforming the Parameters

While the block decomposition is a transformation of the pixel data into a new parameter space, which results in more efficient coding, additional transformations of the parameters aid in entropy reduction. A few transformations will be discussed here.

8.6.2.1. Color Age

The input images consist of 8-bit pseudo-color pixels. The colors of punted pixels, and to some extent filled blocks, exhibit a large degree of spatial locality. It is common for only a few unique colors to be used in a given region. Text regions typically use only two colors. However, throughout the image, different sets of colors might be used. Thus it is beneficial to code a *relative* property of the colors instead of the colors themselves. Given a stream of pixel colors:

red, blue, brown, brown, brown, red, brown

the *color age* is the number of unique colors present between a given instance of a particular color and the previous instance of that same color. In the above case the color ages are:

red=?, blue=?, brown=?, brown=0, brown=0, red=2, brown=1

It is not possible to determine the color age of the first three colors since they depend on the previous colors encountered. To initialize the system, the color history is set such that the initial color age of each color is its pixel color.

As evidence of the utility of the color age technique, using adaptive Huffman coding, the average number of bits to encode the fill and punt colors in the *screendump* image drops from 3.63 and 3.05 using the actual colors to 2.03 and 1.62 using color age[1].

8.6.2.2. Relative Copy Source

Spatial locality suggests that blocks will often be copied from nearby regions. Thus, the relative sources will be small and non-uniformly distributed, and hence have lower entropy. Additionally, many images, particularly ones with text, have significant spatial structure that can be modeled by coding the source location relatively. For the *screendump* image, coding the source x

1. Lempel-Ziv compression of punted pixels did not appear to yield better compression.

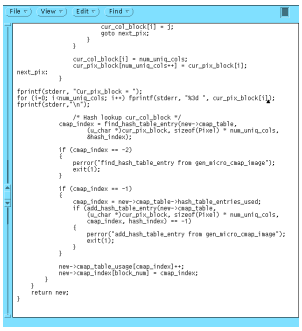
coordinate relatively reduces its entropy from 7.58 bits/pixel to 7.26 bits/pixel and coding the source y coordinate relatively reduces its entropy from 7.41 bits/pixel to 5.56 bits/pixel.

8.6.3. Huffman Coding

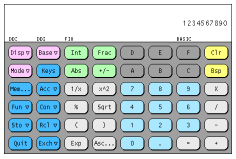
After the parameters are transformed, conventional entropy coding techniques are used to exploit the skewed probability distributions present. Adaptive Huffman codes are used since the alphabet sizes are large enough not to require arithmetic coding. The various parameters are coded independently. Joint coding of height and width did not perform as well due to undersampling effects: when pairs of parameters, such as block width and height, are considered jointly, there are more unique symbols, and thus the overhead of introducing new symbols is greater. Using joint coding, the number of symbols is proportional to the product of number values of each parameter while using independent coding, the number of symbols is proportional to the sum. Joint coding might be more advantageous for large sample sets (i.e. large images) and if there was a strong correlation between two parameters.

8.7. Results

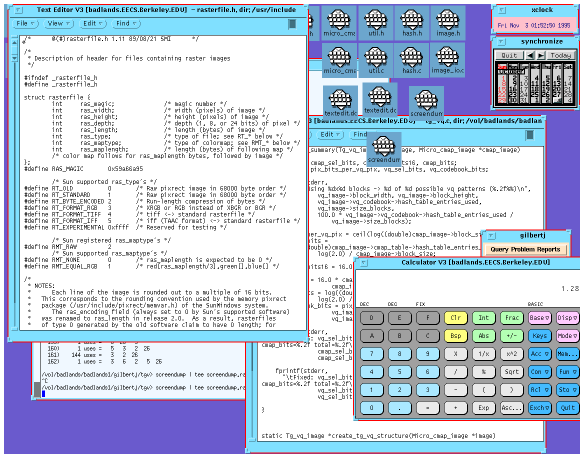
The FABD algorithm was evaluated on several images which typify the class of images described in Section8.2.1. *Calculator*, *textedit*, *paper*, and *screendump*, shown in Figure 8.6, are discrete-tone snapshots of the type used in remote computation. The first three show single applications while *screendump* shows a number of applications running. These images stress the compression algorithm on both simple and complex images. *Paper5*, *paper5_big*, and *paper9*, shown in Figure 8.7, are bi-level images generated from PostScript® files, differing in density and content which will allow performance variations over varying image detail to be analyzed. *Paper5* contains a typical page of text in a proportional font at 130dpi while *Paper5_big* contains the same



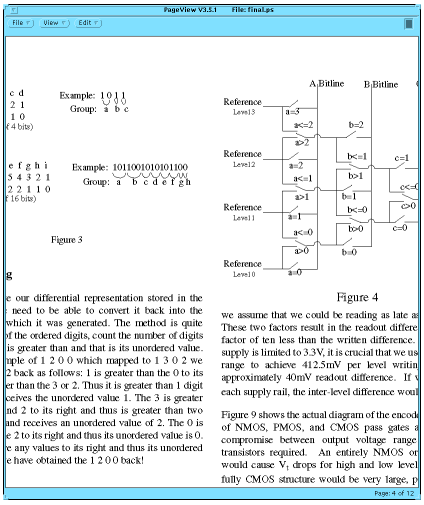
Textedit



Calculator

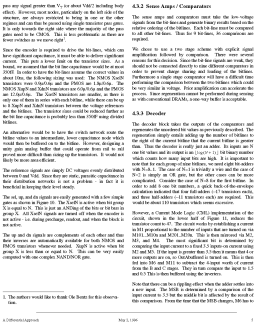


Screendump

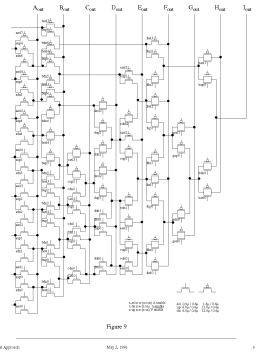


Paper

FIGURE 8.6. Discrete-tone pseudo-color test images



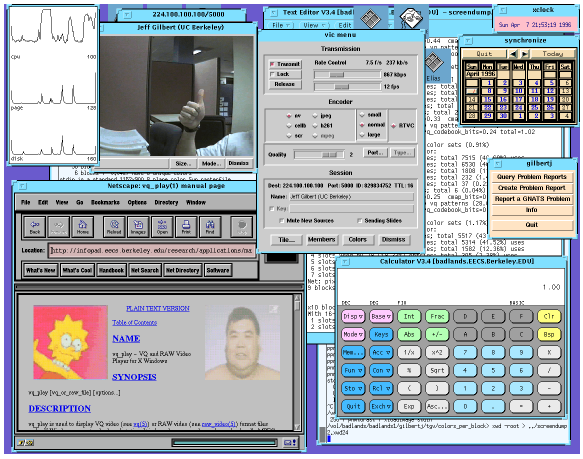
Paper5 / Paper5_big



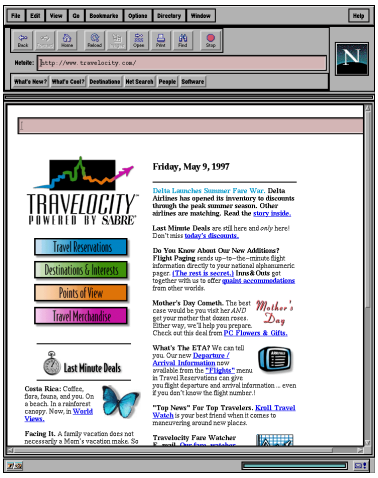
Paper9

FIGURE 8.7. Bi-level test images

text rendered at 200dpi. *Paper9* is a simple circuit schematic rendered at 150dpi. Finally, *screendump2* and *netscape*, shown in Figure 8.8, are primarily discrete-tone images that have



Screendump2



Netscape

FIGURE 8.8. **Hybrid discrete / continuous tone test images**

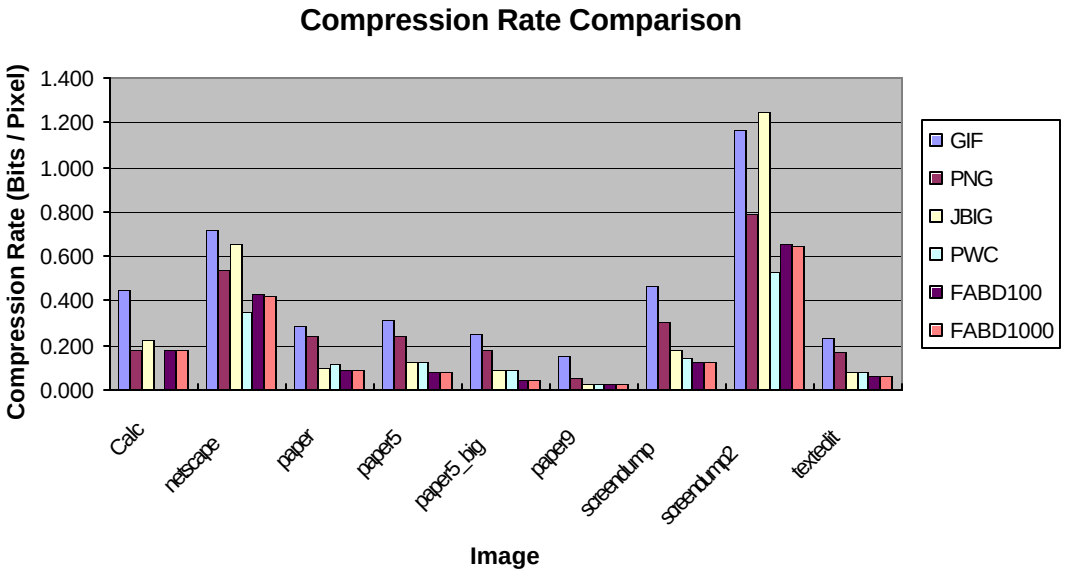


FIGURE 8.9. **Graph of compression rates for various techniques. FABD100 and FABD1000 refer to FABD using maximum search depths of 100 and 1000 respectively.**

some sizable continuous-tone regions. These are useful to analyze performance of the algorithm on images not completely in the anticipated domain.

The compression levels obtained by FABD, GIF, PNG, JBIG, and PWC are shown in Figure 8.9 and Table8.2. The ratios indicate FABD's compression advantage over the other techniques. The top and bottom FABD results for images correspond to maximum search depths of

	FABD	GIF		PNG		JBIG		PWC	
	Bpp	**Bpp**	**Ratio**	**Bpp**	**Ratio**	**Bpp**	**Ratio**	**Bpp**	**Ratio**
calculator (456x298)	0.176	0.446	2.540	0.177	1.010	0.228	1.298	???	???
	0.177		2.525		1.004		1.290		???
paper (836x993)	0.091	0.283	3.127	0.242	2.673	0.102	1.128	0.115	1.272
	0.091		3.104		2.653		1.119		1.263
screendump (1152x900)	0.124	0.467	3.767	0.309	2.497	0.177	1.430	0.146	1.181
	0.126		3.711		2.460		1.408		1.163
textedit (593x646)	0.064	0.238	3.720	0.168	2.629	0.080	1.246	0.081	1.271
	0.064		3.705		2.618		1.241		1.266
paper5 (1132x1465)	0.078	0.310	3.966	0.241	3.082	0.125	1.599	0.251	1.607
	0.083		3.733		2.901		1.506		1.513
paper5_big (1675x2168)	0.044	0.248	5.675	0.182	4.162	0.086	1.970	0.087	1.987
	0.048		5.169		3.791		1.794		1.810
paper9 (1245x1611)	0.029	0.154	5.323	0.056	1.951	0.026	0.888	0.025	0.869
	0.031		4.926		1.806		0.822		0.804
netscape (741x938)	0.426	0.717	1.684	0.537	1.263	0.651	1.529	0.352	0.827
	0.427		1.680		1.260		1.525		0.825
screendump2 (1152x900)	0.650	1.166	1.794	0.794	1.221	1.252	1.925	0.526	0.809
	0.652		1.789		1.218		1.921		0.807

TABLE 8.2. **Compression rates for various techniques.**
The top numbers in each pair correspond to FABD max depth of 1000 while the bottom correspond to a FABD max depth of 100. The ratio is the other method's coding rate divided by FABD's.

100 and 1000 respectively. A depth of 100 typically sacrifices at most 10% of the compression obtained using a depth of 1000. Table8.3 shows the time required by the techniques to compress the images. The breakdown of the bit usage for FABD coding of each image is shown in Table8.4.

Since FABD is non-progressive, the JBIG compression was also performed in the non-progressive mode which delivers better compression than the default progressive mode. The JBIG bitplane decomposition is shown in Table8.5.

	FABD 100		FABD 1000		GIF		PNG	
	Time (Sec)	**Rate (Pix/Sec)**	**Time (Sec)**	**Rate (Pix/Sec)**	**Time (Sec)**	**Rate (Pix/Sec)**	**Time (Sec)**	**Rate (Pix/Sec)**
calculator (456x298)	0.4	340,000	0.8	170,000	0.3	450,000	0.4	340,000
paper (836x993)	2.5	33,0000	4.6	180,000	2.3	360,000	2.1	400,000
screendump (1152x900)	3.2	320,000	5.2	200,000	2.9	360,000	3.2	320,000
textedit (593x646)	1.1	350,000	1.6	240,000	1.1	350,000	1.1	350,000
paper5 (1132x1465)	5.4	310,000	9.4	180,000	4.3	390,000	7.4	220,000
paper5_big (1675x2168)	10.4	350,000	18.0	200,000	9.8	370,000	18.4	200,000
paper9 (1245x1611)	4.8	420,000	7.2	280,000	5.5	370,000	10.1	200,000
netscape (741x938)	2.3	300,000	3.9	180,000	1.9	370,000	2.1	330,000
screendump 2 (1152x900)	4.4	240,000	6.4	160,000	2.9	360,000	3.2	320,000

TABLE 8.3. **Compression times for dictionary-based techniques on 168Mhz Sun Ultra 2.**

It is readily apparent that the algorithm outperforms the one-dimensional dictionary-based techniques GIF and PNG on all images, dramatically so on most images. FABD outperforms the two-dimensional statistical JBIG on all but one image and PWC on all but three images. It is worthwhile to discuss the performance on the images grouped by type of image.

FABD outperforms GIF on the discrete-tone images *calculator*, *paper*, *screendump*, and *textedit* by a factor of 2.5 to 3.8 due to its ability to exploit the two-dimensional redundancy. It similarly out compresses PNG by a factor of about 2.5 on three of the four images. However, its compression is similar to PNG on the small *calculator* image as it cannot find many large two-dimensional regions. It compresses 12% - 43% more efficiently than JBIG and 16% - 27% more

	Total Rate (bpp)	Copy Blocks			Fill Blocks			Punt Pixels		
		Blocks	Ave Size (bits)	Net Effect (bpp)	Blocks	Ave Size (bits)	Net Effect (bpp)	Pixels (% of total)	Ave Size (bits)	Net Effect (bpp)
calculator (456x298)	0.176	357	22.67	0.060 (34%)	398	12.30	0.036 (20%)	3439 (2.5%)	2.36	0.060 (34%)
paper (836x993)	0.091	1652	27.73	0.055 (60%)	899	13.51	0.015 (16%)	4897 (0.6%)	2.29	0.014 (15%)
screendump (1152x900)	0.124	2233	25.97	0.056 (45%)	1818	13.83	0.024 (19%)	15591 (1.5%)	2.21	0.033 (27%)
textedit (593x646)	0.064	429	27.22	0.030 (47%)	349	14.91	0.014 (22%)	2077 (0.5%)	2.56	0.014 (22%)
paper5 (1132x1465)	0.078	3283	28.48	0.056 (72%)	1563	12.22	0.012 (15%)	4822 (0.3%)	1.75	0.005 (6%)
paper5_big (1675x2168)	0.044	3900	29.88	0.032 (73%)	1787	13.17	0.006 (14%)	4972 (0.1%)	1.78	0.002 (5%)
paper9 (1245x1611)	0.029	1237	28.92	0.018 (62%)	855	14.81	0.006 (21%)	2619 (0.1%)	1.91	0.002 (7%)
netscape (741x938)	0.426	1352	25.25	0.049 (12%)	1701	14.17	0.035 (8%)	48636 (7.0%)	4.57	0.320 (75%)
screendump2 (1152x900)	0.650	2544	25.64	0.063 (10%)	2769	15.14	0.040 (6%)	128113 (12.4%)	4.26	0.527 (82%)

TABLE 8.4. Bit breakdown for FABD 1000.
Note that the results relating to copy and fill are per block while the punt is per pixel. Fraction of contribution that is not copy, fill, block is overhead associated with block type, colormap, etc.

	Total Rate	K bytes in Bitplanes
calculator (456x298)	0.228 bpp	1.3 / 1.1 1.1 / 0.2
netscape (741x938)	0.651 bpp	11 / 7 / 6 / 6 7 / 6 / 6 / 7
paper (836x993)	0.102 bpp	0.5 / 0.5 / 9.5
paper5 (1132x1465)	0.125 bpp	26
paper5_big (1675x2168)	0.086 bpp	39

	Total Rate	K bytes in Bitplanes
paper9 (1245x1611)	0.026 bpp	6
screendump (1152x900)	0.177 bpp	12.8 / 3.8 / 3.1 1.8 / 1.5
screendump2 (1152x900)	1.252 bpp	22 / 19 / 20 / 19 21 / 21 / 22 / 19
textedit (593x646)	0.080 bpp	3.3 / 0.2 / 0.3

TABLE 8.5. JBIG Bitplane decompression

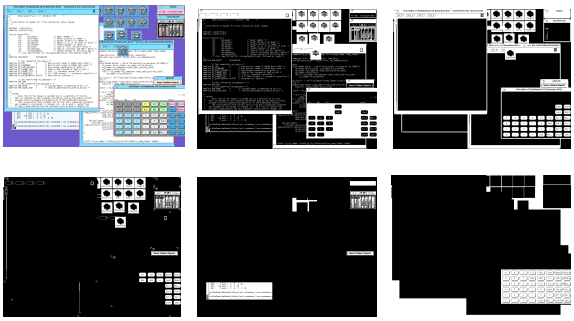


FIGURE 8.10. Bitplane decomposition of *screendump* image. Original is on far left while the monochrome images formed from each of the 5 bit-planes are shown to its right. The bit planes account for 12.8k, 3.8k, 3.1k, 1.8k, and 1.5k bytes of the JBIG image respectively.

efficiently than PWC on these images based on JBIG and PWC's inability to code multiple pixels at once. The JBIG bit-plane decomposition of *screendump* is shown in Figure 8.10. As can be seen from the figure as well as Table8.5, most of the information is, by chance, in bitplane 0 so there is not too much redundancy across bit planes and JBIG has a chance at efficient coding. However, quite often text and other structures will be striped over multiple planes if its foreground and background colors differ in more than one bit. In these cases, JBIG will be less efficient.

FABD allows 3.7 to 5.6 times more compact coding than GIF and 1.8 to 4.2 times more compact coding that PNG on the bi-level images *paper5*, *paper5_big*, and *paper9*. PNG's superior bit-packing compared to GIF is probably responsible for much of the difference in the results of the two techniques. However, since there is a significant amount of two-dimensional repetition, FABD outperforms both. FABD outperforms JBIG on *paper5* and *paper5_big* due to its ability to exploit the repetitive patterns in the image at a high level. While JBIG efficiently models each pixel, FABD spots letters, words and sometimes phrases that are used more than once. As the resolution is increased, the compressed file size does not increase dramatically since the number of blocks remains roughly constant. The number of pixels more than doubled but only 15% more blocks are required. JBIG very slightly outperforms FABD on *paper9* because *paper9* is so simple, consisting of mostly horizontal and vertical lines, which JBIG can very accurately model at

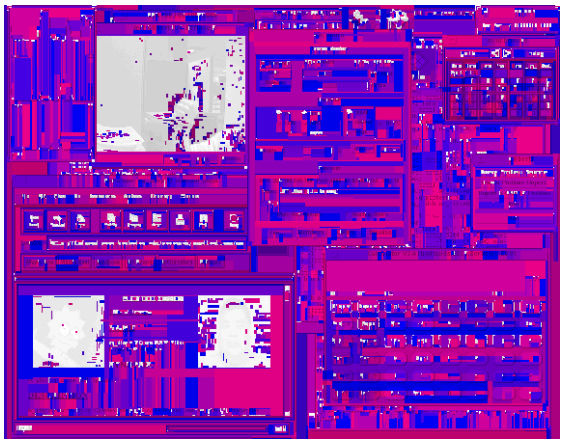


FIGURE 8.11. Block decomposition of screendump2 image. The white regions represent punted pixels.

the pixel level. PWC performs similarly to JBIG since in the case of bi-level images, the two techniques are roughly equivalent.

JBIG is tailored to bi-level images so it is notable that FABD can achieve better compression on this class of images. However, JBIG is well suited for scanned and/or half-toned images as well, which FABD will not compress as compactly. These types of images have a probabilistic regularity but low deterministic regularity. Most of the pixels can be predicted correctly but, many cannot. Thus JBIG will incur a coding penalty for the wrong pixels but FABD will have to reduce the size of the entire blocks, effectively incurring a penalty on all of the pixels in the block.

Lastly, *netscape* and *screendump2* are primarily discrete-tone color images with some sizable continuous-tone regions. FABD is still able to outcompress GIF and PNG on these images due to its improved performance in the discrete-tone parts of the image. As seen in Table8.4, approximately 80% of the FABD bits are used for the 10% of the pixels which are punted. The continuous regions of the images do not lend well to block matching and are thus punted. Figure 8.11 shows the block decomposition of *screendump2* with punted pixels shown in white. In *screendump2*, each punted pixel requires more than 4 bits to code. Since typically punts are rare, a

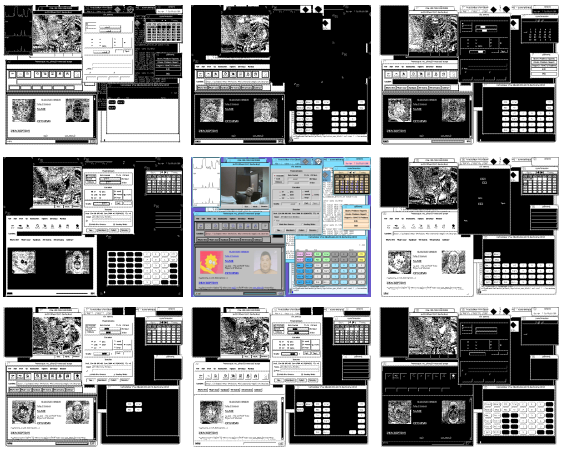


FIGURE 8.12. Bitplane decomposition of *screendump2* image. Original is in center while bitplane images surround it.

simple coder was used but for improved coding of hybrid images, a more sophisticated technique could be used such as lossless JPEG, or possibly even a lossy coding. FABD still outperforms JBIG by a factor of 1.5 to 2 on these images due to the fact that the images do not split well across bit-planes, as seen in Figure 8.12. Additionally, the continuous regions are not well suited to JBIG compression. However, the continuous regions are exactly what PWC is designed for and thus it can code in 20% fewer bits than FABD.

8.8. Conclusion

This chapter describes how two-dimensional global structure can be effectively exploited to achieve efficient coding of discrete-tone images. While GIF and PNG are limited to one-dimensional global structure and JBIG and PWC only use a local context, FABD is able to obtain the best of both. Due to FABD's lossless nature, efficient matching is possible. The two-dimensional flexible automatic block decomposition provides a different method of compressing images which outperforms one-dimensional dictionary and two-dimensional statistical techniques on many images, and could be combined with more sophisticated entropy coding techniques to achieve even greater performance.

CHAPTER 9 *Development & Analysis Environment*

9.1. Introduction

Many of application-independent compression techniques described in this part of the thesis were prototyped in the context of the InfoPad project, previously described. This chapter describes the development and analysis environment created during the InfoPad project which was used to develop, debug, analyze, and improve the algorithms previously presented, as well as further the research of others in the InfoPad research group working on topics ranging from wireless networking protocols to CMOS high-bandwidth radio design.

The design environment allowed full development of the software infrastructure and applications before the actual InfoPad hardware was deployed. In this way, the hardware and software development could proceed concurrently. Additionally, debugging hooks in the system allow emulation and analysis in the software domain that would be more difficult or impossible using the actual hardware system.

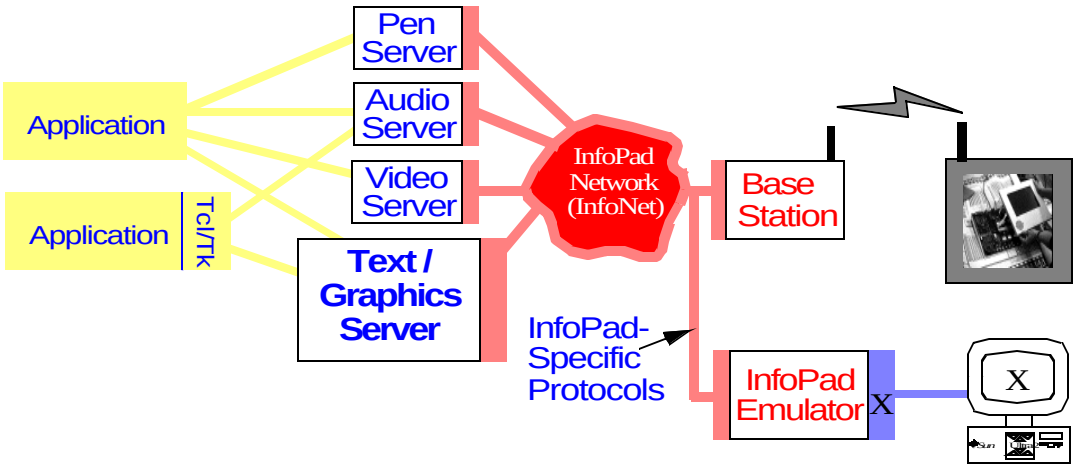


FIGURE 9.1. **InfoPad development environment**

9.2. *Networking Environment*

Figure 9.1 shows the InfoPad development environment. Standard Unix / X applications shown on the far left communicate with InfoPad-specific *type-servers* which translate the standard protocols into InfoPad-specific protocols. The type-servers operate on different types of data: pen, audio, video, and text / graphics, and are described in sections that follow. The type-servers communicate the InfoPad-specific data over the InfoNet networking infrastructure. InfoNet manages tasks such as routing and hand-off of networking data as well as overall pad state maintenance, tracking and control. A nameserver database manager is used to keep track of the state of various components in the system, including versioning information as well as operational status. The InfoNet networking layer is typically overlaid on top of standard TCP/IP to allow the InfoPad network to run on multiple machines.

Remote terminals or “pads” can connect in one of two ways as shown on the right side of the figure. Hardware pads connect wirelessly through hardware / software basestations. The base stations connect to the rest of the network via standard IP protocols and to the wireless pads through custom or commercial radios[49]. Each basestation is responsible for a particular cell and

as hardware pads migrate between cells, their connections are handed off via InfoPad-specific cell-servers and gateways.

Alternatively, an all-software environment can be used by connecting to InfoNet with a software InfoPad Terminal Emulator described below. The emulator uses the same protocols as the hardware pads, allowing the type-servers and InfoNet to operate exactly as if a hardware pad is in use. However, the emulator displays its data on any X Window terminal, allowing greater availability. Debugging and analysis hooks allow performance characterization and emulation control.

9.3. Emulator

The InfoPad terminal emulator, shown in Figure 9.2, allows emulation of the InfoPad hardware terminals, debugging of the InfoPad software components, and analysis of protocol and system performance[1]. The emulator connects to the InfoPad network just as hardware terminals via basestations do, but instead presents its user interface to any X Window terminal. The emulator supports text / graphics, video, audio, and pen traffic as well as control messages. Several pop-up windows described below control detailed aspects of the emulator operation.

The emulator is written as a hybrid C / Tcl/Tk application where Tcl/Tk code controls the user interface and high-level control, and underlying C code is used to interface to InfoNet and perform per-packet time-critical processing. This allows the flexibility and rapid prototyping capabilities of Tcl/Tk with the low-level processing power of C.

1. The emulator was originally developed by Brian Richards.

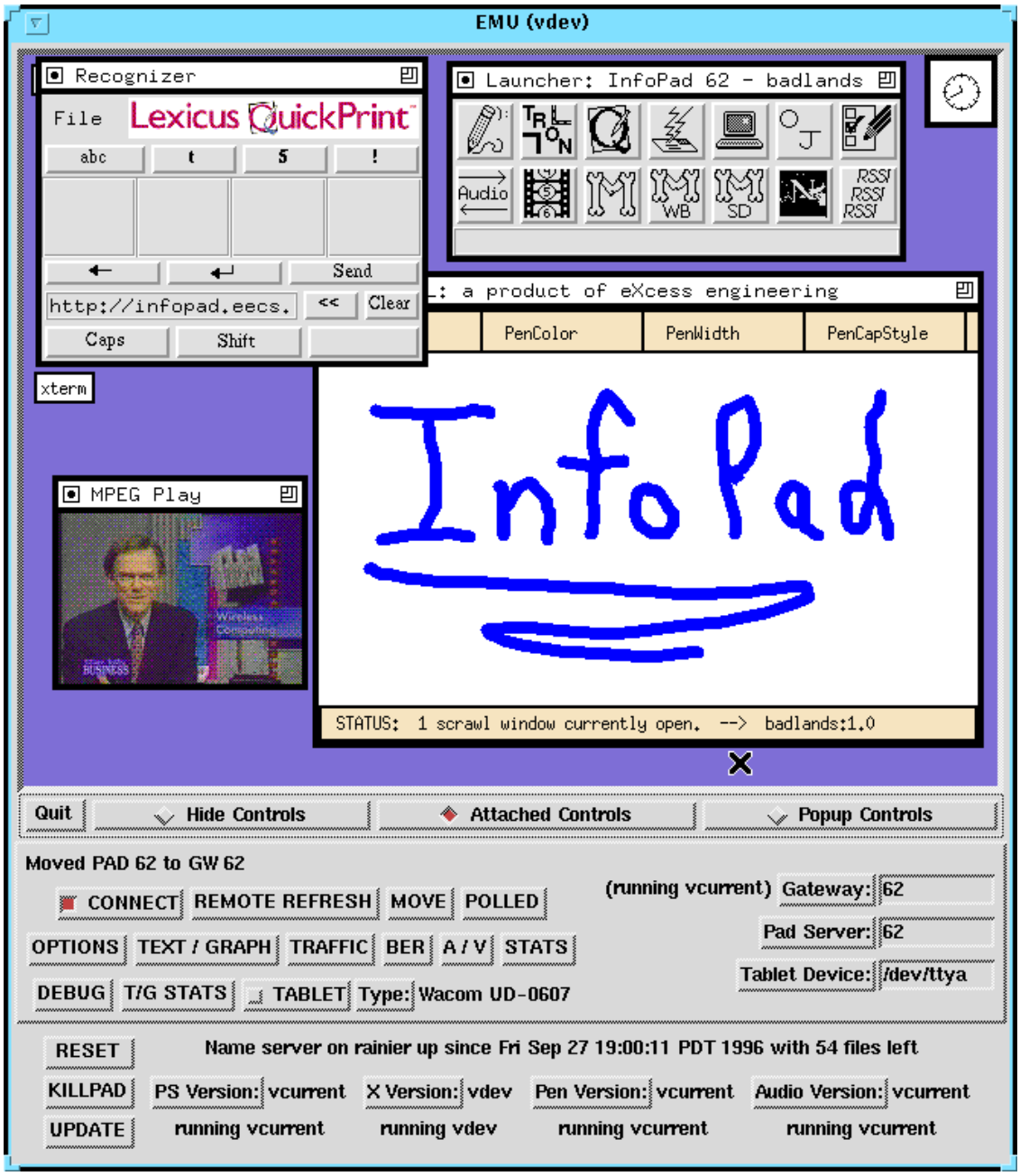


FIGURE 9.2. **InfoPad terminal emulator**

9.3.1. Operation

The main window shown in Figure 9.2 emulates the text / graphics display as will be described below in the next section. Below that are controls which give the user control beyond that which the pad hardware supports.

On the right, the user can select which gateway and pad server number they want to connect to. The "GATEWAY:" pull-down menu queries the name server to see which gateways are currently running and displays their numbers and the version of the gateway code that they are running. If the gateway or cellserver are down, this is indicated. The "PAD SERVER:" pull-down menu queries the nameserver to display which pad servers are operational and their versions. Finally the "TABLET DEVICE:" pull-down allows the user to enter which port the pen tablet is connected to. The "Type:" pull-down menu allows selection of the tablet type and the "TABLET" toggle button controls whether a connection to the tablet is attempted.

The bottom row of indictors show further status information from the nameserver. Its location and time it was started are displayed. Below, the status of the currently selected pad, X (text / graphics), pen, and audio server are displayed. It is retrieved from the Name Server periodically. When any critical event occurs, such as selecting a new pad server or gateway, the emulator polls for status more often for a while. The "UPDATE" button forces the status to be reread. The pull-down menus for "PS VERSION", "X VERSION", "PEN VERSION", and "AUDIO VERSION" control the version of the respective servers which is used if they are auto-started by connecting to a pad cluster which is not running. The "KILLPAD" button kills the currently selected pad server, causing termination of all associated type servers as well.

The user can connect to and disconnect from a pad cluster using the "CONNECT" button. The "REMOTE REFRESH" button causes the display to be updated. The "MOVE" button forces a hand-off to a new gateway while the "POLL" button indicates that the emulated receive signal

strength has been queried by a cell server. The "OPTIONS", "TEXT/GRAPHICS", "TRAFFIC", "BER", "A/V", "STATS", "DEBUG", and "T/G STATS" buttons invoke pop-up display presented below.

9.3.2. Text / Graphics Display Support

The text / graphics data is displayed in the main emulator window. The 640x480 monochrome mode used by the pad hardware is supported. Additionally, color modes of varying size which support some of the techniques outlined in Chapter 4, Chapter 5, and Chapter 6 are supported to allow research into future protocols.

The pop-up dialogs shown in Figure 9.3 control various aspects of the operation of the text / graphics subsystem. The Start-up Options dialog is used to set configuration options which are sent to the nameserver and passed to the text / graphics servers when they are auto-started. The "X CONFIG" field is used to set the X configuration file that is used to control pad user and session preferences. It is used instead of login authentication. The "MODE" controls if the X server is started in monochrome or a color mode while the "SCREEN SIZE" is used to select the size of the emulated screen when a color mode is used. When the text / graphics server is running, it periodically transmits the screen size and mode parameter to the emulator so that the emulated screen size is adjusted properly. The "RATE LIMIT" and "MAX REFRESH" sliders are used to adjust the initial rate limiting and refresh rates. These can be adjusted during operation using the traffic control and monitoring popup described below. The "MAX PACKET SIZE" and "BUFFERED WRITES" boxes are used to control the packet size and whether the network interface is buffered. The "INTER-PACKET DELAY:" slider is used to limit the rate that packets are generated. Finally the "CURRENTLY RUNNING:" line indicates the results from querying the name server as to the configuration of the currently running text / graphics server.

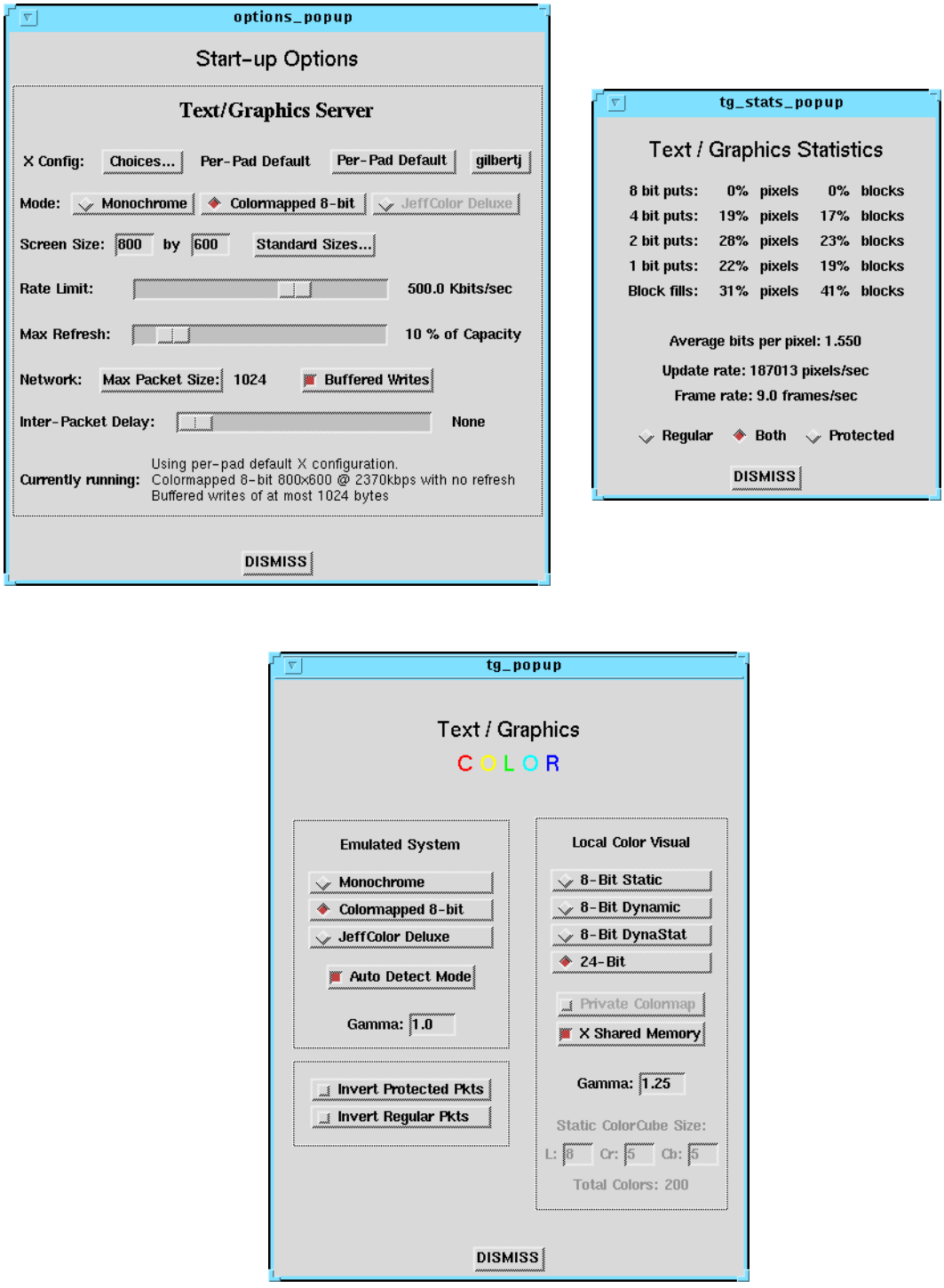


FIGURE 9.3. InfoPad emulator text / graphics pop-up controls.
Top-left is the text / graphics start-up options dialog, top-right is the text / graphics statistics dialog, and bottom center is the text / graphics control dialog.

The text / graphics control dialog controls the current emulated display mode. The emulator will emulate "MONOCHROME", "COLORMAPPED 8-BIT", or "JEFFCOLOR DELUXE" (TGVQ) displays. The "AUTO DETECT MODE" enables decoding of packets from the text / graphics server indicating which type of display is being used. The "GAMMA:" box allows the gamma correction factor to be applied to the of the emulated screen data to be entered. Gamma correction is applied on-the-fly. The "INVERT PROTECTED PKTS" and "INVERT REGULAR PKTS" buttons allow regular or protected packets to be inverted on the screen for easier identification. Recall that the protected packets are those which are used for asymtotic reliability refresh, have higher forward error correction applied, and are dropped if an error is detected in their data.

Emulation of a terminal window which could potentially contain any configuration of colors on an X Window display possibly with limited 8-bit colormapped display requires on-the fly mapping of colors. The Local Color Visual section allows the user to select the mode that the emulator text / graphics X Window is displayed. The basic problem is that the emulated terminal screen can potentially contain any color. However, since the emulator runs as a standard X Window application, it cannot be guaranteed to be able to render all colors. The Local Color Visual box allows the user to select the mode that the screen should be emulated. The emulator can run on 8-bit colormapped or 24-bit true-color displays. On 24-bit true-color displays, there is not a problem since any possible color configuration can be rendered since each pixel's red, green, and blue values can be individually set. In 8-bit colormapped displays, the emulator can choose to co-exist with other applications' colormap requests, in which case it can only use some fraction of the 256 physical colormap entries, otherwise it can use its own colormap, so it can control all 256 colormap entries, but the screen will flash when the mouse enters and exits the application since the emulator colormap is only active when the cursor is in the emulator window. The former is chosen by deselecting the "PRIVATE COLORMAP" box while the latter is chosen by selecting it. If in 8-bit mode, the "8-BIT STATIC" mode refers to a mode that a static color-cube of specified dimensions is allocated

and colors to be displayed are chosen from this color cube. "8-BIT DYNAMIC" mode entails allocating colors as the emulated display needs them, and any colors that cannot be allocated are represented by the closest color that can be allocated. "8-BIT DYNASTAT" is a hybrid of the two techniques that first allocates a static color-cube and then allocates additional requested colors dynamically. In this way, a fixed base set of colors will always be present, but often exact matches will be possible. Finally the "X SHARED MEMORY" button indicates if the X Window shared memory image transfer protocol should be used. If the emulator is running on the same machine as the X server it is being displayed on (not to be confused with the text / graphics server servicing the pad), the X shared memory protocol allows image data to be transferred in shared memory, avoiding copying throughout the networking subsystem. This reduces the computation load of rendering, thus allowing higher frame rates.

The "Text / Graphics Statistics" pop-up displays statistics of the text / graphics data received. The "8-BIT PUTS", "4-BIT PUTS", "2-BIT PUTS", "1-BIT PUTS", and "BLOCK FILLS", and "AVERAGE BITS PER PIXEL" are used for analysis of a compressed bitmap transmission algorithm. They display the percentage of pixels and blocks that are sent using 8, 4, 2, 1, and 0 bits per pixel as well as the average bits per pixel. The "UPDATE RATE" indicates the net pixel display rate averaged over the last second, and the "FRAME RATE" indicates the frame update rate achieved calculated from new-frame packets received from the text / graphics server. The "REGULAR", "PROTECTED", "BOTH" selection indicates which types of update packets the above measurements apply to.

9.3.3. Audio and Video support

The emulator supports emulation of pad audio input and output as well as VQ-video output through the Audio / Video pop-up shown in Figure 9.4.. The audio is coded as 8 kHz, 8-bit μ-law

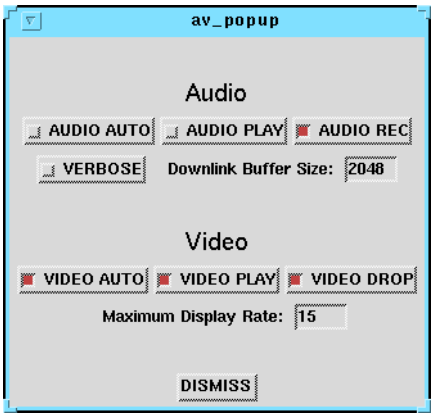


FIGURE 9.4. InfoPad emulator audio / video pop-up dialog

PCM coded samples, and is accessed directly through the Sun Workstation's `/dev/audio` device. The uplink and downlink can be individually enabled through the "AUDIO PLAY" and "AUDIO REC" buttons and an automatic mode is enabled through the "AUDIO AUTO" button, whereby the audio is enabled as soon as the first downlink audio packet is received. Diagnostic data is displayed if the "VERBOSE" button is checked.

Low latency audio data, required for synchronized audio and video as well as effective video conferencing, requires that the number of samples of buffered audio is kept low. The "DOWNLINK BUFFER SIZE:" box allows the amount of data that is buffered in the emulator and the `/dev/audio` device buffer to be limited. If more data than is allowed is queued, the extra data is dropped, emulating a limited size downlink buffer. By varying the size of this buffer, the effects of network jitter and packetization can be explored and the requirements of the hardware downlink audio buffer size can be determined.

The Audio / Video popup also controls the real-time color 128x240 VQ video display. The actual display of the VQ data is performed by the video utility `vq_play` described below, but the emulator is responsible for controlling when vq_play is opened and closed as well as combining video packets into complete VQ video frames and sending them to `vq_play`. Thus the use of a separate stand-alone application for video display is hidden from the user but allows greater mod-

ularity and code reuse. The VQ video window is opened and closed via the "VIDEO PLAY" button. If "VIDEO AUTO" is enabled then the VQ video window is opened whenever VQ video arrives. The frame rate can be manually limited for performance reasons by entering the maximum desired frame rate in the "MAXIMUM DISPLAY RATE" box and selecting "VIDEO DROP". This can be used to assure that emulation of the video screen does not impact the performance of the other subsystems. Frames are dropped to assure that the aggregate rate does not exceed the specified limit.

9.3.4. Traffic Monitoring and Control and Debugging Hooks

Figure 9.5 shows the emulator traffic and debug dialogs which are used to monitor downlink traffic rates and latencies, optionally limit downlink traffic rates, and monitor uplink traffic rates. At the top, the rate in packets per second, kilobits per second, and average bytes per packet are shown for each of the individual uplink and downlink data types as well as the aggregate uplink and downlink traffic. The text / graphics data is further subdivided into regular and protected traffic where as previously described, the protected traffic is used for asymtotic reliability and is dropped if in error.

Below the traffic rate display, a measurement of the latencies of the downlink traffic is presented. Minimum, average, and maximum latencies over the past second are displayed. This information is obtained by comparing time-stamps placed in the packets by their senders with the time that the packets are processed by the emulator. Clock skew, which could occur if the packets are sent by a different Unix host than the emulator is running on, is removed by assuming that the lowest-latency packet ever received during the lifetime of the emulator-pad connection will be 0 ms. This is based on the assumption that most delays are due to queuing and not transport delays. The latency measurements are particularly useful when combined with the downlink rate limiting.

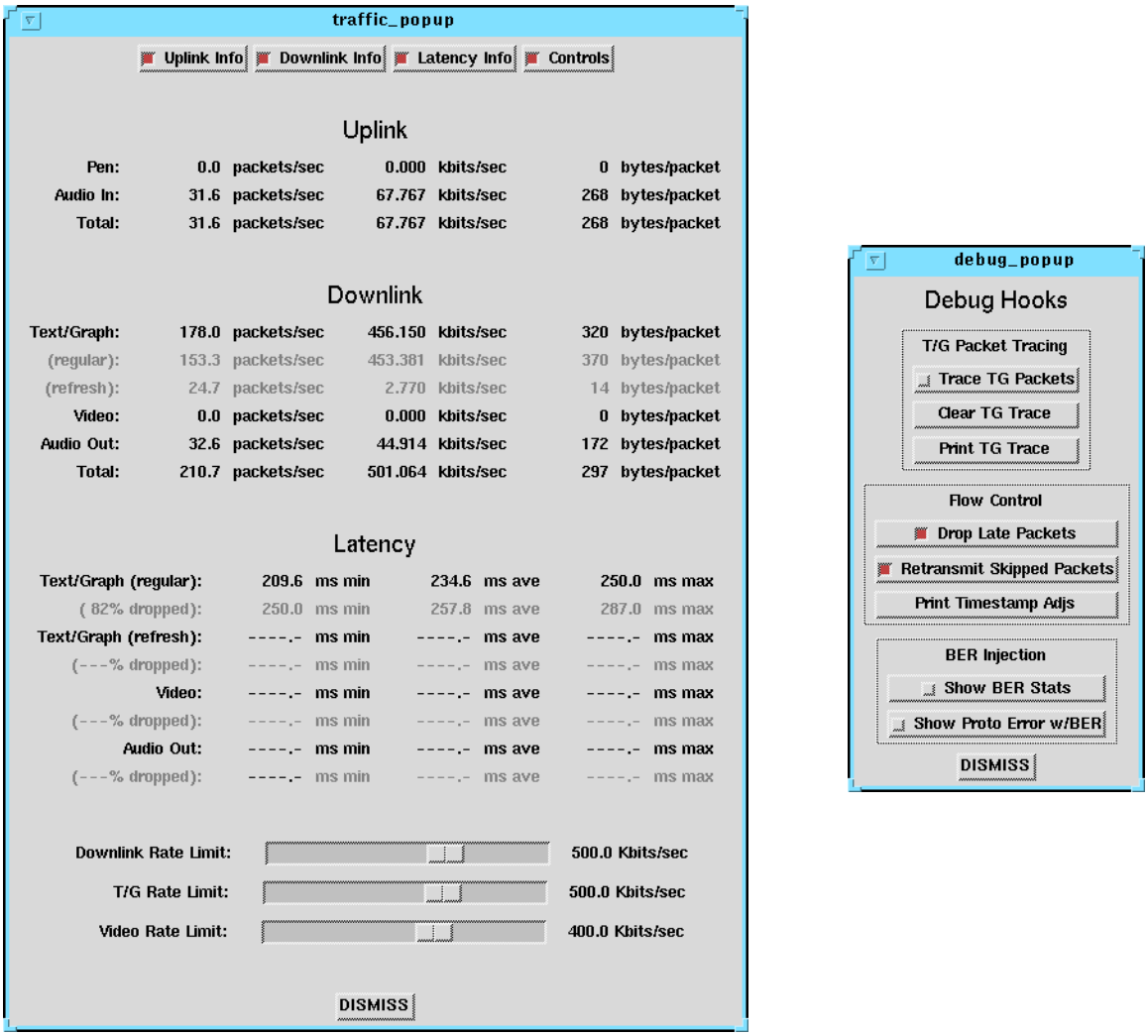


FIGURE 9.5. **Infopad emulator traffic and debug pop-up dialogs**

The "DOWNLINK RATE LIMIT", "T/G RATE LIMIT", and "VIDEO RATE LIMIT" sliders are used to control the downlink sender rates by sending messages to the gateway, text / graphics server, and video server (`send_vq` described below) requesting rate limiting. The sliders are logarithmic to allow fine control over a large range of rates. The far right position is "unlimited" whereby the senders are instructed not to limit their data generation.

The Debug Hooks popup is used to expose various debugging hooks in the emulator. In particular text / graphics packet statistics can be gathered and information about incoming T/G

packets can be displayed. The Flow Control options allows the user to select if late packets are dropped as the gateway would drop them for flow control. The "RETRANSMIT SKIPPED PACKETS" controls if NACK packets are sent back to the sender. Finally "PRINT TIMESTAMP ADJS" displays the timestamp offsets used to cancel out clock skew. The BER Injection is used in conjunction with the BER popup window (not shown) which can be used to automatically corrupt the data stream at a specified Bit Error Rate to observe the effects on protocol reliability.

9.4. Text / Graphics Server

The Text / Graphics algorithms were developed by augmenting an X Window server to support the Split X design[1]. The X Windows server was amenable to modification due to its modular architecture and freely available source code. The X Windows server is designed to be ported to different display adapters, and thus the InfoPad port is implemented as a new display adapter. Upon initialization, a new slave thread is spawned which communicates via InfoNet to the pad. The virtual framebuffer is placed in shared global memory and semaphores and mutexes are used for synchronization. The manual page for XInfoPad can be found in Appendix A.11..

9.5. Video Support

The VQ video subsystem of the InfoPad system was developed using a set of utilities which generate VQ video clips, play the clips on standard workstations, play the clips through the InfoPad network to an emulator or hardware pad, and manipulate and display the clips. The manual pages of the applications to perform these tasks are listed in Appendix A and the applications are described below.

1. Note that Richard Han and/or Brian Richards did the original work on the integration of InfoPad hooks into the X server although I expanded upon it significantly through the implementation of the virtual framebuffer algorithms, color support, as well as support of transmission of block, instead of just line, packets.

VQ video files used to save and exchange the VQ video clips are stored in a custom file format detailed in Appendix A.13. (*vq (5)*). The video format allows arbitrary dimension video clips with codebook updates both at the beginning and throughout the video clip. The latter would allow for new VQ codebooks to be sent between major scenes for improved image quality. Video files can also be exchanged in a RAW format, described in Appendix A.12. (*raw_video (5)*), which allows for arbitrary dimensions of the Y, I, and Q components. These RAW videos can then be transcoded into VQ videos as described below. Thus they are a useful interchange format which allows non-InfoPad applications to generate VQ videos.

VQ videos can be generated from MPEG and RAW clips using `mpeg2vq` as detailed in Appendix A.4. (*mpeg2vq (1)*). `mpeg2vq` allows resizing of video clips and transcoding from particular subregions of the source clip. It allows the video frame rate to be specified as well as frame dropping to obtain a desired frame rate. `mpeg2vq` can generate adaptively-coded or fast-coded video clips as discussed in Section5.2.. Additionally, video clips can be coded to an existing arbitrary codebook.

VQ and RAW videos can be played on standard X workstations using vq_play as detailed in Appendix A.9. (*vq_play (1)*).

VQ clips coded for a 128x240 display can be sent to an InfoPad terminal or emulator using the `send_vq` utility as described in Appendix A.6. (*send_vq (1)*). send_vq can be used to inject text / graphics, audio, and other types of data into the InfoPad network for general purpose debugging. Rate limiting options can be specified to prevent overflow on bandlimited links and codebook information can be included or overridden. send_vq will also deliver synchronized audio and video if an audio file is specified.

VQ codebooks can be displayed and extracted using the `Codebook2ras`, `show[vq]codebook[y]`, and `vq2codebook` utilities as described in Appendix A.1., Appendix A.7., and Appendix A.10.. `Codebook2ras` converts a codebook into a Sun Rasterfile which can then be viewed. The codebook images in this thesis were generated with that utility. `show[vq]codebook[y]` is a shortcut to extract and display the codebooks. Finally `vq2codebook` extracts a codebook from a VQ video clip. This codebook can then be sent manually using `send_vq` or another video can be coded to this codebook using `mpeg2vq` as previously described.

PART III *Application-Specific Transmission*

CHAPTER 10 *Optimization of Web for Bandlimited Links*

10.1. Introduction

This chapter applies the techniques and methodology developed for application-independent text / graphics and image transmission to the web. Since web browsing is an interactive process and downloading a web page can take several seconds to several minutes over slow links, the information presented to the user during this time is important. New metrics and visualization techniques to illustrate and quantify web page loading are presented. Given the insight afforded by the metrics, a methodology to improve web access using a new technique, globally progressive interactive web delivery, is proposed. This technique views the web delivery process as the remote display of a web page, similar to application-independent transmission previously described, and entails applying progressive coding to the document transmission process in its entirety. It also allows the user to explicitly direct link bandwidth to images of interest. This glo-

bally progressive interactive framework has been prototyped without modifying either existing web browsers or servers through the use of a web proxy and browser-side Java applets. The framework allows for both protocol and image compression research in a platform-independent manner. Methods for integrating the architecture into existing web infrastructure for greater performance and ability to scale are discussed [31].

10.2. Motivation

Delivery is everything. "The web experience" is much more than the web pages visited - it also encompasses the speed of access and the quality of information delivered. While browsing over a T3 line can be highly productive, web access over dial-up or wireless modems often leads to the phenomenon known as the "World Wide Wait". Network congestion and bottlenecks within the Internet can also limit the gains of a high-speed last link. However, as will be shown, web performance over these slow links can be dramatically improved through efficient link scheduling and data compression tailored to web transmission. Although some previous work has accelerated web access through lossy image compression [25][41][65], this, by necessity, results in a reduction in image quality. Transport protocol modifications designed to reduce the total web page delivery time [52][9][54] also improve web performance, but further gains can be achieved by combining image coding and networking techniques as described in this chapter.

It is important to view web browsing as a form of *remote display*, similar to the application-independent transmission described previously in this dissertation. Browsing is not the bulk transfer of information for off-line storage, but rather the "real-time" rendering of a web page on a remote client. Thus the speeds at which various parts are rendered are useful metrics to gauge the browsing experience.

Web browsing is an *interactive* process where the user often has specific goals when browsing. The goals may be to read the text, glance at all of the pictures, or closely examine a single picture. In order to maximize the utility to the user, it is important to incorporate these goals as best as possible. Currently the level of interaction is limited to aborting the downloading of a page or suppressing all image downloads until they are explicitly requested. Thus, the user is at the mercy of the delivery system and must wait for the image or images of interested to be delivered. A method is proposed for incorporating feedback to allow the user to guide the bandwidth utilization throughout the downloading process: by simply moving the mouse cursor into an image's window, the image will gain full use of link bandwidth.

Incremental deployment is essential in global Internet applications such as the web. It is infeasible to expect all existing web browsers and servers to be converted. Two levels of incremental deployment of the protocols are proposed. Firstly, by deploying intermediate proxies to perform image and protocol conversion, the improved delivery system can be used on slow links with modified web browsers to view content on conventional web servers. Secondly, a web proxy / Java applet prototyping framework is used which does not require the web browsers or servers to be modified. The proposed system is benchmarked and a discussion of the strengths and weaknesses of the framework is presented.

10.3. Background / Previous Work

Web pages are multi-object documents. They consist of a main HTML object, which contains the text and formatting information, and zero or more additional objects such as inline images and Java applets. The main HTML object contains references to other objects by specifying their uniform resource locator (URL). These objects are retrieved individually by the web browser from the web server using the HyperText Transfer Protocol (HTTP).

HTTP/1.0 [11] uses a separate TCP connection to retrieve each object. Several concurrent connections are maintained between the browser and server to load multiple objects at once in order to hide TCP connection establishment and slow-start [42] delays. However, this increases overhead and network congestion, thus impacting scalebility [66]. Typically the number of open connections is limited to about 4.

HTTP/1.1 [23] improves upon the delivery protocol through support of persistent connections [54] between the servers and browsers. By using a single connection to sequentially deliver all of the HTML and images in a web page, the connection establishment and slow-start delays can be amortized over the cost of the entire page, the connection packet overhead can be reduced, and better responsiveness to network congestion is achieved. HTTP/1.1 also supports compression of the HTML objects via the deflate coding of the public domain zlib compression library [28]. (This is a hybrid LZ77 [72] / Huffman coding [39]). When applied to HTML data, it can typically result in a greater than 3x reduction in the size of the HTML file [52].

TCP sessions [9][53] have been proposed to allow sharing of state between related TCP connections, such as those connecting the same host pair, at the transport layer. If any connection in the session experiences congestion, all connections can reduce their windows. In this way the network-friendly behavior of HTTP/1.1 can be obtained while decoupling sensitivity to losses between different image transfers. Additionally, only the server-side TCP stack needs to be modified - no browser or server modifications are necessary.

The MUX protocol [51], a work in progress which is part of the next generation HTTP effort (HTTP-ng) [50], provides a method to layer lightweight multi-session delivery on top of a reliable stream-oriented transport such as TCP. The MUX protocol can be used by multiple applications to share the same transport stream as the main HTTP connection in an application independent manner.

The WebTP project [70] is addressing transport requirements of the World Wide Web through their "User-Centric Web Transport Protocol". This is a receiver-driven architecture to improve upon scaling of web-servers while retaining TCP-friendliness. It uses rate-based flow control and does not require hard state at the sender.

10.3.1. Previous Work in Web Acceleration

UC Berkeley's Transend [25][26], Intel's QuickWeb [41], and Spectrum's FastLane [65] improve upon web access over slow links by reducing image size via lossy compression[1]. Web proxies are used to transform the images through resolution and color reduction. Users of Quick-Web have noted that it can cause significant degradation of image quality [44] and the service has been discontinued. Use of FastLane also results in image quality degradation [7]. The three systems allow the user to explicitly load the original undegraded images, but no further user interaction is supported. In contrast, the globally progressive interactive system described here does not cause reduction of the final quality of the images delivered, and allows greater user interaction.

10.4. Quantifying Web Page Downloading

Since downloading web pages can take from several seconds to several minutes using modem links, the information presented to the user during this period is critical. For instance, if it requires one minute to download a page, it is clearly better to have most of the text and images present after 10 seconds with the remaining filling in over the minute, than to have nothing until the whole page appears at the one minute mark. Yet in both cases the total time to load the page is the same. **Thus the total time to load a given page is often not an accurate measurement of the utility of the loading process.**

1. Transend is also designed to operate with clients of varying computational and display capabilities and perform other types of manipulations.

Object	Name	Bytes	Object	Name	Bytes
HTML	index.html	41,261	Image1	horoscopes120.gif	4,313
Image2	ad_info.gif	92	Image3	still_468x60.gif	9,374
Image4	cnnin_logo.gif	2,225	Image5	icon_arrow_left.gif	261
Image6	white.gif	35	Image7	roof_top.gif	1,205
Image8	search_infose.gif	441	Image9	right_corner.gif	102
Image10	top_ruins_ap.jpg	17,774	Imaeg11	custom_clint.jpg	6,112
Image12	video_transp.gif	170	Image13	cnn_website.gif	2,980
Image14	web_services.gif	11,623	Image15	custom_arrow.gif	137
Image16	tv_generic.gif	1,291	Image17	health2.gif	2,124
Image18	thumbnail.jpg	2,580	Image19	week_in_review.gif	986
Imge20	diana.jpg	2,507	Image21	infoseek_logo.gif	555
Image22	pointcast.gif	6,983	Image23	red_468.gif	10,486

TABLE 10.1. Contents of typical web page (CNN Interactive - www.cnn.com). Shading denotes images not initial visible when viewed on a 1024x768 screen.

In order to fully encapsulate the loading process as viewed by the user, it is necessary to visualize and quantify the loading of the constituent parts of the page, and not just the time to load everything. A *web-page loading graph* can be used to more precisely illustrate and quantify the effects of different transfer protocols on a typical web document. The CNN Interactive (tm) home page (http://www.cnn.com/) will be used as an example. The constituent objects are listed in Table10.1 in order of their appearance and the shaded entries correspond to images that the user would have to scroll to see when using a 1024x768 pixel web browser window because they are not initially visible.

10.4.1. An Example of Concurrent HTTP/1.0-Style Loading

Figure 10.1 shows a web page loading graph of the document loaded using the conventional HTTP/1.0-style concurrent loading protocol. The graphs in this section display timings using a simulated constant 3KB/sec link similar to a 28.8kbps modem, in order to best illustrate the perfor-

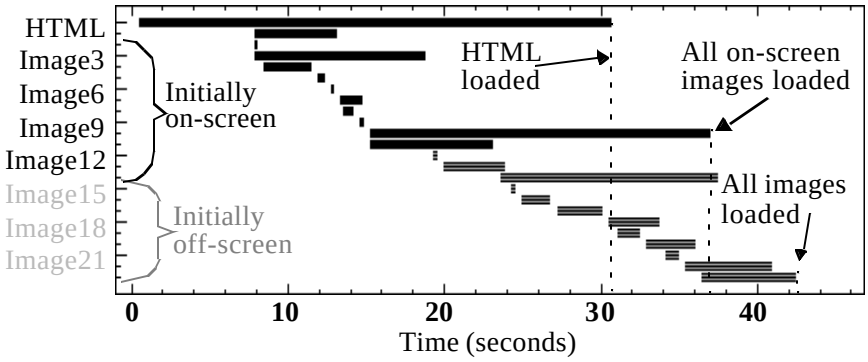


FIGURE 10.1. Web-page loading graph using concurrent loading of up to 4 connections

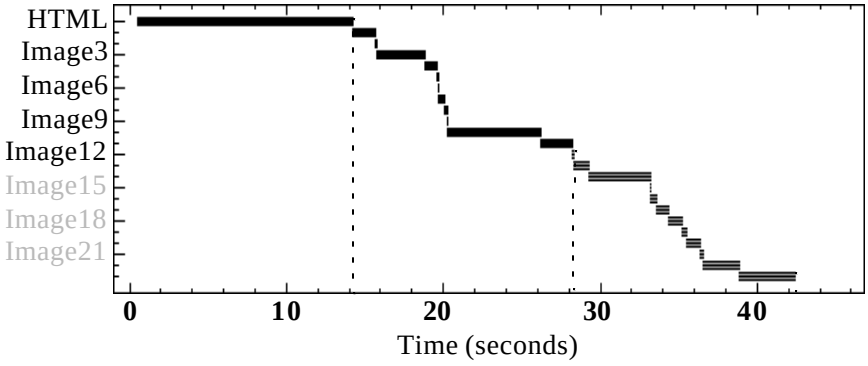


FIGURE 10.2. Web-page loading graph of sequential loading protocol

mance of the different loading styles apart from implementation specifics. The results shown in Section 10.6.5. are based on actual data collected from our prototype proxy / applet system. The objects in the page are listed on the vertical axis while time traverses the horizontal axis, with the lighter gray used for images outside of the viewable window. The bars indicate the time during which a particular object is being delivered.

In order to quantify the loading process, it is useful to define the following metrics: the time that the HTML is loaded, the time that initially visible images are loaded, and the time to load the entire page. The time to load the HTML determines how long it will take to view the text on the page, and the time to load the initially visible images determines when the user thinks that the page has loaded. In the example we can see that the HTML is loaded within 30.6 seconds, all visible

	Time to text	Time to visible document	Time to entire document
Concurrent	30.6 sec	36.9 sec	42.4 sec
Sequential	14.3 sec	28.2 sec	42.4 sec
Conc. w/comp.	6.6 sec	29.4 sec	31.8 sec
Seq. w/comp.	3.7 sec	17.7 sec	31.8 sec

TABLE 10.2. Summary of concurrent and sequential loading

images are loaded within 36.9 seconds, and the entire page is not loaded until 42.4 seconds have elapsed.

10.4.2. An Example of Sequential HTTP/1.1-Style Loading

The web-page loading graph for the same web document loaded using the conventional HTTP/1.1-style *sequential loading protocol* is shown in Figure 10.2. Using the sequential loading, the total time to load the entire web document remains 42.4 seconds but the time to see the visible images has dropped to 28.2 seconds, simply by reordering the data being sent. Additionally, the time to see the text is reduced to 14.3 seconds - less than half of the time required using concurrent loading. Thus, despite the fact that the total loading time is the same as the previous case, the delivery orders have very different consequences for the user.

10.4.3. HTML Compression

If HTML compression is used, the size of the HTML object in our example is reduced by about a factor of 4, from 41261 to 9950. The corresponding web-page loading graphs are shown in Figure 10.3 and Figure 10.4. While the total time to load the document is only reduced by about 25% from 42.4 seconds to 31.8 seconds, the time to see the text is reduced by a factor of 4 and the time to see the entire visible part of the document is reduced by about 20% and 40% for the concurrent and sequential modes respectively. However, the delivered document is identical to the original. These results are summarized in Table10.2.

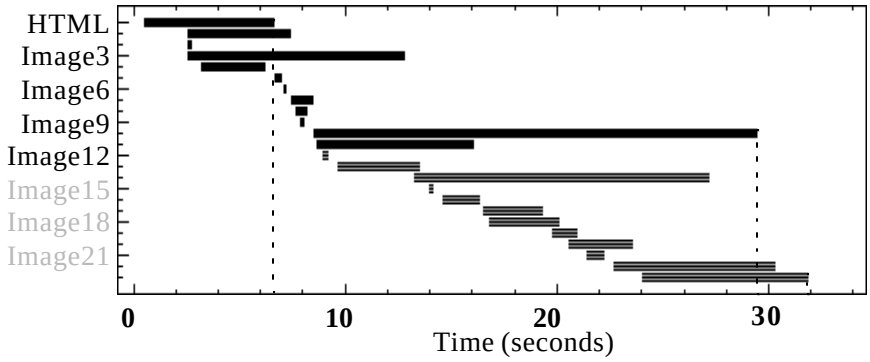


FIGURE 10.3. Concurrent loading and HTML compression

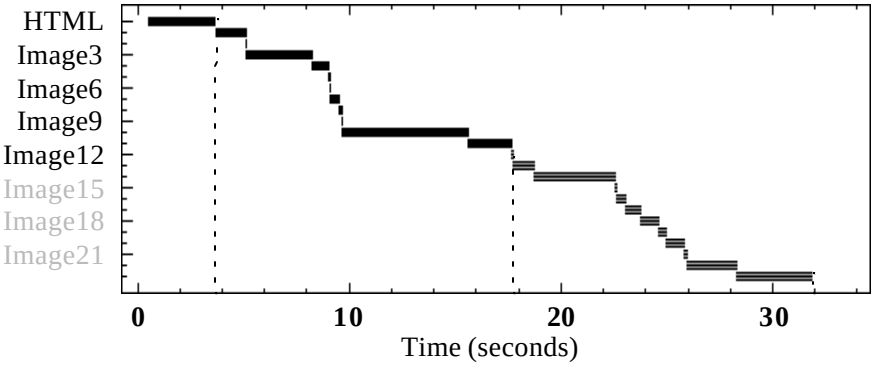


FIGURE 10.4. Sequential loading and HTML compression

10.5. Globally Progressive Interactive Web Delivery

Once metrics and methodology are in place to examine web document delivery, it is possible to investigate alternatives. Although sequential transmission is an improvement over concurrent delivery, using progressive and interactive techniques, significant improvements can be realized.

10.5.1. Globally Progressive Transmission

Progressive image transmission entails sending a layered coding consisting of a low quality version initially, followed by refinements. Thus, after the initial bytes are received, a complete, albeit non-aesthetically pleasing, version can be displayed, and as time progresses, the image qual-

ity improves. Progressive techniques are ideally suited for transmission of human-viewed data over band-limited links since they allow the user to quickly deduce the salient features and only require the user to wait for the full load times to see full detail.

10.5.1.1. Existing Progressive Image Formats

Progressive image formats are already common on the web. CompuServe's lossless Graphical Interchange Format (GIF) [17] has an "interlaced" mode in which rows are sent in a progressive manner. Portable Network Graphics (PNG) [59], a graphics format which has been formally accepted as a standard MIME type by the World Wide Web Consortium, has a mode that is progressive in both rows and columns to achieve better subjective quality after a smaller amount of data has been received. Both GIF's and PNG's progressive modes result in some reduction in compression rate due to the increase in local entropy. PNG also enjoys a patent-free status and typically achieves 5%-25% better compression than GIF [56] by using zlib's LZ77 / Huffman compression instead of GIF's Lempel Ziv Welch (LZW) [72][71] compression and packing sub-byte pixels.

The Joint Picture Expert Group's JPEG includes a lossy DCT-based coding technique which has a progressive mode that divides layers by spatial frequency and quantization level [56]. The number of layers and their composition can be varied, and are specified in the image header. Figure 10.5 shows an example comparing JPEG's non-progressive and progressive (PJPEG) modes. The baseline JPEG quantization levels are used, resulting in a coding rate of 1.84 bits/pixel which, from an informal survey, is typical of JPEG images on the web. As can be seen from the lower set of images, a very crude version of the image is available after receiving only 8% of the progressive data. After 19%, enough detail is available to easily discern the contents of the image. Subsequent data provides further detail, with the later layers only becoming noticeable

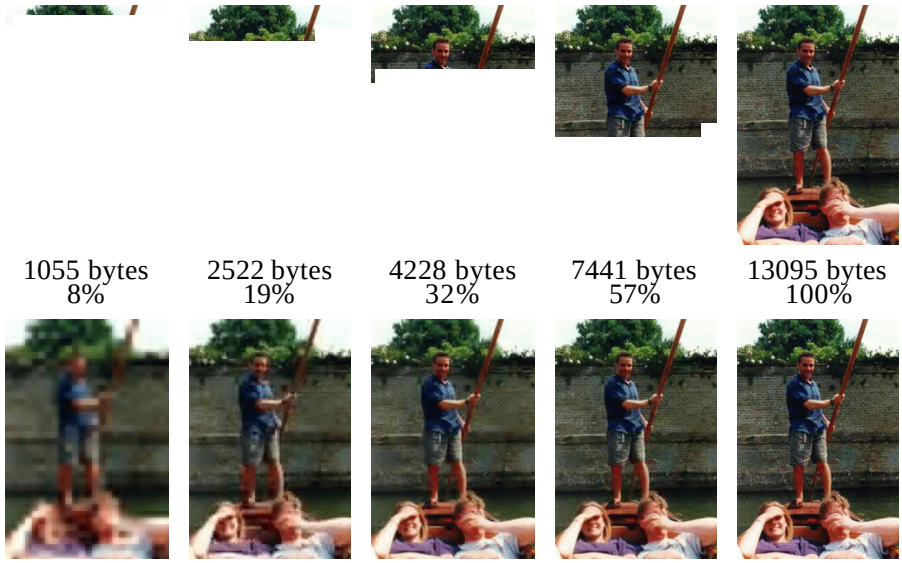


FIGURE 10.5. Example of progressive images.
JPEG (top) and PJPEG (bottom) showing display of incomplete data. Image was quantized to JPEG default level yielding 1.84 bits/pixel coding.

under closer inspection. It is important to note that this is exactly what lossy image compression proxies such as QuickWeb, TranSend, and Fastlane rely upon - by performing lossy image compression they are removing some of the fine detail, similar to that in the final layers of PJPEG. While the degradation is not severe to cursory inspection, it does reduce the final image quality. However, by using progressive techniques, the end quality need not be sacrificed at all.

10.5.2. Locally Progressive Delivery

Progressive image formats are of limited benefit when using standard loading protocols since they are *locally progressive* - each image is progressively coded but the document as a whole is not. Otherwise stated, the loading process itself is not progressive. Figure 10.6 and Figure 10.7 show web page loading graphs of locally progressive images loaded concurrently and sequentially. The lighter parts to the left correspond to coarser layers while the darker parts to the right correspond to refinements. For illustrative purposes, the layers have been divided according to the division of the lower images in Figure 10.5.

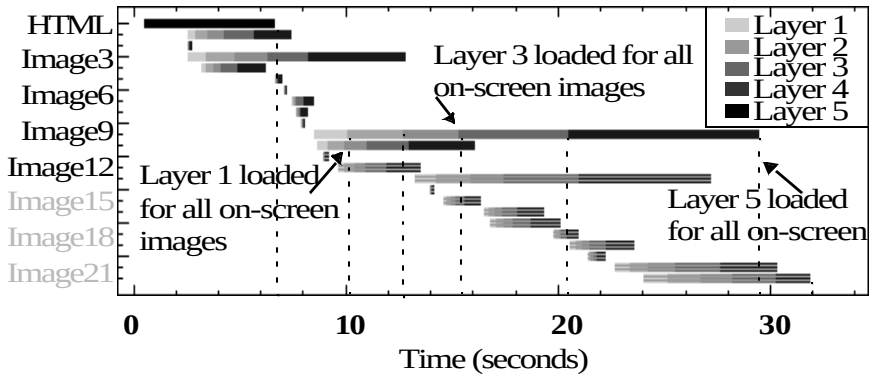


FIGURE 10.6. Simul. loading w/ locally progressive images (w/ HTML compression)

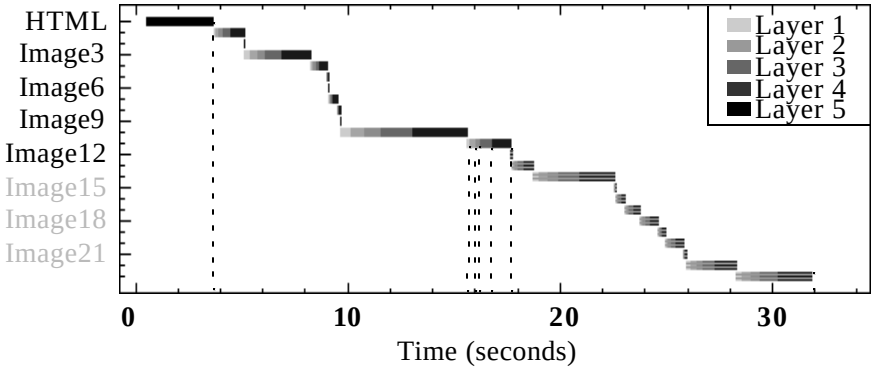


FIGURE 10.7. Sequential loading with locally progressive images (w/ HTML compression)

The amount of time to deliver enough of all of the images to see a certain number of layers in each image can be used to quantify the progressive delivery. As can be seen from Figure 10.6, the time to concurrently load at least the first layer of each of the images is 10.1 seconds, to load the first two layers is about 12.7 seconds, three layers is 15.3 seconds, four layers is 20.5 seconds, and the time to load all layers for all visible images is still 29.4 seconds. Figure 10.7 shows that the layers in each image for the sequential case are loaded in rapid succession since the images have access to the full link bandwidth.

10.5.3. Globally Progressive Delivery

However, by using *globally progressive* loading, further benefits can be achieved. Globally progressive loading considers the whole document as a progressive object and displays a coarse

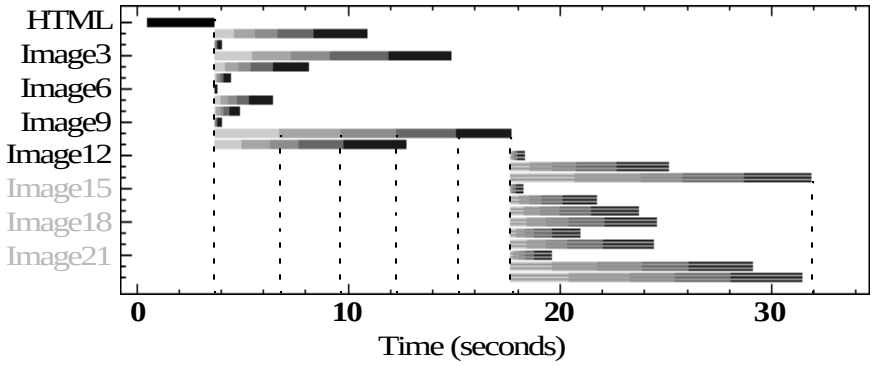


FIGURE 10.8. Globally byte-wise progressive loading (w/ HTML compression)

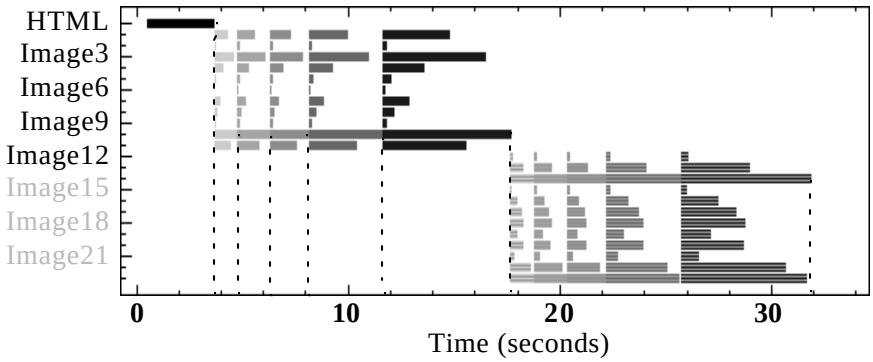


FIGURE 10.9. Globally layer-wise progressive loading (w/HTML compression)

version followed by more refined versions. In the case of a web document, the HTML might be considered the coarsest layer since it is required to decode the rest of the document and conveys the bulk of the information. The next layer of the document would include the first layer of all images and subsequent layers of the document would include subsequent layers of each image. By using globally progressive loading, the user is very quickly presented the text and coarse versions of all visible images and can rapidly proceed to analyze the page's contents. This is particularly useful when combined with interactive loading described in the next section.

Figure 10.8 shows globally progressive loading using a byte-wise equality metric such that the scheduling attempts to keep the number of transmitted bytes of each visible image similar, and Figure 10.9 shows globally progressive loading using a layer-wise equality metric dictating that a

Protocol			Timings						
HTML Compr	**Concur / Sequen**	**Progressive**	**Text**	**Visible Layer 1**	**Visible Layer 2**	**Visible Layer 3**	**Visible Layer 4**	**Visible All Layers**	**Complete Document**
No	Conc	No	30.6 sec	n/a	n/a	n/a	n/a	36.9 sec	42.4 sec
No	Sequen	No	14.3 sec	n/a	n/a	n/a	n/a	28.2 sec	42.4 sec
Yes	Conc	No	6.6 sec	n/a	n/a	n/a	n/a	29.4 sec	31.8 sec
Yes	Sequen	No	3.7 sec	n/a	n/a	n/a	n/a	17.7 sec	31.8 sec
Yes	Conc	Locally Progressive	6.6 sec	10.1 sec	12.7 sec	15.3 sec	20.5 sec	29.4 sec	31.8 sec
Yes	Sequen	Locally Progressive	3.7 sec	15.8 sec	16.0 sec	16.3 sec	16.8 sec	17.7 sec	31.8 sec
Yes	**Both**	**Bytewise Globally Progressive**	**3.7 sec**	**6.8 sec**	**9.6 sec**	**12.3 sec**	**15.1 sec**	**17.7 sec**	**31.8 sec**
Yes	**Both**	**Layerwise Globally Progressive**	**3.7 sec**	**4.8 sec**	**6.3 sec**	**8.2 sec**	**11.7 sec**	**17.7 sec**	**31.8 sec**

TABLE 10.3. Summary of delivery methods and performances

given layer in one visible image should not be loaded until the previous layers are loaded in all other visible images. While the layer-wise metric allows earlier layers to be loaded for all images sooner, the byte-wise metric is typically preferable since it allows link scheduling to be independent of image coding and also prevents large images from severely delaying smaller images. For these reasons, the byte-wise metric is used in our prototype system described in Section10.6..

Table10.3 shows a summary of the comparison between locally and globally progressive loading. As can be seen, the time to load all visible layers is the same for the sequentially loaded locally progressive case as the globally progressive cases, but in the locally progressive case, the benefits of progressive loading are minimal. In the simultaneously loaded, locally progressive case, there is a delay until the base layers of some of the later images are loaded, as well as a delay in finishing the visible images due to competition with images that are not visible. However, in the globally progressive cases, the text is shown very quickly, followed immediately by coarse versions of all images and then refinements.

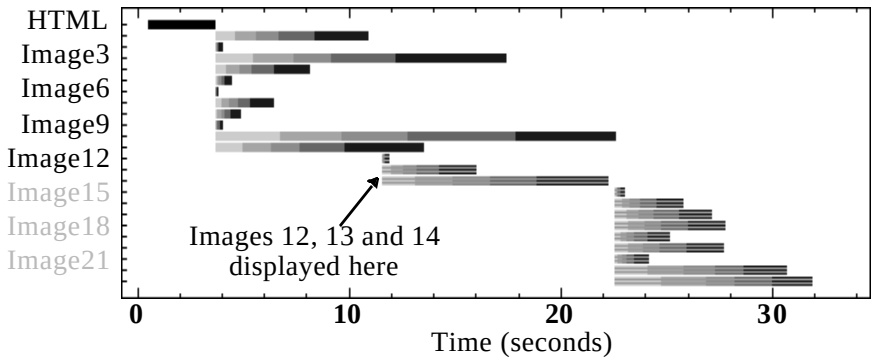


FIGURE 10.10. Globally progressive loading with images scrolled to during loading

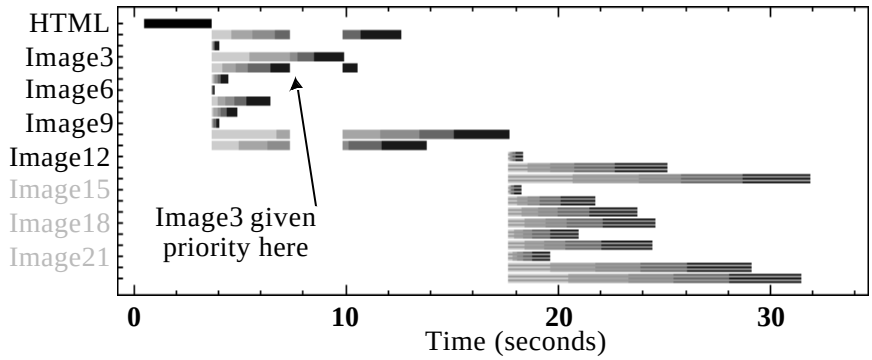


FIGURE 10.11. Interactive loading with image interactively selected by user

If an image that is not initially displayed is made visible during the loading process, it is boosted to the same priority as the other visible images so that bandwidth is initially dedicated to only that image until it has loaded to the same degree as the other images. Then all visible images resume loading in unison. An example is shown in Figure 10.10. In this example, images 12, 13 and 14 were previously off screen but then scrolled into view after about 12 seconds. Similarly, the priority of images that are scrolled off screen can be reduced[1].

1. However, detecting images scrolling off-screen is not possible using the Java prototype.

While the above assumed strict priority scheduling, a stochastic algorithm such as Lottery Scheduling [69] could be used to assure progress of all images. Additionally, techniques such as Class Based Queuing (CBQ) [24] could be used to increase flexibility and allow more complex policies. Web page designers could also incorporate delivery and image transcoding hints into image tags to assert more control over the delivery process.

Globally progressive delivery is also well suited to take advantage of networks with variable Quality of Service (QoS). There is an implicit ordering of the importance of the data with text being most important, initial bytes or layers of images being a little less important, later bytes or layers being still less important, and off-screen images being even less important. While it is currently used to prioritize data within the web connection, it could also be used to prioritize data transmission across multiple connections, web or otherwise. For instance, if text over any web connection is given higher priority than images over any web connection, then even as the network becomes congested, the text delivery performance will not suffer as much. Likewise, if off-screen images are given lower priorities, the downloading of long pages by some users will not hamper the interactive operation of other users. The prioritization could be used within a web server to ensure timely servicing of text and coarse image requests.

10.5.4. Interactive Operation

While globally progressive loading rapidly delivers coarse versions of all visible images, the user must still wait for all refinements, even if they are only interested in a single image in detail. However, by allowing the user to easily instruct the system as to which image they are interested in, this image can be loaded more quickly by dedicating all available link bandwidth to it. Since the layers are loaded in synchrony, the user is able to quickly determine which image or images are most interesting. One method for incorporating user image preference is to detect when the mouse cursor is inside a particular image, and give that image priority. Figure 10.11

shows the effect of selecting image 3 after its two first layers have been displayed. As can be seen, in this way, the entire refined image can be loaded by the 10 second mark, a little more than half of the time required in the absence of user intervention. Further interactivity could include targeting exactly which parts of the image are transmitted first. Explicit targeting may be useful in cases of large images where semantic quality requires high fidelity, such as maps.

When interaction is allowed, the amount of data buffered in the connection between the web server and web browser must be limited in order to allow rapid response. Once the user preference is detected and transmitted to the web server (or proxy), the high-priority image can be queued, but this data will not reach the browser until all other data queued in that connection has been delivered (unless out-of-band signalling or more sophisticated transport protocols are used). For instance, using a 28.8k baud modem connection, in order to obtain a one-second response time, at most 3.6K bytes can be buffered even in the absence of network congestion. The buffered data consists of the data queued in the kernel buffers as well as the packets queued in the network routers. Kernel-level scheduling of the images can be used to eliminate delay due to the former[55] while TCP window-size limiting would have to be used to reduce the latter. Severely limiting the amount of queued data can reduce link utilization, particularly for high-bandwidth, high-latency links such as satellite links.

10.6. Transport Protocol Prototyping via Web Proxies and Java Applets

In order to prototype the globally progressive interactive delivery scheme, a proxy-based architecture allowing full control over image delivery and display was designed. This architecture can be used as a test-bed to develop and experimentally deploy a range of network protocols not restricted to HTTP or even TCP. Additionally, non-standard image compression techniques can be used over the link.

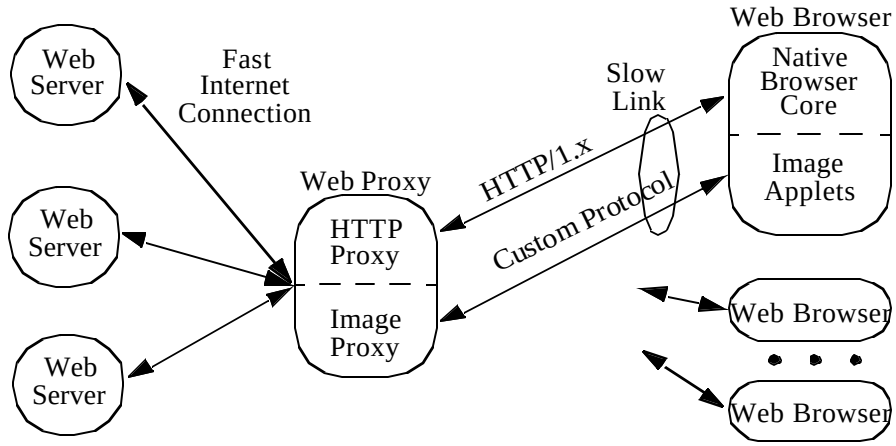


FIGURE 10.12. Web proxy / Java applet framework

10.6.1. Proxy Operation

While conventional web proxies can transform the content of the web pages, they must work within the confines of the HTTP protocol specifications. For instance, they cannot alter the number of connections that the browser opens, change the ordering of the requests, or respond to fine-grained user interactivity - thus precluding implementation of a protocol similar to the one just described. However, through the use of Java applets to load and display the images on the page, much greater control is possible. A block diagram of the framework is shown in Figure 10.12.

The globally progressive interactive delivery is best implemented by using a single multiplexed, prioritized connection between the web browser and web server or proxy. However, in the case of the proxy / applet system described here, separate HTML and image connections are required to allow the HTML to go directly to the browser core while the images are sent to the Java applets over a single multiplexed connection.

Original HTML:

```
For my homepage, click <A HREF=/~me> here <IMG
SRC=home.gif></A>
```

Modified HTML:

```
For my homepage click <A HREF=/~me> here
<A HREF="~me">
<APPLET CODE="SpeedImage.class"
   WIDTH=54 HEIGHT=39
   ARCHIVE="http://proxy.com/SpeedImage.jar"
   CODEBASE="http://proxy.com/">
   <PARAM NAME=imageHostPort VALUE=9999>
   <PARAM NAME=src
       VALUE="http://home.com/~me/home.gif">
   <PARAM NAME=href
       VALUE="http://home.com/~me">
</APPLET>
</A>
```

FIGURE 10.13. Example of HTML modification to embed image applets

10.6.2. HTTP Proxy Design

The HTTP Proxy substitutes Java applet tags for image tags, as shown in Figure 10.13, as it sends pages to the browser, causing applets to appear where the images were. The applets are supplied the URL of the image to display as well the host name and port of the Image Proxy. If the width and height are not specified in the IMG tag, they are determined by prefetching the image and decoding its header since the applet dimensions *must* be specified in the APPLET tag (unlike IMG tags where the dimensions *should* be specified.) HTML compression is achieved via HTTP/1.1 transport encoding.

10.6.2.1. Streaming HTML Conversion

One important technique used in both the HTTP Proxy and the Image Proxy is that of *streaming conversion*. Streaming conversion entails converting both images and HTML text "on-

the-fly" whenever possible, instead of first retrieving the entire objects before processing them. This is critical if the response from the web server is slow due to network congestion or server loading. If the proxy waits for the entire object to be received before sending any part of it out, the time for the user to receive any part of it will be substantially increased, and the total time to retrieve the object can be doubled. For the HTTP Proxy, streaming conversion means that the HTML is transformed and compressed on the fly. While compression and simple textual substitution would not be problematic, incorporating image size information in applet tags requires asynchronous retrieval and parsing of image headers to determine their sizes.

10.6.2.2. Link Scheduling

The HTTP Proxy performs limited link scheduling by tracking the amount of data outstanding in HTML and non-HTML (typically image) links. It does not send data on non-HTML links until the amount of HTML data outstanding is below a given threshold. The image proxy also suppresses custom image data until the amount of HTML data outstanding is below a given threshold in order to effect the text prioritization needed for globally progressive transmission as described in Section10.5. The fine-grain inter-image scheduling takes place in the Image Proxy.

10.6.3. Image Proxy Design

The Image Proxy is responsible for retrieving images from the web servers, transforming them, and sending them over the custom managed link to the Image Applets. A strict priority round-robin system is used with priorities dynamically specified by the Image Applets depending on whether they are on-screen and where the cursor is.

10.6.3.1. Image Conversion

As the images are retrieved, their type is determined from the HTTP meta-data. Type-specific conversion is performed to generate progressive versions. JPEG images are converted to Pro-

gressive JPEG (PJPEG) using the Independent JPEG Group's JPEG library [40]. This conversion is lossless and does not significantly effect compression rate. GIF images are converted to a lossless format similar to interlaced PNG.

Additionally, conversion from GIF to PJPEG is attempted and the PJPEG image is used whenever a high-quality lossy JPEG conversion results in a reduction in size compared to the PNG-like coding. By keeping the JPEG quality setting high, discrete-tone images not well suited to lossy compression will compress less compactly with JPEG than with a lossless coding such as GIF or PNG, and thus the lossless coding will be used. However, photographic images are often stored on web sites in lossless GIF while lossy JPEG coding dramatically reduces the size of the image while not resulting in any perceptible degradation. Since progressive coding is used, a high quality setting can be used while still allowing quick delivery of a coarse version of images.

10.6.3.2. Streaming Image Conversion

As described previously, an important aspect of effective proxy operation is forwarding data as soon as possible to reduce user-perceived delay. However, conversion to progressive formats requires the entire image to be present. In order to minimize delay, the original data is streamed to the browser until the entire image is loaded and converted. A switch to the progressive coding is done when advantageous. For the case of JPEG to PJPEG conversion, a heuristic of requiring less than half of the data being sent is used. For the case of GIF to PJPEG conversion, the size of the PJPEG data has to be less than the number of bytes of the GIF image remaining. Typically the total increase in the amount of data sent due to streaming image conversion is not large, yet it can substantially improve perceived latency.

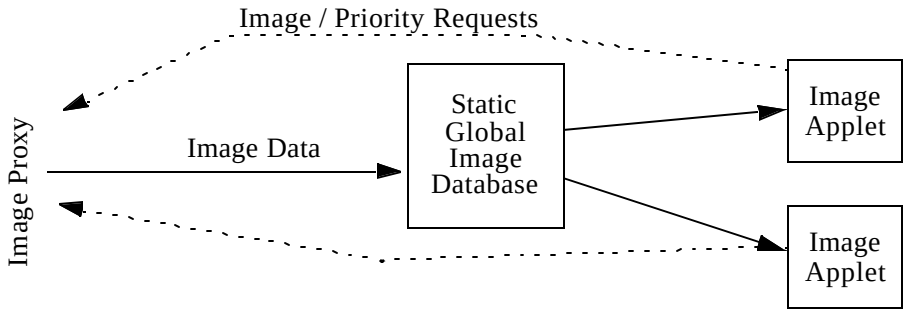


FIGURE 10.14. Image applet design

10.6.3.3. Link Scheduling

The Image Proxy explicitly manages the link to the Image Applets. The image priorities are determined by the Image Applets (described next) as dictated by the globally progressive interactive delivery scheme, and communicated to the Image Proxy. Within a priority class, images sending processed data (progressively coded images) are given priority to those sending unprocessed data since the unprocessed data may have to be flushed, as previously described, if the processing proves to be advantageous.

10.6.4. Image Applet Design

Image Applets request the data for their images from the proxy and display it. They are nearly indistinguishable from images they replace, responding to mouse clicks to follow web links as standard images do. The Image Applets also respond to keyboard commands to create new windows which are copies of the images as well as zoom and pan within the images. Thus the document is an active entity that can be manipulated.

Figure 10.14 shows the internal architecture of the Image Applets. There is one applet per image on the page, though the images communicate through shared static objects. Static Java objects are shared among all applets running in the same Virtual Machine - i.e. all applets running in the same browser. In particular, a shared image database is used to track which images have

Text	5.1 sec	Visible Layer 4	16.9 sec
Visible Layer 1	10.1 sec	Visible All Layers	20.0 sec
Visible Layer 2	12.3 sec	Complete Document	33.3 sec
Visible Layer 3	14.4 sec		

TABLE 10.4. Performance of Java / proxy system on example CNN Interactive page

been loaded already and allows the same image to be shown in multiple applets without requiring the image to be loaded more than once. The loading of the images is centralized via a single Java thread which contacts the Image Proxy and manages all browser-proxy communication.

10.6.5. Proxy / Applet Performance

In order to evaluate both conventional and proposed globally progressive interactive loading protocols, the SpeedSurfer client-side proxy described in is used as shown in Figure 10.15. The client PC was a Pentium II/266 PC running Microsoft Internet Explorer 4.0 web browser under Windows NT 4.0 and connected to the Internet via a 28.8k baud modem. The web proxy and local web server ran on a Sun UltraSparc 2 workstation.

The web page loading graph depicting loading the example page from a local server using conventional HTTP/1.0 is shown in Figure 10.16. The initial vertical line in each object's row indicates when the HTTP request is issued while the bars indicate response timing. The loading pattern has similarities to both Figure 10.1 and Figure 10.2 due to the interaction of the multiple simultaneous TCP/IP connections. Although up to four requests are open simultaneously, due to TCP/IP's adaptive congestion control, new flows receive less bandwidth than existing connections until equilibrium is reached.

The graph of loading the page using the globally progressive interactive technique via the proxy / applet prototype is shown in Figure 10.17 and summarized in Table10.4. The perfor-

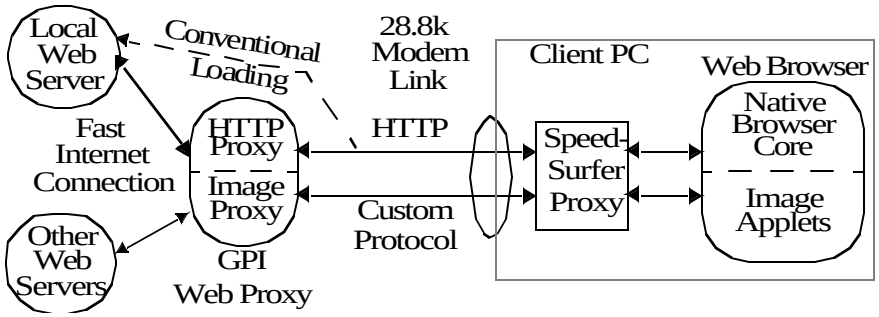


FIGURE 10.15. **Performance evaluation setup**

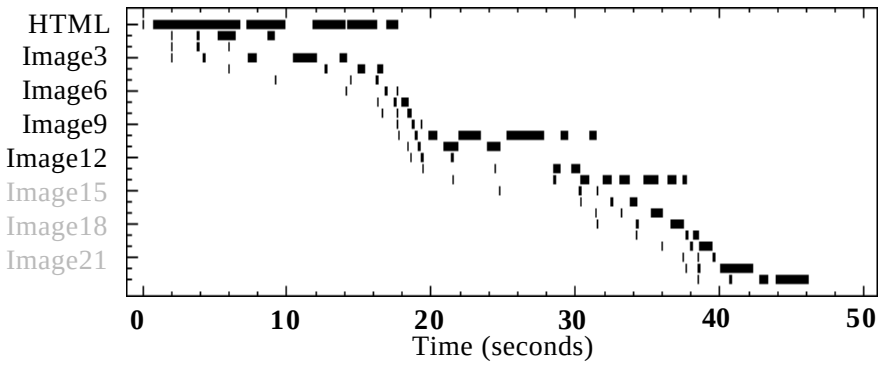


FIGURE 10.16. **Trace of conventional HTTP/1.0 concurrent loading**

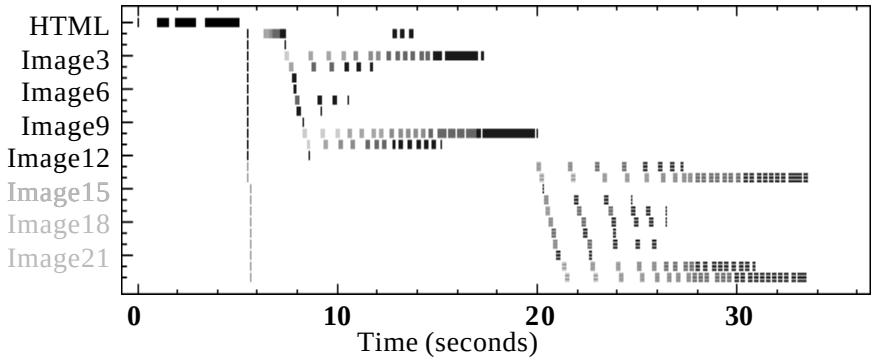


FIGURE 10.17. **Collected trace of proxy / applet operation**

Time (sec)	Amount (bytes)	Time (sec)	Amount (bytes)	Time (sec)	Amount (bytes)
1.543	1460	3.956	1460	5.098	1960
2.855	2540	4.857	2540		

TABLE 10.5. **Data Reception of HTML Object of Figure 10.17. Packet times are in seconds since request.**

mance is improved significantly over conventional loading but is slightly slower than predicted. Comparing Figure 10.17 to Figure 10.8 illustrates a few notable differences. The increased time to text can be seen in the segmentation of the HTML object bar indicating unsteady flow of data due to TCP/IP slow-start. The TCP/IP data reception timing shown in Table10.5 further verifies this. The increased time to the first layer of images is an artifact of the Java implementation: the web browser does not start the Image Applets until the entire web page has been loaded and thus their requests are delayed.

The time to perform on-the-fly compression and image conversion did not contribute significant overhead. Gzip compression of the 41261 byte HTML object to 9950 bytes required only 0.10 seconds while conversion of the 17774 byte / 47600 pixel top_ruins_ap.jpg image from JPEG to PJPEG required 0.12 seconds. Conversion of the same image from GIF to PJPEG would require 0.16 seconds. These all result in converted data rates of approximately 1 Mbit/second and thus a high performance workstation can support the compression and conversion for roughly thirty 28.8k baud modems. Caching and distributed computing could further increase scalability.

10.7. Conclusions and Future Directions

With the explosive growth of the Internet and increasing proliferation of low-bandwidth wireless access, efficient and expedient delivery of web content has become more important than ever. This optimized delivery is only possible through carefully analysis of the factors affecting loading speed. As has been shown, by viewing web delivery as a form of remote display, and

combining networking and image compression techniques, significant gains have been demonstrated. Additionally, this point of view yields opportunities for further developments as described below.

10.7.1. Integration with Existing Web Infrastructure

While the Web Proxy / Java Applet architecture described in the previous section is useful to evaluate and optimize the globally progressive interactive web delivery, further gains can be achieved by incorporating the methodology into existing web servers, proxies, and browsers and by building upon current HTTP protocols and related work described in Section10.3.. Public-domain open-source browsers and servers / proxies provide a state-of-the-art starting point [3], [48], [72]. The functionality of the Image Applets can be directly incorporated into the browser while the functionality of the web proxy can be integrated into the web server / proxy.

Interoperability is achieved through protocol negotiation on connection. A mechanism such as MUX can be used to support transmission of multiple images over a single link. One disadvantage of MUX using TCP is its use of a single buffered connection, which forces multiplexing to occur before buffering. This can result in additional buffering delay, impacting interactive switching of image priorities as described previously. Alternatively, TCP sessions could be employed if explicitly prioritized queuing is added [55]. TCP sessions have the advantages of application independence and removing false-dependencies across images. However, the use of TCP sessions requires server kernel modifications and may result in greater kernel overhead and resource utilization since a new TCP connection is required for each image.

10.7.2. Transparent Content Negotiation

The globally progressive interactive web delivery requires the ability to access both transcoded versions of images as well as the original. The underlying protocol must be able to

specify which image conversions should occur. While HTTP/1.0 provides very crude content negotiation via “accept” headers, it lacks sufficient expressive capabilities, which can lead to name-space conflicts. Transparent Content Negotiation, however, as described in [23] allows for a more flexible, powerful mechanism which allows “variants” of an object to be described and named differently than the original object.

Integration of an automatic content conversion mechanisms into a web proxy or server also allows delivery of improved formats, such as PNG, JBIG [4], [61], or wavelets [60], while retaining originals in the highly compatible legacy formats - GIF and JPEG. The web servers or proxies can automatically negotiate with the browser to determine the best mutually supported format, and perform conversion. In this way, the latest image coding techniques can be used without sacrificing compatibility.

10.7.3. Scalability through Server / Proxy Caching of Processed Data

Since the same progressive image codings are used for both high-bandwidth and low-bandwidth links, transcoded images can be cached and reused. When coupled with streaming conversion, this allows conversions to be delayed when the load on the server gets high, and yet assure that performance is never worse than without the conversions. However, if several users access a given web site via different web proxies, the objects on the site must be converted by each of the proxies. By integrating the functionality of the proxy into the server, the server / proxy can process the objects on the site only once per creation or update, regardless of how many geographically separated users access the objects. By transmitting the converted objects over the Internet, the delay of the wide-area access is mitigated by the globally progressive interactive delivery. The same scalability for dynamically created objects cannot be obtained but most images are not dynamically created, even at sites that use dynamically created content.

CHAPTER 11 *Application-Level Link Management*

In this chapter, techniques for optimizing the transmission of text / graphics information for a specific application are presented and compared to the application-independent methods previously described. While the application-independent techniques of Part II allow any application to be used remotely, further gains are often achievable by optimizing for the particular application. The specific application that is used as a case study in this chapter is a Java-based VLSI layout viewer called “WebChip”. This application is chosen because it is both information- and display-intensive and requires interactive operation.

11.1. WebChip - An Interactive Java-based VLSI Layout Viewer

WebChip is Java-based VLSI layout viewer that allows users to embed active layouts in their web pages instead of only using static images. It is information-based as it is designed to view a large layout database remotely. It is also display-intensive, particularly considering it is designed to operate on a relatively restricted Java virtual machine. While the implications of WebChip in terms of application-specific text / graphics and image transmission are described in this

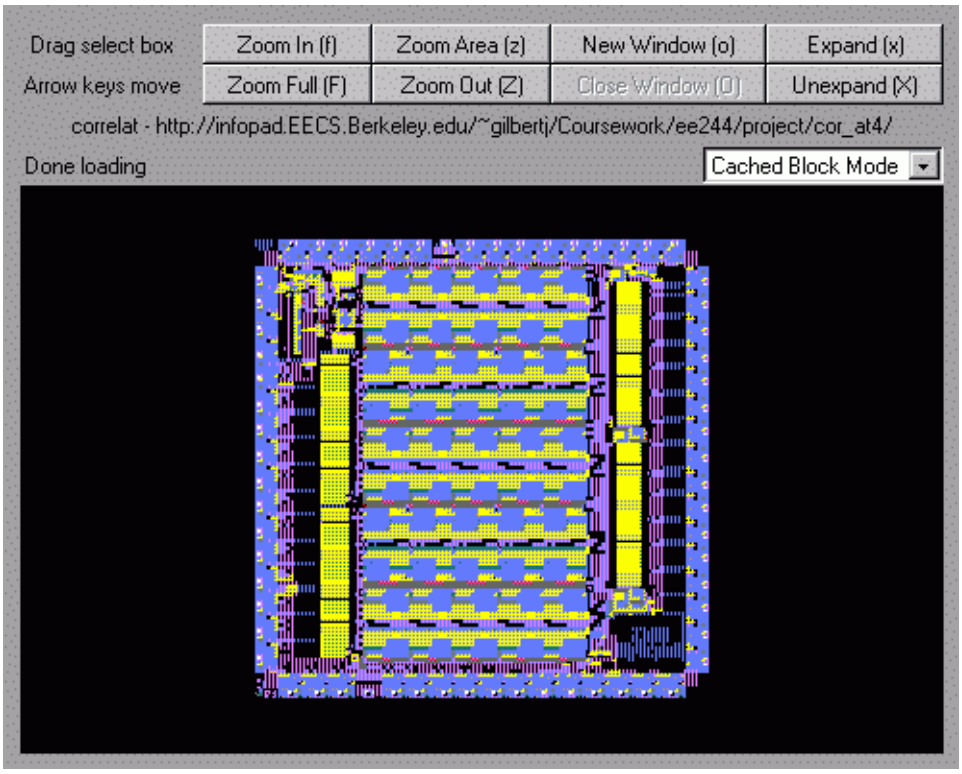


FIGURE 11.1. Example layout viewed with WebChip

chapter, further details about its use and operation can be found in Appendix B (*The WebChip Applet*).

WebChip, like other instances of application-specific text / graphics and image transmission, could be designed as a local client that did not know about remote transmission and instead relied upon a text / graphics server to perform application-independent transmission as described in earlier chapters. While this would allow full functionality, typically with reduced design time, it would also result in severely degraded interactivity, increased bandwidth utilization, and increased latency over low-bandwidth links.

Using application-independent transmission would require all layout data to be fully rendered before transmission and thus whenever the particular desired view changed, the image would have to be re-transmitted. Application-specific transmission allows client-side caching of view-invariant data, such as layout cells. Also, rendered data in the form of primitive draw com-

mands or bitmaps, is typically less compact then the source data it came from, since it has greater expressive capability.

The WebChip application is used to present application-level techniques to hide the effects of slow networking and image rendering. Many of the issues in application-independent text / graphics and image transmission as well as web-specific delivery find parallels in application-specific domain as described below.

There is a need to combine compression and link scheduling to most effectively utilize the limited bandwidth link. Compression alone will yield some benefit, but only in conjunction with link scheduling can very low-bandwidth links be managed.

Progressive techniques can be used, as before, to assure that the user is delivered the coarsest information first, so that in a very short amount of time a crude version of the entire layout can be seen. While in the case of web delivery, a crude version refers to reduced spatial resolution, in the case of a layout viewer, this refers to seeing only higher-level cells. The data within the cells could also be arranged progressively.

As with the application-independent and web-specific cases, it is critical to infer user intent to guide the use of the limited bandwidth link. In the case of the layout viewer, the intent is manifest in the choice of cells to layout cells to expand. This information is then provided to the networking layer so that the expanded cells are given higher loading priority and their data is transferred sooner.

In order to retain interactivity during the loading process, the user interface, networking, and rendering layers of the application must be decoupled. In this way, the user interface will not block because the application is waiting on data from the network, or because the application is in the middle of a compute-intensive drawing operation. Once this decoupling has occurred then the

constraints on the networking and rendering layers can be relaxed while still achieving interactivity.

Even large, highly detailed VLSI layouts can be viewed interactively over a slow link. The key is that at any given time the user's immediate requirements of data are typically modest, although they do need the ability to view the entire layout when desired.

11.2. Techniques to Increase Speed

11.2.1. Display Techniques

The simplest way to produce stipples is using the AWT drawimage to copy pre-determined swaths of semi-transparent stipples. The swaths are basically sets of IndexedColorMap images. Rendering is limited to within the window and clip rect to avoid unnecessary computations. Subcells are only recursed into if they contain some part in the current clip rect or window. This has the advantage of relative simplicity although the swaths have to be generated and determining where on the screen to draw layers is non-trivial given a hierarchical design with transformations.

This method is too slow, particularly under Unix. Unix currently uses interpreted Java while the PC implementations use just-in-time compilation. More importantly in Unix, the display is handled by another process - the X server, and each drawing request is very costly. Under Windows 95/NT on a PC, the drawing can be performed by the Java application directly using DirectDraw, and thus the overhead is substantially less. By running the "top" utility in Unix, it is clear that in AWT mode, the X server is indeed consuming the bulk of the cycles.

11.2.1.1. Image Blocks

Performance can be improved upon by performing the rendering to a byte array in Java on a cell by cell basis. Once a cells has been rendered into a byte array, it is displayed using a single

drawimage request. In this way, far fewer drawing requests are made. Instead of using the AWT's semi-transparent copy, the block mode routines that generate the byte data use something more akin to stippled rectangle drawing. The code has been unrolled and optimized to achieve respectable performance, even under interpreted Java.

11.2.1.2. Cell Image Caching

A more architecturally significant improvement comes by a technique dubbed *cell image caching.* Most designs are hierarchical while currently most display programs display “flattened”. If many copies of a given cell are displayed, the cells are rendered one at a time. This can lead to very slow rendering of large expanded layouts.

Instead, it is beneficial to save the image of a cell once it is rendered at a given magnification, and then upon having to render it again later, the image can be retrieved instead of having to redraw the layers from scratch. Since hierarchical designs include rotated and flipped subcells, either the cell image rendering has to be able to rotate and flip the images, or else an image has to be cached for each rotation. WebChip uses the latter option due to Java's lack of support for rotation. Not all browsers supported the 10 argument drawimage which allows flipping. There can be up to 8 different combinations of rotations and flips. The transform matrix is examined to determine the orientation.

In the case of stippled drawing, the stipple patterns must always align or else different layers in different cells can obscure each other. It suffices to round out the box sizes to the basic granularity of the stipple which is two. I.e. in the 8x8 stipples, most transparent patterns, except nwell, repeat on a 2x2 basis.

A cell image caching system must be able to revert back to either non-cached blocks or primitive AWT in case there is insufficient memory for the images, or the cell image rendering will be more costly than the primitives. WebChip has both modes of fall-back.

While cell image caching is very useful for WebChip, allowing it in some cases to render more rapidly than Magic or Cadence, it could also be extended to those tools. It is a generic technique useful for dealing with hierarchy.

11.2.2. Loading Techniques

Since the time to load layout over slow links is an important factor in interactivity, ways to improve it were investigated.

11.2.2.1. Compression

The first way to reduce load-time is to compress the data being loaded. Magic files are plain ASCII text which has the advantage of being human-readable, but not particularly compact. Converting the layout into a binary form and doing application-specific compression would yield very good compression at the expense of reduced portability. An alternative is to use Java's GZIP compression support on the ASCII cells[1].

WebChip supports reading both compressed and uncompressed layout files. If it does not find one type then it looks for the other. It assumes that most of the cells in a particular design will be compressed, or most will not. If it finds one compressed file, it checked for the next one being compressed before reverting to searching for the uncompressed version.

11.2.2.2. Concurrent Loading

Additional reductions in loading time can be obtained by issuing multiple layout cell requests simultaneously. This technique is commonly performed by web browsers when retrieving the images associated with a particular page. It amortizes the TCP/IP connect time cost. However, opening too many connections simultaneously can result in a loss in performance particularly over a modem link as the connections are never able to stabilize. For these reasons WebChip is

1. This was suggested by Michael Shilman.

currently set up to handle 3 concurrent loads. Additionally, by allowing the layout editor to issue the requests in the order it sees fit, it can request the top cells that the user is trying to view.

11.3. Techniques to Deal with Work In Progress

As mentioned, even once all efforts have been made to accelerate loading and display, additional mechanisms should be put in place to make the best of the speed that can be obtained. This section describes ways to operate effectively with non-negligible load and display times.

11.3.1. Hiding Slow Loading

Conventional CAD systems load the design database and then use it. Web-based agents, however should consider the loading process a significant part of their operation and thus allow the user to interact and perform useful work during this time. WebChip does just this.

Typically cells are created when they are loaded. Instead, WebChip creates cells the first time they are referenced. Then they are scheduled to be loaded as soon as possible. In the meantime, however, the database is still consistent and can be manipulated. The display subsystem knows how to render a cell that has not been loaded. (Typically their bounding box is drawn in gray.) Additionally, the loading subsystem places a call-back to the rendering subsystem as the cell is loaded so the user can get an up-to-date view.

11.3.2. Hiding Slow Display

Existing systems typically also consider the rendering process as something that starts and is then completed before further interaction can occur. In very fast systems operating on small designs, this is not a problem. However, on slower systems or very large designs, the user often has to interrupt the display process. WebChip allows interaction even during display by maintain-

ing a separate display thread for each active view window. The GUI interaction occurs in yet another thread so that it is not blocked.

Proper accounting keeps the display thread focused on the correct task. This is how, for instance, the selection box can be manipulated while redraw is in progress. (The actual mechanism for maintaining the selection box is to actually create 4 Canvas objects for each of the 4 sides and these are mapped over the view window.)

11.4. Conclusions and Future Work

This chapter has presented a method for interactive display of large VLSI layouts over the web. The WebChip applet achieves this through a number of optimizations focused on reducing and managing load and display time. Although the viewer is written in Java, similar techniques could be used on any platform.

CHAPTER 12 *Development Environment*

In this chapter, the development environment for the application-specific techniques is presented. First presented is *netem*, a network emulator which is used to model bandlimited channels. Next presented is the *SpeedSurfer* client-side proxy which is used to perform client-side link monitoring and analysis. Last described is *SurfServ*, the SpeedSurfer server-side proxy, which is also used to prototype the globally progressive interactive web delivery as described in Section 10.6..

12.1. Netem - Network Emulator

`netem`, detailed in Appendix A.5. (*netem (1)*), is a network emulator that can be used to emulate bandlimited links in real-time in order to study the effects of bandwidth limitations on system performance and user interaction. `netem` was used extensively in the development of the Globally Progressive Interactive web delivery protocol described in Chapter 10, as well as the WebChip VLSI layout viewer described in Chapter 11.

`netem` allows multiple simultaneous connections to be emulated, and operates at the stream level. Currently only TCP/IP connections are supported and emulation occurs at the TCP/IP connection level. `netem` emulates bandlimited links with fixed individual or total link capac-

ity. It can also emulate transport latency. The modeling is performed with an emphasis on high throughput rather than IP packet-level modeling accuracy. Thus to obtain highly accurate models of the interaction of multiple TCP/IP streams, or their reactions to packet loss or congestion, a tool such as the Network Simulator - `ns` [68] should be used.

`netem` is configured by specifying a set of connections that it should forward and emulate from the local machine to some remote host. For instance, consider that netem is running on a machine called emhost.eecs.berkeley.edu. It could be configured to forward connections from port 1234 to port 23 on otherhost.eecs.berkeley.edu via the command:

```
netem 1234=otherhost.eecs.berkeley.edu:23
```

This would allow emulation of telnet connections, which by default use port 23. If a user now telnets to port 1234 of emhost.eecs.berkeley.edu, a telnet session to otherhost, through netem is established. This would operate the same as a direct telnet connection to otherhost. However, a slow modem link can be emulated if rate limiting is specified via:

```
netem -rate_limit_bps 25000
      1234=otherhost.eecs.berkeley.edu:23
```

Now connections to port 1234 of emhost are still forwarded to otherhost, but only at a maximum data rate of 25Kbps. Note that a typical 28.8kbps modem connection will result in throughput closer to 20-25Kbps due to byte-level and packet-level synchronization and overhead and prevailing phone line and Internet conditions. If multiple connections to port 1234 of emhost are established then the total rate of all connections are limited to 25Kbps in order to simulate a shared link. The `-rate_limit_individual` option can be used to allocate 25Kbps for each connection to emulate independent links.

Fixed latency can be specified via the `-latency_ms` option as in:

```
netem -rate_limit_bps 25000 -latency_ms 200
      1234=otherhost.eecs.berkeley.edu:23
```

This adds the addition restriction that data incurs a delay of 200ms while being forwarded. This can emulate latency due to network queues and packetization delays.

In addition to forwarding connections to fixed addresses, netem can act as a web proxy and determine which host to connect to via the data stream. The following command demonstrates this:

```
netem -rate_limit_bps 25000 1234=web
```

Now if a web browser is used with its HTTP proxy set to emhost port 1234 then all web pages viewed with the web browser are directed through the proxy and will consequently be subjected to rate limiting. In this way, the performance of a slow link can be experienced on a network without such slow links, and the effects of different networking protocols can be explored.

Netem can also print connection rate information as well as log the data being transferred on the emulated connections by varying the `-verbosity` level. One such example is shown in Figure 12.1. Other features and options are described in Appendix A.5. (*netem (1)*).

12.2. SpeedSurfer - PC Client-Side Proxy

The *SpeedSurfer* client-side proxy is a Windows 95/NT application that acts as a local client-side proxy as well as arbitrary TCP/IP connection forwarder. It can be used in conjunction with SurfServ, described in the next section, to encapsulate multiple TCP/IP connections across a lossy bandlimited link in an explicitly managed manner. Additionally, it can perform real-time data flow analysis and generate real-time web-page loading graphs described in Section 10.4..

```
Using buffer of size 128
Port mappings:
        Port 1234 -> Web
Accepted client from 128.32.62.75 port 1794 as connection 0
Got (128) “GET http://Badlands.EECS.Berkeley.EDU:8090/~gilbertj/
HTTP/1.0\r\n
Proxy-Connection: Keep-Havering
User-Agent: Mozilla/4.5 [en] (WinN" from client 0
Http server Badlands.EECS.Berkeley.EDU:8090 is 128.32.139.53 port
8090
Connection 0 is to Badlands.EECS.Berkeley.EDU:8090 to perform:
        GET /~gilbertj/ HTTP/1.0
Sending to server 0 "GET /~gilbertj/ HTTP/1.0\r\n
Proxy-Connection: Keep-Alive\r\n
User-Agent: Mozilla/4.5 [en] (WinN"
Finished writing 90 to server
Got (128) "T; I)\r\n
Pragma: no-cache\r\n
Host: Badlands.EECS.Berkeley.EDU:8090\r\n
Accept: image/gif, image/x-xbitmap, image/jpeg, image/pjpeg, ima"
from client 0
Finished writing 128 to server
Got (95) "ge/png, */*\r\n
Accept-Encoding: gzip\r\n
Accept-Language: en\r\n
Accept-Charset: iso-8859-1,*,utf-8\r\n
\r\n" from client 0
Finished writing 95 to server
Got (128) "HTTP/1.0 200 Document follows\r\n
Server: CERN/3.0A\r\n
Date: Mon, 20 Mar 2000 18:44:42 GMT\r\n
Content-Type: text/html\r\n
Content-Length: " from server 0
Finished writing 128 to client
Got (128) "4108\r\n
Last-Modified: Mon, 07 Jun 1999 00:24:28 GMT\r\n
\r\n
<!doctype html public "-//w3c//dtd html 4.0 transitional//en">\r\n
<html>\r\n
<h" from server 0
Finished writing 128 to client
Got (128) "ead>\r\n
   <meta http-equiv="Content-Type" content="text/html; char-
set=iso-8859-1">\r\n
   <meta name="Author" content="Jeff Gilbert"" from server 0
Finished writing 128 to client
Got (128) ">\r\n
   <meta name="GENERATOR" content="Mozilla/4.5 [en] (WinNT; U)
[Netscape]">\r\n
   <title>Jeff Gilbert's Home Page</title>\r\n
```

FIGURE 12.1. **Example diagnostic printouts. Generated by `netem 1234=web -verbosity 31 -rate_limit_bps 20000`**

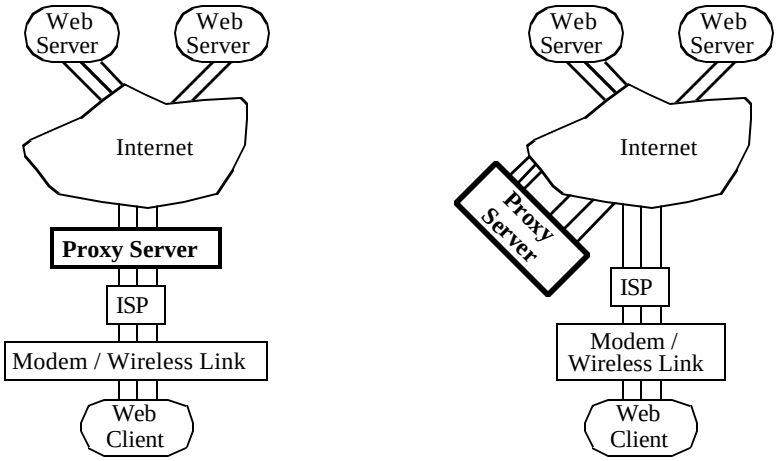


FIGURE 12.2. Two views of server-side proxies. The view on the left is semantic while the view on the right is closer to actual implementation.

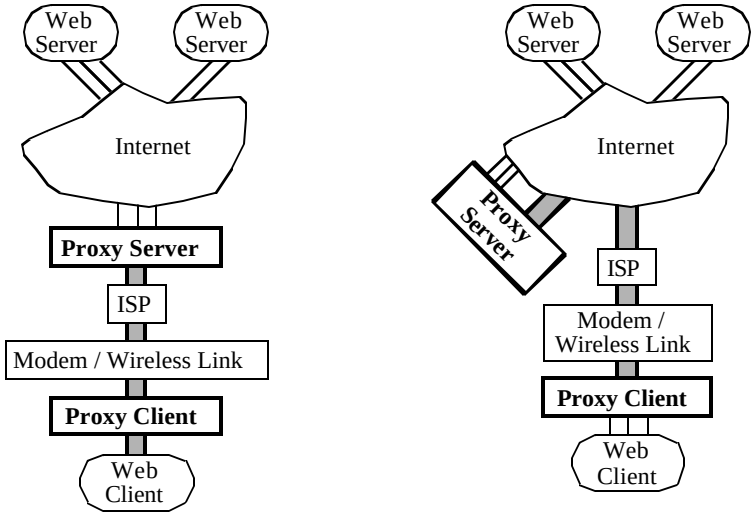


FIGURE 12.3. Two views of client-side and server-side proxies for better link control. Again the view on the left is semantic while the view on the right is closer to actual implementations. Note that the proxy server should be placed *near* the ISP for optimal performance if it cannot be replicated at each ISP.

While a summary of the SpeedSurfer application is presented here, details of its operation can be found in Appendix C (*The SpeedSurfer Application*).

12.2.1. Client-Side Proxies

Conventional (server-side) web proxies only reside on the web-server side of slow links as shown in Figure12.2. In order to use these proxies, the address of the web proxy is specified to a

web browser. The browser then sends all requests to the proxy instead of the address of the requested web page. While this setup means that the client does not have to run any special software, it also means that the protocol going over the slow or lossy link must be the protocol that the web browser understands. This can significantly limit the flexibility and power of the system. The Java applets described in Section 10.6. are one way to obtain greater flexibility without browser modification but are limited to web delivery, and have limited interface capability and performance constraints.

However, if a web proxy is run on the client's side of the link as well, as shown in Figure 12.3, then the data travelling over the modem or wireless link can be fully controlled to best exploit the characteristics of the link. In this scenario, the web browser is told that there is a proxy located on the client's machine. Thus all requests are routed through the client-side proxy. The client-side proxy, in turn, knows how to contact the server-side proxy. Although the user has to run a proxy on the same side of the slow link as the browser, the actual web browser need not be modified. Additionally, the client and server proxies are standard applications and the network infrastructure need not be modified.

The same proxy architecture can be used for non-web connections. Most applications such as telnet, ftp, and X Windows can be instructed to connect to a different location - one of the client proxy's ports, to effect the connection. A single client proxy can accept connections on multiple ports to handle multiple services. Since all (or most) connections over the constrained link go through the client and server proxies, centralized link management is possible, leading to more efficient link utilization.

12.2.2. Link Management Using Client-Side and Server-Side Proxies

The client-side proxy architecture can be used to investigate link management from a networking perspective since all outgoing network connections are passed through the client / server proxy pair. Thus all traffic over the limited bandwidth links can be centrally controlled. In particular, link aggregation was explored where all incoming TCP/IP connections are combined into one persistent connection with the server-side proxy.

Multiplexing the multiple transient streams through one persistent TCP/IP connection has several benefits in terms of increasing performance. It eliminates the per-connection TCP/IP connection establishment overhead and slow-start delays. It also reduces contention between the multiple connections. Additionally, the combination and prioritization of the links can be controlled by the two proxy servers. In the case of web access, it can be used to experiment with coercing HTTP/1.0 connections into an HTTP/1.1-like stream.

Furthermore, some of the overhead of TCP and IP headers of small packets can be amortized over multiple connections since the information needed to demarcate the various streams in the single TCP connection is much less than the overhead if each was its own TCP connection. Additionally, since there is a single dominant stream, it is safe to disable the Nagle algorithm which can delay packetization.

Finally, since the TCP/IP connections between the web server and browser have been divided into two parts on the server side, buffering can occur at the server-side proxy. The server-side proxy smooths out the bursty traffic that comes from the remote web servers over the Internet. When a direct connection is made from the web browser to the servers, this kind of buffering cannot occur effectively as losses due to congestion in the Internet will cause degradation in link per-

formance. The client / server proxy pair could be used to perform lossy or lossless source coding or encryption without modifying either the end-server or end-client.

The SpeedSurfer and SurfServ proxies are used to analyze a performance limitation in the Berkeley dial-in modem pool. By multiplexing multiple independent web connections and limiting the amount of data queued for transit over the modem link to 2000 bytes, extraneous TCP/IP errors and time-outs are eliminated, allowing transmission at about 90% of the link capacity compared to only about 50% link capacity without the proxies. [30]

In addition to adding more flow control to the TCP/IP connection, UDP packets could be used instead to allow full redesign of the reliable protocol. This would likely be necessary for adequate performance over a wireless link. Alternatively other flavors of TCP/IP such as Vegas could be investigated. Based on the previous observations, it seems that Vegas would be much better suited since it avoids forced congestion and packet loss.

12.3. SurfServ - SpeedSurfer Server / Progressive Proxy

`SurfServ`, detailed in Appendix A.8. (*SurfServ (1)*), is a unix-based application that performs the necessary connection establishment, multiplexing, and demultiplexing to support the SpeedSurfer client-side proxy. The SurferServ is optimized to handle multiple SpeedSurfer sessions efficiently. Simple techniques such as minimizing data copying and efficient use of select() make it possible for the SurferServ to have a minimal load on the CPU. Each session with a distinct SpeedSurfer client is forked into a new process so that the connections will not adversely affect each other, and also to get around the per-process limitation of 64 open files. The SpeedSurfer sessions are long-lived so that the impact of the new process creation is minimal.

SurfServ also acts as the server-side proxy for the Globally Progressive Interactive Delivery prototype described in Section 10.6. It performs the HTML image to applet tag translation as well as managing the image delivery links from the Java applets. The connection to the web servers is made using HTTP/1.1 with persistent connections whenever possible, reducing the number of new web browser connections. Thus a new process can be used for each new connection and a new process is used for connection to the Java applets. This reduces open-file limitations. Individual light-weight threads are used to perform the progressive image transcoding so that the transcoding of one image does not impact that of another.

PART IV *Conclusions*

CHAPTER 13 *Conclusions and Future Directions*

13.1. Network Requirements

There are many common themes which pervade the various types of text / graphics and image transmission described in this thesis. This section distills the common networking requirements in effort to propose new services which will allow modular reuse of these capabilities. By coupling the applications more closely with the network protocols, but keeping the packet-level scheduling in the operating system, the necessary agility can be retained while not sacrificing efficiency.

In this section, messages refer to the atomic unit of communication where a part of a message is of no value until the whole message is received. (Others have referred to this as an *atomic data unit*.) A stream refers to one ordered set of messages where in-order delivery of the messages

is required. A packet is the unit of data physically transferred over the network and a connection subsumes the entire set of data transferred for the application.

13.1.1. Lightweight, Independent Streams

One recurring theme seen throughout this thesis is that text, graphics, and image data transmission involves multiple lightweight streams. The granularity and size depend on the particular application. For the case of the web, the streams are simply the objects in the web page. Application-specific instances define their own granularities - the WebChip application uses independent streams for each VLSI layout cell. For the case of the bitmap-based approaches, the ultra-light streams correspond to the individual blocks in the image. The conventional primitive approach allows one stream per application. The hybrid approach uncovers much more packet independence allowing disjoint sets of primitives to occupy separate streams.

Uncovering parallelism and removing false dependencies is almost always beneficial. Parallel computer architectures require this for efficiency, multiple-issue microprocessors can exploit it, networking can exploit it to better cope with packet loss. By exposing the true dependencies to the networking layer, the application need not individually manage the sub-streams but can enjoy the benefits of their management.

The MUX protocol [51] described in Section10.3.1. implements lightweight streams over TCP/IP which does expose the interface to applications, but layering upon TCP/IP introduces false-dependencies in that the use of a single TCP/IP stream will cause losses in one sub-stream to delay data in another sub-stream.

13.1.2. Explicit Message Interdependence

In the case of the hybrid approach, simple streams do not suffice to expose all dependencies. More complicated directed acyclic graphs are required to express the fact that some primitives can

be dependent on multiple other streams. Thus more complex message interdependence exists. Streams are a special case where there are linear dependency graphs, and thus general purpose interdependencies can be used to describe streams as long as efficiency is not compromised.

Explicit packet interdependence can also be used to improve sending of individual images in web transmission if the independent regions in the data are noted. This can expose further parallelism and further reduce latency due to loss (see Section3.2.2.) in highly lossy environments.

13.1.3. Dynamic Reprioritization of the Streams

Interactive text / graphics and image transmission requires dynamic reprioritization of the data streams to obtain low latency over bandlimited and/or lossy links. Explicit prioritization allows better link management to assure that the data that needs to arrive quickly is delivered first. In the case of application-independent transmission, it allows low-bandwidth, high-impact data such as text and non-image graphics primitives to be delivered quickly. It also allows regions of interest, such as where the cursor is, to be delivered more quickly than other areas. For web transmission, it allows on-screen images, to be delivered before off-screen images and images where the cursor is to be delivered before other on-screen images. Application-specific instances such as WebChip use prioritization to assure that base cells are transferred before underlying cells.

13.1.4. Message Unqueing

In addition to reprioritizing messages, some cases, such as the application-independent transmission, benefit from being able to unqueue messages that have been enqueued earlier. This is primarily used for removal of redundant messages when one text / graphics primitive or bitmap update is superseded by a later one. In these cases, correctness still is maintained if the earlier, stale packet is delivered, though it will consume additional bandwidth. Thus as soon as the text / graphics server knows that the message is no longer useful, it will unqueue it. In this way, if the

message is queued in the network buffers, it can be removed before it is sent, and if it has already been sent, no bandwidth will be wasted attempting retransmissions if it is not delivered. This was also proposed as *implicit annihilation* in [37].

13.1.5. Rate, Flow, and Congestion Control

While applications desire tight control over how their share of available bandwidth is used, they still need to coexist in a global Internet with other traffic streams. Thus the sum of their traffic must behave like one or multiple TCP/IP streams to promote fair resource utilization. This also allows applications which run over varying networks to adapt to the network and conditions. Care must be taken to prevent loss due to wireless link corruption from being confused with packet loss. Techniques to adapt TCP/IP to wireless links, such as SNOOP, can be used while retaining the multi-stream architecture [8].

13.1.6. Notification of Packet Arrival

As we have seen in the bitmapped and hybrid approaches, the applications often keep additional state associated with data in flight, and must be notified of the delivery of the data in order to free the state. Thus they often need feedback from the transport system when successful data delivery occurs. Additionally, if reliable transmission at the network level is not used then a time-out notification back to the application is useful.

13.2. Conclusions

Compression is not enough.

This thesis has shown that image and data compression alone are not sufficient to obtain interactive performance over many bandlimited and lossy links. Link scheduling, progressive techniques, and suppression of stale data in conjunction with image and data compression yield a

far more effective strategy for compensating for bandwidth limitations and data loss. It is critical to view interactive text / graphics and image transmission as a user-based activity and optimize the results as experienced by the user. This entails determining which information the user is most interested in and sending it first, and realizing that this interest can change dynamically over time. Increased concurrency of delivery of data on various parts of the image is required to allowed continued interactivity in the face of reduced connectivity. When these steps are taken, interactive operation using a large screen or of a large design significantly improves over low bandwidth, high error-rate links.

13.3. Future Directions

While Section 13.1. defines some criteria for a network protocol that would be useful to a wide range of text / graphics and image applications, there are still many details to be resolved and an implementation has yet to be developed and deployed.

Similarly, Chapter 7 describes a hybrid approach to text / graphics and image transmission with many ideas and motivations grounded in previous completed work, but the hybrid approach itself has not been implemented. It would be useful to implement the hybrid approach into a thin client and either a modified X server, as used in the InfoPad system, or Windows Terminal Server, to allow bandwidth-efficient remote access to windows applications over wireless lossy links.

Chapter 10 describes how web access over slow and/or lossy links can be improved by using globally progressive interactive web delivery. Analysis and a Java / proxy prototype are presented, as are the drawbacks of the prototyping method. More seamless deployment on a large scale can be achieved by integrating the client-side functionality into a popular web browser and integrating the server-side functionality into a popular web server and proxy. Several of the

advantages are described in that chapter. Additionally, by using the improved networking services previously described, improved performance, particularly over wireless links, can be achieved.

This future research could further confirm that although compression is an important ingredient in efficient text / graphics and image transmission over bandlimited lossy links, *compression is not enough*.

Bibliography

[1] Elan Amir, Steve McCanne, and Hui Zhang, “An Application Level Video Gateway,” Proc. ACM Multimedia '95, San Francisco, CA, November 1995.

[2] Thomas E. Anderson, David E. Culler, David A. Patterson, and the NOW Team, “A Case for Networks of Workstations: NOW.” *IEEE Micro*, Feb 1995.

[3] The Apache Group, “The Apache Web Server Project.” http://www.apache.org.

[4] Arps, R.B. and T.K. Truong, Comparison of International Standards for Lossless Still Image Compression. *Proceedings of the IEEE*: 82:889-899, June 1994.

[5] AT&T. Virtual Network Computing homepage. http://www.uk.research.att.com/vnc

[6] P. Ausbeck, Jr., “Context Models for Palette Images”, in *Proc. of the 1998 Data Compression Conference*, (Snowbird, Utah), pp. 309-318, April 1998.

[7] Backman, Dan, “Spectrum Lets You Dial In On The FastLane,” Network Computing Online, http://techweb.cmp.com/nc/907/907sp4.html.

[8] H. Balakrishnan, S. Seshan, E. Amir, and R. Katz, “Improving TCP/IP Performance in Wireless Networks,” Proc 1st ACM Conference on Mobile Computing and Networking (MOBICOM), Berkeley, CA, November 1995.

[9] Balakrishnan, Hari, and Venkat Padmanabhan, Srinivasan Seshan, Mark Stemm, and Randy H. Katz, "TCP Behavior of a Busy Internet Server: Analysis and Improvements." Proceedings of IEEE Infocom '98. March 1998.

[10] Bell, Timothy C., John G. Cleary, and Ian H. Witten, Text Compression, Prentice Hall, Englewood Cliffs, NJ, 1990.

[11] Berners-Lee, Tim, R. Fielding, H. Frystyk., "Informational RFC 1945 - Hypertext Transfer Protocol -- HTTP/1.0," MIT/LCS, UC Irvine, May 1996. http://www.w3c.org/Protocols/rfc1945/rfc1945

[12] Boutell, T., T. Lane et al. "PNG (Portable Network Graphics) Specification," W3C Recommendation, October 1996, RFC 2083, Boutell.Com Inc., January 1997. http://www.w3c.org/Graphics/PNG.

[13] A. Broder and M. Mitzenmacher. "Pattern-based compression of text images," in *Proc. of the 1996 Data Compression Conference*, (Snowbird, Utah), pp. 171-180, March 1996.

[14] R.W. Brodersen, A.P. Chandrakasan, S. Sheng, "Design Considerations for Portable Systems", IEEE International Solid-state Circuits Conference, pp. 168-169, February 1993.

[15] Chandrakasan, Anantha, "Low Power Digital CMOS Design," Ph.D. Dissertation, University of California, Berkeley. Berkeley, California, 30 August 1994.

[16] Citrix Corporation. "ICA Technology." http://www.citrix.com/products/ica.asp

[17] CompuServe Incorporated, "Graphics Interchange Format - Version 89a," 1987-1990. http://www.w3c.org/Graphics/GIF/spec-gif89a.txt

[18] C. Constantinescu and R. Arps, “Fast Residue Coding for Lossless Textual Image Compression” in *Proc. of the 1997 Data Compression Conference*, (Snowbird, Utah), pp. 397-406, March 1997.

[19] Converse, D et al. - X Consortium, “Low Bandwidth X Extension.” ftp://ftp.x.org/pub/R6.4/xc/doc/specs/Xext/lbx.mif

[20] Cornelius, David, “XRemote: a serial line protocol for X,” 6th Annual X Technical Conference, Boston, MA 1992.

[21] David E. Culler, Andrea Arpaci-Dusseau, Remzi Arpaci-Dusseau, Brent Chun, Steven Lumetta, Alan Mainwaring, Richard Martin, Chad Yoshikawa, Frederick Wong, “Parallel Computing on the Berkeley NOW”. JSPP'97 (9th Joint Symposium on Parallel Processing), Kobe, Japan .

[22] Danskin, John, “Compressing the X Graphics Protocol,” Ph.D. Dissertation, Princeton University. Princeton, New Jersey, January 1995.

[23] Fielding, R., J. Gettys, J.C. Mogul, H. Frystyk, T. Berners-Lee, “RFC 2068 - Hypertext Transfer Protocol -- HTTP/1.1,” UC Irvine, Digital Equipment Corporation, MIT. http://www.w3c.org/Protocols/rfc2068/rfc2068

[24] Floyd, Sally and Van Jacobson. Link-sharing and resource management models for packet networks. *IEEE/ACM Transactions on Networking*, 3(4):365-386, August 1995.

[25] Fox, Armando, Steve Gribble, Yatin Chawathe, and Eric Brewer, “The Transend Service,” http://transend.cs.berkeley.edu/about

[26] Fox, Armando, Steven D. Gribble, Yatin Chawathe, Eric A. Brewer, and Elan Amir, "Adapting to Network and Client Variability via On-Demand Dynamic Distillation," Proceedings Seventh International Conference on Architectural Support for Programing Languages and Operating Systems (ASPLOS-VII), October 1997, Cambridge, MA.

[27] Fulton, Jim, and Chris Keat Kaatarjiev, "An update on low bandwidth X (LBX)," *Proceegins of the 7th Annual X Technical Conference*, January 1993, O'Reilly and Associates.

[28] Gailly, J. and M. Adler, "ZLIB documentation and sources," available at ftp://ftp.uu.net/pub/archiving/zip/doc

[29] Gettys, James, and Philip L. Karlton, and Scott Mcgregor, "The X Window System, Version 11," *Software Practice and Experience* vol. 20(S2), S2/35-S2/67. October 1991.

[30] Gilbert, Jeff. "Optimizing Web Access Over Modem (and Wireless) Links Using Client-Side Proxies." EE228a Project Report. December 4, 1997.

[31] Gilbert, Jeffrey M. and Robert W. Brodersen. "Globally Progressive Interactive Web Delivery." *Proceedings 1999 IEEE Infocom*. New York. 21-25 Mar 1999.

[32] Gilbert, Jeffrey M. and Robert W. Brodersen. "A Lossless 2-D Image Compression Technique for Synthetic Discrete Tone Images." *1998 IEEE Data Compression Conference*. Snowbird, Utah. 28 Mar - 1 Apr 1998. p.359-368.

[33] Gilbert, J.M. and Yang, W. "A Real-time Face Recognition System using Custom VLSI Hardware." *Proceedings 1993 Computer Architectures for Machine Perception*. New Orleans, LA. 15-17 Dec 1993. p.58-66.

[34] Gilbert, Jeffrey M. “A Real-time Face Recognition System using Custom VLSI Hardware.” Undergraduate honors thesis, Harvard College. Cambridge, MA. May 1993.

[35] Graphon Corporation, “Graphon Corporation Homepage,” http://www.graphon.com

[36] Richard Han and David G. Messerschmitt, “A Progressively Reliable Transport Protocol for Interactive Wireless Multimedia.” *Multimedia Systems* vol 7(2): pp. 141-156, 1999.

[37] Richard Y. Han and David G. Messerschmitt, “Asymtotically Reliable Transport of Multimedia / Graphics over Wireless Channels.” *Proceedings Multimedia Compution and Networking*, San Jose, CA, Jan 29-31, 1996.

[38] P. G. Howard, “Lossless and Lossy Compression of Text Images by Soft Pattern Matching” in *Proc. of the 1996 Data Compression Conference*, (Snowbird, Utah), pp. 210-219, March 1996.

[39] Huffman, D.A., “A Method for the Construction of Minimum Redundancy Codes,” Proceedings of the Institute of Radio Engineers, September 1952, Volume 40, Number 9, pp. 1098-1101.

[40] Independent JPEG Group’s JPEG library. ftp://ftp.uu.net/graphics/jpeg.

[41] Intel Corporation, “QuickWeb homepage”. http://www.intel.com/quickweb

[42] Jacobson, Van, “Congestion Avoidance and Control,” Proceedings of ACM SIGCOMM ‘88, p. 314-329. Stanford, CA, August 1988.

[43] M. Kuhn. Software kit for jbigkit. Available via anonymous ftp from ftp.unierlangen.de as /pub/doc/ISO/JBIG/jbigkit-0.7.tar.gz.

[44] McWilliams, Brian, “Intel Announces Server-Side Browser Accelerator,” PC World Online, January 20, 1998. http://www.pcworld.com/news/daily/data/0198/980120192859.html

[45] Microsoft Corporation. NetMeeting Home. http://www.microsoft.com/netmeeting.

[46] Microsoft Corporation. "Windows NT Terminal Server Edition." http://www.microsoft.com/ntserver/terminalserver/default.asp

[47] Moffatt, A., "A note on the PPM data compression algorithm," Research Report 88/7, Department of Computer Science, University of Melbourne, Parkville, Victoria, Australia.

[48] Mozilla Home Page, http://www.mozilla.org

[49] S. Narayanaswamy, S. Seshan, E. Brewer, R. Brodersen, F. Burghardt, A. Burstein, Y.-C. Chang, A. Fox, J. Gilbert, R. Han, R. H. Katz, A.. Long, D. Messerschmitt, J. Rabaey; "Application and Network Support for InfoPad" *IEEE Personal Communications Magazine*, Mar 1996.

[50] Nielsen, Henrik Frystky, "Hypertext Transfer Protocol - Next Generation" homepage. http://www.w3c.org/Protocols/HTTP-NG/

[51] Nielsen, Henrik Frystky, "MUX Overview" W3C Architecture Domain. http://www.w3c.org/Protocols/MUX

[52] Nielsen, Henrik Frystky, et al., "Network Performance Effects of HTTP/1.1, CSS1, and PNG." W3C NOTE-pipelining-970624. June 24, 1997. http://www.w3c.org/Protocols/HTTP/Performance/Pipeline.html.

[53] Padmanabhan, Venkat and Randy H. Katz, "Addressing the Challenges of Web Data Transport," unpublished.

[54] Padmanabhan, V.N. and J. Mogul, "Improving HTTP Latency," Computer Networks and ISDN Systems, v.28, pp.25-35, Dec. 1995.

[55] Padmanabhan, Venkat - personal correspondence.

[56] Pennebaker, William B. and Joan L. Mitchel, JPEG Still Image Data Compression Standard, New York: Van Nostrand Reinhold, 1993.

[57] J. Poskanzer. Pbmplus - image file format conversion package: http://www.acme.com/software/pbmplus

[58] T. Richardson, Q. Stafford-Fraser, J. Weatherall, K.Wood, A. Harter, C. McLachlan, P. Webster. AT&T Virtual Network Computing homepage. http:// www.uk.research.att.com/vnc.

[59] Roelofs, Greg., "PNG (Portable Network Graphics) Home Page" http:// www.cdrom.com/pub/png

[60] Said, Amir, and William A. Pearlman, "An Image Multiresolution Representation for Lossless and Lossy Compression," SPIE Symposium on Visual Communications and Image Processing, Cambridge, MA, Nov. 1993.

[61] Sayood, K., *Introduction to Data Compression*. San Francisco, CA: Morgan Kaufmann Publishers, Inc., 1996.

[62] Scheifler, Robert W., "The X Window System Protocol." M.I.T. Laboratory for Computer Science. 1988.

[63] Scheifler, Robert and James Gettys. X Window System. Digital Press, Burlington, MA, 1992.

[64] Scheifler, Robert W. and Jim Gettys, "The X Window System." *Transactions on Graphics* 5(2), 79-109, April 1986, and Software Practice and Experience vol 20(S2), S2/5-S2/34, October 1991.

[65] Spectrum Information Technologies homepage, http://www.spectruminfo.com

[66] Spero, Simon E, “Analysis of HTTP Performance Problems,” July, 1994. http://www.w3c.org/Protocols/HTTP/Performance/Pipeline.html

[67] J. A. Storer and J. Reif, “Low-Cost Prevention of Error-Propagation for Data Compression with Dynamic Dictionaries” in *Proc. of the 1997 Data Compression Conference*, (Snowbird, Utah), pp. 171-180, March 1997.

[68] “UCB / LBNL / VINT Network Simulator - ns (version 2)” http://www-mash.cs.berkeley.edu/ns

[69] Waldspurger, Carl A. Lottery and Stride Scheduling: Flexible Proportional-Share Resource Management, Ph.D. dissertation, Massachusetts Institute of Technology, September 1995.

[70] “WebTP Home Page, EECS, UC Berkeley”. http://webtp.eecs.berkeley.edu

[71] Welch, T., “A Technique for High-Performance Data Compression,” Computer, June 1984.

[72] Wessels, Duane, “Squid Internet Object Cache.” http://squid.nlanr.net/

Ziv J. and A. Lempel, “A Universal Algorithm for Sequential Data Compression,” IEEE Transactions on Information Theory, Vol. 23, No. 3, pp. 337-343.

[73] X Consortium, “LBX X Consortium Algorithms.” ftp://ftp.x.org/pub/R6.4/xc/doc/specs/Xext/lbxalg.mif

APPENDIX A

Software Documentation

A.1. Codebook2ras (1)

NAME

codebook2ras - VQ Video Codebook to Sun Rasterfile Converter

SYNOPSIS

codebook2ras < codebook_file > codebook_picture.rs

DESCRIPTION

This short program converts VQ Video codebooks into a form suitable for viewing or further manipulation. (For manipulation, rasttopnm followed by any of the pnm* tools works well.) The codebook files can be generated from vq(5) files via vq2codebook(1) and viewed via many views, particularly xloadimage and xv.

OPTIONS

It does not take any command line arguments. Input must come from the standard input and the output goes to standard output.

OPERATION

Codebook2ras arranges the data in the codebook so that it is easier to interpret (i.e. in 4x4 block, etc) with the Y codebook on top of the I codebook, which is on top of the Q codebook. The whole resultant image is 324x1004.

SEE ALSO

showcodebook(1) mpeg2vq(1) send_vq(1) vq_play(1) vq(5)

AUTHOR

Jeff Gilbert <gilbertj@eecs.berkeley.edu>

NAME

emu - InfoPad Emulator

SYNOPSIS

emu [- GatewayId] [- PadId] options...

DESCRIPTION

The InfoPad emulator is a Tcl/Tk application which emulates the pad at the protocol level. Messages as they would appear over the radio link are interpreted and generated. The emulator connects to a gateway which independently connects to the various servers. The emulator displays both text / graphics output as well as VQ video on a separate popup window. (See also vq_play man page). Pad audio input and output are also emulated using the workstation's microphone and speaker ports. (Currently only Sun SparcStations are supported.) Pen input can be generate either by using the mouse or an external tablet. (Many brands of tablet are supported.) Keyboard input is also available to aid in development.

Additional debugging aids include a display of the radio traffic statistics into and out of the emulator (or pad). This can be used to analyze bandwidth requirements. The effective downlink audio buffer size can be varies to determine the buffering necessary compensate network jitter.

The emulator is tightly coupled with The Name Server to allow the auto-start version of the various servers (pad, t/g, pen, and audio) to be selected during operation. The status of the servers and gateway is continually displayed to quickly identify which are running. Pull-down menus locate running gateways and available pad servers.

OPTIONS

-show_traffic

Causes the traffic window to be displayed initially. Otherwise the TRAFFIC button has to be hit to display it. This can be useful for automated demos.

-show_av

Causes the A/W window to be displayed initially. Otherwise the A/V button has to be hit to display it. This can also be useful for automated demos.

-tablet <tablet type> <tablet device>

Enables tablet support for the pen emulator. <tablet type> must be: scriptel, gazelle0, gazelle1, wacom_old, or wacom_ud. <tablet device> must be the UNIX file name of the tablet device - for example /dev/ttya.

-helpp

Lists the above options. Note that -help will not work as Tcl/Tk intercepts it before the emu application can get at it. Any invalid command-line option will work.

OPERATION

When emu is started, it displays the main window which consists of the emulated black and white text / graphics display with various widgets below it which control the operation of the emulator. If a *GatewayId and PadId are both specified on the command line then the emulator connects to the specified pad server through the specified gateway at start-up. Otherwise the emulator comes up in an unconnected mode.*

The following buttons, menus, and text entry widgets at the bottom of the window control the emulator operation:

QUIT

Quits the emulator. Will disconnect first if necessary.

CONNECT

The indicator in the connect button is on if the emulator is currently connected to a pad server through a gateway. If there is no current connection, (i.e. the indicator is off) pressing CONNECT button attempts to connect to the pad server and gateway specified in the text entry boxes. If there is already a current connection (i.e. the indicator is on) then the connection is broken.

REMOTE REFRESH

Causes an xrefresh to be sent to the text / graphics server to force a full refresh of the display.

STATS

Displays a pop-up with neat BER and CELL POWER bars which can be manipulated but actually do nothing as of yet. They were put in by original author for functionality not yet fully implemented. It did not hurt anything and may well prove useful soon so I did not remove it.

TRAFFIC

Displays the traffic pop-up which shows packets/sec, kilobits/sec, and average bytes per packet for each of the uplink and downlink data types as well as the overall downlink and uplink statistics. The display is updated every second.

AUTO REMAP

Selects automatic remapping, which like the STATS box has all of the hooks necessary to allow handoff (??) but is not supported by the rest of the system. Currently if AUTO REMAP is on then when it get polling packets the cell power bars in the STATs window are randomly varied.

POLLED

This button is actually just used as an indicator of the reception of polling packets. These are generated automatically by the pad server. When received, the POLLING button is flashed. They are received periodically whenever there is an active connection.

MOVE

Causes the current connection to be broken and a connection to the new padserver and gateway specified in their text entry boxes to be established.

A V

Displays the A/V (audio and video) pop-up window which allows control of the audio and video emulation. The window has the following controls:

Audio Play - If on then downlink audio is sent to workstation /dev/audio.

Audio Rec - If on then uplink audio is read from workstation /dev/audio.

Audio Auto - If on, the audio play and rec are turned on upon reception of downlink audio data and turned off upon disconnection. By default, this is selected. Note that only one program can be connected to a Sparc's /dev/audio at a time so if AF is running, the emu cannot connect to the speaker and microphone. A message will be displayed and audioless operation can continue.

Verbose (audio) - Logs audio downlink buffer information to the console if selected.

Downlink Buffer Size - Selects the amount of audio data that can be buffered in the emulator. This should be set to the amount of buffering which would occur between the gateway and the pad Codec (thus it includes Tx chip fifo and audio chip fifo.) Altering this allows determination of the proper buffer size required to tolerate network jitter but also not delay the audio too much.

Video Play - Displays the VQ Video window for downlink VQ video. The vq_play(1) program is used for this. Clicking on it again closes the window.

Video Auto - If on, the video window is opened upon reception of any VQ Video data and closed upon disconnection. By default, this is selected.

Video Drop - When selected, the Maximum Display Rate value is used to make sure that only that many frames per second are sent to the vq_play program. Additional frames are discarded in the emulator. It is necessary to use frame dropping to prevent the vq_play program and X Windows server from consuming too much CPU time making the emulation unrepresentative. Setting it to about 15 frames per second still gives reasonable performance without causing a bottleneck.

If Video Drop is not selected then all video data received is sent to the vq_play program and the emulator can block waiting to write into its pipe to vq_play. By default, it is selected.

TABLET

Enables and controls tablet support. If tablet is selected then the TYPE menu must be set to the tablet type and InfoPad pen emulation using the tablet occurs. The Tablet Device text entry box should be set to the device (for example /dev/ttya) that the tablet is connected to.

GATEWAY

This pull-down menu and text entry box allow selection of the gateway to connect through. A number can be entered directly into the text box or else the current running gateways (from the name server) are listed in the pull-down if the left mouse button is clicked on the Gateway button.

The status of the currently selected gateway and cell server are displayed to the left of the word “Gateway”. The status is either “not running” if neither are running, “running” if both are running, “CS down” if the gateway is up but the cell server is down, or “GW down” if the opposite is true.

PAD SERVER

This pull-down menu and text entry box allow selection of the pad server to connect to. A number can be entered directly into the text box or else all registered pad servers are listed in the pull-down menu if the left mouse button is clicked on the Pad Server button. Their status is also shown in the menu (unless it is onDemand.)

TABLET DEVICE

Selects the unix device to connect to for the tablet. The pull-down menu has a few common choices.

SERVER STATUS AREA

The status of the currently selected pad, X (text / graphics), Pen, and audio server are displayed at the bottom of the main emulator window. It is retreived from the Name Server periodically. When any critical event occurs, such as selecting a new pad server or gateway, the emulator polls for status more often for a while. Additionally, the UPDATE button forces the status to be reread.

The Pulldown menus for "PS Version", "X Version", "Pen Version", and "Audio Version" control the version of the respective servers which is used if they are autostarted by connecting to a pad which is not running.

KILLPAD

This button kills the currently selected pad server. This causes (or should cause) the x, pen, and audio servers to go down as well.

ENVIRONMENT

The environment variable EMUVERSION controls which version of the emulator is started. If this is not set then it will default to vcurrent, which is the most current stable version.

The DISPLAY environment variable should be set to where the emulator window should go.

SEE ALSO

vq_play(1)

BUGS

Uplink audio can stop unexpectedly under high downlink video traffic. If this happens, simply go to A/V window and stop and restart the AUDIO REC.

The vq_play program blocks on input if no video data is sent to it so it will not redraw itself in response to an X paint request. Need to send stay alive packets out to VQ play.

AUTHOR

Currently maintained and improved by Jeff Gilbert <gilbertj@eecs.berkeley.edu>

Initial work and developement until about October 1994 by
Brian Richards <richards@eecs.berkeley.edu>

A.3. imgcomp2d (1)

NAME

imgcomp2d - Two-dimensional fast automatic block decomposition compressor / decompressor

SYNOPSIS

imgcomp2d [options] filename

DESCRIPTION

imgcomp2d is the compression program used to perform research into the Flexible Automatic Block Decomposition (FABD) algorithm. The application performs compression, decompression and also generates diagnostic images which depict the size and location of copy and fill blocks. The application allows many parameters of the compression to be tailored to investigate tradeoffs between compression time and efficiency.

OPTIONS

-help

Show this message

-compress

Compress ras->FABD

-uncompress

Uncompress FABD->ras

-compress_test

Compress, uncompress & verify

-write_in_ras

Write input ras file to stdout

-no_write_out

Don't write any output file

-test_bit_pack

Test bit packing code

-make_diag_image

Make diagnostic image

-make_diag_image_slow

Slower method

-make_diag_image_really_slow

You guessed it...

-save_distribs

Used with -make_diag_image to save param distributions to files

-min_fill_wid_no_search n

Minimum fill width,

-min_fill_hei_no_search n
-min_fill_area_no_search n

height and area to avoid copy search

-min_copy_block_width n
-min_copy_block_height n
-min_copy_block_size n

Minumum copy block width, height, and unmarked pixels to consider.

-max_copy_block_width n
-max_copy_block_height n

Maximum copy block width and height to consider

-min_fill_block_area n
-min_fill_block_size n

Minimum width*height and unmarked pixels for a fill

-max_fill_block_width n
-max_fill_block_height n

Maximum fill block width and height to consider

-max_matches_to_try n

This is the maximum number of matches for each block to try

-one_pass_mode n]

Single more accurate pass

-two_pass_mode n

Coarse size mod 4x4 pass followed by mode accurate pass on winner (default)

-verbosity n

0=quiet, higher=noisier
1=Just summary info
2=Also show progress
3=Also save stats to stat.doc
4=Also print as go along
5=Print too much stuff

SEE ALSO

“Gilbert, Jeffrey M. and Robert W. Brodersen. “A lossless 2-D image compression technique for synthetic discrete tone images.” 1998 IEEE Data Compression Conference. Snowbird, Utah. 28 Mar - 1 Apr 1998. p.359-368.

BUGS

What’s a bug?

AUTHOR

Jeff Gilbert <gilbertj@eecs.berkeley.edu>

A.4. mpeg2vq (1)

NAME

mpeg2vq - MPEG and RAW to InfoPad VQ and RAW Video Transcoder

SYNOPSIS

mpeg2vq [mpeg_file] [options...]

DESCRIPTION

The MPEG and RAW to VQ and RAW Video transcoder converts MPEG or RAW files into VQ or RAW video format files (see vq(5) and raw_video(5)). The mpeg decoding capabilites of the transcoder are taken from mpeg_play. MPEG files can be converted directly to VQ or else the RAW format can be use as an intermediary. This could be for further processing or inspection and additionally the RAW file format can be generated by external sources which would like to be shown on InfoPad. A RAW-to-RAW conversion is possible but would only be useful for resizing or resampling in L, Cr, Cb space.

Many different modes of operation are supported, allowing coding speed and accuracy to be traded off. A fast coding technique can be used to code at frame rate while adaptive codebook methods allow higher quality for off-line VQ file generation. Frames can also be dropped to code a 30 fps movie at 15 fps to reduce network and bandwidth requirements at playback.

GENERIC OPTIONS

-help

Print help message describing options.

-raw_in

Read in a RAW video file (raw_video(5)) instead MPEG.

-nob

Skips over MPEG type B frames in source if reading MPEG.

-nop

Skips over MPEG type P frames in source if reading MPEG.

-eachstat

Shows MPEG statistics if reading MPEG.

-quiet

Suppresses diagnostic messages.

-max_frames <number>

Convert only <number> frames. The rest are discarded.

-frames_per_sec <number>

This value is placed in the VQ or RAW file as the desired frame rate. If this is not specified then the value is taken from the source file. If the source file also does not contain frame rate information then a default of 30 frames per second is used.

-skip_every <number>

Causes the specified number of frames to be skipped for every frame that is transcoded. Thus -skip_every 9 causes only 1 out of 10 frames to be transcoded resulting in a reduction in bandwidth requirements of 10. To halve the frame rate, use -skip_every one. The frame rate recorded in the file is automatically corrected. Defaults to 0.

-src_geometry <w>x<h>+<x>+<y>

Describes the rectangle within the MPEG or RAW source movie from which the image should be taken. This can be used to zoom into a particular part of the source image. Defaults to the full size of the source. If the source contains luminance and chrominance planes at different resolutions then the largest dimensions are used as a reference and the others are scaled accordingly.

-dest_geometry <w>x<h>

Specifies the dimensions of the target VQ screen or RAW file. For VQ, this is measured in descompressed pixels - i.e. the number of Y pixels - and defaults to the InfoPad video screen dimensions of 128x240. For RAW outputs, this is with respect to the largest plane (usually the luminance plane) and defaults to the source dimensions. The relative sizes of the luminance and chrominance planes in a RAW output file can be set using the -raw_ratios option below.

VQ OPTIONS

-dump_codebook

For VQ, specifies that the codebook should be placed at the beginning of the VQ file. This is on by default.

-no_dump_codebook

For Vq, specifies that the codebook should not be placed in the VQ file.

-codebook_file <filename>

Code to an existing codebook file. A full codebook search method is used which is slower (a few frames per second on a Sparc10) than the fast coding method (20-25 fps on a Sparc10, over 25-40 fps on a sparc 10) but obtains better results if the codebook is good. Codebooks can be extracted from vq(5) files using vq2codebook(1). The vq files (and hence codebooks) can be generated with this program using codebook adaptation (see -adapt_frames below).

-fast_coding

Uses a fast codebook hand-constructed out of different shapes and intensity variations. It conists of solid blocks, horizontal, vertical, and diagonal gradations. Since mpeg2vq knows the exact hierarchical nature of the codebook, it is able to quickly determine which entry is best matched. Most of the transcoding time is spent decoding the MPEG. This option is on by default. If it is on, it implies -halfres_coding unless -fullres_coding is specified.

-uniform_coding

Uses a codebook consisting of solid entries - i.e. each of the 4x4 pixels has the same value. This results in a very blocky VQ picture and is mostly only good for debugging purposes. It is also very fast.

-halfres_coding

Causes image to be coded at half-resolution. Thus if the destination format is 128x240 (i.e. InfoPad) - a 64x120 image is computed and broken into 2x2 blocks. This results in an increase in coding speed but should not be used except for with fast and uniform coding as it will introduce errors unless the 4x4 codebook blocks are smooth. This is on by default with fast and uniform coding but can be overridden with -fullres_coding.

-fullres_coding

Opposite of -halfres_coding. Causes images to be coded at their full resolution. Should be used with any full codebook search as implied with -codebook_file or -adapt_frames.

-no_VQconv

Suppresses the VQ conversion step. Only the MPEG decoding is performed. Can be used to tell how much time is going into the MPEG decoding and how much is going into image resizing and VQ coding.

-adapt_frames number

Enables codebook adaptation. The codebook is determined by running a K-means type adaptive algorithm to generate a representative set of vectors for the codebook. The first number of frames are used for adaptation. The adapted codebook is then saved in the VQ file and also used for coding. The -adapt_global_threshold option below can be used to for the codebook to be recomputed in the middle of the movie.

-adapt_global_threshold <number>

Causes codebook (re)adaptation to occur if the coding error ever exceeds the specified threshold. The codebook is re-adapted and placed in the vq file so the the player or hardware decoder knows that is has changed. This could be used for scene-level codebook adaptation. Currenly not supported well by hardware due to flashes on the screen that happen during codebook updates. adapt_frames must also be specified.

-error_tolerance <number>

This is used with full-codebook search methods (either via -adapt_frames or -codebook_file. It causes the search for the best codebook entry to stop when the error drops below the specified limit. Since the codebook entry that was used in the previous frame is used as the initial guess for the current frame, this allows quicker coding espe-

cially for movies where some parts of the image are stationary. Setting this too high can result in a reduction in coding quality. coding

-print_vq_error

Causes the VQ coding error for each frame (for each of Y, I, and Q) to be displayed.

RAW OPTIONS

-raw_out

Specifies to write raw video instead of VQ. Identical to -raw_ratios 2 2 1 1 1 1. (see below).

-raw_ratio <L_h> <L_v> <Cr_h> <Cr_v> <Cb_h> <Cb_v>

Specifies to write raw video instead of VQ but using a particular ratio of L (luminance), Cr (red chrominance), and Cb (blue chrominance) frame sizes. The exact value of the numbers does not matter, only their ratio. The ratios of L_h : Cr_h : Cb_h gives the ratio of the widths of the L, Cr, and Cb frames in the raw output frame. Similarly, the ratios of L_v : Cr_v : Cb_v gives the ratio of the heights of the L, Cr, and Cb frames in the raw output frame. Specifying a zero height or width for a particular plane causes it not to be present in the out. A standard 4:1:1 encoding (which is default) is specified via 2 2 1 1 1 1. A 4:2:2 encoding would be 2 1 1 1 1 1. A grayscale image (luminance only) can be produced using 1 1 0 0 0 0.

If a filename is not specified then the standard input is assumed.

OPERATION

The operation of the transcoder can be broken down into five distinct pieces: MPEG or RAW decoding, Image resizing, colorspace conversion, Vector Quantization, and Codebook Adaptation (optional). If RAW video is generated instead of VQ, the last three pieces are not used.

-> 1) MPEG or RAW decoding

Mpeg2vq starts by decoding the MPEG or RAW file into a virtual frame buffer. For MPEG, this part was taken directly from mpeg_play. The frame buffer is in luminance-chrominance format with separate buffers for L (luminance), Cr (red chrominance), and Cb (blue chrominance). The L buffer is at twice the resolution of the Cr and Cb buffers in both dimensions. (Just as the VQ's Y is at twice the resolution in both dimensions of the I and Q buffers.) The image frame buffers are sized according to the source MPEG's dimensions.

For RAW video file decoding (or really just reading), the L, Cr, and Cb buffers are sized in accordance with the size of the image planes in the file. If image planes are not present in the file, they are simply not read.

-> 2) Image resizing

The next step is to resize the images into the size needed for the VQ coding or RAW

file output. For VQ, if -fullres_coding is in effect then the L buffer is resized to the -dest_geometry size, and the Cr and Cb buffers are resized to half of this size (although they start out half as large in each direction anyhow so the rescaling is by the same amount.) If -halfres_coding is in effect then all image buffers are reduced further by a factor of two in each direction.

For RAW output, the images are resized to the size specified by dest_geometry and the raw_ratios. The dest_geometry defaults to the source geometry and the raw_ratios default to 4:1:1. For RAW output, the process is complete and the resized image buffers are written out.

-> 3) Colorspace Conversion (VQ only)

For VQ, next the Cr and Cb image buffers are converted into I and Q space using a linear combination. Cr/Cb space differs slightly from I/Q space.

-> 4) Vector Quantization (VQ only)

Finally for VQ, the resized, colorspace-converted Y, I, and Q frame buffers are passed to the vector quantization module which attepts to fit the 4x4 pixel blocks (or 2x2 for half-res coding) to the current codebook. This can be with the fast, uniform, or full search methods. This generates a set of codes which are written into the VQ file.

-> 5) Codebook Adaptation (VQ only - optional)

If selected via the -adapt_frames option (and possibly -adapt_global_threshold as well), the codebook is adapted to the set of pixel vectors to derive a codebook well suited for the particular images present. This is done by considering all 4x4 pixel clusters of a given type (Y, I, or Q) over all frames to be adapted as a set of regular 16 element vectors.

The codebook adaptation requires an initial estimate of the codebook. This is the codebook that would be used if -adapt_frames was not specified: the fast (deterministic) codebook if -fast_coding, or an existing codebook if -codebook_file was specified.

The K-means clustering algorithm adapts the codebook as follows: The codebook is used to code the vectors. Then each codebook entry is recomputed as the average of all image vectors for which it is the best match. Thus the codebook entries are modified to better represent the vectors that it is representing. Then the vectors are recoded and the codebook recomputed until the total coding error stop decreasing.

A couple of extra steps are used to ensure that the codebook represents the diversity in the image. Firstly, the 256 codebook entries are compared to each other and if two are too similar then one is "freed" up for use by some other vector. The vectors which matched to the freed codebook entry are then assigned to the one that it was similar to. Next, the unused codebook entries are filled with the input image vectors which had the greatest coding error. This is done to ensure codebook diversity.

ENVIRONMENT

None.

SEE ALSO

vq_play(1) send_vq(1) vq2codebook(1) codebook2ras(1) showcodebook(1) showvqcodebook(1) raw_video(5) vq(5)

BUGS

Does not have aspect_pad and aspect_crop options which will either pad the image with black on the sides or crop off excess in order to make sure that the aspect ratio is correct.

If -adapt_frames is specified as more frames than there actually are, it will not transcode at all.

Much better image quality could be achieved if the pad could receive codebook updates in-line inbetween scenes and the -adapt_global_threshold could be used.

AUTHOR

Jeff Gilbert <gilbertj@eecs.berkeley.edu>

Based on UCB mpeg_play application for MPEG decoding.

A.5. netem (1)

NAME

netem - Flow-level network emulator

SYNOPSIS

netem [options] [proxy_port[u]=server_spec ...]

Server_spec is one of:

web	Web proxy
<server_port>	Fixed port on same host
<server_host>:<server_port>	Fixed host on another host

Add u to proxy_port to use UDP instead of TCP (*not implemented yet*)

DESCRIPTION

netem is a multi-connection flow-level network emulator that can be used to emulate bandlimited links with fixed individual or total link capacity and/or transport latency. Netem can also print connection rate information as well as display the data being transferred on the emulated connections. The modeling is done at the connection level and not the packet level with an emphasis on high throughput rather than packet-level modeling accuracy. Thus the intracacies of TCP/IP are not modeled but it will give a good idea of how applications and algorithms will react to link limitations. Netem will forward arbitrary network connections as well as web proxy requests.

OPTIONS

-help

Show this message

Rate Limiting Options:

-rate_limit_bps n

Limit rate to n bps

-limit_total

Limit total rate through gateway (default)

-limit_invidivual

Limit on a per-connection basis

-buffer_size_ms n

Amount of data to buffer if rate limiting

-min_buffer_size n
-max_buffer_size n

Minimum and maximum number of bytes to buffer

-buffer_size n

Give exact buffer size in bytes

-tcp_connect_time_ms n

Wait n ms after TCP connection after TCP connection

-udp_packet_overhead_ms n

Add n ms delay to UDP packets delay to UDP packets (*not implemented yet*)

-latency_ms n

Delay all packets at least this much

Packet Dropping Options (UDP only - not implemented yet):

-packet_drop_rate 0.XXXX

Drop this fraction of packets

-bit_drop_rate 0.XXXX

Drop packets containing any errant bits according to this fraction of bits

Other options:

-verbosity n

What to print. bit-OR these:

1=Show (dis)connections
2=Show xfer sizes
4=Show xfer data
8=Show errors
16=Show traffic rates
(default is 25)

-max_connections n

Maximum number of connections to support at once

-find_free_proxy_port

Try successive proxy ports if busy (default)

-no_find_free_proxy_port

Don't find_free_proxy_port

SEE ALSO

SurfServ(1) Gilbert, Jeffrey M. and Robert W. Brodersen. “Globally Progressive Interactive Web Delivery.” Proceedings 1999 IEEE Infocom. New York. 21-25 Mar 1999.

BUGS

UDP forwarding / limiting not implemented yet

AUTHOR

Jeff Gilbert <gilbertj@eecs.berkeley.edu>

A.6. send_vq (1)

NAME

send_vq - VQ Video Player and T/G, Audio, and Radio Tester for InfoPad

SYNOPSIS

send_vq [input_file] [options...]

DESCRIPTION

send_vq is utility that allows direct interfacing to the InfoPad peripherals either via the InfoPad network or directly through the GPIB interface. It has two primary purposes. First, it is used to test the network, pad hardware, and emulator. It is also used to play VQ videos on the pad (as opposed to vq_play(1) which plays them on a workstation).

As a tester, send_vq can write videos to the VQ video screen, sound files to the audio downlink port, test graphics patterns to the text / graphics, and arbitrary data to the radio transmitter. It can read and display or store data from the audio uplink port, pen port, and radio receiver. It allows for fixed rate control over the GPIB and can route messages via the ARM processor in a basestation board to a remote pad. It also has a bandwidth test mode to determine the maximum bandwidth the the GPIB hardware can support on the downlink side.

As a VQ video player, send_vq parses the vq(5) format generated by mpeg2vq(1) allowing both codebook and frame data to be sent. It can do rate control, including frame dropping if it sees it is getting behind. Additionally, synchronized audio can be played coordinated with the video. Send_vq can also be used to generate a C include file to be linked in with the ARM code to specify a default VQ codebook on power-up.

GENERAL OPTIONS

-help

Print help message describing options.

-gpib_id <number>

Specifies that direct interface to the GPIB should be used. The <number> is the GPIB talker/listener address. (Hardware is currently set for 4). Setting the environment variable GPIBID to a number makes the GPIB connection a default if neither -gpib_id nor -pad_id are specified. The command-line options always override the environment variables. Both uplink and downlink tests are supported over the GPIB interface.

-pad_id <number>

Specifies that test should be performed over the InfoPad network. The <number> specified is the ID of the pad server to send data to. Setting the environment variable

PADID to a number makes the network connection a default if neither -gpib_id nor -pad_id are specified. The command-line options always override the environment variables. Note that only downlink tests are available over the network connection. Uplink tests must be performed using a direct GPIB connection.

-show_gpib_data

Show GPIB data packets being sent. For debugging purposes.

-loop

Repeat the specified test. Only valid for downlink tests (including VQ video playing). Input must not come from standard in for video, audio, and radio downlink tests.

-quiet

Don't print quite so much diagnostic infomation to stderr. This is the default - override with -no_quiet.

-no_quiet

Print lots o' stuff to stderr. Information depends on the test.

-gpib_rate_limit <kbps>

When sending over GPIB, set a cap on the rate of downlink data. The number is specified in kilo-bits per second and should be a multiple of 8. Note that for large packets, this can become approximate and you should check the rate that it says that it is sending at, shown at the end of the test. This is independent from video rate control. (Video rate control is more accurate as well)

-gpib_buffer_size <n>

Buffer size that the GPIB rate limiting uses for accounting purposes. It should larger than the packets that will be transmitted but no larger than the buffering on the actual pad.

-send_via_arm

Send GPIB packets to the ARM (acting as the TX chip) to send over the radio. Use to send over GPIB to remote pad. Packets will still be tagged with correct destination (via type field.) You almost definitely have to use -gpib_rate_limit (above) if you use -send_via_arm.

-single_byte_packets

Use old single byte GPIB protocol. Used for testing a long time ago. Probably of no use to anyone now.

UPLINK TEST OPTIONS

These options are used for all (audio, pen, radio) uplink tests. Uplink tests may only be performed directly over the GPIB.

-gpib_gcr1 <hex_num>

Specifies the value, in hex, to be programmed into the GPIB Control Register 1 for

uplink tests. This controls several modes of operation of the GPIB interface. The default is currently 88.

-gpib_gcr2 <hex_num>

Specifies the value, in hex, to be programmed into the GPIB Control Register 2 for uplink tests. This controls several modes of operation of the GPIB interface. The default is currently 84.

-gpib_base_addr <hex_num>

IP Bus base address (in hex) for GPIB controller. This is where peripherals will be instructed to send their data when it should go to the host. The default is 70.

VIDEO OPTIONS

-rate_control

Times when video frames are send and waits if it is ready too soon for the next frame or drops frames if it is too late. The frame rate is specified in the vq(5) file but can be overridden by the -frame_rate option.

-rate_limit

Will wait to play frames to make sure that it is not playing too fast but will not drop any frames if it is playing too slow. This option is the default but can be overridden by the -rate_control and -no_rate_limit options.

-no_rate_limit

Disables frame rate control - i.e. plays the video at peak channel bandwidth.

-frame_rate <fps>

Override the frame rate specified in the VQ file. Affects -rate_control and -rate_limit.

-no_codebook_updates

Causes codebook updates found in vq(5) stream (including initial codebook information) to be ignored. Typically used to play videos that have been coded with a codebook that is known to the pad already (such as with the -fast_coding option of mpeg2vq(1)). In this way the codebook does not have to be sent and there are no chances of codebook corruption.

-codebook_file <name>

Sends the specified codebook (even if -no_codebook_updates is specified) before sending the VQ file. Note that unless the -no_codebook_updates is specified and the VQ file contains a codebook (as most do) then the initial codebook will immediately be overwritten by the ones in the VQ file. Codebook files can be generated via vq2codebook(1).

-no_resend_codebooks_on_error

Usually, if sending over the GPIB and an error is encountered, the full codebook is resent as the GPIB error could have been from a power-down or board reset. To prevent this from happening, use this flag.

-video_sound_file <file>

Specifies an audio file to play with video. The audio file is in Sun 8-bit u-law, 8 kHz .au format. The audio is send to the AF server specified by AUDIOFILE or DISPLAY. This can be either an AF running on a workstation, or the InfoPad Audio Server, which is an AF server which sends its audio to the pad. The audio is synchronized to the video in that the au file is played at exactly 8000 samples per second so if the -rate_control flag is specified, the video rate can be controlled exactly. The -video_delay_sound option (below) can be used to adjust the relative starting points of the audio and video for a perfect match.

-video_delay_sound <ms>

This options is used to specify a delay between when the video and audio are sent. This can be used to compensate for network, hardware, and recording latencies. The number is specified as the number of milliseconds to delay the audio relative to the video. Negative values cause the audio to be sent before the video.

-all_y_code <n>

The -all_y_code option causes all Y codebook data to be changed to the specified value before being sent off. Thus a normal codebook update would cause the entire Y codebook to be changed to the value specified. <n> is in actual hardware format - i.e. an integer between 0 and 63 with 0 representing black and 63 white. Setting the entire Y codebook to one value causes it to ignore the Y frame data, removing the Y frame buffer from the test path.

-all_i_code <n>

The -all_i_code option causes all I codebook data to be changed to the specified value before being sent off. Thus a normal codebook update would cause the entire I codebook to be changed to the value specified. <n> is in actual hardware format - i.e. a 1 bit sign and 5 bit magnitude integer between 0 and 63 with 0 through 31 representing +0 through +31 (tending towards more yellow) and 32 through 63 representing -0 through -31 (tending towards more cyan). Setting the entire I codebook to one value causes it to ignore the I frame data, removing the I frame buffer from the test path. For example, to see the video in black and white (Y only), specify -all_i_code 0 -all_q_code 0.

-all_q_code <n>

The -all_q_code option causes all Q codebook data to be changed to the specified value before being sent off. Thus a normal codebook update would cause the entire Q codebook to be changed to the value specified. <n> is in actual hardware format - i.e. a 1 bit sign and 5 bit magnitude integer between 0 and 63 with 0 through 31 representing +0 through +31 (tending towards more magenta) and 32 through 63 representing -0 through -31 (tending towards more lime-green). Setting the entire Q codebook to one value causes it to ignore the Q frame data, removing the Q frame buffer from the test path.

-all_y_data <n>

Sets all Y frame data sent to one value. <n> is the value (between 0 and 255) to set the

Y frame data to. All accesses to the Y codebook should then be to the specified location and the screen (at least as far as the Y plane is concerned) should be the repeating pattern of the selected entry.

-all_i_data <n>

Sets all I frame data sent to one value. <n> is the value (between 0 and 255) to set the I frame data to. All accesses to the I codebook should then be to the specified location and the screen (at least as far as the I plane is concerned) should be the repeating pattern of the selected entry.

-all_q_data <n>

Sets all Q frame data sent to one value. <n> is the value (between 0 and 255) to set the Q frame data to. All accesses to the Q codebook should then be to the specified location and the screen (at least as far as the Q plane is concerned) should be the repeating pattern of the selected entry.

-gen_codebook_prom_include

If this flag is specified then a C include file containing the codebook data in a format suitable for the pad ARM code is written to the standard out. It contains the packets to be send to the Video chip to initialize the codebook.

-video_base_addr <hex_num>

For GPIB operation, send_vq has to know how to address the video packets going over the IP Bus. This corresponds to the address of the video chip. (The lower four bits are always 0.) The value is specified in hex and its default is 60. For InfoPad network operation, this is not used.

AUDIO TEST OPTIONS

-test_audio

Specifies that a downlink audio test is to be performed. The input file (or the standard input if no input file is specified) is interpreted as being a 8kHz, 8-bit u-law audio file. It is sent out at a rate-controlled 8kHz.

-record_audio <dest_file>

Does audio uplink test. The audio chip is initialized to record and send data over the GPIB to the host. The host then records the data (8 bit, 8kHz) in the specified <dest_file>. If the file name starts with a "+" then the output is sent both to the file (after removing the "+") as well as the /dev/audio of the host. This uplink test, as well as all uplink tests, can only be performed directly over the GPIB.

-record_audio_pen <dest_file>

Same as above by tries to interpret and display pen packets if they are interleaved. Probably will not work as it was a hack.

-test_audio_in

Test audio uplink printing status messages to the screen. Does not record data.

-record_audio_send_video <dest_file>

Tests simultaneous audio uplink with video downlink. The <dest_file> is again the file to record the data in. The video is taken from the supplied file name or standard in otherwise. May or may not work - it was a hack.

-audio_base_addr <hex_num>

For GPIB operation, send_vq has to know how to address the audio chip over the IP Bus for both data and control - i.e. the address of the audio chip. (The lower four bits are always 0.) The value is specified in hex and its default is 40. For InfoPad network operation, this is not used.

RADIO TEST OPTIONS

-test_xmit

This flag specifies test transmitter mode. This can test the TX chip (or ARM) over the radio or wired link. The input file (or standard input) describes the packets to be sent. The packets are directed to the GPIB on the other end of the link so that the -test_recv and -test_recv_timestamp options can be used at the other end to recover the data. (This test cannot be run using the InfoPad network infrastructure.) The -loop option (above) can be used to repeatedly send the packets. The -gpib_rate_limit option (above) should be used to limit the transmit rate as no flow control is employed. This test can only be performed using the direct GPIB method. The file is in ASCII with each packet represented by a single line and numbers within the line to are 2 digit hex (lower and upper case letter are ok) numbers. The following sample specifies four packets to be sent for a total of 15 bytes (plus IPN and radio headers):

```
01 02 03 04 05 06
07 08 09 0A
0B 0C
12 34 56
```

-test_xmit_timestamp

This test is as above except that additionally the transmit time of each packet is recorded and printing, along with the transmitted data, to standard out. The time is given both relative to the first packet as well as to the previous packet sent. The output should be piped to a file or else the significant delays may occur while printing it to the screen in real-time. The transmit timestamp is useful in conjuction with the receive timestamp (see -test_recv_timestamp) to determine fifoing delays and mis-ordering. A sample output from -test_xmit_timestamp looks like this:

```
00000000 ms (    +0 ms) ( 6 bytes): 01 02 03 04 05 06
00000001 ms (    +1 ms) ( 4 bytes): 07 08 09 0A
00000003 ms (    +2 ms) ( 2 bytes): 0B 0C
00000004 ms (    +1 ms) ( 3 bytes): 12 34 56
```

-test_recv

This is the receiver end of the radio / wired link test. This can be run on the receiver

side any time before the transmit test (above) is run on the transmit side (in order not to lose any packets.) This test must be run using a direct gpib connection - i.e. not be using the InfoPad network. It will output to standard out in the same format that the -test_xmit expects: ASCII with one packet per line and each byte represented as a 2 character hex number. In this way, the file transmitted and the file received can be compared using the diff command. The output should be piped to a file or else the significant delays may occur while printing it to the screen in real-time.

-test_recv_timestamp

Same as above but the output file includes timestamps and message sizes as with the -test_xmit_timestamp option. This can be used to quickly scan which packets are of incorrect size and also what the relative timings are. To convert a file with timestamps (generated by this command) to one without - such as generated by -test_recv and used by -test_xmit and -test_xmit_timestamp, the following sed command will work:

```
sed "s/.*: //g" INPUT_FILE_NAME
```

BANDWIDTH, TEXT / GRAPHICS, and PEN TESTING OPTIONS

-test_bandwidth

Runs a test of the GPIB downlink bandwidth. (This test cannot be run over InfoPad Network.) Sends packets (all bytes 0) of varying size over GPIB link and measures time it takes. It tests for packets of length 4 bytes to 1 megabyte. It automatically determines how many packets to send to get accurate results. The progress is reported as it goes along and then a summary, like the following, is printed (this was from a slow machine):

Packet Size	Packets/Sec	KBytes/Sec	KBits/Sec
4	679.947	2.720	21.758
8	679.496	5.436	43.488
16	386.124	6.178	49.424
32	376.748	12.056	96.447
64	591.566	37.860	302.882
128	521.651	66.771	534.170
256	221.645	56.741	453.929
512	306.220	156.785	1254.278
1024	177.285	181.540	1452.321
2048	115.004	235.529	1884.234
4096	59.150	242.277	1938.218
8192	15.795	129.390	1035.119
16384	16.649	272.783	2182.260
32768	5.525	181.039	1448.309
65536	3.086	202.272	1618.173
131072	1.528	200.263	1602.102
262144	0.888	232.913	1863.307
524288	0.431	226.084	1808.669
1048576	0.242	253.708	2029.666

-test_tg

Runs text / graphics downlink test. The test exercises all four graphics primatives of the T/G chip. First an image of a prototype pad is displayed using the Block (B) mode. Then the horizontal (H) mode is used to make a little lined pattern in the center. Next, another small section is drawn using the vertical (V) mode. Next a reverse-video image of the prototype is shown using the Protected Block (PB) mode. Then a small area is scrolled around using the block mode again. Finally, the whole image is scrolled around using the block mode. The downlink bandwidth used for this last test is printed out. Use -pause_time option (below) to have it pause between tests. Use -no_quiet (above) to print info about tests being performed. The T/G tests can be run over InfoPad Network or directly through GPIB.

-pause_time <secs>

This specifies the number of seconds to pause between graphics tests. Defaults to 1.

-tg_base_addr <hex_num>

For GPIB operation, send_vq has to know how to address the t/g packets going over the IP Bus. This corresponds to the address of the t/g chip. (The lower four bits are always 0.) The value is specified in hex and its default is 50. For InfoPad network operation, this is not used.

-test_pen

Used to test the pen uplink. The pen chip is configured and pen packets are printed to the screen. This test, like all other uplink test, must be performed over the GPIB directly.

-pen_base_addr <hex_num>

For pen uplink tests, which must be over the GPIB, send_vq has to know how to address the pen chip to configure it. The number here is the base address the lower four bits are always 0. The value is specified in hex and its default is 30. For InfoPad network operation, this is not used.

If an input file is not specified but the particular test (or VQ send) requires it then the standard input is assumed.

FILES

/tools/ui/movies/adapt_vq and /tools/ui/movies/fast_vq

Contains sample VQ files that were converted using mpeg2vq(1). The MPEG source files are in /tools/ui/movies/mpeg.

ENVIRONMENT

The GPIBID and PADID environment variables are used if either -gpib_id nor -pad_id are specified. If they are not and GPIBID is set to a number then testing is performed over the GPIB to a board whose talker/listener address is the number. If PADID is set to a number then the tests are run through the InfoPad network through the specified pad server.

If syncronized video and audio are requested via the -video_sound_file flag, the audio goes through AudioFile (AF). The AF libraries look use AUDIOFILE to determine where to send the audio. It should be set the same way you would set your DISPLAY variable. If AUDIOFILE is not set then DISPLAY is used.

SEE ALSO

emu(1) mpeg2vq(1) vq_play(1)

BUGS

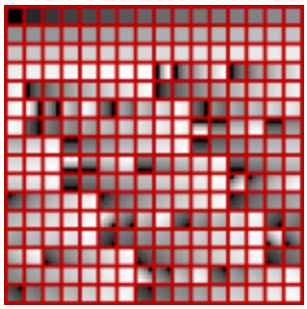
None, of course.

AUTHOR

Jeff Gilbert <gilbertj@eecs.berkeley.edu>

A.7. show[vq]codebook[y] (1)

NAME



showcodebook, showcodebooky, showvqcodebook, showvq-codebooky - VQ Video Codebook Display Utilities

SYNOPSIS

showcodebook codebook_filename

showcodebooky codebook_filename

showvqcodebook vq_video_filename

showvqcodebooky vq_video_filename

DESCRIPTION

These tiny scripts just pipe the output of codebook2ras(1) into the xloadimage viewer for X, optionally zooming in on the Y part of the codebook and optionally stripping the codebook out of a vq(5) video file. (vq2codebook(1) can also do this).

OPTIONS

Showcodebook and showcodebooky must be supplied the name of a codebook file (one that could be fed into mpeg2vq(1)). showvqcodebook and showvqcodebooky must be supplied the name of a VQ video file (one that could be played with vq_play(1) or send_vq(1)).

OPERATION

Since codebook2ras generates a SUN rasterfile which xloadimage can display, it is basically a no-brainer. It does specify -gamma 2.25 to xloadimage to gamma correct and make the image more readable. Showvqcodebook and showvqcodebooky first pass the vq file through vq2codebook to extract the codebook. Showcodebooky and showvqcodebooky pass options to xloadimage to make it only show the Y part of the codebook.

SEE ALSO

codebook2ras(1) mpeg2vq(1) send_vq(1) vq_play(1) vq(5)

AUTHOR

Jeff Gilbert <gilbertj@eecs.berkeley.edu>

A.8. SurfServ (1)

NAME

SurfServ - Server for SpeedSurfer connection proxy as well as server for Globally Progressive Interative Web Delivery transformational proxy.

SYNOPSIS

SurfServ [options]

DESCRIPTION

SurfServ is both a server for the SpeedSurfer proxy application as well as for the Globally Progressive Interative Web Delivery transformational proxy. Most of the options are related to the Globally Progressive Interative Web Delivery transformational proxy since the client-side SpeedSurfer application provides controls most of the operation of the server. Connection of either type are accepted on the same port and the type of connection is determined by initial handshaking sequence.

As a server for the SpeedSurfer proxy, SurfServ accepts connections from one or more SpeedSurfer clients and will make connections on behalf of them. The SpeedSurfer / SurfServ pair can be used to force all connections through a single TCP/IP host. The pair can also perform detailed link traffic analysis for web data and display aggregate flow information for all types of data.

As a transformational proxy for the Globally Progressive Interative Web Delivery, SurfServ appears like a standard web proxy, accepting HTTP/1.0 or HTTP/1.1 connections from web browsers. For HTML connections, it transforms image tags into Java applet tags. It then also supports connections back from the Java applets and will manage a single explicitly multiplexed link to deliver the images to the applets as defined by the Globally Progressive Interative Web Delivery algorithm.

OPTIONS

-help

Show this message

-port n

Port to listen on

-webProxy host:port

Forward all proxy requests here

-no_image_applets

Don't use applets for images

-no_include_image_size

Don't add size to unsize images

-no_compress_html

Don't compress HTML (HTTP/1.1)

-no_html_priority

Don't give priority to HTML

-force_image_applets

Use applets whenever possible

-max_html_for_non_html n

Max # bytes of html that can be queued to queue non-html

-max_html_in_flight n

Max # bytes of html buffered

-max_image_in_flight n

Max # bytes of images buffered

-max_speedimg_in_flight n

Max # bytes of custom buffered. Further limited via down link window

-browser_port_offset

Amount to add to proxy port

-applet_dir_url

Where to get speedImage applet. Default is
http://badlands.EECS.Berkeley.EDU:8090/~gilbertj/ttt

-scans scan_file

Progressive scan file to use otherwise use default

-verbosity n

What to print. bit-OR these:

1=Show proxy connections

2=Show errors

4=Show connects/disconnects

64=Some debugging

128=More debugging

(default is 71)

-max_sessions n

Maximum number of connections & sessions to support at once

-find_free_proxy_port

Try successive proxy ports if busy (default)

-send_timing_packets

Send timing packets that SpeedSurfer will log

-dont_send_timing_packets

Don't send timing packets that SpeedSurfer will log

SEE ALSO

netem(1) Gilbert, Jeffrey M. and Robert W. Brodersen. “Globally Progressive Interactive Web Delivery.” Proceedings 1999 IEEE Infocom. New York. 21-25 Mar 1999.

BUGS

None that I’d care to discuss

AUTHOR

Jeff Gilbert <gilbertj@eecs.berkeley.edu>

NAME

vq_play - VQ and RAW Video Player
for X Windows

SYNOPSIS

vq_play [vq_or_raw_file] [options...]

DESCRIPTION

vq_play is used to display VQ video (see vq(5)) or RAW video (see raw_video(5)) format files under X Windows on a standard workstation. It is based on mpeg_play and can also handle MPEG files if passed a command-line switch. All of the display modes of mpeg_play are supported (i.e. dithering, shared-memory etc) for playback of VQ, RAW, and MPEG files. Gamma correction and zooming are additionally supported for all three types as well. For VQ and RAW file playback, it additionally supports frame rate control.

NOTE: The parts of this man page which describe mpeg_play features are taken from the mpeg_play man page. See authors section for list of those responsible for mpeg_play.

GENERIC OPTIONS

-help

Print help message describing options.

-dither dither_option

Selects from a variety of dither options. The possible values are:

ordered - ordered dither.

ordered2 - a faster ordered dither. This is the default.

mbordered - ordered dithering at the macroblock level. Although there is a noticeable decrease in dither quality, this is the fastest dither available.

fs4 - Floyd-Steinberg dithering with 4 error values propogated.

fs2 - Floyd-Steinberg dithering with 2 error values propogated.

fs2fast - Fast Floyd-Steinberg dithering with 2 error values propogated.

hybrid - Hybrid dithering, a combination of ordered dithering for the luminance channel and Floyd Steinberg 2 error dithering for the chrominance channels. Errors are NOT propogated properly and are dropped all togethor every two pixels in either direction.

hybrid2 - Hybrid dithering as above, but with error propogation among pixels. 2x2 - A

dithering technique using a 2x2 pixel area for each pixel. The image displayed is 4 times larger than the original image encoded. Random error terms are added to each pixel to break up contours and gradients.

gray - Grayscale dithering. The image is dithered into 128 grayscales. Chrominance information is thrown away.

color - Full color display (only available on 24 bit color displays).

none - no dithering is done, no image is displayed. Used to time decoding process.

mono - Floyd-Steinberg dithering for monochrome displays.

threshold - Floyd-simple dithering for monochrome displays.

-l_range num_colors

sets the number of colors assigned to the luminance component when dithering the image. The product of l_range, cr_range and cb_range should be less than the number of colors on the display.

-cr_range num_colors

sets the number of colors assigned to the red component of the chrominace range when dither ing the image. The product of l_range, cr_range and cb_range should be less than the number of colors on the display.

-cb_range num_colors

sets the number of colors assigned to the blue component of the chrominace range when dither ing the image. The product of l_range, cr_range and cb_range should be less than the number of colors on the display.

-loop

makes the player loop back to the beginning after reaching the end.

-no_display

dithers, but does not display, usually used for testing and timing purposes.

-quiet

Suppresses diagnostic messages

-shmem_off

Don't use X shared memory for image buffer. Using shared memory is faster but can sometimes cause problems.

-gamma correction_val

Specify the amount to gamma correct the images. Gamma correction warps (brightens) the image to compensate for the non-linear effects of monitors. Sun monitors require about a 2.25 gamma correction while other displays, such as LCD or LiveBoard may require more or less. If this option is not specified then the VQ_PLAY_GAMMA environment is checked for and used. If it does not exist then a default gamma correction of 2.25 is used. Note that this gamma correction is applied to mpeg videos played with the -mpeg option so aliasing mpeg_play to `vq_play -

mpeg' can be used to gamma correct when you mpeg_play. A gamma correction value of 1.0 implies no correction.

-zoom percentage

Display the the image percentage/100.0 times wider and taller than it actually is. Percentages below 100 cause the image to appear smaller on the screen while percentages above 100 cause the image to appear larger. The default is 100 which causes the image to appear exactly the same size. NOTE: resizing is not available for MPEG images. To zoom an MPEG image, pass it through mpeg2vq(1) with the -raw_out flag and pipe to vq_play with the -raw flag. Then zooming and rate control will be available.

-xzoom percentage

Set only the width zoom factor.

-yzoom percentage

Set only the height zoom factor.

-rate_limit

Limit playback speed to the frame rate specified in the vq(5) file or by the -frame_rate option. The playback speed may be slower as frames are not dropped with this mode. Does not work with MPEG playback. Convert to raw first via mpeg2vq(1) -raw_out and pipe to vq_play with the raw flag.

-rate_control

Constrain the playback speed to the exact frame rate specified in the VQ file or by the -frame_rate option. If the playback is going too slow then frames are dropped to keep on schedule. Does not work with MPEG playback. Will usually not work unless the source is a file and not a pipe.

-frame_rate n

Override the VQ file's specified frame rate. Only meaningful with -rate_limit or -rate_control. Does not work with MPEG playback.

VQ OPTIONS

-vq

Specifies that file is a VQ file. (I.e. negates -mpeg and -raw). Since this is the default, this option will probably not be needed.

-VQCodebook filename

Initial VQ codebook to use for lookup. Mostly for use with VQ files which do not have codebook information (i.e. generated with mpeg2vq(1)'s -no_dump_codebook option. VQ files with codebooks at the beginning will override this option. Not valid if -mpeg is specified.

RAW OPTIONS

-raw

Specifies that the input file follows the raw_video(5) file format.

MPEG OPTIONS

-mpeg

Specifies that file is an MPEG file. All of the MPEG decoding capabilities of mpeg_play are built in to vq_play as well as gamma correction.

-eachstat

Shows MPEG statistics. Requires -mpeg.

-nob

Skips over MPEG type B frames in source. Requires -mpeg.

-nop

Skips over MPEG type P frames in source. Requires -mpeg.

If an input file is not specified then the standard input is assumed.

NOTES

The display is automatically stretched vertically to account for the unusual aspect ratio of the InfoPad display (128x240.)

FILES

/tools/ui/movies/fast_vq

Contains sample VQ files that were converted using the fast coding method of mpeg2vq(1). These all use the same codebook but can be generated in realtime.

/tools/ui/movies/adapt_vq

Contains sample VQ files that were adaptively coded using mpeg2vq. These have to be generated off-line but of course can be played in real-time.

/tools/ui/movies/mpeg

The mpeg files from which the VQ files were converted.

ENVIRONMENT

If -gamma is not specified then the environment variable VQ_PLAY_GAMMA is used as the gamma correct value. If it is not does not exist then a default value of 2.25 is used. This is pretty good for Sun monitors.

SEE ALSO

mpeg2vq(1) send_vq(1) vq2codebook(1) codebook2ras(1) showcodebook(1) showvq-codebook(1) vq(5)

BUGS

None.

It might be nice to be able to show the image larger (enlarge before dither) but this is not high priority.

AUTHORS

Addition of VQ and RAW support as well as gamma correction and zooming to mpeg_play (yielding vq_play) by Jeff Gilbert <gilbertj@eecs.berkeley.edu>.

----> mpeg_play written by:

Ketan Patel - University of California, Berkeley, kpatel@cs.berkeley.edu

Brian Smith - University of California, Berkeley, bsmith@cs.berkeley.edu

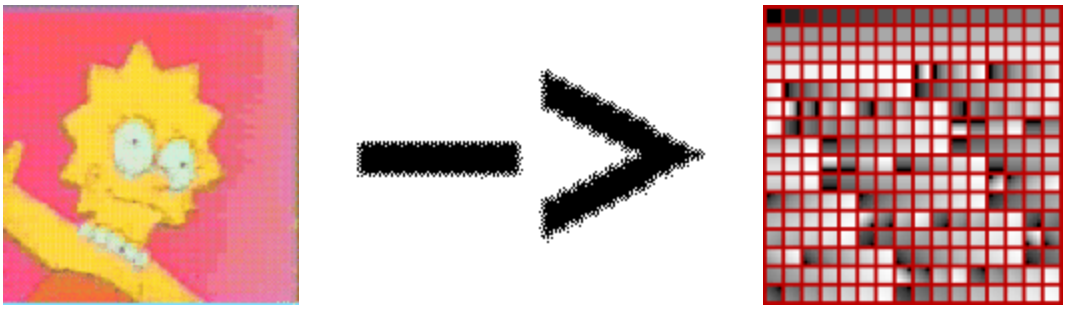
Henry Chi-To Ma - University of California, Berkeley, cma@cs.berkeley.edu

Kim Man Liu - University of California, Berkeley, kliu@cs.berkeley.edu

Steve Smoot - University of California, Berkeley, smoot@cs.berkeley.edu

Eugene Hung - University of California, Berkeley, eyhung@cs.berkeley.edu

A.10. vq2codebook (1)



NAME

vq2codebook - VQ Video file Codebook Extractor

SYNOPSIS

vq2codebook [vq_file] [options...]

DESCRIPTION

This tiny script just extracts the codebook from the specified vq(5) file and writes it to the standard out. This can then be fed back into the mpeg2vq(1) program to transcode new videos to an existing video's codebook. Alternatively it can be fed into codebook2ras(1) to generate a SUN rasterfiles for use in documentation. (Use showvqcodebook(1) to just view the codebook being used in a VQ file.)

OPTIONS

If a vq file name is not specified, standard in is assumed.

OPERATION

The script assumes that the codebook follows directly after the VQ header (as it will if it is generated with mpeg2vq) so it simply extracts the 12288 bytes following the 44 byte header.

SEE ALSO

codebook2ras(1) mpeg2vq(1) send_vq(1) showcodebook(1) showvqcodebook(1) vq_play(1) vq(5)

AUTHOR

Jeff Gilbert <gilbertj@eecs.berkeley.edu>

A.11. XInfoPad (1)

NAME

XInfoPad - Modified X server used as InfoPad Text / Graphics server

DESCRIPTION

XInfoPad is the modified X server used as InfoPad Text / Graphics server. It connects to applications as a standard X server but sends display to the specified PadServer using the virtual framebuffer approach for adaptive bandwidth control. Packets are formated according to InfoPad specifications and communicated using InfoNet networking routines.

SYNOPSIS

XInfoPad [:<display>] [options]

NON X-STANDARD OPTIONS:

(The standard options to X servers are not presented here - see X (1) man page)

-pad_id N

InfoPad emulation, on Pad N

-ipn_rate_limit kbps

Rate-limit the downlink traffic. 0 means unlimited. Default is 200.

-ipn_unbuffered_write

Use IPNWrite instead of IPNWriteBuffered

-ipn_interpacket_delay us

Wait this many us after an IPN send

-max_packet_size n

Maximum number of bytes in IPN packet. Default is 512.

-refresh_rate kbps

Maximum background refresh rate. Default 10kbps.

-invert_video

Invert video. Default no.

-send_everything

Constantly send whole framebuffer - not just updates

-allow_mark_filename XXXX.c

Only allow mark requests from these mfb/cfb files

-deny_mark_filename XXXX.c

Don't allow mark requests from these mfb/cfb files

-dont_grow_marks

Don't grow marks to cover whole 32x1 or region before merging. Useful debugging

-show_mark_requests

Show mfb/cfb mark requests

-show_mark_flushes

Show mfb/cfb mark flushes

-show_merged_marks

Show mfb/cfb merged marks

-show_offscreens

Show also mfb/cfb mark/flushed offscreen

-show_sent_blocks

Show blocks sent by InfoSlave

-show_only_filename XXXX.c

Show only requests from these mfb/cfb files

-show_skip_filename XXXX.c

Don't show requests from these mfb/cfb files

SEE ALSO

codebook2ras(1) mpeg2vq(1) send_vq(1) showcodebook(1) showvqcodebook(1) vq_play(1) vq(5)

AUTHOR

The Split X topology was originally developed by Richard Han, Brian Richards, and Trevor B. The XInfoPad server using the virtual framebuffer architecture was developed by Jeff Gilbert <gilbertj@eecs.berkeley.edu> since ~1995.

NAME

raw_video - InfoPad file format for RAW (non-VQ, non-MPEG) Video files

SYNOPSIS

#include <raw_vid_file_fmt.h>

OVERVIEW

The RAW Video file format is used primarily as a means of using non-MPEG video sources in the InfoPad environment. The RAW file format allows images specified in raw luminance and chrominance pixels to be used by the transcoder (mpeg2vq(1)) and video player (vq_play(1)). (Luminance pixels are also know as L or Y while the chrominance is know as I and Q, Cr and Cb, or U and V. MPEG's definition of Cr Cb is used.) The size of the three image planes is arbitrary. I.e. MPEG/VQ type 4:1:1 ratios where the Cr (I) and Cb (Q) are sampled only a quarter as densly can be specified as well as 4:2:2 or any other combination. Additionally, grayscale movies can be specified by setting the Cr and Cb frame dimensions to 0. A frame rate can also be specified to allow vq_play to rate control and mpeg2vq to put this into the vq(5) file.

The RAW Video file has two parts: a header containing frame size and rate information (as well as additional optional header information) followed by the frame data. All frames contain full updates of any of the planes (L, Cr, Cb) that are non-zero sized. Thus the file looks like:

```
HEADER
EXTRA HEADER DATA
L DATA FRAME 0
Cr DATA FRAME 0
Cb DATA FRAME 0
L DATA FRAME 1
Cr DATA FRAME 1
Cb DATA FRAME 1
   ...
```

or in the case of grayscale:

```
HEADER
EXTRA HEADER DATA
L DATA FRAME 0
```

```
        L DATA FRAME 1
        L DATA FRAME 2
        L DATA FRAME 3
           ...
```

HEADER

The header is defined by the following structure:

```
        /* All numbers should be in net order */
typedef struct _Raw_video_file_header {
        u_long          magic_number;
        u_short         major_version;
        u_short         minor_version;
        u_long          width_L;
        u_long          height_L;
        u_long          width_Cr;
        u_long          height_Cr;
        u_long          width_Cb;
        u_long          height_Cb;
        u_long          image_type;
        u_long          frames_per_sec;
        u_long          extra_data_len;
} Raw_video_file_header;
```

The *magic_number field always contains the following constant:*

```
        #define RAW_VIDEO_MAGIC_NUM     0x52415756 /* `RAWV' */
```

The major_version and minor_version fields contain major and minor version that the file conforms to. Currently 1 and 0 respectively.

Next the L, Cr, and Cb dimensions are specified via width_L, height_L, width_Cr, height_Cr, width_Cb, and height_Cb, fields. These are specified in pixels and can be arbitrarily sized with respect to each other. Additionally setting any of the widths and heights to 0 denotes that that plane is not present in the data. Any of the three planes may be omitted, although most commonly either all three will be present for full color, or else just the luminance (L) plane will be present and the chrominance (Cr and Cb) planes will be absent.

image_type currently must be 0 but may at some time be used to support other image formats or colorspaces.

frames_per_sec is the number of frames per second that the video was recorded at. It is a 32 bit integer but is stored * 65536 so it can be as high as 65536 frames per second and has the accuracy of 1/65536th of a frame per second. The last field in the header is extra_data_len which is used to specify the size of additional header information which followes the header. This could be used to add a frame offset table or other information to the VQ file.

After the header is the optional header data. This data is ignored by vq_play and send_vq.

FRAME DATA

The frame data consists the pixels from the L, Cr, and Cb frames in standard raster order. The header dimensions determine the number of bytes expected. All data are single unsigned bytes. The L data ranges from 0 being darkest to 255 being white while the Cr and Cb data are biased around 128. If Cr or Cb data is not present (i.e. Cr_width==0 or Cb_width==0), it is assumed to be all 128's while if L data is not present it is assumed to be all 0's.

SEE ALSO

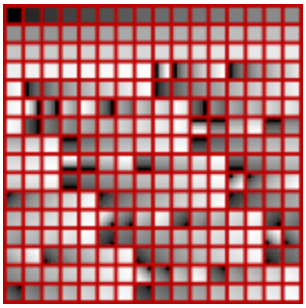
vq_play(1) mpeg2vq(1) vq(5)

AUTHOR

Jeff Gilbert <gilbertj@eecs.berkeley.edu>

NAME

vq - InfoPad file format for VQ Video files



SYNOPSIS

#include <vq_file_fmt.h>

OVERVIEW

The VQ file format is used by the InfoPad VQ Video programs as a means of specifying video streams. The file format is rich enough to be used for other types of VQ files or to add extensions to the existing video. The video transcoder (mpeg2vq(1)) and VQ video player (vq_play(1)) both can operate on VQ files of any frame size. The video send utility (send_vq(1)), however, requires a fixed size due to hardware constraints.

First the general format will be described, followed by how it should be filled in for Info-Pad hardware compatible files.

The vq file has three parts: a header containing frame size and rate information (as well as additional optional header information) followed by codebook update sections interleaved between video data sections. The codebook update sections can specify that no entries are to be updated. Thus the file looks like:

```
HEADER
EXTRA HEADER DATA
CODEBOOK UPDATE SECTION
VIDEO FRAME DATA
CODEBOOK UPDATE SECTION
VIDEO FRAME DATA
CODEBOOK UPDATE SECTION
VIDEO FRAME DATA
   ...
```

HEADER

The header is defined by the following structure:

```
        /* All numbers should be in net order */
typedef struct _VQ_file_header {
        u_long          magic_number;
        u_short         major_version;
        u_short         minor_version;
        u_long          width;
        u_long          height;
```

```
        u_long          frame_size;
        u_long          codebook_entries;
        u_long          codebook_entry_size;
        u_long          frames_per_sec;
        u_long          extra_data_len;
} VQ_file_header;
```

The *magic_number field always contains the following constant:*

```
        #define VQ_MAGIC_NUM     0x49505651 /* `IPVQ' */
```

The major_version and minor_version fields contain major and minor version that the file conforms to. Currently 1 and 0 respectively. The width and height fields are measured in I and Q entries which are one eighth the number of uncompressed pixels. So for the InfoPad hardware resolution of 128x240 this would be width=128/8=16 and height=240/8=30. framesize is the size of a frame in bytes which for InfoPad VQ is width * height * 6. (Since Y is at double the resolution in both directions, there are 4 Y bytes per 1 I and 1 Q). codebook_entries is the number of entries in all codebooks. For InfoPad, this is 256*3=768. codebook_entry_size is the size in bytes of each entry. For InfoPad, this is 16 bytes. (One byte for each pixel in the 8x8 block.) frames_per_sec is the number of frames per second that the video was recorded at. It is a 32 bit integer but is stored * 65536 so it can be as high as 65536 frames per second and has the accuracy of 1/65536th of a frame per second. The last field in the header is extra_data_len which is used to specify the size of additional header information which followes the header. This could be used to add a frame offset table or other information to the VQ file.

After the header is the optional header data. This data is ignored by vq_play and send_vq.

CODEBOOK UPDATES

The header and optional header data are followed by pairs of codebook update and frame data sections. The codebook update sections may contain as little as a single integer specifying that no codebook updates happen before the next frame. Alternatively any number of section in the codebook may be updated. The format of the codebook update sections is as follows:

```
        ENTRIES_TO_UPDATE
                If ENTRIES_TO_UPDATE != 0
        FIRST_ENTRY_NUM
        CODEBOOK DATA

        ENTRIES_TO_UPDATE
                If ENTRIES_TO_UPDATE != 0
        FIRST_ENTRY_NUM
        CODEBOOK DATA

                ...
        ENTRIES_TO_UPDATE = 0
```

ENTRIES_TO_UPDATE and FIRST_ENTRY_NUM are both 32-bit integers (in net byte

order). ENTRIES_TO_UPDATE indicates how many entries in the codebook are to be updated. It must not be negative or exceed the codebook_entries (768 for InfoPad) value specified in the VQ file header. A 0 value indicates that the codebook update section is done and the frame data will follow.

Note that any program that reads VQ files (such as vq_play(1) and send_vq(1)) should check to see if ENTRIES_TO_UPDATE is the magic number #defined as VQ_MAGIC_NUM (0x49505651 = `IPVQ') expected for the magic_number field in the header. If this is the case, it should assume that two (or more) VQ files have been concatinated together. It should parse the new header, possibly making sure that it is consistent with the old one or making many necessary modifications (like resizing the play window), and procede. In this way, larger VQ files can be created by simple concatination of a number of smaller clips.

FIRST_ENTRY_NUM specifies the first codebook entry to be updated (in this block) and should be between 0 and codebook_entries - ENTRIES_TO_UPDATE to specify a valid region. For the InfoPad hardware, entries 0 through 255 specify the Y codebook, entries 256 through 511 specify the I codebook, and entries 512 through 767 specify the Q codebook.

CODEBOOK DATA is codebook_entry_size (16 for InfoPad) * ENTRIES_TO_UPDATE bytes of codebook data. For InfoPad, the codebook data is organized in 16 byte blocks in A to P order in the file:

```
A B C D
E F G H
I J K L
M N O P
```

The Y Codebook values are 0..255 unsigned binary - 0=black, 255=white. The I and Q Codebook values are 0..255 offset binary: 128 means 0, 127 is -1, 129 is 1 etc. Note that this is slightly different from the InfoPad hardware: it uses 6 bit sign-magnitude format aligned to 6 bits. The send_vq(1) program will, however, correct the format in order to keep the VQ file format as simple and flexible as possible.

FRAME DATA

The frame data consists of the new frame of Y data followed by the new frame of IQ data. (The I and Q data is interleaved due to hardware design.) The Y frame data consists of width * height (from header) * 4 bytes of Y frame data (one byte per 4x4 block) present in normal scan order - top left to top right down to bottom right. The IQ frame data consists of width * height (from header) * 2 bytes of IQ frame data. The reason that this is half as many bytes as the Y frame data is that the I and Q are subsambled by 2 in each direction (for a reduction by 4 in data) but both the I and Q are present, making the reduction only by a factor of 2. The I and Q data are interleaved in foursomes of 2 apiece - i.e:

```
I  I  Q  Q      I  I  Q  Q      I  I  Q  Q   ...
0  1  0  1      2  3  2  3      4  5  4  5   ...
```

where the 0,1,2,3,... again correspond to the normal scan order.

SEE ALSO

vq_play(1) send_vq(1) mpeg2vq(1) vq2codebook(1) codebook2ras(1) showcodebook(1) showvqcodebook(1) raw_video(5)

AUTHOR

Jeff Gilbert <gilbertj@eecs.berkeley.edu>

APPENDIX B *The WebChip Applet*

This appendix describes the details of the WebChip applet, used to illustrate concepts in application-level link management described in Chapter 11.

B.1. Motivation

The web has proven to be an effective medium for exchanging information and documentation. It has been an invaluable aid in enabling researchers and industry to share information. Its use in Electronic Design Automation (EDA) is also expanding rapidly.

However, currently it does not provide an adequate solution to the problem of displaying integrated circuit layout in a manner that is readily accessible to a large number of people with varying compute and network resources. Integrated circuit designers wishing to document layout on the web are limited to placing static GIF images of their design in the pages. Alternatively they can leave a link to the actual layout files for others to download and run with their particular layout editor. This is, of course, contingent upon the user having access to a compatible viewer or editor. Even if so, it can be a time consuming process at best.

With the WebChip Interactive Java VLSI Layout Viewer, the designer can now embed the actual layout in the web pages and remote users are able to actively navigate the layout by zooming, panning, and expanding and unexpanding subcells. WebChip accepts standard Magic layout files and does rapid rendering of hierarchical designs including the sub-cell transformations, arrays, label justification, and layer display. WebChip is tailored to the task of remote access and customizable to work with any fabrication process.

Interactive viewing of large layouts over the web brings up a number of interesting challenges which WebChip addresses. One set of challenges relates to obtaining the greatest viewing speed and lowest latency possible. One factor which impedes this is the bandwidth limitation of the link connecting the user to the remote web server. Often this is a slow modem or wireless link which would not seem congenial to large layout databases. Another problem is that even when the local host receives the data, it has to be able to render it quickly to obtain interactive navigation. However, the local machine may not have sufficient compute power, or in case of the web, the local machine may be using interpreted Java which further hampers performance. In order to address these issues, a display optimization called cell image caching was developed, which greatly accelerates the display of hierarchical layouts. In many cases, WebChip, written in Java, can display fully expanded layout faster than Magic's or Cadence's layout editor! The technique of cell image caching, however, could be applied to existing commercial tools as well to yield even faster display. In order to address the issue of loading speed, compression and link scheduling are used to deliver the layout data to the local host.

No matter how well data is transferred and the display is optimized, it will inevitably still be too slow in some circumstances. However, this problem can be masked by allowing the user to deal with "work in progress". They should be able to effectively navigate all portions of the layout that they have received even during the loading process. Additionally, to mitigate slow redrawing,

the user must be able to interact with the system even while it is still busy displaying something else[1].

B.2. Operation Tutorial

Although no Magic code is used in WebChip, the GUI was modeled in some ways after Magic, as Magic is the best layout editor! The WebChip viewer uses Magic files for input. It can used gzipped magic files to reduce load time. (Described in Section 11.2.2.1..) It could be modified to also accept CIF or other formats but Magic delivered all of the necessary functionality. Many of the issues in interactive web-based operation, of course, do not have a parallel in the standard Magic implementation.

B.2.1. New Window / Close Window

To start our tutorial, open another view of the sample layout. This can be done in a multitude of way: Either click on the New Window button on the applet at the top of this page, type an "o" anywhere in the same applet or type an "o" in the following copy of the same applet:

The window can be resized and the layout will grow or shrink to keep the same amount in view while maintaining a 1:1 aspect ratio. Windows can be closed either by using the standard window manager hooks or else by clicking on the Close Window button or typing "O" in them.

B.2.2. Showing / Hiding Control Buttons and Labels

If you opened a new window from the applet directly, you will notice that the control buttons are not present. Typing space in the applet toggles the display of the control buttons and labels. (This works both on the embedded version as well as stand-alone copies.)

1. I believe that neither Magic nor Cadence do this.

B.2.3. The Selection Box

To effect many of the WebChip commands, a region has to be selected. This is done by dragging the mouse with the left mouse button pressed. A white selection rubber band box will appear. Clicking in the window will make it go away. (This differs slightly from Magic as Magic does not allow a drag but rather clicks to set box corners.)

B.2.4. Expand / Unexpand

The layout viewer is a hierarchical viewer. It only shows the layout and subcells in a given cell if they are "expanded." Initially all subcells are "unexpanded" so that a box with this top cell and instance name appears, indicating the layout bounds. Drag a box in the layout window that crosses some unexpanded subcell. Now press "x" or click on the Expand button. The cells under it are expanded. If you drag a box around the whole layout and press "x", all subcells are expanded. (This can be specified as the default by setting the applet parameter defaultExpand to anything.)

To unexpand a region, drag the box so it crosses or contains the cell to be unexpanded and press "X" or click on the Unexpand button. (The expand / unexpand semantics are exactly as in Magic.)

B.2.5. Zooming and Panning

Zooming into a specified area can be performed by dragging a box around the area and typing "z" or clicking on Zoom Area. To magnify the current region of interest, type "f" or click on Zoom In. Similarly, to zoom out, the "Z" key or Zoom Out button can be used. Zoom Full or the "F" key sets the zoom to see the full layout. Lastly, the arrow keys can be used to pan left, right, up and down. If the shift key is used, it moves in bigger jumps while the control key causes movement in smaller jumps.

B.2.6. Redisplay

Finally, control-L can be used to refresh the screen if ever necessary (or just to investigate the drawing speed)

B.2.7. Status Panel

Below the buttons is displayed the name of the top cell in the design followed by a dash and the location of the design. In our case, the name of the design is http://Infopad.EECS.Berkeley.EDU/~gilbertj/Coursework/ee244/project/cor_at4 and the top cell is correlat. (This is a dual 64x64 image correlator that I built as part of a PC-based real-time face recognition system for my undergraduate honors thesis. See the thesis [34] or conference paper [33] if you are interested!)

B.2.8. Design File Loading Status Indicator

As the files are loading, the status is shown below the design name. Note that it loads up to 3 design cell files concurrently.

Loading multiple cells concurrently can help amortize TCP connection times as is commonly done by web browsers. Using a single managed stream would have all of the benefits that it has for the web as outlined in Chapter 10. Also note that as the design is loading, the cells that are visible (i.e. all of their ancestors have been expanded) will be displayed in gray until they are loaded. Once they are loaded, the outlines turn yellow if they are unexpanded or the layout is shown if they are expanded. Meanwhile, feel free to navigate the design as it is loading!

All design cell information is managed centrally such that if multiple instances of the WebChip applet reference the same design, they will share the same database. Thus the multiple instances do not use more memory or incur the load-time penalty multiple times. By specifying a different initial zoom region to different WebChip applet instances, multiple views of the same layout can be included at low computational cost and without additional bandwidth utilization.

B.2.9. Display Mode Choice Button

Lastly, the mode choice button which now indicates “Cached Block Mode” allows the user to change the display method being used. This is described in Section 11.2.1.

B.3. Configuration

Several applet parameters control WebChip. The file parameter is used to specify the top-level URL of the file to load. This is also the design name. (The design name without the top-level cell name is shown in the control panel.) The URL can specify a .mag magic file or a .mag.gz gzipped magic file or not have any extension as described in Section 11.2.2.1.

The style parameter is used to specify a URL for the display style. See below for more on style files.

The noControlButtons and noControlLabels options are used to specify whether the control buttons and labels should initially appear. The space bar will always toggle this. It is useful when embedding many layouts or small ones to not have the control buttons and labels as they consume space.

The defaultExpand controls the initial state of subcells. If it is not set to anything then sub-cells are initialized as unexpanded until explicitly expanded. If it is set to anything then they are expanded by default. This allows the initial view to appear similar to a die photo.

B.3.1. Style Files

The style files control all display aspects of WebChip, including layer colors, stipple patterns, crossing, and outlining. The style files also allow background color, sub-cell color, and unloaded cell colors to be specified. The files are standard ASCII and can be customized to viewer

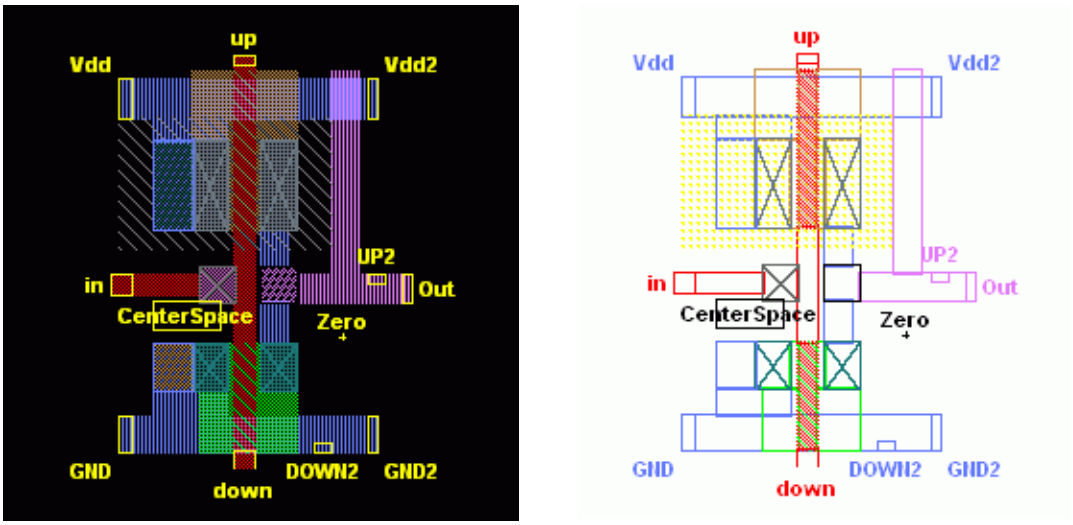


FIGURE B.1. Effects of style files on layout presentation.
The two figures are of the same layout but different style files.

preference as well as different technologies. Figure B.1 shows an example of two different style files used on the same design.

The style file is similar in format to Magic technology files. The file must start with the literal LayoutStyleFile. This is followed by two numbers specifying the stipple height and width which should be 2, 4, or 8. This is followed by a set of color and layer definitions. WebChip draws layers as some combination of a stipple pattern, an optional solid outline, and an optional solid cross. The layer stippling is specified in terms of a swath the size of the stipple. Each pixel in the swath can be some opaque color or transparent. While the style file could be specified in terms of RGB or transparent for each pixel in each stipple, an extra level of indirection was introduced to aid in style file design. The indirection is accomplished by having a set of user-named colors and then specifying the stipple and layer information in terms of the named colors. The file is similar to a magic tech file in that sections are demarcated with << section >>. Here section can be “layer LayerName” or << colors>> corresponding to layer and color definitions. C++ style // comments can be included (thanks to the Java tokenizer) and sections and colors can be redefined.

```
LayoutStyleFile
8 8     // Stipple size

<< colors >>
trans       transparent
background  0   0   0
label       255 255 0
subcell     255 255 0
missingCell 128 128 128

red         255 0   0
green       0   255 0
blue        0   0   255
purple      255 0   255
black       0   0   0
white       255 255 255

polyRed     255 0   0
m1Blue      60  101 185
m1Blue      100 120 255
m2Purple    125 88  157
m2Purple    255 128 255
m2Purple    230 120 255
ndiffGreen  0   255 0
pdiffBrown  200 150 75
ndcDrkGrn   30  110 110
ndcCross    30  120 120
pdcGray     100 110 120
pdcCross    110 120 130
nwellGray   128 128 128
pcPurp      200 100 200
pcOut       100 100 100
via1        175 124 255
via1        230 120 255
via2        0   0   0
nsc1        30  110 110
nsc2        100 120 255
psc1        200 150 75
psc2        100 120 255

<< layer metal1 >>
mono_stipple    m1Blue
    0 0 0 0 0 0 0 0
    1 0 1 0 1 0 1 0
    0 0 0 0 0 0 0 0
    1 0 1 0 1 0 1 0
    0 0 0 0 0 0 0 0
    1 0 1 0 1 0 1 0
    0 0 0 0 0 0 0 0
    1 0 1 0 1 0 1 0

mono_stipple    m1Blue
    1 0 1 0 1 0 1 0
    1 0 1 0 1 0 1 0
    1 0 1 0 1 0 1 0
    1 0 1 0 1 0 1 0
    1 0 1 0 1 0 1 0
    1 0 1 0 1 0 1 0
    1 0 1 0 1 0 1 0
    1 0 1 0 1 0 1 0

<< layer metal2 >>
mono_stipple    m2Purple
    0 1 0 1 0 1 0 1
    0 0 0 0 0 0 0 0
    0 1 0 1 0 1 0 1
    0 0 0 0 0 0 0 0
    0 1 0 1 0 1 0 1
    0 0 0 0 0 0 0 0
    0 1 0 1 0 1 0 1
    0 0 0 0 0 0 0 0

mono_stipple    m2Purple
    0 1 0 1 0 1 0 1
    0 1 0 1 0 1 0 1
    0 1 0 1 0 1 0 1
    0 1 0 1 0 1 0 1
    0 1 0 1 0 1 0 1
    0 1 0 1 0 1 0 1
    0 1 0 1 0 1 0 1
    0 1 0 1 0 1 0 1

<< layer m2contact >>
outline via2
color_stipple
  trans via1  trans via1  trans via1  trans via1
  via1  trans via1  trans via1  trans via1  trans
  trans via1  trans via2  trans via1  trans via2
  via1  trans via2  trans via1  trans via2  trans
  trans via1  trans via1  trans via1  trans via1
  via1  trans via1  trans via1  trans via1  trans
  trans via2  trans via1  trans via2  trans via1
  via2  trans via1  trans via2  trans via1  trans

<< layer polysilicon >>
mono_stipple polyRed
    1 0 1 0 1 0 1 0
    0 1 0 1 0 1 0 1
    1 0 1 0 1 0 1 0
    0 1 0 1 0 1 0 1
    1 0 1 0 1 0 1 0
    0 1 0 1 0 1 0 1
    1 0 1 0 1 0 1 0
    0 1 0 1 0 1 0 1

<< layer polycontact >>
cross pcOut
mono_stipple pcPurp
    1 0 1 0 1 0 1 0
    0 1 0 1 0 1 0 1
    1 0 1 0 1 0 1 0
    0 1 0 1 0 1 0 1
    1 0 1 0 1 0 1 0
    0 1 0 1 0 1 0 1
    1 0 1 0 1 0 1 0
    0 1 0 1 0 1 0 1

<< layer ndiffusion >>
mono_stipple ndiffGreen
    1 0 1 0 1 0 1 0
    0 1 0 1 0 1 0 1
    1 0 1 0 1 0 1 0
    0 1 0 1 0 1 0 1
    1 0 1 0 1 0 1 0
    0 1 0 1 0 1 0 1
    1 0 1 0 1 0 1 0
    0 1 0 1 0 1 0 1

<< layer pdiffusion >>
mono_stipple pdiffBrown
    1 0 1 0 1 0 1 0
    0 1 0 1 0 1 0 1
    1 0 1 0 1 0 1 0
    0 1 0 1 0 1 0 1
    1 0 1 0 1 0 1 0
    0 1 0 1 0 1 0 1
    1 0 1 0 1 0 1 0
    0 1 0 1 0 1 0 1

<< layer nwell >>
mono_stipple nwellGray
    1 0 0 0 0 0 0 0
    0 1 0 0 0 0 0 0
    0 0 1 0 0 0 0 0
    0 0 0 1 0 0 0 0
    0 0 0 0 1 0 0 0
    0 0 0 0 0 1 0 0
    0 0 0 0 0 0 1 0
    0 0 0 0 0 0 0 1

<< layer ndcontact >>
cross ndcCross
mono_stipple ndcDrkGrn
    1 0 1 0 1 0 1 0
    1 1 1 1 1 1 1 1
    1 0 1 0 1 0 1 0
    1 1 1 1 1 1 1 1
    1 0 1 0 1 0 1 0
    1 1 1 1 1 1 1 1
    1 0 1 0 1 0 1 0
    1 1 1 1 1 1 1 1
```

FIGURE B.2. Style file used to produce image in Figure 11.1.
Continued on next page.

```
<< layer pdcontact >>
outline pdcCross
cross pdcCross
mono_stipple pdcGray
    1 0 1 0 1 0 1 0
    1 1 1 1 1 1 1 1
    1 0 1 0 1 0 1 0
    1 1 1 1 1 1 1 1
    1 0 1 0 1 0 1 0
    1 1 1 1 1 1 1 1
    1 0 1 0 1 0 1 0
    1 1 1 1 1 1 1 1

<< layer ntransistor >>
color_stipple
      ndiffGreen trans      polyRed    trans      polyRed    trans      polyRed    trans
      trans      ndiffGreen trans      polyRed    trans      polyRed    trans      polyRed
      polyRed    trans      ndiffGreen trans      polyRed    trans      polyRed    trans
      trans      polyRed    trans      ndiffGreen trans      polyRed    trans      polyRed
      polyRed    trans      polyRed    trans      ndiffGreen trans      polyRed    trans
      trans      polyRed    trans      polyRed    trans      ndiffGreen trans      polyRed
      polyRed    trans      polyRed    trans      polyRed    trans      ndiffGreen trans
      trans      polyRed    trans      polyRed    trans      polyRed    trans      ndiffGreen


<< layer ptransistor >>
color_stipple
      pdiffBrown trans      polyRed    trans      polyRed    trans      polyRed    trans
      trans      pdiffBrown trans      polyRed    trans      polyRed    trans      polyRed
      polyRed    trans      pdiffBrown trans      polyRed    trans      polyRed    trans
      trans      polyRed    trans      pdiffBrown trans      polyRed    trans      polyRed
      polyRed    trans      polyRed    trans      pdiffBrown trans      polyRed    trans
      trans      polyRed    trans      polyRed    trans      pdiffBrown trans      polyRed
      polyRed    trans      polyRed    trans      polyRed    trans      pdiffBrown trans
      trans      polyRed    trans      polyRed    trans      polyRed    trans      pdiffBrown

<< layer nsubstratencontact >>
outline nsc2
color_stipple
    nsc1   trans  nsc1   trans  nsc1   trans  nsc1   trans
    trans  nsc1   trans  nsc2   trans  nsc1   trans  nsc2
    nsc1   trans  nsc2   trans  nsc1   trans  nsc2   trans
    trans  nsc1   trans  nsc1   trans  nsc1   trans  nsc1
    nsc1   trans  nsc1   trans  nsc1   trans  nsc1   trans
    trans  nsc2   trans  nsc1   trans  nsc2   trans  nsc1
    nsc2   trans  nsc1   trans  nsc2   trans  nsc1   trans
    trans  nsc1   trans  nsc1   trans  nsc1   trans  nsc1


<< layer psubstratepcontact >>
outline psc2
color_stipple
    psc1   trans  psc1   trans  psc1   trans  psc1   trans
    trans  psc1   trans  psc2   trans  psc1   trans  psc2
    psc1   trans  psc2   trans  psc1   trans  psc2   trans
    trans  psc1   trans  psc1   trans  psc1   trans  psc1
    psc1   trans  psc1   trans  psc1   trans  psc1   trans
    trans  psc2   trans  psc1   trans  psc2   trans  psc1
    psc2   trans  psc1   trans  psc2   trans  psc1   trans
    trans  psc1   trans  psc1   trans  psc1   trans  psc1

<< layer error_p >>
mono_stipple white
    0 0 0 0 0 0 0 0
    0 1 1 0 0 1 1 0
    0 1 1 0 0 1 1 0
    0 0 0 0 0 0 0 0
    0 0 0 0 0 0 0 0
    0 1 1 0 0 1 1 0
    0 1 1 0 0 1 1 0
    0 0 0 0 0 0 0 0
```

FIGURE B.2. Style file used to produce image in Figure 11.1. Continued from previous page.

The colors can be specified in terms of red, green, and blue triples or the literal transparent. For example:

```
<< colors >>
green       0        255        0
trans       transparent
polyColor   255      0          0
```

would define the color green as expected, define a color named trans to be transparent and define polyColor to be red.

The colors are used in subsequent layer sections as well as to control certain global defaults and colors. The background color is used to set the background color of the layout. (It is important that this is in the style file since the background has to go with the other colors.) The label color is used as a default color for labels. This can be overridden on a per-layer basis as well as shown below. The subcell color is used to determine what color to draw unexpanded subcell boxes and labels in. Finally, the missingCell color is used to determine in which color to draw cells that could not be found or have not been loaded yet. Note that all of these colors can be an RGB triple or the literal transparent. For instance, to not show missing cells, the following line should be placed in a colors section:

```
missingCell      transparent
```

Once the colors are defined, the layers can be defined. Each layer has 4 attributes: label color, outline color, cross color, and fill stipple. The first three are defined by single lines specifying the literal and a predefined color while the fill stipple can be defined in one of three ways. If no stipple information is included then it is transparent. Otherwise it can be defined as a color_stipple in which each pixel in the stipple is defined as a color (which can be transparent.) Lastly, mono_stipple is an abbreviation used for the common case where a stipple is just one color

and transparent. Here the color is specified followed by a series of 1's and 0's of the size of the stipple.

The following illustrates part of a typical style file:

```
LayoutStyleFile
4  4              // This is the stipple size

<< colors >>
trans          transparent
red            255      0      0
green            0     13     34
white          255    255    255
pdcGray        100    110    120
pdcCross       110    120    130
brown          200    150     75


<< layer ptransistor >>
outline             green
color_stipple       brown   trans  red     trans
                    trans   brown  trans   red
                    red     trans  trans   brown
                    red     trans  trans   brown

<< layer pdcontact >>
outline pdcCross
cross pdcCross
mono_stipple pdcGray  1 0 1 0
                      1 1 1 1
                      1 0 1 0
                      1 1 1 1
```

If WebChip encounters layout data on a layer not included in the style file, a warning is printed and layout on that layer is ignored.

The order of the layers in the style file determines the order that they are drawn and thus the order of precedence. The stipples should be carefully arranged such that all likely combinations of layer overlaps produces viable results. For instance if metal1 and metal2 have the same stipple patterns but different colors, when they overlapped, whichever one was drawn last would be hidden. Thus the stipple patterns should be at least partially offset.

APPENDIX C *The SpeedSurfer Application*

This appendix describes the operation of the SpeedSurfer client-side proxy application and the format used to communicate with the SurfServ server-side proxy. These are both presented in Chapter 12.

C.1. SpeedSurfer Operation

SpeedSurfer is a Windows NT-based client-side proxy as described in Section 12.2., written using Microsoft's Visual C++ development environment and Microsoft Foundation Classes (MFC). SpeedSurfer is a standard user-mode application and does not require special installation. The only step needed to browse through the SpeedSurfer is to set the web browser's Proxy setting to localhost port 2000. (The port 2000 is user-selectable as shown in SectionC.1.4.).

SpeedSurfer manages the GUI as well as the client proxy interface. It is readily configurable and can automatically initiate contact with the server-side proxy. SpeedSurfer allows for up to 5 user-configurable tunnels in addition to the web proxy. It allows web proxy chaining whereby the connections are forwarded to another proxy rather than going to the web server specified by the web address. SpeedSurfer also analyzes the traffic rates delivered to the web browser facilitating

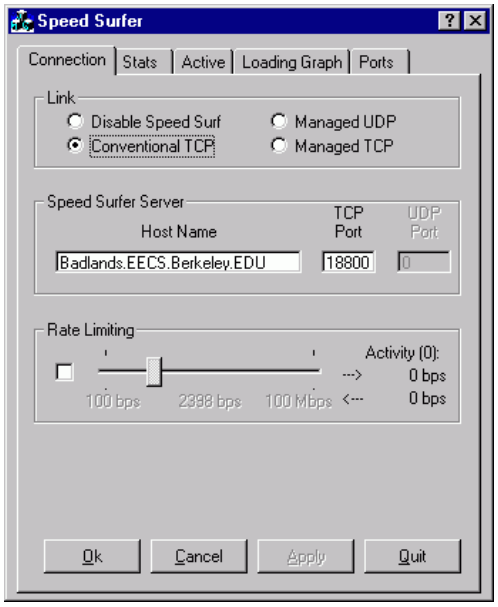


FIGURE C.1. SpeedSurfer connection page

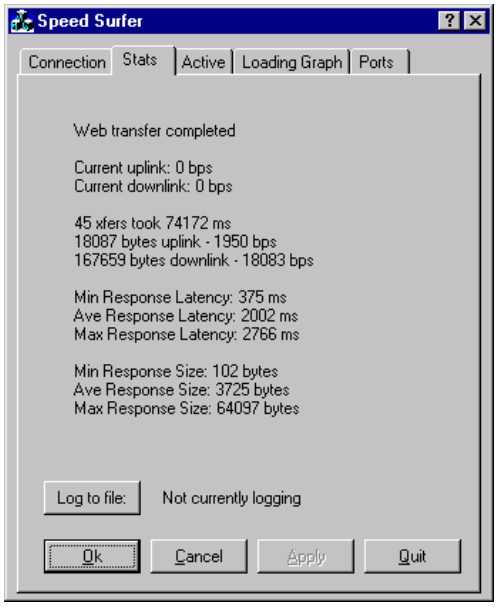


FIGURE C.2. SpeedSurfer stats page

evaluation. It can be run in "enabled" mode which implements the full proxy-proxy system or "disabled" mode where the SpeedSurfer routes HTTP traffic directly through it, maintaining multiple transient conventional TCP connections for analysis purposes only.

C.1.1. Connection Page

The connection page (see FigureC.1) is used to control the SpeedSurfer's basic mode of operation. At the top the user can select whether to disable the SpeedSurfer (but still enable statistics gathering), or if it is enabled what mode to run the proxy-proxy link in. Currently only the two TCP options are implemented. The managed / conventional TCP refers to whether the additional flow control is used. The user can also specify where the SurferServ can be found. When the mode is changed from Disable SpeedSurfer to one of the other modes and the Apply button is pressed, the SpeedSurfer initiates contact with the SurferServ.

Rate limiting may be implemented at a later time. The number of active links and current uplink and downlink rates are displayed.

C.1.2. Stats Page

The stats page, shown in FigureC.2 displays network traffic statistics in real-time. It displays the current uplink and downlink traffic rates and number of active links as well as information about the current web page load. This information includes the duration of the load, the number of transfers (data items) thus far encountered, the number of uplink and downlink bytes transferred, and average uplink and downlink traffic rates over the load. Lastly, it displays the minimum, average, and maximum response latency and size. The response latency is defined as the duration between the web browser delivering a request and getting the first byte of response data. A new web page download is detected whenever there are no connections through the web proxy port for more than 1 second.

The “Log to File” button can be used to send collected data into a log file for later post processing. The graphs in Chapter 10 are generated off-line using such a log file. The logs contain time-stamped itemization of when connections are established, what items are requested, detailed timing of data transfer, as well as when the connections are closed. A sample log file is shown in Figure C.3.

C.1.3. Loading Graph Page

The loading graph page, shown in Figure C.4., shows real-time presentations of the loading graphs used to quantify web page loading time in Section 10.4. Three views of the loading graph are supported - “TOTAL BYTES”, “TOTAL RATE”, and “LOADING GRAPH” as depicted in the figure.

```
0 startAt 0
0 request GET /~gilbertj/ HTTP/1.1
0 host badlands.eecs.berkeley.edu:8090
0 recv 182 at 841
0 recv 2920 at 2043
1 startAt 2083
2 startAt 2103
0 recv 4096 at 2994
0 recv 1176 at 3214
1 request GET /~gilbertj/me_harry_jim_small.jpg HTTP/1.1
1 host badlands.eecs.berkeley.edu:8090
2 request GET /~gilbertj/eegsa_small.gif HTTP/1.1
2 host badlands.eecs.berkeley.edu:8090
1 recv 184 at 3655
0 recv 1257 at 3965
0 doneAt 3965
0 total 9631
GET /~gilbertj/ HTTP/1.1 size 9631
2 recv 181 at 3985
1 recv 1460 at 5047
1 recv 1460 at 5127
2 recv 542 at 5367
2 doneAt 5367
2 total 723
GET /~gilbertj/eegsa_small.gif HTTP/1.1 size 723
1 recv 1176 at 6249
1 recv 1460 at 6749
1 recv 1460 at 7250
1 recv 1176 at 7260
1 recv 1460 at 8252
1 recv 1460 at 8752
1 recv 1176 at 9253
1 recv 1864 at 9443
1 doneAt 9483
1 total 14336
GET /~gilbertj/me_harry_jim_small.jpg HTTP/1.1 size 14336

0 startAt 0
0 request GET /~gilbertj/cnn2/ HTTP/1.1
0 host badlands.eecs.berkeley.edu:8090
0 recv 183 at 851
0 recv 2920 at 1983
1 startAt 2003
1 request GET /~gilbertj/cnn2/cnn.js HTTP/1.1
1 host badlands.eecs.berkeley.edu:8090
0 recv 4096 at 2684
0 recv 1176 at 2864
0 recv 4096 at 3595
1 recv 183 at 3675
```

FIGURE C.3. Example SpeedSurfer log file

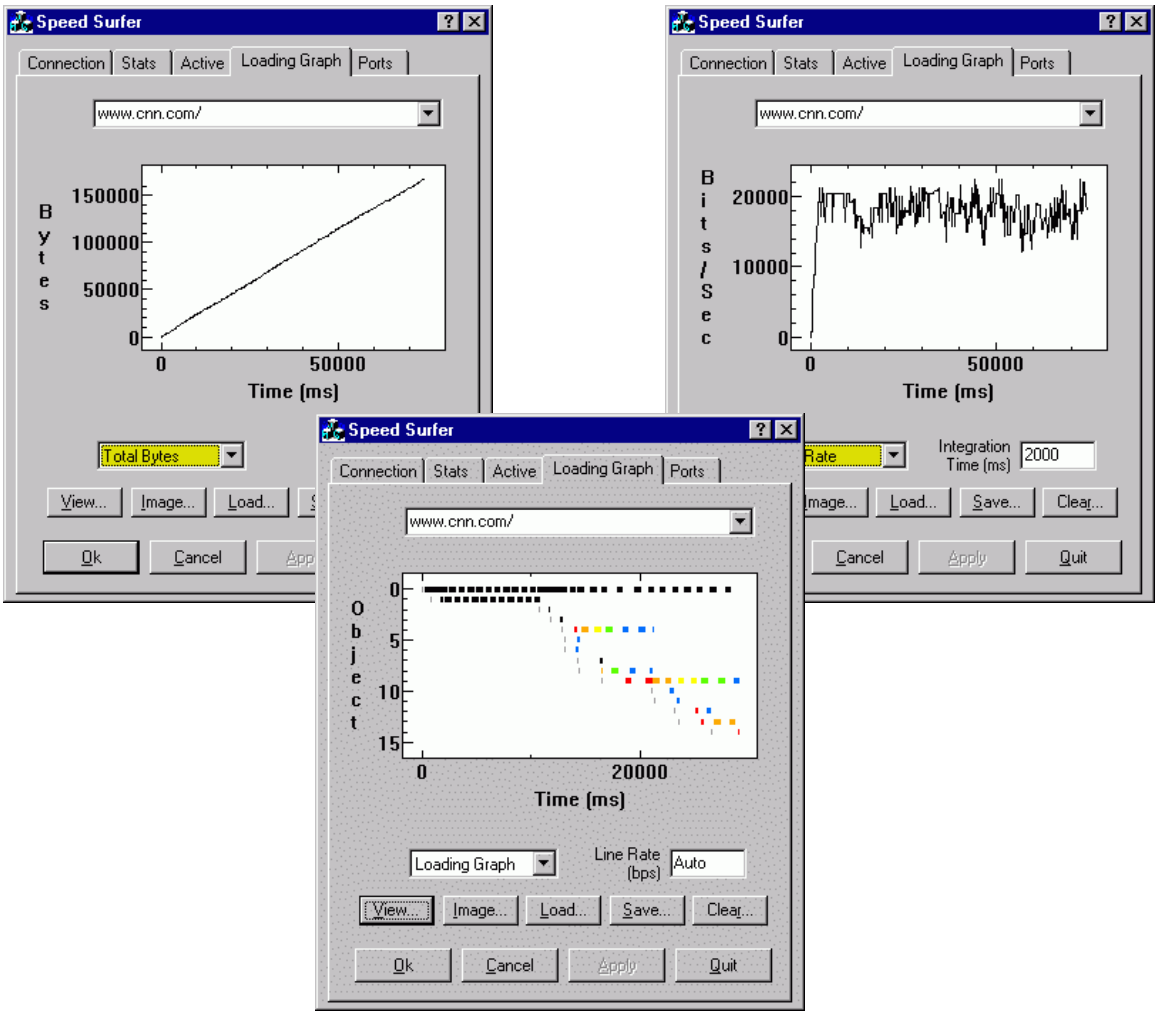


FIGURE C.4. SpeedSurfer loading graph page.
Three views shown: total bytes (top left), total rate (top right) and loading graph (bottom center)

Each figure has time across the X axis. The three types of graphs can be zoomed using the mouse to view particular regions in greater detail.

The TOTAL BYTES option graphs the total number of bytes of web traffic received since the beginning of the web transfer. This is, by definition, is a monotonic non-decreasing function. The instantaneous rate of data delivery is the slope of the line. Thus flat portions indicate stalls.

The TOTAL RATE option graphs the short-term averaged rate of data transfer across all connections. Thus it is the derivative of the TOTAL BYTES graph. The size of the sampling window is

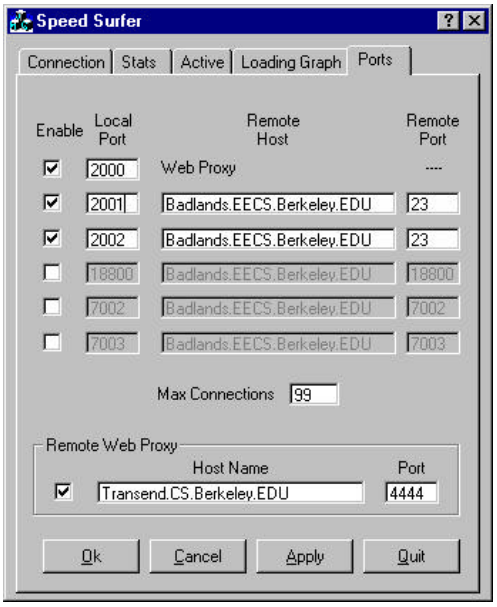


FIGURE C.5. SpeedSurfer ports page

controlled by the INTEGRATION TIME (MS) field. This allows the trade-off of graph detail and amount of sampling noise to be adjusted.

The LOADING GRAPH option shows the web page loading graph of Section 10.4. in real time. The LINE RATE (BPS) field is used to input the line rate to adjust the length of the bars in the graph. This is required since the bars represent the amount of time a transfer corresponds to but in reality all that is known is the number of bytes that are transferred. The line rate is the constant for this conversion.

C.1.4. Ports Page

The ports page, shown in FigureC.5, allows the user to select incoming and outgoing ports. In the example, if a connection is made to localhost:2001 then a connection will be made to Badlands.EECS.Berkeley.EDU:23. If the mode set in the connection page is not “Disabled” then this connection will be set up through the proxy-proxy link and the connection will originate from the

server proxy. Thus, to hold a telnet session to Badlands through the proxy mechanism, the user would just run telnet, setting the host to localhost and port to 2001. The maximum number of simultaneous connections can also be set. If this many links are already in progress then no new ones will be accepted. Finally, a remote web proxy can be specified to allow the web proxy chaining.

C.2. Proxy-Proxy Link Protocol

As previously mentioned, the client proxy and server proxy communicate via a custom protocol that allows embedding of multiple connections in the link, as well as maintaining flow control. Additionally, an echo packet facility allows the client proxy to determine the round-trip time as well as the number of bytes in transit by measuring the differences between packet transmission and reception times. The protocol used between the two proxies will be briefly described here for the purpose of better illustrating their operation.

The protocol is design to push as much of the complexity as possible into the client proxy to keep the server proxy scalable since multiple simultaneous sessions may be running on a given server. Additionally, this eases design since all GUI interaction occurs in the client proxy. The CPU burden presented by both proxies is minimal though the addition of compression would increase the necessary computation.

The proxy-proxy link is a single TCP/IP connection with the stream consisting of control and data packets of a form understood by the two proxies. The packet formats are shown in TableC.1. The server proxy is a general purpose multiplexor / demultiplexor that does not understand HTML or HTTP at all. All data parsing is performed by the client proxy. Currently link-

Field	Size	Meaning
Packet header present at beginning of all packets		
SYNC	2 bytes	Must be 0x24CE. Used to assure synchronization
MSG	2 bytes	Control packet type or data link ID
LEN	2 bytes	Length of remaining bytes in message

ClientInit message is sent from client->server on connection. Could contain setup info.		
HEADER	6 bytes	See above. Msg=CLIENT_INIT, Len=0

ServerInit message is sent from server->client in response to ClientInit. Could have more info.		
HEADER	6 bytes	See above. Msg=SERVER_INIT, Len=0

NewLink message sent from client->server to establish new link.		
HEADER	6 bytes	See above. Msg=NEW_LINK, Len=6 + hostNameLen
ID	2 bytes	ID for new connection.
PORTNUM	2 bytes	TCP Port Number of new connection
PRIORITY	1 byte	Priority - not currently used
WINDOWSIZE256	1 byte	Used for link flow control - not currently used
HOSTNAME	N bytes	Name of host to connect to

CloseLink message sent in either direction to signal link closure		
HEADER	6 bytes	See above. Msg=CLOSE_LINK, Len=2 + errorStrLen
ID	2 bytes	ID of link to close
ERRORSTR	N bytes	For server->client gives reason for closure. Empty for no error.

CloseAck message sent server->client to allow ID reuse		
HEADER	6 bytes	See above. Msg=CLOSE_ACK, Len=2
ID	2 bytes	ID of link closed

EchoMsg message echoed by server and used by client for diagnostics - like ping packet.		
HEADER	6 bytes	See above. Msg=ECHO_MSG, Len=N
DATA	N bytes	Generic data. Client uses timestamp and downlinkBytesRcvd.

Window message used to impose additional flow control (currently downlink only)		
HEADER	6 bytes	See above. Msg=ECHO_MSG, Len=4
NEWTOTAL	4 bytes	Allowable total bytes sent this session. -1 for no flow control

TABLE C.1. Proxy-proxy packet protocol

level flow control is not used but is provisioned. This is not a problem unless applications on either end of the link are slower than the link, which is typically not the case.