Networking Requirements and Solutions for a TV WWW Browser

Theodoros P. David

Thesis submitted to the Faculty of the Virginia Polytechnic Institute and State University in partial fulfillment of the requirements for the degree of

Master of Science
in
Electrical Engineering

Scott F. Midkiff, Chair
Nathaniel J. Davis, IV
Willard W. Farley, Jr.

September 17, 1997
Blacksburg, Virginia

Keywords: WWW, TV Browser, Asymmetric Networks, Fast Ethernet, Distribution Algorithms.

Networking Requirements and Solutions for a TV WWW Browser

Theodoros P. David

(ABSTRACT)

Most people cannot access the World Wide Web (WWW) and other Internet services because access requires a complex and expensive computer. Moreover, the bandwidth offered to the general public is mostly limited by today's analog modems through standard telephone lines. An inexpensive, easy-to-use Internet/WWW access service, able to handle bandwidth-intensive, multimedia-oriented WWW pages, is currently not available to the general public.

Some concrete proposals, and even completed products, have been provided by the computer and communications industry. Nonetheless, these solutions are either still too expensive and complex or have limited bandwidth. One service that introduces a simple and inexpensive way to access the Internet, while providing high bandwidth, is the IVDS/WWW Browser. Controlled by a central WWW Browser Server, the WWW pages are displayed on a standard television set using an IVDS (Interactive Video and Data Service) Decoder Box, an inexpensive hardware device. The user can access the WWW using a remote control device. This thesis presents the networking requirements and solutions for the IVDS/WWW Browser Server and the IVDS Decoder Box.

ACKNOWLEDGMENTS

I would like to thank Dr. Scott F. Midkiff, my advisor, for his continued advice, proofreading, and direction throughout the research period. I would also like to thank the other members of my thesis committee, Dr. Nat Davis and Woody Farley, as well as the other members of the IVDS/WWW Browser research team, Aaron Hawes, John Stanhope and John Deighan for their valuable support, ideas, and research hints.

The project was funded by Mr. Fernando Morales of Interactive Response Services, Inc. and the Virginia Center for Innovative Technology through the Center for Wireless Telecommunications, whose support is greatly appreciated.

Finally, I would like to thank my parents, Papasotirios and Artemis, for their continued support and encouragement.

TABLE OF CONTENTS

LIST OF FIGURES

Number *Page*

LIST OF TABLES

Number *Page*

Chapter 1 - INTRODUCTION

1.1 Overview

Despite the explosive growth of the World-Wide Web (WWW), or simply the “Web,” and the Internet in general, the percentage of the world's population that is connected to the world's largest network is not impressive. While the world's population is almost 6 billion, an estimated group of only 30 million people have accessed the Internet [Zak96][CLC96].

Due to the high cost of personal computers (PCs), coupled with the high degree of technical sophistication required to operate them, over 65% of the US population alone still does not own a PC [CLC96][Fin96]. They are, therefore, unable to gain access to the “information super-highway.” So, an inexpensive, easy-to-use Internet/WWW access service for the general public, could be attractive. Proposals have already been made by the computer and communications industry to achieve this goal. The most prominent are the Network Computer (NC) proposed by Oracle, Inc., the Network Personal Computer (NetPC) by Microsoft Corp. and Intel Corp., and the WebTV device already on the market by Philips and Magnavox [Hal97]. The first two proposals, although they are reduced versions of today’s PC, still contain enough complexity and cost to scare away the general public. The WebTV approach has managed to remove most of the complexity from the end-user, but the cost is still high. Furthermore, the available bandwidth that these devices provide for home users is usually limited by the capabilities of today’s analog modems.

The most obvious existing device to be used for a WWW browsing service is the television (TV) set. Firstly, it can be used as the monitor to display Web pages. Secondly, unlike the PC, the TV can be found in virtually all United States (US) residences [Cic95]. A project

sponsored by Interactive Response Services (IRS), Inc. and conducted by the Center for Wireless Telecommunications (CWT) at Virginia Tech has investigated the possibilities of a WWW service that makes use of the TV set. Part of this Interactive Video and Data Services (IVDS) project was to develop a technique for users to view Web pages on their TV set and control the browser using remote control devices. The idea is to remove the complexity and high cost from the end-user device so that it will eventually be as available to the masses as the telephone and television set today.

1.2 Problem Statement

The IVDS/WWW Browser service consists of two main parts: the IVDS/WWW Browser Server and the IVDS/WWW Decoder Box. The Browser Server is a software package that runs on a fast server computer, while the Decoder Box is an independent hardware device to be connected to or near the television set. My research for this project involved the networking requirements for the successful interfacing of these two parts. For the Decoder Box, I have designed the network interface and identified correct and cost-effective networking hardware. For the IVDS/WWW Browser Server, I have developed a client-server application so that the Server can efficiently support a large number of Decoder Boxes. And, finally, I have investigated several network technologies that could be used to connect the Browser Server and Decoder Box.

1.3 Contents of Thesis

In Chapter 2, I give background on the network technologies involved in the IVDS project. First, I discuss previous work on techniques to provide interactive information to the general public. Then, I give information about the Fast Ethernet (FE) protocol and the different physical media standards that have emerged for FE. In this chapter I also discuss the general

asymmetric networking concept as well as how FE can be used in asymmetric networks. Finally, I present the IVDS wireless return path which is required to control the TV WWW Browser.

Chapter 3 deals with the IVDS/WWW system in general, explaining the operation from the moment a user requests a Web page with the remote control until the page is displayed on the television set. In this chapter I also discuss the specific requirements for the Browser Server-to-Decoder Box network.

In Chapter 4, I present the detailed system design for the Browser Server-to-Decoder Box network. For the Browser Server interface, I explain the Ethernet Management and Ethernet Interface software modules that handle multiple decoder boxes as clients. The uniqueness of this client-server application is that we practically have no return path from the Decoder Box back to the Browser Server. Hence, we cannot apply any known error correction techniques, which are critical to some services. To compensate for this I have developed an "error prevention" technique which I also present in Chapter 4. My research for the Decoder Box network interface is also described. I explain the different network chips considered, the selected chips' functionality and circuit design. Finally, I give more details about the selected network technology.

In Chapter 5, I explain how I tested the network interface and I present the results obtained. From these results I make some extrapolations for the final product.

The last chapter, Chapter 6, is a summary of the work completed as well as further work required. The Browser Server's Ethernet Management and Ethernet Interface code is provided in Appendix A and complete timing results are presented in Appendix B. The circuit design for the Decoder Box is presented in Appendix C.

Chapter 2 - BACKGROUND

This chapter starts with a review of previous work on ways to provide interactive information to the general public and identifies the network media used in each case. It then provides background information about the Fast Ethernet standard which is the technology used in the interface between the IVDS/WWW Browser Server and the Decoder Boxes. The deployment of the IVDS/WWW service requires the use of existing public networks, so an overview of the asymmetric network concept is also presented. Finally, I discuss the IVDS wireless return path which is common to all IVDS services and is used to send the user's requests back to the IVDS servers.

2.1 Previous Work

The TV brought a revolution in providing information to the general public. Its passive nature, however, does not allow users to control the information they receive. Numerous interactive TV trials have tested consumer behavior. These experiments have employed different infrastructures, as well as different switching and transmission techniques, in order to meet the high bandwidth requirement of multimedia services [CMC95]. Services like pay per view, video on demand, home shopping, and home banking run on an ordinary TV equipped with a "set-top box" that performs MPEG-2 decoding [Gai96]. Another technique to provide information to a user through the TV is Teletext. Provided by most TV stations in Europe, Teletext makes use of the unused Vertical Blanking Interval (VBI) of a TV signal. The TV set receives all the data and the user selects the information he or she is interested in with the remote control. Note that the interaction is between the remote control device and the TV set only. The small bandwidth, however, limits the data to ASCII text.

After the Internet, and the WWW in particular, became popular, many companies decided to incorporate Internet capabilities into consumer electronics devices. The Gateway 2000 DestinationTV incorporates the hardware necessary for Internet access into the TV box [Day96]. It uses an analog modem to access the Internet. WebTV is a set-top box device that uses the TV set as a monitor and an analog modem to access the Internet [Day96]. Both the DestinationTV and the WebTV use either a remote control or a wireless keyboard for user interaction. The NC is a standard rather than a product. It defines the minimum hardware and software requirements to access either the Internet or local intranets [Dav97]. It can be in the form of a TV set-top box, a video-phone style device, or a scaled-down PC, complete with keyboard, monitor, and processor. Microsoft's and Intel's NetPC falls into the latter category. X-terminals, although introduced much earlier, can also be categorized as NCs. The network capabilities of current NCs range from analog modems to Ethernet cards [Hal97] [Fin96].

2.2 Fast Ethernet

Fast Ethernet technology has been selected as the network technology to be used between the IVDS/WWW Browser Server and the Decoder Boxes. Below, I discuss several high-speed technologies and explain the reasons for selecting FE over them. Then, I present the technical details of the FE standard adopted by the Institute of Electrical and Electronics Engineers (IEEE) as 802.3u.

Ethernet is the most widely used type of local area network (LAN) in use today. It was invented at the Xerox Palo Alto Research Center in the 1970s by Dr. Robert M. Metcalfe and was officially defined by the IEEE in the IEEE 802.3 standard [Beg82]. It is a three-layer architecture that corresponds to layers 1 and 2 of the Open Systems Interconnection (OSI) network reference model. The OSI Physical and Data Link layers are replaced by the Physical, the Medium Access Control (MAC), and the IEEE 802.2 Logical Link Control (LLC) layers. The IEEE 802.3

standard uses unslotted persistent Carrier Sense Multiple Access with Collision Detection (CSMA/CD) with binary exponential backoff [IEEE90].

2.2.1 High Speed Networks

Standard Ethernet has a theoretical maximum bandwidth of 10 Mbps. Although this is considered enough bandwidth for today's network applications, new factors are pointing to the need for more bandwidth. First of all, new networking applications like video-conferencing, imaging, and multimedia databases are already driving 10-Mbps Ethernet to its limit. At the same time, more and more computers are being added to LANs. The extraordinary leaps in computer performance move the networking performance bottleneck from the computer's processing power to the network bandwidth [3Com96]. High speed network standards have been around for a while. Fiber Distributed Data Interface (FDDI) is a fiber-optic, token ring network system running at 100 Mbps. It is often deployed as a backbone network with links to centralized clusters of servers. Copper Distributed Data Interface (CDDI) uses the same protocol as FDDI, but is modified to run over less expensive copper cabling. Some of FDDI/CDDI's strengths include redundancy and fault tolerance, built-in network management, and guaranteed access. Nevertheless, the high cost and complexity for desktop connections have prevented their widespread use [BeG92].

Another high-speed technology is Asynchronous Transfer Mode (ATM). ATM can accommodate both wide area network (WAN) and LAN connections since it provides a broad range of bandwidth, currently between 25 Mbps and 622 Mbps. ATM is intended to be used for real-time interactive multimedia applications because of ATM's built-in ability to merge voice, video, and data. What has limited its extensive deployment, however, are the high cost of ATM equipment, installation complexities, and problems in interoperability with existing networking equipment [BeG92] [Sta93].

A less expensive high-speed network technology is the new 100VG-AnyLAN, running at 100 Mbps. Though originally branded by its developer, Hewlett-Packard, as “Fast Ethernet,” this standard is not true Ethernet. It is based on a new MAC protocol, Demand Priority Access Method (DPAM), that is different than the CSMA/CD MAC Ethernet standard. The new MAC transports standard Ethernet frames, but using a “demand priority” mechanism [CMC95]. It was later extended to transport token ring frames as well, since Hewlett-Packard has designed 100VG-AnyLAN to be a low-cost, FDDI-like solution. Hence, the IEEE has moved the AnyLAN standardization from the Ethernet Working Group (802.3) to the new 802.12 Working Group.

2.2.2 The IEEE 802.3u Fast Ethernet Standard

When it became obvious that 10 Mbps would not be an enough data rate for many emerging multimedia applications, the IEEE 802 committee began to define the standard for 100-Mbps LANs. Between the two approaches presented to the committee, 100Base-T and 100VG-AnyLAN, only 100Base-T preserved the original Ethernet CSMA/CD MAC protocol. So, the IEEE 802.3 working group has adopted 100Base-T. The new standard, IEEE 802.3u, is a supplement to the original IEEE802.3, and it is widely known as Fast Ethernet [3Com96].

Among the most common high-speed network technologies, Fast Ethernet is expected to get the lion’s share of the high-speed LAN market. Its main advantages are as follows [3Com96].

- High performance: Based on the proven CSMA/CD MAC protocol, it gives ten times the performance of standard Ethernet.
- Standards-based technology: IEEE 802.3u is simply an extension of the original IEEE 802.3 standard. It is the same reliable, robust, and economical technology used in today’s dominant network standard.
- Cost-effective migration: Fast Ethernet can use the most common Ethernet wiring. Network administrators can use existing network analysis and management tools used for Ethernet. Application and networking software will work unchanged on both Ethernet and Fast

Ethernet. Due to the 10/100-Mbps Auto-Negotiation feature, defined in IEEE802.3u and integrated into most 100BaseT cards, both standards can work on the same LAN.

- Wide industry support: This is probably Fast Ethernet's biggest advantage over the rest of the high-speed technologies. An initial group of computer and communications industry leaders including Intel, 3Com, and Sun Microsystems, known as the Fast Ethernet Alliance, has successfully promoted the IEEE802.3u standard and ensured interoperability between different Fast Ethernet products.

2.2.2.1 Fast Ethernet Frame Format

The design of the Fast Ethernet MAC reduces the bit time, the duration of each bit transmitted, of the 10 Mbps CSMA/CD MAC by a factor of 10, enabling a ten-fold boost in the data rate. Since the Ethernet MAC specification is already speed-independent, there is no change in functionality. Packet format, packet length, error control, and management information are identical to 10BASE-T. The format of the Ethernet frame is shown in Figure 2.1 [IEEE90].

	Preamble	SFD	Destination Address	Source Address	Type / Length	Data	CRC
Bytes:	6	1	6	6	2	46 - 1500	4

Figure 2.1: Fast Ethernet frame format.

Each frame is preceded by at least a 6-byte (48-bit) Preamble sequence of 1s and 0s. This is required for synchronization of the two parties since the destination node can determine the source node's clock based on this sequence. The Start Frame Delimiter (SFD), 10101011, signals the beginning of the Ethernet frame. The 48-bit (6-byte) Destination Address and Source Address that follow are unique for each Ethernet card, and hence they are usually called the *hardware addresses*. The next two bytes can be either the type or the length of the encapsulated data. This is because there are currently two Ethernet standards in the industry; the original "DIX" Ethernet developed by Digital Equipment Corp., Intel Corp., and Xerox Corp., uses the field as *type*, while

the IEEE 802.3 standard uses the field as *length*. Fortunately, none of the valid IEEE 802.3 *length* values are the same as the Ethernet *type* values, making the two frame formats distinguishable. The actual data follows in the next field which should be of size 46 to 1500 bytes. Padding is used if the data is less than 46 bytes. The last field is the Cyclic Redundancy Check (CRC), a checksum that detects errors in the rest of the frame [IEEE90].

2.2.2.2 Main Components of the IEEE 802.3u Standard

As discussed in the previous section, the MAC layer is identical for both Ethernet and Fast Ethernet. Below this layer, however, some modifications were made to accommodate the 100-Mbps bandwidth of Fast Ethernet. Figure 2.2, shows the main components of the IEEE 802.3u standard [3Com96].

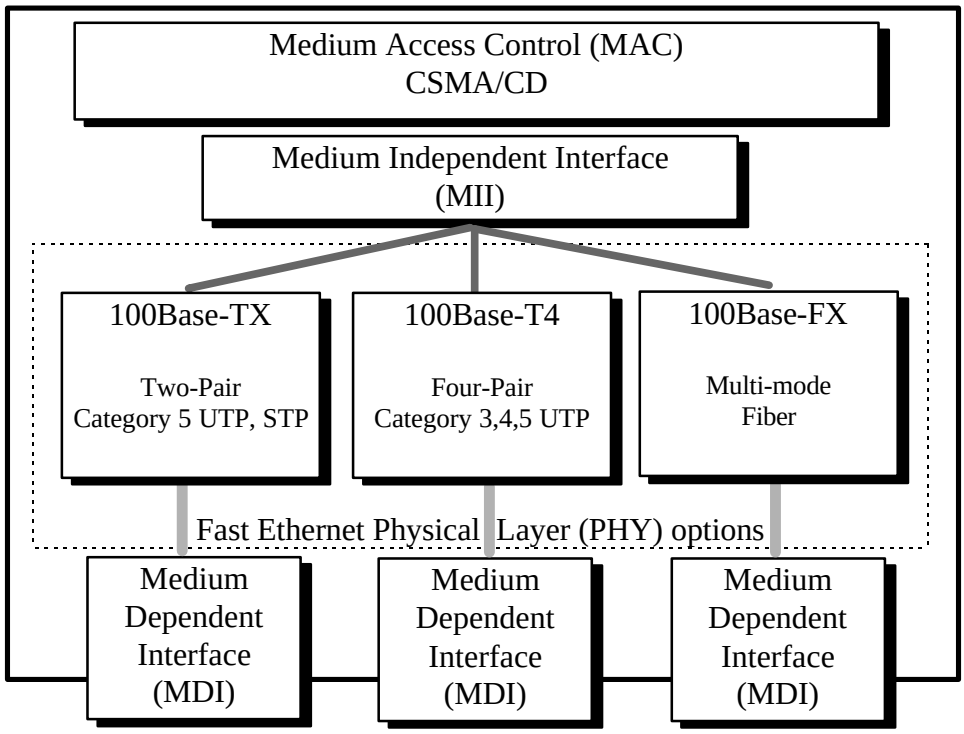

Figure 2.2: Main components of the IEEE 802.3u (Fast Ethernet) standard.

The Medium Independent Interface (MII) is an optional set of electronics that provides a way to link the Ethernet MAC functions in the network device with the Physical Layer Device (PHY) that sends signals onto the network medium. An MII may optionally support both 10 Mbps and 100 Mbps data rates. The MII is designed to make the signaling differences among the various media segments transparent to the Ethernet chips in the network device.

The physical properties of transmission are more difficult to deal with at 100 Mbps than at 10 Mbps. 100BASE-T supports three physical layers [3Com96] [CMC95]:

- 100BASE-TX, a two-pair system for data grade (EIA 568 Category 5) Unshielded Twisted Pair (UTP) and Shielded Twisted Pair (STP) cabling as defined by ISO/IEC 11801,
- 100BASE-T4, a four-pair system for both voice and data grade (Category 3, 4, or 5) UTP cabling as defined by ISO/IEC 11801, and
- 100BASE-FX, a two-strand multi-mode fiber system, as defined by ISO 9314.

The 100BASE-TX and 100BASE-FX media standards are both based on the FDDI/CDDI physical media dependent (PMD) specification developed and approved by the American National Standards Institute (ANSI) X3T9.5 committee. Because these chips are widely available, all of the first Fast Ethernet products support 100BASE-TX. The transmission scheme uses a block-code known as 4B5B, creating a transmission frequency of 125 MHz [NSC97b]. More detailed information about the 100BASE-TX medium standard is provided in Sections 4.2 and 4.3.

The 100BASE-T4 media standard was provided to make it possible to use lower-quality twisted-pair wire for 100-Mbps Ethernet signals. Category 3 cables have poor noise performance above 25MHz. To work on category 3 wire the 100BASE-T4 standard uses four pairs of wire. The signal is split among the wires and encoded using a block code known as 8B6T [3Com96].

2.3 Asymmetric Networking Concept

One major constraint for universal Internet access is the medium for the Internet connection. Most often the bandwidth bottleneck is at the user end of the connection [Gil95]. This “last-mile” user connection is the connection from the telephone company's (telco) end-office (EO) to the user's home. The overwhelming majority of today's Internet users use telephone lines via slow modems to access the Internet [Hal96]. The latest analog modems offer bandwidth of only up to 56 kbps [New97]. Web pages that are rich in multimedia content have made modem access largely unacceptable.

Studies have shown that the information “produced” by the majority of Internet users is only a small fraction of the information “consumed” by them [Hal96]. The telecommunications industry is taking advantage of this fact and is developing new technologies based on asymmetrical networking. The concept is to give huge downstream bandwidth to the user, but allow only a fraction of it for upstream bandwidth from the user. In fact, to the users’ advantage, there is a race among cable network companies, telephone companies, and satellite companies, each giving their own answer to users’ request for more bandwidth [Str96]. The details of these technologies is beyond the scope of this thesis. [Hal96] and [Str96] discuss and compare all three types of technologies. More information about Internet over cable TV networks can be found at [Gil95] and [Cic95]. The telco’s answer to high-bandwidth is Asymmetric Data Subscriber Line (ADSL). [Mot97] gives a detailed technical primer about ADSL. Currently, the only satellite Internet service is DirectPC® by Hughes Network Systems [Hug97]. Table 2.1 gives a summary of the asymmetric technologies.

As seen in Table 2.1, all three technologies require an additional end-user device. Each of these devices connects to a PC through a 10BASE-T network interface card (NIC). In the case of the satellite networks, an analog modem is also needed to dial the user’s Internet Service Provider (ISP) to send information back to the Internet. The satellite dish is strictly a receiver, so it cannot

transmit information back to the satellite [Str96]. The big advantage of these technologies is that they use existing public telecommunication networks.

Table 2.1: Asymmetric Network Technologies

Technology	Medium	End-user Device	Downstream Bandwidth	Upstream Bandwidth
Internet over cable networks	Coaxial cables	Cable modem	40 Mbps	2 Mbps
ADSL	Twisted copper wires	ADSL modem	9 Mbps	1.5 Mbps
Satellite networks	Wireless *	Satellite dish, adapter, analog modem	400 kbps	56 kbps *

* The upstream path is served with analog modems through a telephone line.

Among the three asymmetric public network technologies, the most prominent seems to be cable modems that use cable TV networks. The huge downstream bandwidth they provide may allure many home users. The fact that many cable modems are already on the market, while others are in the final trial stages at different cable networks around the world gives them a definite advantage in the market [Hal96]. ADSL modems have just appeared in the market with a downstream bandwidth of only 6 Mbps. Nevertheless, the dedicated bandwidth will attract many users concerned about privacy and that need fixed bandwidth. As for satellite networks, they need to provide more and reliable bandwidth. These will be ideal for isolated areas where there are no ADSL or cable networks installed [Str96] [Dav96].

2.3.1 Use of Fast Ethernet in Asymmetric Networks

Despite the growing popularity that Fast Ethernet enjoys in LANs, practical restrictions prevent it from being as popular in Medium or Metropolitan Area Networks (MANs) and WANs. Its use is almost prohibitive in end-user networks. A technical limitation is that FE preserves the 100-meter maximum UTP cable length from the hub to the desktop. This, however, can be by-passed with the use of repeaters in routers, bridges, or switches. Each time data passes through

one of these devices, it enters a new collision domain, allowing distances and hops to start again from zero [Sta93]. The major problem is the existing wiring to homes. It is everybody's wish that the "Information Highway" will someday consist exclusively of fiber-optic cables. But that day is likely not in the near future since installing new cables to each home is expensive. Moreover, FE cannot work on either telephone copper wires or cable coaxial wires. As discussed in Section 2.1, the most practical approach to provide high bandwidth to the home is the asymmetric networking concept.

By merging the two technologies we can provide most of their benefits to end-users, including:

- inexpensive hardware due to the popularity of Fast Ethernet,
- use of standard protocols and software applications, since Fast Ethernet is an extension of standard Ethernet, and
- use of existing wiring to the home.

Copper telephone wires exist to virtually all houses in the US. Nevertheless, ADSL's maximum 9 Mbps downstream bandwidth is even less than standard Ethernet's 10 Mbps symmetrical bandwidth [New97]. Cable modem technology, on the other hand, offers up to 30 Mbps (and growing) downstream bandwidth which can accommodate much of the FE's 100 Mbps bandwidth. Cable companies command a small, but still impressive, network that dates from the 1960's and covers 90 percent of US homes [Cic95] [Con95]. The overwhelming majority of cable modems today interface with PCs through a standard 10BASE-T Ethernet card. It is expected that since Fast Ethernet is the natural successor of Ethernet, cable modem manufacturers will eventually provide Fast Ethernet interfaces to their products. The IEEE has set up a new committee to specify the standards for Internet over cable technologies, IEEE 802.14 [Str96]. I discuss the integration of Fast Ethernet technology and asymmetric networks in more detail in Chapters 3 and 4.

2.4 IVDS Wireless Return Path

Although each of the asymmetric network technologies provide their own upstream path, CWT has designed and developed a unique return path to transmit user's commands. The IVDS wireless return path system consists of a standard TV/VCR/cable remote control, an IVDS AudioLink unit, and an IVDS Repeater unit [Fra96].

The user sends commands to the AudioLink with the remote control keypad. The AudioLink, which should be in the vicinity of the television set, is able to read the signals from the remote control. It then transmits these commands to the Repeater at a frequency range of 902-928 MHz, using random spread spectrum modulation [Gre96]. The Repeater collects data messages from AudioLinks located within a service area of 1,000 homes (the expected population around a 550-meter radius) and routes them through the Internet or other networks to a server using either a physical line or the Cellular Digital Packet Data (CDPD) service. At a rate of up to 56 kbps, the Repeater re-transmits the users' commands to the IVDS server using the Point-to-Point Protocol (PPP) [Fra96].

2.5 Summary

This chapter has provided background information about previous and existing interactive information services. Among the high-speed network technologies reviewed here, Fast Ethernet seems to be the most suitable for the IVDS/WWW Browser service. It can be incorporated with any of the asymmetric network technologies to deploy the IVDS/WWW service in existing public networks. Cable TV networks will make the most out of Fast Ethernet's high bandwidth and, hence, are more appropriate to be the medium to carry the FE frames. The IVDS wireless return path is independent of any specific asymmetric technology which allows flexibility in the decision to deploy a specific type of network.

Chapter 3 - PROBLEM STATEMENT

This chapter gives an overview of the IVDS system with emphasis on the WWW Browsing service and then concentrates on the networking requirements that are considered in this thesis. Section 3.1 describes the different parts of the IVDS/WWW Browser system that are necessary to complete the operational cycle from the moment a user requests a Web page until the requested Web page is displayed on the TV screen. Section 3.2 focuses on the networking requirements between the IVDS/WWW Browser Server and the Decoder Boxes.

3.1 IVDS/WWW System

The IVDS/WWW Browser system developed at the CWT makes use of Fast Ethernet and asymmetric networking. Figure 3.1 shows the client-end part of the system, while Figure 3.2 shows the server-end components of this system. The operational cycle starts the moment a user issues a command using his or her remote control device. The user's command is transmitted to the IVDS/WWW Browser Server through the IVDS wireless return path. At the Browser Server, the command is processed and a new screen is created, if needed. The screen is transmitted to the correct Decoder Box using the FE. The FE frames are injected into the cable TV network. At the user-end the Decoder Box collects the Ethernet frames and reconstructs the screen. Depending on whether the Decoder Box is located in (or on) the house or in the neighborhood, the NTSC signals are either sent directly to the TV set or injected back into the Internet channel of the cable TV network. A step-by-step description of the IVDS/WWW Browser system segments is presented in the next sections.

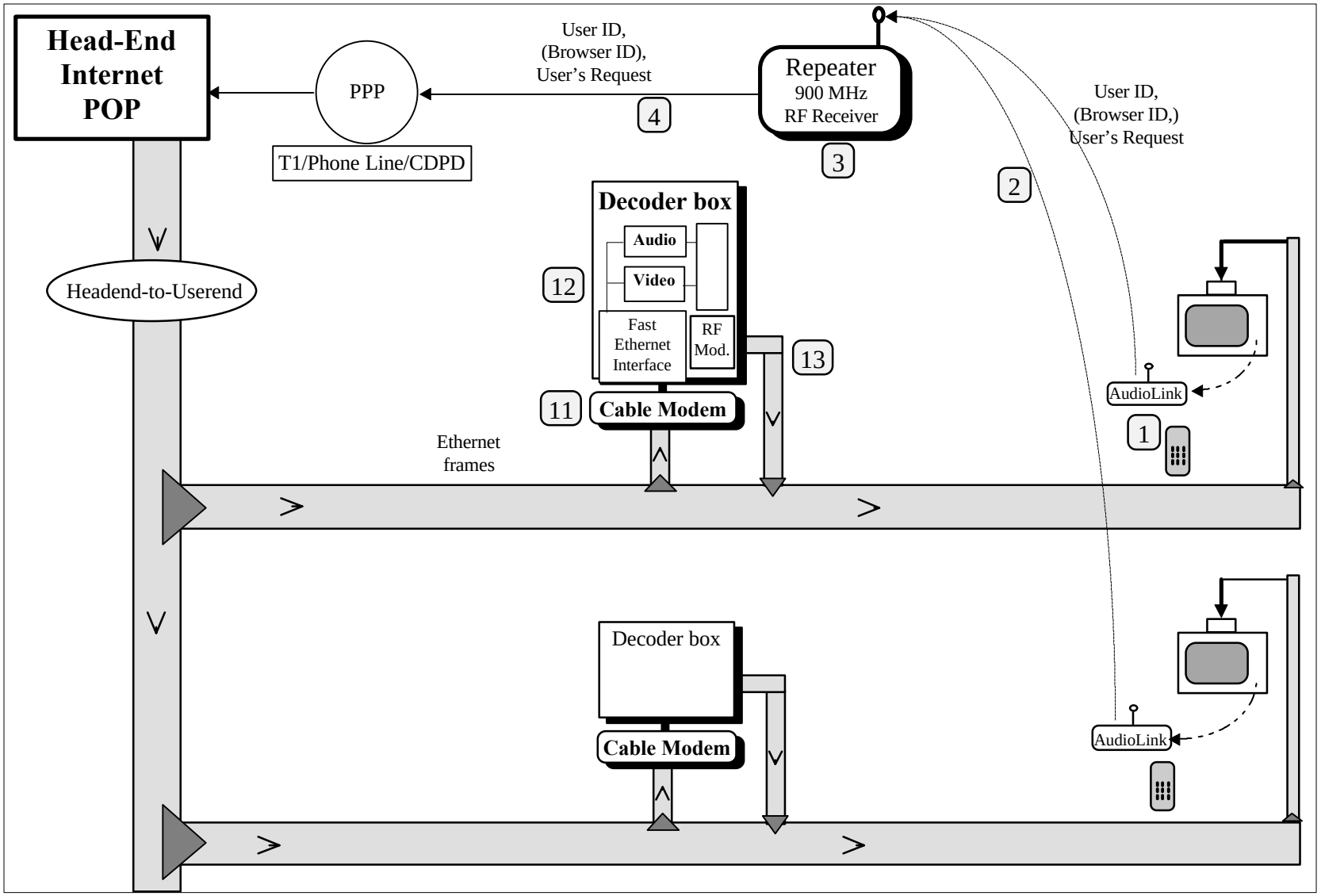

Figure 3.1: IVDS/WWW Decoder Box.

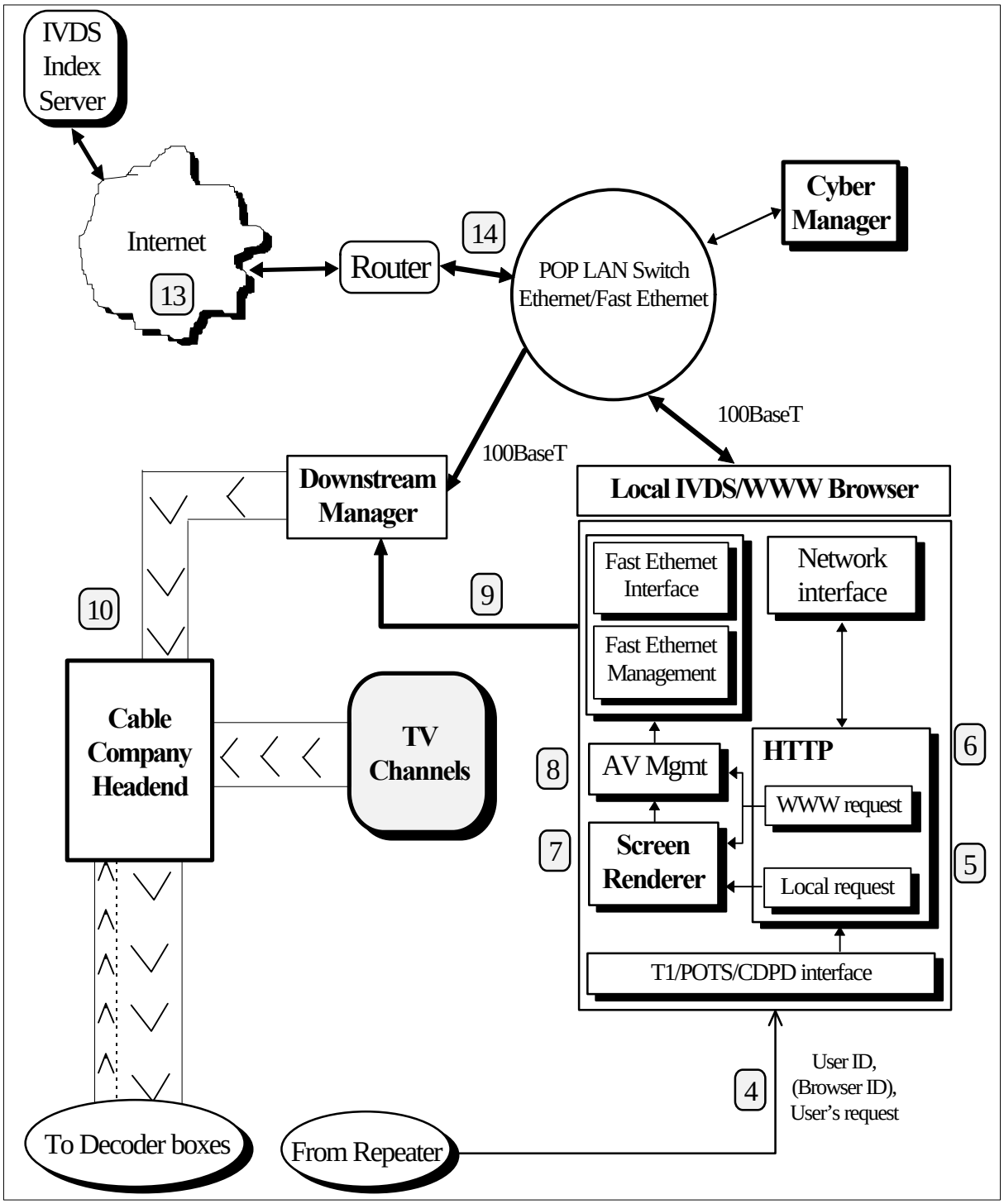

Figure 3.2: IVDS/WWW Browser Server.

3.1.1 IVDS Wireless Return Path

The upstream path is common to the entire IVDS system. It consists of a TV/VCR/Cable remote control, the IVDS AudioLink, and the IVDS Repeater. As shown in Figure 3.1, in step 1 the user, responding to instructions displayed on the TV screen or voice from the AudioLink speaker, uses the standard TV/VCR/Cable remote control to make his/her request. The interface format of the remote control and the TV display is described in detail in [Erg96]. The AudioLink should be near the TV set so that it can receive the infrared signal from the remote control. It should, also, be able to pick-up audio signals from the television set's speaker. These audio signals, which are in a frequency that cannot be detected by the human ear, are essential for the operation of IVDS services other than Web browsing. In step 2, the AudioLink transmits the user's request, along with the User ID and the Decoder Box ID. The 100 mW transmission is done at a 902-928 MHz frequency range using random spread spectrum modulation [Gre96]. The operation of a CWT-designed device, called the Repeater, constitutes step 3 of the IVDS Wireless Return path.

The Repeater is located in the neighborhood, serving about 1,000 AudioLinks. With an RF sensitivity of 100 dBm, it collects the AudioLinks' signals using the 902-928 MHz frequency range. Each Repeater can have up to four receiver boards powered by Motorola 6805 microcontrollers. All received messages are checked for data integrity using a CRC code. It then queues the messages and, at step 4, transmits them to the IVDS server using either the Cellular Digital Packet Data service or a physical line. The Repeater communicates with an Internet Service Provider (ISP) using the Point-to-Point Protocol (PPP). The software required for the operation of the Repeater runs on a Motorola 68306 processor [Fra96].

3.1.2 IVDS/WWW Browser Server

The Browser Server, shown in Figure 3.2, is itself connected to the Internet and receives the Web-related messages sent to it by the Repeaters (step 4). Based on the User ID, the Browser Server directs the request to the appropriate Browser thread. If the AudioLink request requires

only a local action, e.g. page down, the Browser makes the changes to the Web page and sends it to the Screen Rendering module. In step 5, if the AudioLink request requires new data from the Internet, e.g. a new Web page or audio clip, the Browser converts the AudioLink request to a HyperText Transfer Protocol (HTTP) request, and sends it to the Internet through a network interface (step 6). Upon arrival, the audio data is directed to the Audio-Video Management (AV Mgmt) module, while the video/graphics data is directed to the Screen Rendering module. The Screen Rendering module (step 7) renders the Web page to a 640-by-480 pixel screen, using a standard Red-Green-Blue (RGB) format. Each pixel can be either 8 bits or 24 bits long. The RGB data is forwarded to the AV Mgmt module.

The AV Mgmt module (step 8) is responsible for creating the AV packets from the raw data it receives. The format of the AV packet is shown in Figure 3.3 [Haw97]. The Mode field is specified as shown in Table 3.1.

	Mode	**Starting Address**	**Number of Samples**	**Data**
Bytes:	1	3	2	< 1494

Figure 3.3: IVDS/WWW AV packet format.

Table 3.1: AV Packet Mode Field Specifications

Mode	**Bit:**	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**
24-bit Color		x	0	0	0	x	x	x	x
8-Bit Color		x	0	0	1	p	p	p	p
New Audio		x	0	1	0	x	x	x	x
Continuing Audio		x	0	1	1	x	x	x	x

x = not used,
p = used as a 4-bit address to specify which palette
to use to convert the 8-bit color to 24-bit color.

The *Starting Address* field is used in the case of 24-bit or 8-bit color video data to specify the starting screen address into which the first pixel data of the AV packet should be written. The

Number of Samples field specifies the number of color or audio samples that the AV packet contains. The size of each sample varies depending on the Mode field. 24-bit color and 8-bit color are obviously 3 bytes and 1 byte long, respectively. New and Continuing Audio samples are both 2 bytes (16 bits) long [Haw97]. One restriction in the creation of the AV packets is that the total length of each, both header and data, should not be more than the maximum number of bytes that the data field in the Ethernet frame can hold, i.e. 1500 bytes. Each Browser's AV Mgmt module sends the AV packets to the Fast Ethernet Management and Interface modules which create the Ethernet frames, schedule their transmission, and transmit them to the correct Decoder Boxes (step 9).

3.1.3 IVDS/WWW Downstream Path

In merging Fast Ethernet and asymmetric network technology, the Ethernet frames are injected in the data channel(s) of the cable network with the cable company's Downstream Manager (step 10). The total bandwidth of data channels depends on the cable company's policy [Gil95]. A limiting factor, however, is that at this point cable modems can receive only up to 40 Mbps [Dav97]. The Ethernet frames are received by the cable modem assigned to the Decoder Box.

3.1.4 IVDS/WWW Decoder Box

The Decoder Box needs to have the appropriate network interface to receive the Ethernet packets from the cable modem (step 11). Based on the type of the data, audio versus video, the Decoder Box reassembles the data to either a 640-by-480 screen and encodes it into NTSC format or uses a Digital-to-Analog Converter (DAC) to reproduce the audio (step 12). Since the Decoder Box can be located either in (or on) the house or in the neighborhood, there are two ways to send the NTSC signals to the television set. If the Decoder Box is in the house it can act as a set-top box that is connected directly to the TV. If it is in the neighborhood, an RF modulator is necessary to re-inject the NTSC signal into the data channel of the Cable Network (step 13). The Web pages are displayed on the TV from the cable set-top box [Haw97].

3.2 Browser Server - to - Decoder Box Networking

The distribution of the Web material, both video and audio, from the Browser Server to the Decoder Boxes consists of three parts:

- Browser Server Network Interface (Transmission)
- Communication Medium
- Decoder Box Network Interface (Reception)

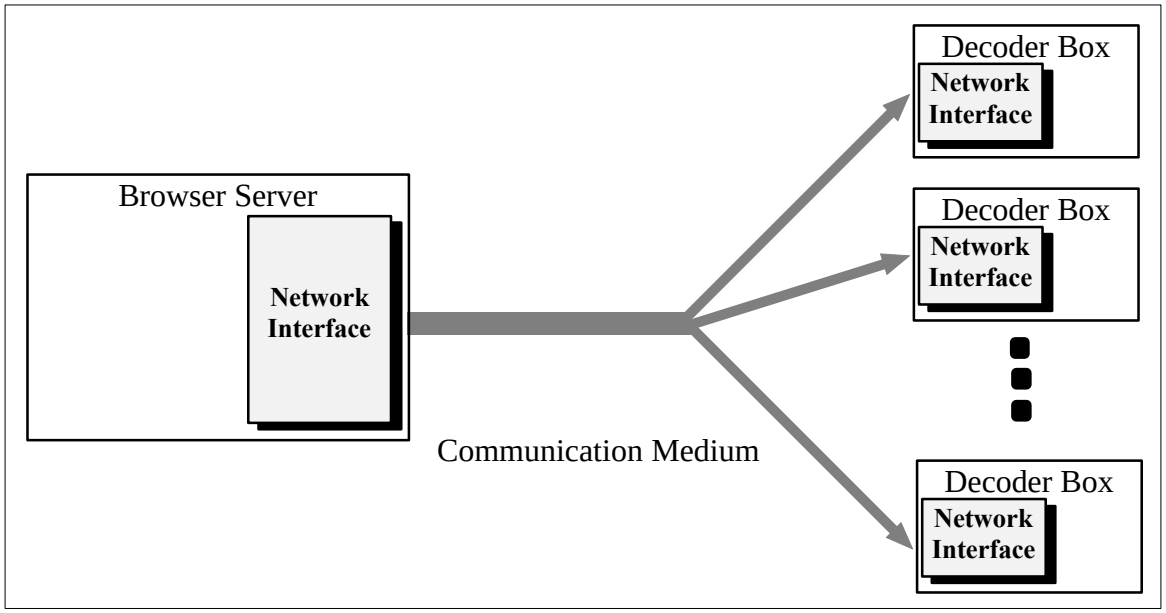

Figure 3.4: IVDS/WWW Browser Server to Decoder Box networking.

The Browser Server should be able to service up to 256 Decoder Boxes. The AV packets created by each of the Browser threads should be scheduled for transmission according to the Mode of each packet and the order of their creation. The latter depends on the order of requests by each user. The Browser Server network interface should be able to distribute the AV packets to the correct Decoder Boxes. The communication medium should be able to handle the bandwidth created by the Browser Server network interface. Finally, the Decoder Box network

interface should be able to connect to the communication medium, receive all the packets sent from the server for this particular Decoder Box, and provide the rest of the Decoder with only the packets ready to be processed.

3.2.1 Browser Server Network Interface

The IVDS/WWW Browser server is a software package that resides on a fast computer. As explain in Section 3.1, for each active Decoder Box, a new Browser thread is spawned to handle each user's requests. The AV Packet Management (Mgmt) module of each browser thread sends the AV packets to a Network Interface which is common to all active browser threads.

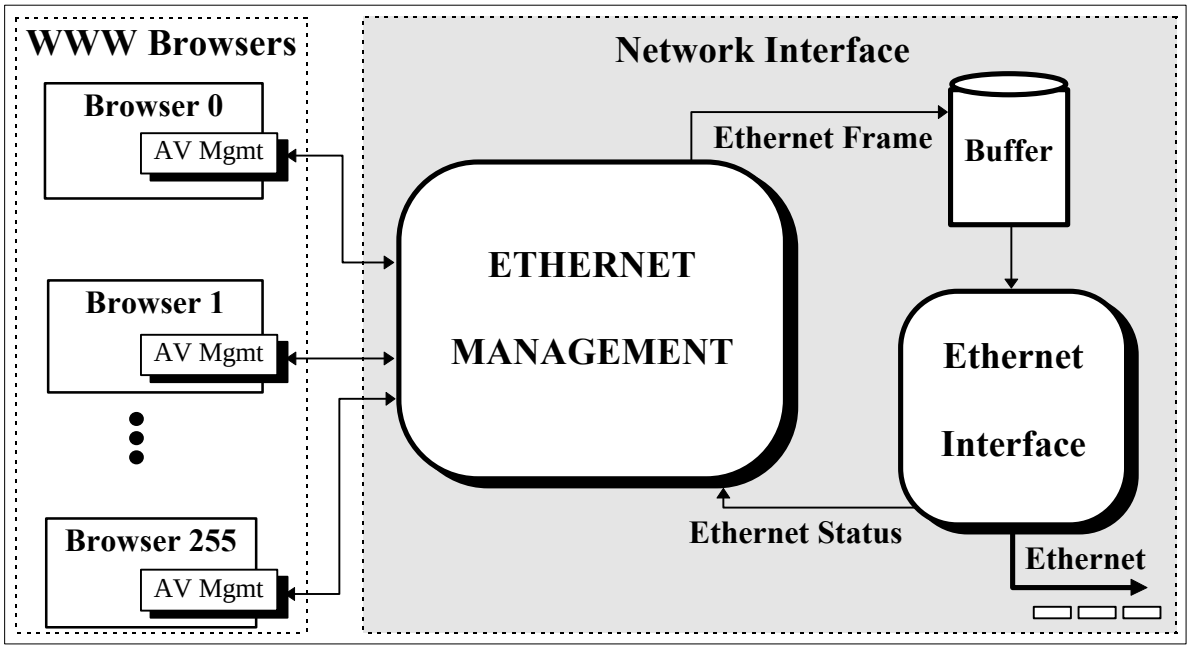

Figure 3.5: Browser Server Network Interface.

The Network Interface should have bi-directional communication with the AV Packet Mgmt function for each browser. While receiving the AV packets for a Decoder Box, the Network Interface should be able to give feedback to the corresponding AV Packet Mgmt about the status of its packets. The responsibilities of the Network Interface are:

- identification of the correct Decoder Box ID based on the browser thread,
- scheduling of the AV packets so that they will not arrive at the correct Decoder Box too late or too early, which is critical for the audio as well as any motion pictures or animations on the Web page,
- fair time allocation for each active Decoder Box,
- encapsulation of the AV packets into Ethernet frames using as destination address the Ethernet address of the Decoder Box, and
- transmission of the raw Ethernet frames from the computer's Fast Ethernet Network Interface Card.

The fact that the system must deal with real-time audio, or even video, makes the synchronization of the Server and the Client (Decoder Box) an important issue. If the network delays data such that the client has no more data to play, an unpleasant gap in the audio or video occurs. Moreover, in our case the client is hardware based with a fixed amount of buffer memory. Hence, if a packet arrives too early at the Decoder Box and its memory is full, the packet will be lost. The usual solution is to implement a client-server protocol that buffers data at the client during intervals when data can be transferred from the server to the client faster than it can be played and then reduces the amount of buffered data during intervals when the data is being played faster than it can be delivered by the server. Another adjustment scheme is to control the rate at which packets are sent over a connection, the objective being to limit this rate to the capacity of the connection. In first-in first-out (FIFO) networks such as the Internet, this capacity changes with time because the number of connections routed through a given link varies and because sources do not sent data at a constant rate.

In both of these approaches the server can receive feedback about the state of the network from the client. Because of the nature of the IVDS wireless return path, the IVDS/WWW Browser Server has practically no feedback from the Decoder Box. Hence, a new protocol had to be developed to handle network errors (see Section 4.1.2.2.1).

3.2.2 Decoder Box Network Interface

At the IVDS Decoder Box, the network interface should be entirely hardware based, as is the rest of the device. The network interface should be able to:

- receive the electrical signals from the wire,
- convert signals into raw Ethernet frames,
- provide the Ethernet frames to the MAC layer (Custom Controller).

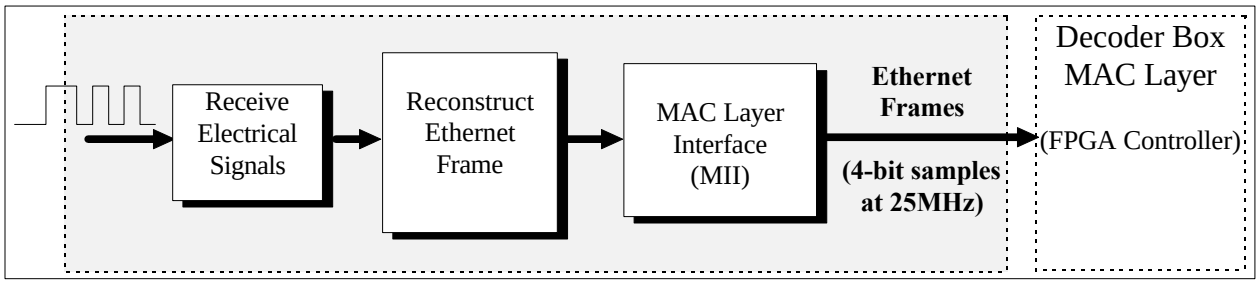

Figure 3.6: Decoder Box Network Interface.

Despite the fact that Fast Ethernet is a relatively new technology, there are already many chip-sets available that fulfill the above requirements [3Com96]. The research for this part of the project was to not only identify the correct chips, but also the least expensive since the cost of the Decoder Box is intended to be as low as possible. Selection from among the three different physical layer standards of Fast Ethernet is also an important issue since this will define which chips we will select.

3.2.3 Communication Medium

Research on the communication medium to be used to transfer the Ethernet frames from the Browser Server to the Decoder Box consists of two parts. First we identified which of the three Fast Ethernet physical layer standards is best suited for this application. As discussed in Section 2.2.2.2, each standard is to be used on a different type of cable, unshielded twisted pair

(UTP), shielded twisted pair (STP), or fiber optic cable. Nonetheless, since none of these cables are deployed in large scale at the “last-mile” end-user connection, we also need to determine which of the upcoming asymmetric networking technologies is best suited to accommodate the transmission of the Fast Ethernet frames.

3.3 Summary

The Browser Server to Decoder Box network interface requires software design at the Browser Server, research on communication medium technologies, and hardware design at the Decoder Box. Both, the Browser Server and the Decoder Box network interfaces should follow the modular design guidelines, making them independent of the design of the rest of the project. At the same time, they should have concrete interfaces to the other modules to provide smooth interaction with them.

Chapter 4 - SYSTEM DESIGN

This chapter presents the design of the Network Interface between the IVDS/WWW Browser Server and the Decoder Box. In Section 4.1, I explain the Ethernet Management module (EMM) and the Ethernet Interface module (EIM) that constitute the Browser Server Network Interface. Section 4.2 discusses the Decoder Box Network Interface design, while Section 4.3 presents results for the communication medium.

4.1 Browser Server Network Interface

Based on the requirements for the IVDS/WWW Browser Server considered in Chapter 3, I developed a software-based Network Interface to handle multiple Decoder Boxes. The Network Interface consists of an Ethernet Mgmt module and an Ethernet Interface module. Both modules share two data structures as explained next.

4.1.1 Data Structures

For every active Browser/Decoder Box the EMM maintains entries in two data structures: Decoder Info and Latest Priority Queue (LPQ). The Decoder Info data structure holds the information for the Decoder Boxes, audio and video queues that contain the AV packets waiting for transmission to the Decoder Box, and timing and scheduling information. The C language definition of the data structure is shown in Figure 4.3.

The LPQ is implemented as an ordered linked list. Each entry of the LPQ contains the *Decoder id*, the *type* of the data to be transmitted (audio or video), the latest transmission *time*, and the earliest transmission *e_time*. The insertion function of the LPQ uses the *time* field as the key and inserts the entry at the appropriate position. In other words, the LPQ is always sorted in

an ascending order based on the *time* field. Hence to get/delete the minimum transmission time, we simply need to get/delete the top element of the LPQ. I have implemented the LPQ as a linked list because linked lists provide better performance for large sets (our application has a maximum of 256 entries) than arrays. When we insert a new browser/decoder into the LPQ we do not have to shift the remaining copies, but only adjust some pointers.

```
typedef struct  {
  int     id;               /* browser - decoder id                      */
  char    DecoderID[20];    /* decoder box ethernet address              */
  int     fd;               /* UNIX domain socket for this decoder       */
  long    FreeVideoMem;     /* the free video memory at any time         */
  long    VideoMemRate;     /* the Video Memory Rate                     */
  queue   video_queue;      /* the queue containing the video packets    */
  double  LastVideoTime;    /* Last time a video packet was transmitted  */
  double  EarliestVideoTime; /* Earliest time a video packet can be sent  */
  double  LatestVideoTime;  /* Latest time a video packet can be sent    */
  long    FreeAudioMem;     /* the free audio memory at any time         */
  long    AudioMemRate;     /* the Video Memory Rate                     */
  queue   audio_queue;      /* the queue containing the audio packets    */
  double  LastAudioTime;    /* Last time an audio packet was transmitted */
  double  EarliestAudioTime; /* Earliest time an audio packet can be sent */
  double  LatestAudioTime;  /* Latest time an audio packet can be sent   */
} decoder_info;
```

Figure 4.1: Decoder Info data structure definition.

4.1.2 Ethernet Management Module

The responsibilities of the Ethernet Mgmt module are, first, to receive the AV packets from multiple browser threads of the Browser Server and, second, to schedule the AV packets to be transmitted by the EIM.

4.1.2.1 Communication with the Rest of the IVDS/WWW Browser Server

The AV Mgmt module of each active browser should pass the AV packets to the Network Interface as efficiently as possible. Since all the software for the Browser Server will reside on the same computer, it is reasonable to use a communication method that takes advantage of the fact that the software shares the same operating system (OS) kernel. The most obvious way is to use shared memory between the two processes. One problem with this option, however, is portability

because different versions of UNIX have different limits on shared memory [Ste90]. Another problem with shared memory is that when the server and the client are unrelated processes, the server cannot identify the client [Ste92]. So the client's id should be written every time the client sends a packet to the server. Moreover, due to the immense processing requirements of the Browser Server software, later versions might end up running on a distributed environment. In this case, especially, shared memory will not be a viable option. Use of network protocols, on the other hand, would give us complete independence, but with prohibitive overhead. The UNIX domain protocols, presented in the next section, are the best solution for our application because of their portable code and low processing requirement [Ste96].

4.1.2.1.1 UNIX Domain Sockets

The UNIX domain protocols are a form of interprocess communication (IPC) that are accessed using the same sockets Application Programming Interface (API) that is used for network communication. Figure 4.1 shows a client/server application communicating on the same host computer written using the Internet protocols (left) and the UNIX domain protocols (right).

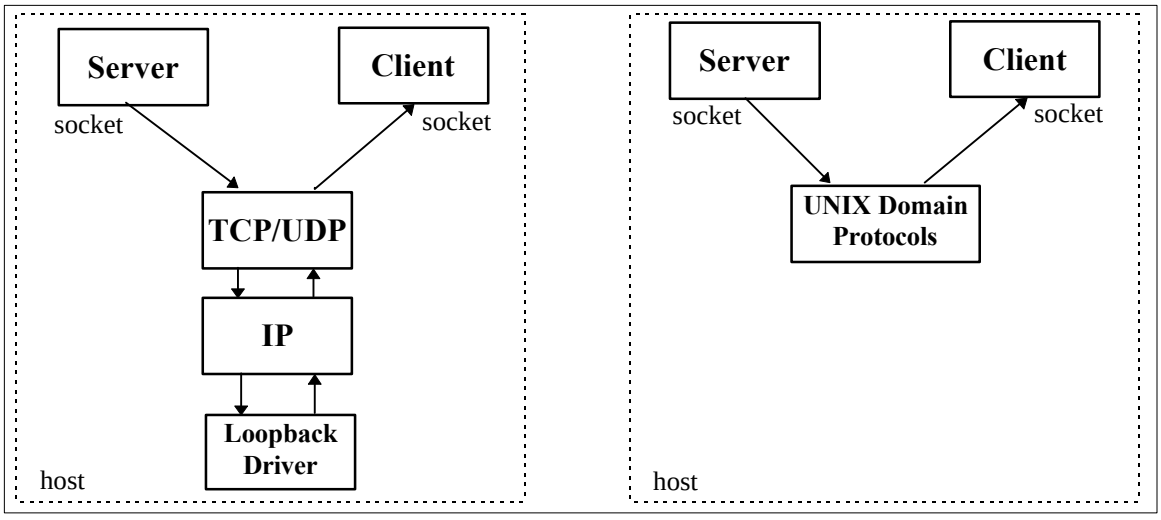

Figure 4.2: Client/server protocol comparison of applications running on the same host.

When the server sends data to the client using the Internet protocols, the data is processed by TCP or UDP, then by IP, sent to the loopback driver where it is placed onto IP's input queue, then processed by IP, then TCP/UDP, and finally passed to the client [Ste94]. Although transparent to the client and the server, this amount of processing is not required when the data never leaves the host. The UNIX domain protocols require less processing, i.e. they are faster, because they are designed for applications whose data never leaves the host. There is no checksum to calculate or verify, there is no potential for data to arrive out of order, flow control is simplified since the kernel can control the execution of both processes, etc. Table 4.1 presents a summary of the performance of UNIX domain sockets versus TCP sockets [Ste96].

Table 4.1: Comparison of Unix Domain Socket Throughput versus TCP [Ste96]

Kernel	**Fastest TCP (KB/sec)**	**Unix domain (KB/sec)**	**% increase Unix/TCP**
DEC OSF/1 V3.0	14,980	32,109	114
SunOS 4.1.3	4,877	11,570	137
BSD/OS V1.1	3,459	7,626	120
Solaris 2.4	2,828	3,570	26
AIX 3.2.2	1,592	3,948	148

While other forms of InterProcess Communication (IPC), such as message queues, shared memory, and named pipes, can also provide these same advantages, the advantage of the UNIX domain protocols is that they use the same, identical sockets interface that networked applications use; clients call *connect*, servers call *listen* and *accept*, both use *read* and *write*, and so on. The other forms of IPC use completely different APIs, some of which do not interact nicely with sockets and other forms of input/output (I/O) [Ste90].

The UNIX domain protocols provide both a stream socket, similar to a TCP byte stream, and a datagram socket, similar to a UDP datagram. The names used to identify sockets in the UNIX domain are pathnames in the filesystem. This is also an important advantage over other forms of IPC. The ability to pass a descriptor between unrelated processes across a UNIX domain

socket allows the Network Interface module to identify the calling browser thread, from which it can retrieve the correct Decoder ID.

4.1.2.1.2 The AV Packets Reception Algorithm

Using the *select()* function, the EMM acts as a server with the active browser threads as clients. The EMM opens a master UNIX domain socket with an address known to all the clients. The AV Mgmt module (client) of each browser thread requests a new connection by passing to the EMM (server) the socket address through the master UNIX domain socket. The new UNIX domain socket address is simply a path in the filesystem. It is of the form *ivdsbrowserXXX*, where XXX is the number of the browser. The EMM opens the new connection and the client starts sending the AV packets, as shown in Figure 4.3.

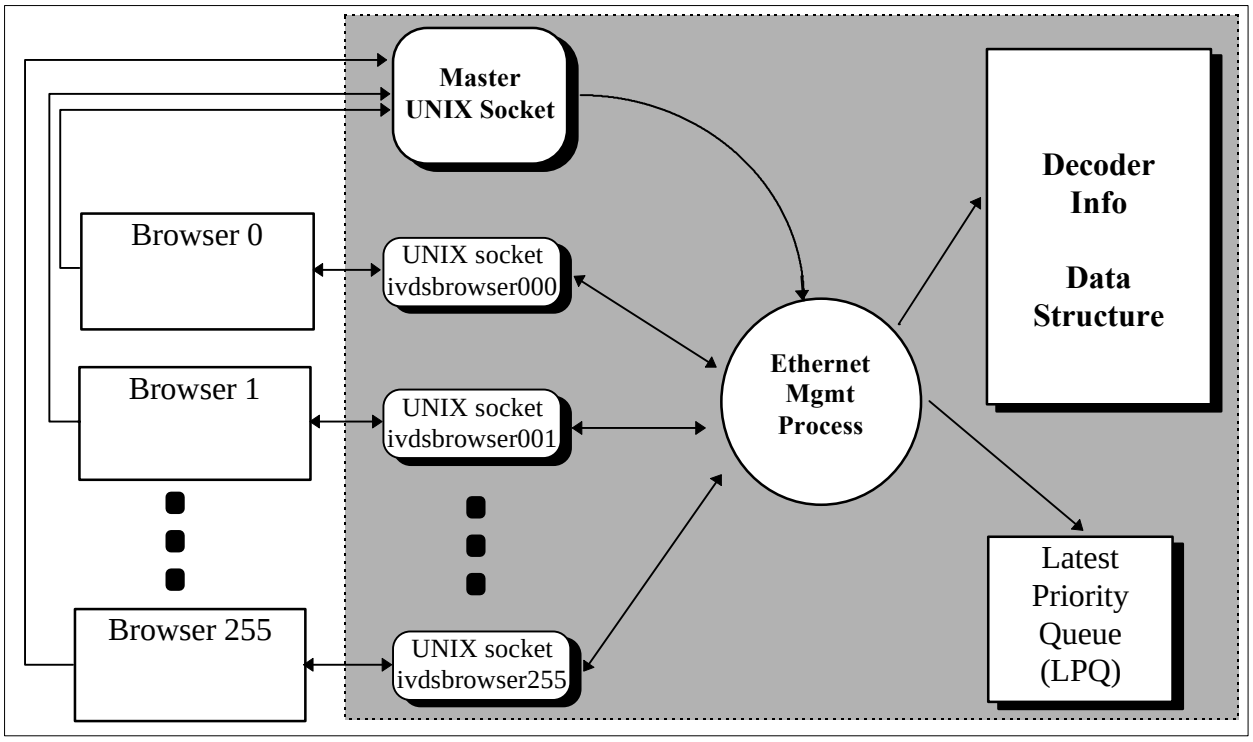

Figure 4.3: Passing AV packets to the Ethernet Mgmt module.

Based on the socket address, the EMM determines the Decoder Box for which the data is intended and stores it in the appropriate Decoder Box data structure. When data for a new

Decoder Box arrives, the EMM creates a new Decoder Info entry, and inserts an entry in the LPQ. The fixed values for each Decoder Box are kept in a file. The EMM determines the *id* and the *fd* fields from the UNIX domain socket client (browser) address. Based on the *id* it initializes the fields *DecoderID, FreeVideoMem, VideoMemRate, FreeAudioMem, AudioMemRate* to the values of the corresponding Decoder Box. Although, except from the *DecoderID* (Ethernet address), the rest of the aforementioned fields are the same for all Decoder Boxes, I keep them all in a file for flexibility in future versions of the system. All timing fields are initialized to zero. The video and audio queues are also initialized and receive the first AV packets. The EMM inserts the AV packets of type 8-bit and 24-bit color into the *video_queue*, and packets of type New and Continuing Audio into the *audio_queue*. Additional AV packets are inserted in the appropriate queue of the Decoder Info data structure.

4.1.2.2 AV Packet Scheduling

Although, as explained in Chapter 3, we cannot apply any known error correction techniques, we can make use of the fact that the Decoder Boxes are entirely hardware based devices, and, hence, have predictable behavior. They have fixed sized audio and video memory. The transfer rates of the memories, as well as the rest of the semiconductor devices, are also fixed. Since the only node that is expected to transmit in the network is the IVDS/WWW Browser Server, there can be no collisions. Given the fact that the network is uni-directional, even the network latency can be predicted. By incorporating this knowledge for each Decoder Box into the Browser Server Network Interface I have developed an “error prevention” algorithm. With this algorithm the software determines a time window during which the Ethernet frames will arrive at the correct Decoder Box, neither too late nor too early.

4.1.2.2.1 Error Prevention Protocol

The Latest Priority Queue is the heart of the error prevention algorithm. For every active Decoder Box, there is an entry in the LPQ. As shown in Figure 4.4, in step 1 the Ethernet Mgmt

module process reads the top of the LPQ. Based on the value of their *time* fields, it decides *which* Decoder Box needs *what type* of data more urgently.

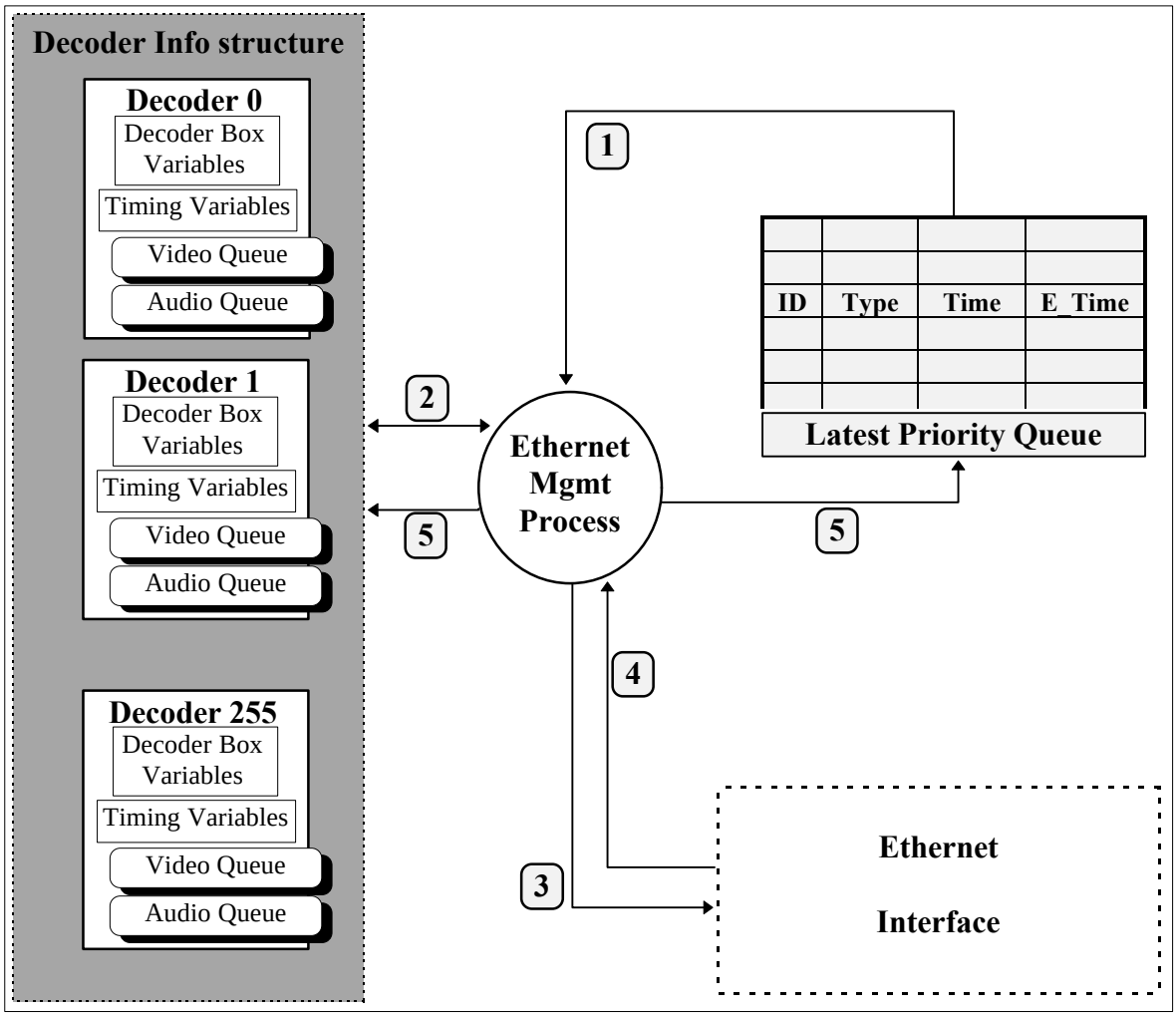

Figure 4.4: AV packet scheduling diagram.

If the current time is less than the earliest transmission time as indicated by the *e_time* field of the LPQ entry, the EMM tries to utilize the time interval; it looks for the next best entry of the LPQ whose *e_time* field is less than the current time. If it cannot find one such entry, the EMM waits until it is safe to send data to the original Decoder Box. In any case, the selected entry is removed from the LPQ. The EMM then accesses the Decoder Info data structure and dequeues

the AV packet from the correct audio or video queue depending on the *type* field of the selected LPQ entry (step 2). At step 3, it sends the AV packet to the Ethernet Interface module for transmission. For better performance there is no data copying between the EMM and the EIM, but simple pointer passing.

The EMM keeps sending audio or video AV packets to the EIM as long as:

- the audio or video queue of the Decoder Info structure is not empty,
- the elapsed time is less than a pre-specified *TIMEPERDECODER* time slot (to achieve fairness among the active Decoder Boxes),
- the next AV packet will not overflow the audio or video memory of the Decoder Box (to make sure that no packet is lost), and
- the *data_type* field of the top entry of the LPQ is not Continuing Audio and its *time* field is less than the current time (to make sure no gap occurs in an audio clip).

Upon successful transmission of this set of AV packets, the EIM returns the last audio or video transmission time for the Decoder Box. Based on previous values of this type's timing variables and the new transmission time, the EMM calculates the new values for this type's timing variables in the Decoder Info data structure. The calculations are described in Section 4.1.2.2.2. Finally at step 5, the EMM updates the timing variables in the Decoder Info data structure. If there are more AV packets to be transmitted a new entry is also inserted in the LPQ. Based on the new timing variables the Ethernet Mgmt module decides the values of the *type, time, and e_time* fields for the entry in the LPQ. The complete algorithm is shown in Figure 4.5, where the LPQ entry is defined in the C language as:

```
struct Entry {
  int        id;
  int        data_type;
  double     time;
  double     e_time;
}
```

```
IF the LPQ is NOT empty {
    /* select which AV packet to send based on the LPQ */
    send ← get_min(latest_pq)
    IF current time < send.e_time
        send ← get_next_best(&latest_pq, current time)
    ELSEIF
        send ← deletemin(&latest_pq)

    i ← send.id
    /* PICK THE NEXT IVDS PACKET FROM THE VIDEO QUEUE */
    IF (send.data_type is VIDEO) AND (current time > Decoder[i].EarliestVideoTime)
      start time ← current time
      packets ← 0
      max_packets ← maximum number of pixels in an AV packet

      WHILE (Decoder[i].video_queue NOT empty) AND
               ((current time - start time) < TIMEPERDECODER)) AND
               (packets + max_packets < Decoder[I].FreeVideoMem)
        Send to the EIM the front(Decoder[i].video_queue) and the Decoder[i].DecoderID
        Get the number_of_pixels in the AV packet from the AV packet header
        packets ← packets + number_of_pixels
        dequeue(&Decoder[i].video_queue);
        IF (latest_pq is NOT empty) AND current time > (get_min(latest_pq).time) AND
           (get_min(latest_pq).data_type is continuing audio)
                EXITWHILE;
        ENDIF
      ENDWHILE

      calculate new timing variables for this Decoder Box (see Section 4.1.2.2.2)
      Decoder[i].LastVideoTime ← current time
    ELSEIF
      /* PICK THE NEXT IVDS PACKET FROM THE AUDIO QUEUE */
      IF (send.data_type is AUDIO) AND (current time > Decoder[i].EarliestAudioTime)
        start time ← current time
        packets ← 0
        max_packets ← maximum number of audio samples in an AV packet

        WHILE (Decoder[i].audio_queue NOT empty) AND
               ((current time - start time) < TIMEPERDECODER)) AND
               (packets + max_packets < Decoder[I].FreeAudioMem)
          Send to the EIM the front(Decoder[i].audio_queue) and the Decoder[i].DecoderID
          Get the number_of_samples in the AV packet from the AV packet header
          packets ← packets + number_of_samples
          dequeue(&Decoder[i].audio_queue);
          IF (latest_pq is NOT empty) AND current time > (get_min(latest_pq).time) AND
           (get_min(latest_pq).data_type is continuing audio)
                EXITWHILE;
           ENDIF
         ENDWHILE

         calculate new timing variables for this Decoder Box (see Section 4.1.2.2.2)
         Decoder[i].LastAudioTime ← current time
    ENDIF
    Insert new entries for this decoder box into the LPQ based on the new timing variables.
ENDIF
```

Figure 4.5: AV packet scheduling algorithm.

In the AV packet scheduling algorithm of Figure 4.5, the following functions and variables are defined as:

- `get_min(pq)` returns the top entry of the priority queue `pq`,
- `delete_min(pq)` deletes the top entry of the priority queue `pq`,
- `get_next_best(pq, time)` returns and deletes the next best entry from the `pq`,
- `front(Q)` returns the front element of video or audio queue `Q`,
- `dequeue(Q)` deletes the front element of video or audio queue `Q`, and
- `TIMEPERDECODER` is the allocated time for transmission per decoder box.

4.1.2.2.2 Timing Calculations

The calculations of the timing variables of the Decoder Info data structure are critical for the successful distribution of the AV packets. Each time the EMM sends a number of AV packets to the EIM, the timing variables are re-calculated. As seen in the previous section, the EMM keeps track of how many AV packets have been transmitted for a specific Decoder Box during the allocated time slot, as well as the last transmission time. Based on these values, the EMM recalculates the Decoder Info's variables as shown in Figure 4.6 for video-type packets and in Figure 4.7 for audio-type packets.

```
1.  freed_mem ← (current time - Decoder[i].LastVideoTime) * Decoder[i].VideoMemRate
2.  Decoder[i].FreeVideoMem ← Decoder[i].FreeVideoMem + (freed_mem - packets);
3.  IF (Decoder[i].FreeVideoMem >= VIDEOMEM) THEN Decoder[i].FreeVideoMem = VIDEOMEM
4.  IF (Decoder[i].FreeVideoMem <= 0) THEN Decoder[i].FreeVideoMem = 0
5.  IF (Decoder[I].FreeVideoMem < LOWER_LIMIT)
6.        Decoder[i].EarliestVideoTime ← current time + (LOWER_LIMIT/Decoder[i].VideoMemRate)
7.  ELSE
8.        Decoder[i].EarliestVideoTime ← 0
9.  IF (Decoder[I].FreeVideoMem > UPPER_LIMIT)
10.       Decoder[i].LatestVideoTime ← current time
11. ELSE
12.       Decoder[i].LatestVideoTime ← current time +
                              ((VIDEOMEM - Decoder[i].FreeVideoMem) / Decoder[i].VideoMemRate)
```

Figure 4.6: Video timing calculations.

Lines 1 and 2 in Figure 4.6 calculate the new free video memory, *FreeVideoMem*, based on the current free video memory, the number of pixels sent, *packets*, and the estimated number of pixels freed from the video memory since the last transmission time, *freed_mem*. Lines 3 and 4 make sure that the free video memory calculated at the Server is within the actual memory range. The calculation of the earliest video transmission time, *EalriestVideoTime*, is done in lines 5 through 8. To avoid the loss of AV packets, the EMM puts a lower limit, *LOWER_LIMIT*, in the free video memory. If the *FreeVidMem* falls below this *LOWER_LIMIT*, the *EarliestVideoTime* is set to the current time plus the time it will take for the Decoder Box to free a *LOWER_LIMIT* amount of video memory. This ensures that next time this Decoder Box can take at least *LOWER_LIMIT* pixels and at most (2 * *LOWER_LIMIT* - 1) pixels. Finally, at lines 9 through 12, the latest video transmission time, *LatestVideoTime*, is calculated. To compensate for minor delays, other than video memory transfer, the EMM introduces an *UPPER_LIMIT* in the free video memory. One such minor delay is the time it takes the Decoder Box to receive the Ethernet frame and put the AV packet into the video memory. If the free video memory is above this *UPPER_LIMIT*, i.e. if the video memory is almost empty, the *LatestVideoTime* is set to the current time. Otherwise, the *LatestVideoTime* is set to the current time plus the time it will take the Decoder Box to free the full video memory.

```
1.  freed_mem ← (current time - Decoder[i].LastAudioTime) * Decoder[i].AudioMemRate
2.  Decoder[i].FreeAudioMem ← Decoder[i].FreeAudioMem + (freed_mem - packets);
3.  IF (Decoder[i].FreeAudioMem >= AUDIOMEM)THEN Decoder[i].FreeVideoMem = AUDIOMEM
4.  IF (Decoder[i].FreeAudioMem <= 0)THEN Decoder[i].FreeVideoMem = 0
5.  IF (Decoder[i].FreeAudioMem < LOWER_LIMIT)
6.         Decoder[i].EarliestAudioTime ← current time + (LOWER_LIMIT/Decoder[i].AudioMemRate)
7.  ELSEIF
8.         Decoder[i].EarliestAudioTime ← 0
9.  IF (Decoder[I].FreeAudioMem > UPPER_LIMIT)
10.       Decoder[i].LatestAudioTime ← current time
11. ELSE
12.       Decoder[i].LatestAudioTime ← current time +
                              ((AUDIOMEM - Decoder[i].FreeAudioMem) / Decoder[i].AudioMemRate)
```

Figure 4.7: Audio timing calculations.

Calculations for the audio timing variables, *EarliestAudioTime* and *LatestAudioTime* are the same as their video counterparts but are based on the *LastAudioTime*, the *FreeAudioMem,* and the *AudioMemRate* variables.

The following are two program output examples that will help in understanding the scheduling algorithm and the timing calculations. The EMM distributes AV packets to five Decoder Boxes. The EMM transmits audio AV packets to Decoders 2 and 4, and 8-bit color video AV packets to Decoders 0, 1, and 3. Each line is printed after the transmission of an AV packet set and has the format:

```
[start time - end time] Type[ID]:
freed_mem, packets sent, FreeMem, evt/eat, lvt/lat
```

where evt/eat is the *EarliestVideoTime* or *EarliestAudioTime* and lvt/lat is the *LatestVideoTime* or *LatestAudioTime*. All the times are in seconds. Constants used are listed in Table 4.2.

The first example in Figure 4.8 shows how the EMM schedules the transmission of AV packets based on the latest video or audio transmission time of each Decoder Box.

Table 4.2: Constants Used in the Timing Calculation Examples

	Audio	**Video**
Total Memory (AV packets)	1024000	32768
Memory Transfer Rate (Hz)	22000	135000
LOWER_LIMIT (AV packets)	4500	4500
UPPER_LIMIT (AV packets)	1019500	28268

In line 3, the free audio memory for Decoder 4 is already below the UPPER_LIMIT so the lat is the current time plus the time it will take for Decoder 4 to empty the audio memory:

$$lat = 24.394784 + \frac{1024000 - 994267}{22200} = 25.734108$$

In line 4, the free video memory is more than the UPPER_LIMIT so the lvt is set to the current time. The same happens to all the video AV packets, until line 15. In line 16, the free video memory of Decoder 1 drops below the UPPER_LIMIT, so its new lvt is:

$$lvt = 25.017176 + \frac{32768 - 25952}{135000} = 25.067665$$

Notice also that the latest video transmission times need not and cannot be enforced. Hence they are merely acting as relative priority values for the LPQ.

```
. . .
    [start time- end time ] Type[id]  freed,  sent,
1.  [24.287980 - 24.338489] Audio[2]: 37656, 32400, FreeMem 1024000, eat 0.000000, lat 24.338489
2.  [24.340173 - 24.340202] Video[3]: 17511,   600, FreeMem   32768, evt 0.000000, lvt 24.340202
3.  [24.343734 - 24.394784] Audio[4]:  2667, 32400, FreeMem  994267, eat 0.000000, lat 25.734108
4.  [24.396582 - 24.446642] Video[0]: 22120, 16800, FreeMem   32768, evt 0.000000, lvt 24.446642
5.  [24.448378 - 24.448409] Video[1]: 22086,   600, FreeMem   32768, evt 0.000000, lvt 24.448409
6.  [24.451583 - 24.502308] Audio[2]:  3636, 32400, FreeMem  995236, eat 0.000000, lat 25.797984
7.  [24.504286 - 24.555888] Video[3]: 29117, 17400, FreeMem   32768, evt 0.000000, lvt 24.555888
8.  [24.557553 - 24.608986] Video[0]: 21916, 18000, FreeMem   32768, evt 0.000000, lvt 24.608986
9.  [24.610734 - 24.661459] Video[1]: 28761, 17400, FreeMem   32768, evt 0.000000, lvt 24.661459
10. [24.663468 - 24.713931] Video[3]: 21335, 17400, FreeMem   32768, evt 0.000000, lvt 24.713931
11. [24.715629 - 24.753087] Video[0]: 19453, 13200, FreeMem   32768, evt 0.000000, lvt 24.753087
12. [24.754716 - 24.805435] Video[1]: 19436, 16800, FreeMem   32768, evt 0.000000, lvt 24.805435
13. [24.807128 - 24.858501] Video[3]: 19516, 18000, FreeMem   32768, evt 0.000000, lvt 24.858501
14. [24.860243 - 24.911757] Video[1]: 14353, 18000, FreeMem   29121, evt 0.000000, lvt 24.911757
15. [24.913743 - 24.964073] Video[3]: 14252, 16800, FreeMem   30220, evt 0.000000, lvt 24.964073
16. [24.965775 - 25.017176] Video[1]: 14231, 17400, FreeMem   25952, evt 0.000000, lvt 25.067665
17. [25.018891 - 25.070209] Video[3]: 14328, 16800, FreeMem   27748, evt 0.000000, lvt 25.107394
18. [25.072005 - 25.122956] Video[1]: 14280, 17400, FreeMem   22832, evt 0.000000, lvt 25.196556
19. [25.124870 - 25.174025] Video[3]: 14015, 16800, FreeMem   24963, evt 0.000000, lvt 25.231840
20. [25.175688 - 25.218431] Video[1]: 12889, 14400, FreeMem   21321, evt 0.000000, lvt 25.303224
21. [25.220083 - 25.270386] Audio[4]: 19438, 31200, FreeMem  982505, eat 0.000000, lat 27.139530
22. [25.274241 - 25.324389] Audio[2]: 18250, 31200, FreeMem  982286, eat 0.000000, lat 27.203398
. . .
```

Figure 4.8: EMM scheduling - Program output example 1.

Since the effect of late AV packets is more unpleasant for audio than it is for color/video packets, the EMM gives higher priority to continuing audio AV packets. If the top entry of the LPQ indicates continuing audio data, with a lat less than the current time, while the current

transmission is not continuing audio packets, the EMM starts sending the audio packets. This can be seen in lines 5 and 6. In line 5, the EMM sends only 600 video AV packets to Decoder 1, and then stops to send audio AV packets to Decoder 2 (line 6).

The second example in Figure 4.9 emphasizes the most critical part of the algorithm, namely how the EMM schedules the transmissions without losing any AV packets.

```
. . .
    [start time- end time ] Type[id] freed,  sent ,
1.  [33.399798 - 33.450162] Video[3]: 1408,  17400, FreeMem   11238, evt  0.000000, lvt 34.489421
2.  [33.450941 - 33.499743] Video[1]: 3873,  16200, FreeMem    5205, evt 34.055299, lvt 34.985891
3.  [33.500595 - 33.550977] Video[0]: 3878,  16800, FreeMem    5285, evt 34.106533, lvt 35.031199
4.  [33.551805 - 33.582883] Video[3]: 1791,  10200, FreeMem    2829, evt 34.138439, lvt 35.245031
5.  [33.583697 - 33.633760] Audio[4]: 6471,  31200, FreeMem  973281, eat  0.000000, lat 35.918400
6.  [33.635139 - 33.685357] Audio[2]: 6356,  31200, FreeMem  972017, eat  0.000000, lat 36.026934
7.  [33.686075 - 33.736438] Audio[4]: 2279,  31200, FreeMem  944360, eat  0.000000, lat 37.323825
8.  [33.737195 - 33.787471] Audio[2]: 2266,  30000, FreeMem  944283, eat  0.000000, lat 37.378327
9.  [33.788182 - 33.838854] Audio[4]: 2273,  31200, FreeMem  915433, eat  0.000000, lat 38.729259
10. [33.839573 - 33.890973] Audio[2]: 2297,  31200, FreeMem  915380, eat  0.000000, lat 38.783766
11. [33.891759 - 33.941873] Audio[4]: 2287,  31200, FreeMem  886520, eat  0.000000, lat 40.134666
12. [33.942577 - 33.948332] Audio[2]: 1273,   3600, FreeMem  913053, eat  0.000000, lat 38.945945
13. [33.949031 - 33.991639] Audio[4]: 1104,  26400, FreeMem  861224, eat  0.000000, lat 41.323891
14. [34.055610 - 34.067631] Video[1]: 7666,   4200, FreeMem    8671, evt  0.000000, lvt 35.297038
15. [34.068399 - 34.089447] Video[1]:  294,   7200, FreeMem    1765, evt 34.645003, lvt 35.830410
16. [34.106802 - 34.119364] Video[0]: 7673,   4200, FreeMem    8758, evt  0.000000, lvt 35.342327
17. [34.120180 - 34.143167] Video[0]:  321,   7800, FreeMem    1279, evt 34.698723, lvt 35.920130
18. [34.144274 - 34.149316] Video[3]: 7646,   1800, FreeMem    8675, evt  0.000000, lvt 35.378427
19. [34.150159 - 34.171321] Video[3]:  297,   7200, FreeMem    1772, evt 34.726877, lvt 35.911765
20. [34.645306 - 34.646923] Video[1]: 7525,    600, FreeMem    8690, evt  0.000000, lvt 35.874923
21. [34.647731 - 34.668788] Video[1]:  295,   7200, FreeMem    1785, evt 35.224344, lvt 36.408269
. . .
```

Figure 4.9: EMM scheduling - Program output example 2.

At line 2, the evt for Decoder 1 is 34.055299, so the EMM schedules AV packets for other Decoders until line 14 where the start time is 34.055610. Note that at line 13 the EMM finishes sending audio packets to Decoders 2 and 4. Nevertheless, none of the other three Decoders can receive other packets, since the current time is less than their evt's. Hence, the

EMM has to wait from 33.991639 until 34.055610 when it can safely send AV packets to Decoder 1. Similarly, at line 15 neither Decoder 0 nor Decoder 3 can receive any packets, so the EMM keeps sending packets to Decoder 1. At the end of this transmission, however, the video memory of Decoder 1 is also full, so a new evt is calculated.

4.1.3 Ethernet Interface Module

The Ethernet Interface receives the appropriate AV packet, encapsulates it in an Ethernet frame, and transmits it using the FE NIC of the host computer. The UNIX OS accesses the raw Ethernet protocol using the same system calls used by the rest of the network protocols.

There are two protocols in UNIX that access the raw Ethernet: snoop and drain. The snoop protocol captures link-layer packets which match a bit-field filter and transports them to that filter's socket. Snoop prepends a header containing packet state, sequence, and a timestamp. The drain protocol receives input packets with network-layer type codes selected from among encapsulations not implemented by the kernel. Snoop enables promiscuous reception to capture packets not destined for this host if an active snoop filter matches such packets, while drain receives non-promiscuously (in the absence of a snooper on the same network interface), filtering packets received by the hardware according to their type codes. Both protocols transmit packets with a link-layer header fetched from the beginning of the user's write buffer. They ignore any destination address supplied to *sendto* or *connect*. However, connecting restricts input to packets originating from the connected address. Therefore, *write* is the likeliest system call to use for raw output.

Raw family address format differs between local and foreign usage. A local AF_RAW *sockaddr* contains a port, identifying the socket to which this address is bound, and the zero-padded name of a network interface. A foreign AF_RAW *sockaddr* contains an opaque link-layer destination address.

From <net/raw.h>:

```
#define RAW_MAXADDRLEN  (sizeof ((struct sockaddr *) 0)->sa_data)
#define RAW_IFNAMSIZ    (RAW_MAXADDRLEN - sizeof(u_short))
struct sockaddr_raw {
        u_short         sr_family;                      /* AF_RAW */
        union {
            struct {
                u_short srl_port;                       /* socket id */
                char    srl_ifname[RAW_IFNAMSIZ];       /* interface name */
            } sru_local;
            struct {
                char    srf_addr[RAW_MAXADDRLEN];       /* raw address */
            } sru_foreign;
        } sr_u;
};
#define sr_port         sr_u.sru_local.srl_port
#define sr_ifname       sr_u.sru_local.srl_ifname
#define sr_addr         sr_u.sru_foreign.srf_addr
```

From <netinet/if_ether.h>

```
/* Structure of a 10Mb/s Ethernet header */
struct  ether_header {
        u_char  ether_dhost[6];
        u_char  ether_shost[6];
        u_short ether_type;
};
```

Figure 4.10: Raw Ethernet data structures.

The arguments used for each system call are described below. For more details on socket system calls see [Ste90]. On initialization the EIM opens a raw Ethernet socket using the *socket()* system call, with the following arguments.

```
socket(AF_RAW, SOCK_RAW, RAWPROTO_SNOOP)
```

It then associates the socket to the desired network interface adapter. This is done with the *bind()* system call. The first argument is the socket id returned by *socket()*, while the second argument is a pointer to the *sockaddr_raw* structure shown in Figure 4.10. The structure's fields

are filled with AF_RAW for *sr_family*, '0' for *sr_port*, and the desired interface adapter's name for *sr_ifname*. At this point the EIM is ready to transmit the raw Ethernet frames.

Whenever the EMM calls the EIM for transmission, it sends the AV packet and the 6-byte Ethernet destination address. To accommodate multiple network interface adapters the EIM also receives the interface name as an argument. From the network interface adapter's name the EIM extracts the Ethernet source address. The destination address is in the format XX:XX:XX:XX:XX:XX and it is converted to a 6-byte value using the *ether_hostton()* system call. The EMM then constructs the raw Ethernet frame as explained in Section 2.2.2.1. It finally transmits the frame using *write()* on the raw Ethernet socket.

4.1.4 Summary

The Browser Server Network Interface consists of two software modules and one or more network interface adapters. The EMM receives arbitrary AV packets from the active browsers using multiple UNIX domain sockets. It then schedules them for transmission so that no packet is lost and, if possible, the packets do not arrive late at the appropriate Decoder Box. The EIM receives the next AV packet, encapsulates it into an Ethernet frame and transmits it to the wire using the desired network interface adapter.

4.2 Decoder Box Network Interface

As explained in Section 3.2.2 the Decoder Box should be entirely hardware based. The complete design is described in [Haw97] and the full circuit schematic is shown in Appendix C.2. The Decoder Box consists of the network interface module, a custom controller module, a main memory module, a video module, and an audio module as shown in Figure 4.11. The custom controller module is a Field Programmable Gate Array (FPGA) that processes the AV packets. The main memory acts as a buffer for the 24-bit or 8-bit video AV packets. It stores both the pixels and the screen address of each pixel. The video module has its own memory to buffer the

pixels and a National Television Standards Committee (NTSC) encoder to convert the RGB pixels to NTSC signals. The audio module has also memory to buffer the audio samples, and a Digital-to-Analog Converter (DAC).

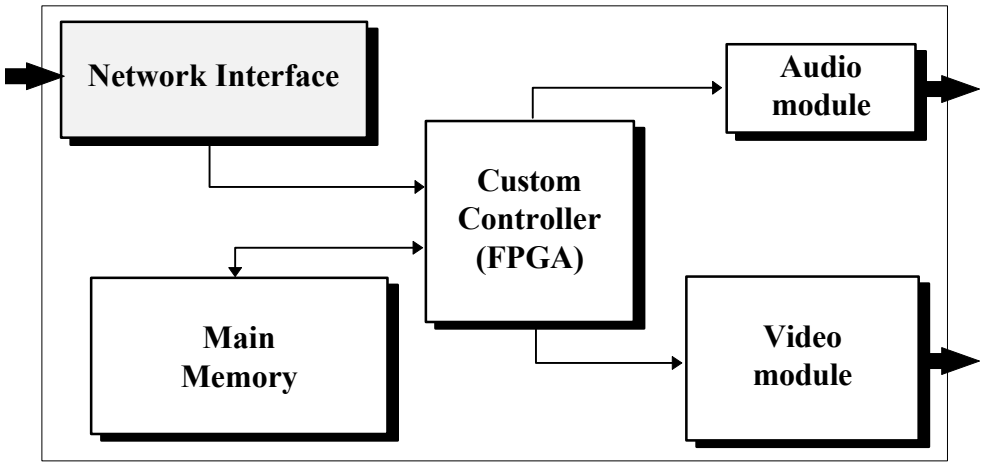

Figure 4.11: IVDS/WWW Decoder Box block diagram.

The Decoder Box Network Interface is a physical layer chipset which is responsible for receiving the Fast Ethernet frames from the CAT-5 UTP cable. It is described in detail in the following subsections.

4.2.1 Physical Layer Chip Set Options

Due to the popularity of the FE standard, the semiconductor industry responded quickly to deliver the chips required to implement the physical layer of this network. Figure 4.12 lists some of the better known Fast Ethernet physical layer chipsets available at the time of this research.

Pulse Engineering, Inc., Cypress Semiconductor, and Quality Semiconductor, Inc. offer some interesting options by combining multiple modules into one chip. This makes the design of the network interface both easy and reliable, especially considering the high speed requirements of

Fast Ethernet. Due to the high cost of multiple-module chips, however, the Decoder Box Network Interface was designed using single-module chips.

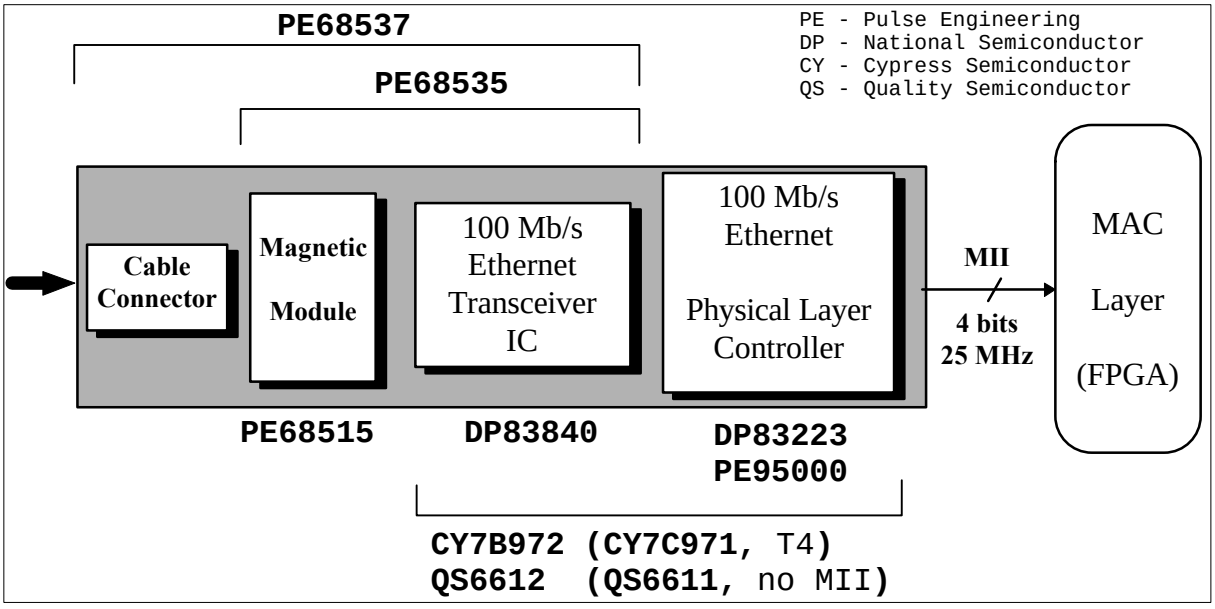

Figure 4.12: Fast Ethernet Physical Layer chipsets.

The most important chips selected, namely the Physical Layer device (PHY) and the 100-Mbps Transceiver device, were the DP83840A and DP83223A, respectively, both by National Semiconductor. Their low cost and reliable performance has led them to get most of the Fast Ethernet physical layer market share [NSC97b]. They are used in most of the 10/100-Mbps NICs and Repeaters. Along with PE-68515, by Pulse Engineering, Inc., as the magnetics module device and an RJ-45 connector, they form the Decoder Box Network Interface.

4.2.2 Decoder Box Network Interface Chipset

4.2.2.1 Connector

An RJ-45 8-pin connector is used to accommodate CAT-5 UTP cable. The RJ-45 connector is common to both 10BASE-T and 100BASE-TX operation.

4.2.2.2 Magnetic module

The PE-68515 by Pulse Engineering, Inc. is a magnetics transformer module that has been specifically designed to interface with the National Semiconductor 10/100 Mbps chip-set [PEI97]. It implements a 10/100-Mbps data-link over data grade UTP-5 UTP cable.

4.2.2.3 100-Mbps Twisted Pair Transceiver

The DP83223A is an integrated circuit functioning at the Physical Medium Dependent (PMD) sublayer. It is capable of driving and receiving either binary or Multi-Level Transmission (three-level) (MLT-3) encoded data streams. It is designed to interface directly with standard-compliant 100BASE-TX, FDDI, and STS-3C ATM chip-sets allowing low cost data links over copper based media. More information about this device can be found in its datasheet [NSC97a]. Since the Decoder Box is a receive-only device, we are using only the reception capabilities of the DP83223A device. It receives the binary or MLT-3 encoded data streams and passes the Non-return to Zero Inverted (NRZI) binary data stream to the PHY sublayer. MLT-3 data streams are carried on UTP cables, whereas STP cables carry binary data streams.

4.2.2.4 10/100-Mbps Ethernet Physical Layer Controller

DP83840A is the chip that performs the main physical layer functions. Again we only need the receive functionality. When operating at 100 Mbps it receives the binary NRZI data stream at 125 Mbps. It passes the data to the MAC layer in nibbles (4-bit words) at 25 MHz for 100-Mbps operation or at 2.5 MHz for 10-Mbps operation. The DP83840A datasheet [NSC97b] provides detailed information about its design and operation.

4.2.2.4.1 100BASE-TX Receiver Functionality

The 100BASE-TX receiver, consists of several functional blocks required to receive and condition the 125-Mbps data stream as specified by the IEEE 802.3u standard. These functions are provided by the DP93840A.

- Clock Recovery Module (CRM): This module accepts a 125-Mbps scrambled NRZI data stream from the DP83223. The CRM locks onto the 125-Mbps data stream and extracts a 125-MHz reference clock which is used by the synchronous receive operations.
- NRZI to NRZ decoder block: The received data stream is NRZI encoded to comply with the TP-PMD standard for 100BASE-TX transmission over CAT-5 UTP cable. The data stream needs to be decoded back to NRZ format before reaching the descrambler block.
- Descrambler block: To control the radiated emissions at the media connector and on the twisted pair cable the data is scrambled. By scrambling the data, the total energy launched onto the cable is randomly distributed over a wide frequency range. Without the scrambler, energy levels at the PMD and at the cable would peak at frequencies related to repeating 5B sequences. This is especially true during continuous transmissions of IDLEs, where an IDLE in 5B NRZ is equal to '11111' [NSC97b]. To reverse the data scrambling process the Descrambler has to generate an identical data scrambling sequence (N) to recover the original unscrambled data (UD) from the scrambled data (SD) as represented by the following equations.

$$SD = UD \oplus N, \quad UD = SD \oplus N$$

- Code-group alignment block: This block receives unaligned 5-bit data from the descrambler and converts it into 5B code-group data. Code-group alignment occurs after the J/K code-group pair (11000 10001) is detected.
- Code-group decoder: The code-group decoder functions as a look up table that translates incoming 5B code-groups into 4B nibbles. The code-group decoder first detects the J/K code-group pair preceded by IDLE code-groups and replaces the J/K with the MAC preamble. Specifically, the J/K 10-bit code-group pair is replaced by the nibble pair (0101 0101). All subsequent 5B code-groups are converted to the corresponding 4B nibbles for the duration of the entire packet. This conversion ceases upon the detection of the T/R code-group pair denoting the End of Stream Delimiter (ESD) or with the reception of a minimum of two IDLE code-groups.

- 100 Mbps Receive State Machine: The DP83840A implements the 100BASE-TX receive state machine diagram as given in ANSI/IEEE Standard 802.3u/ D5, Clause 24 [IEEE95].

In addition to the functional blocks the following monitors and detection blocks complete the receiver part of the DP83840A.

- Collision Detect block
- Carrier Sense block
- Far End Fault Indication block
- Link Integrity Monitor block
- Carrier Integrity Monitor block

4.2.2.4.2 10BASE-T Receiver Functionality

The 10BASE-T receiver operation is independent of the 100BASE-TX receiver operation. The Manchester decoder receives Manchester encoded data from the transceiver, converts it to NRZ data and recovers clock pulses for synchronous data transfer to the MAC layer controller.

4.2.2.4.3 Auto-Negotiation Protocol

The Auto-Negotiation function provides a mechanism for exchanging configuration information between two ends of a link segment and automatically selecting the highest performance mode of operation supported by both devices. Fast Link Pulse (FLP) bursts provide the signaling used to communicate Auto-Negotiation ability information between two devices at each end of a link segment. For further detail regarding Auto-Negotiation, refer to clause 28 of the IEEE 802.3u specification [IEEE95]. The DP83840A supports four different Ethernet protocols (10 Mbps Half Duplex, 10 Mbps Full Duplex, 100 Mbps Half Duplex, and 100 Mbps Full Duplex), so the inclusion of Auto-Negotiation ensures that the highest performance protocol will be selected based on the ability of the link partner.

4.2.3 Network Interface Circuit Design

The Network Interface was designed to operate at a 10 Mbps or a 100 Mbps data rate. The circuit schematic with all the functional connections for the Decoder Box Network Interface is shown in Figure 4.11.

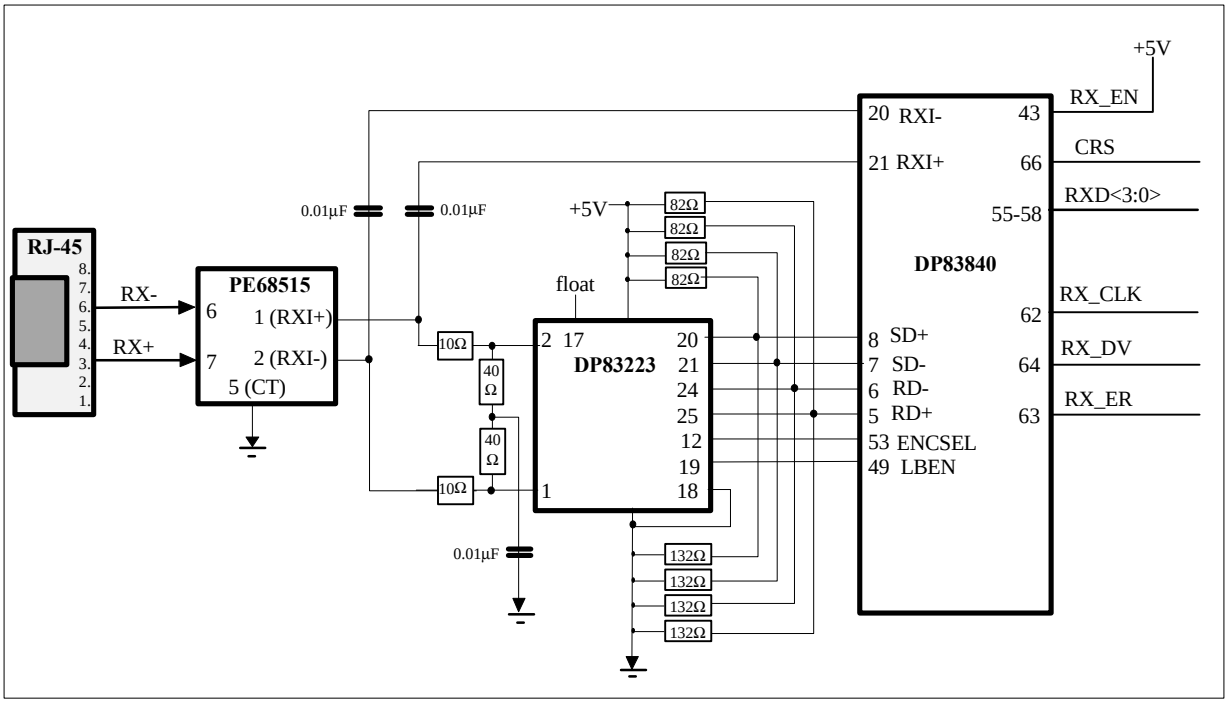

Figure 4.13: Decoder Box Network Interface circuit.

The complexity of the circuit design is reduced because we are only using the receiver functionality of the chip-set. One major part that is only required for the 10/100-Mbps transmitter is a clock generator. For the receiver the clock is recovered from the received signal. Appendix C.1 describes all the pin connections of the DP83840A.

4.2.3.1 Medium Independent Interface

The Network Interface communicates with the rest of the Decoder Box through the Medium Independent Interface (MII). The MII is part of the DP83840A functionality.

Descriptions of the MII pins of the DP83840 chip used in the Decoder Box Network Interface are presented in Table 4.3.

Table 4.3: DP83840A MII Reception Pin Descriptions

Pin Name	Pin #	Description
CRS	66	**Carrier Sense:** This pin is asserted high to indicate the presence of carrier due to receive or transmit activities in 10BASE-T or 100BASE-TX Half Duplex modes.
RX_CLK	62	**Receive Clock:** Provides the recovered receive clock for different modes of operation: • 25 MHz nibble clock in 100 Mb/s mode • 2.5 MHz nibble clock in 10 Mb/s nibble mode • 10 MHz receive clock in 10 Mb/s serial mode
RX_ER	63	**Receive Error:** Asserted high to indicate that an invalid symbol has been detected within a received packet in 100 Mb/s mode.
RX_DV	64	**Receive Data Valid:** Asserted high to indicate that valid data is present on RXD[3:0].
RXD[3] RXD[2] RXD[1] RXD[0]	55 56 57 58	**Receive Data:** Nibble wide receive data (synchronous to RX_CLK, 25 MHz for 100BASE-X mode, 2.5 MHz for 10BASE-T nibble mode). Data is driven on the falling edge of RX_CLK.
RX_EN	43	**Receive Enable:** Active high enable for receive signals RXD[3:0], RX_CLK, RX_DV and RX_ER. A low on this input tri-states these output pins. For normal operation in a node application this pin should be pulled high.

4.2.4 Summary

The Decoder Box Network Interface consists of four semiconductor chips and operates using either the 10-Mbps or the 100-Mbps Ethernet protocol. It receives either Manchester encoded signals or MLT-3 encoded signals for 10-Mbps operation or 100-Mbps operation, respectively. Both types of signals are carried on CAT-5 UTP wires. The Network Interface uses Fast Ethernet's MII to pass the Ethernet frame to the MAC layer controller with additional timing and control information to process the frame.

4.3 Network Medium

4.3.1 Asymmetric Network - Internet over Cable

As seen in Figure 3.2, the Fast Ethernet packets are inserted into the cable network with a downstream manager. Depending on the expected bandwidth, the downstream manager can be as simple as a cable modem or as complex as Hybrid Networks' Downstream Router, CMD-2000 [Hyb97]. The cable network company can allocate a number of TV channels for data services. Table 4.4 lists some techniques to insert digital information into the coaxial cable of a cable network [Bel91] [CaTV97].

Table 4.4: Digital Modulation Techniques on Coaxial Cable

Abbreviation	Modulation Technique	Bandwidth	Data Rate[1]
FSK	Frequency Shift Keying	435 KHz	2.016 Mbps
BPSK (2-PSK)	2-Phase Shift Keying	6 MHz	2.5 Mbps
QPSK (4-PSK)	Quaternary Phase Shift Keying	6 MHz	10 Mbps
64QAM	64-Quadrature Amplitude Modulation	6 MHz	30 Mbps
256QAM	256-Quadrature Amplitude Modulation	6 MHz	54 Mbps

Each TV channel spans a frequency of 6 MHz in the US and 8 MHz in Europe. The most popular technique is to divide the 6-MHz TV channel into three 2-MHz sub-channels. Using 64 Quadrature Amplitude Modulation (QAM) each sub-channel can accommodate a 10-Mbps datastream for a total of 30 Mbps.

At the other end of the cable network, as seen in Figure 3.1, the signals are demodulated using a cable modem. The IVDS/WWW Decoder Box receives the Ethernet packets from the cable modem at a data rate of either 10 Mbps or 100 Mbps, depending on the type of Ethernet interface provided by the cable modem. Almost all of today's cable modems have a 10BASE-T interface, but it is expected that the next generation will likely be equipped with a 100BASE-TX interface.

[1] Data rates refer to the maximum output data rate given the frequency bandwidth on the left.

4.4 Summary

This chapter described the design of the interface between the IVDS/WWW Browser Server and the IVDS/WWW Decoder Box. The Browser Server Network Interface is a software package consisting of the Ethernet Management and the Ethernet Interface modules. It uses UNIX domain sockets to receive the AV packets from the rest of the browser server software. Using the fact that the Decoder Boxes have fixed memory sizes and transfer rates, it schedules the AV packets for transmission so that no packet is lost and, if possible, the packets get to the Decoder Box in time for continuous play of audio or video. The packets are transmitted using the Raw Ethernet protocol. At the Decoder Box, a hardware-based network interface has been designed to receive the AV packets. The Fast Ethernet chip-set receives both 10 Mbps and 100 Mbps datastreams and passes the Ethernet frames to the Custom Controller, implemented by an FPGA, using the MII interface.

Chapter 5 - TESTING AND RESULTS

This chapter describes the testing procedures for the network interface presented in Chapter 4. It also presents the performance results for the Browser Server Network Interface. The Decoder Box has not been manufactured, so no testing could be performed on the Decoder Box Network Interface.

5.1 Testing Intranet

Testing of the Browser Server Network Interface software was performed on a dedicated "intranet" running on 10-Mbps Ethernet as shown in Figure 5.1. The software was running on a Silicon Graphics, Inc. (SGI) Indy computer with the IRIX 5.3 operating system and an R5000 central processing unit running at 150 MHz. For the purposes of the IVDS tests it was also equipped with a second Ethernet NIC, the SGI E++ GIO Ethernet adapter.

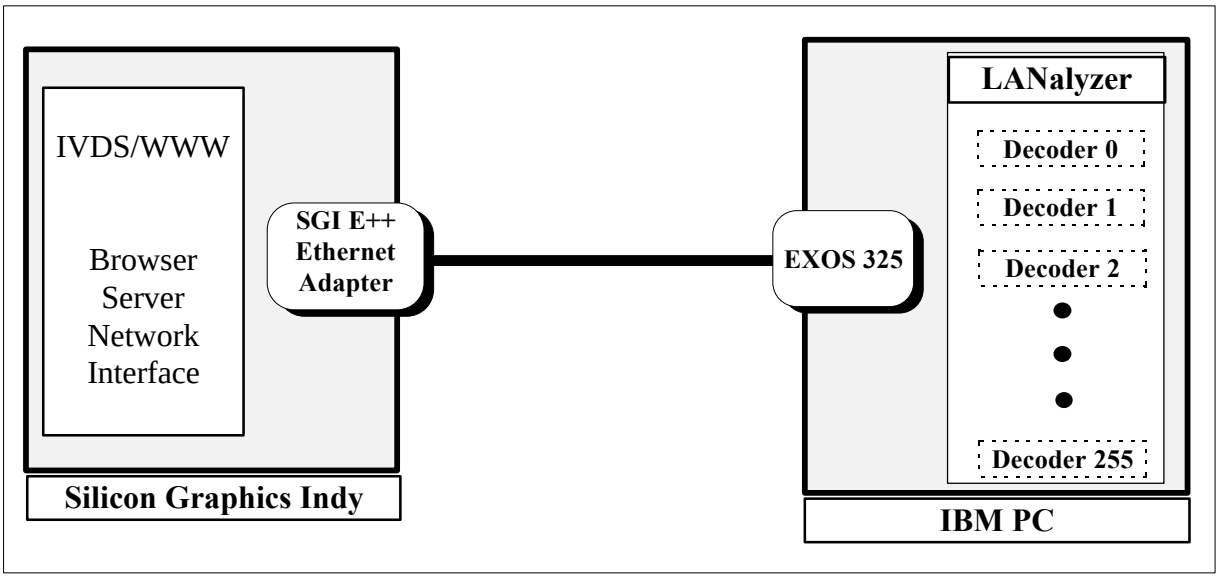

Figure 5.1: Browser Server Network Interface testing "intranet" set-up.

The Ethernet frames were received by the LANalyzer, a computer equipped with special hardware and software able to simulate multiple Ethernet nodes.

5.1.1 LANalyzer

The LANalyzer EX 5000 series network analyzer by Excelan, Inc. consists of a software package and an EXOS 325 board. The LANalyzer software is designed for use on an 80x86-based computer that is running DOS version 3.0 or later. The EXOS 325 board is compatible with Ethernet version 1.0, Ethernet version 2.0, and IEEE 802.3 transceivers [Exc88]. The LANalyzer system is able to monitor, debug, and analyze Ethernet LANs. In normal operation the LANalyzer would be connected to a LAN and receive all packets transmitted from the rest of the Ethernet nodes. The user can see each packet individually and check the headers and data of all protocol layers. More importantly, the LANalyzer can present statistics such as network utilization, number of packets per Ethernet node, average packet size, etc. In our network setup the Browser Server, i.e. the SGI Indy computer, transmits all packets to a number of virtual Decoder Boxes, designated by the destination Ethernet address.

5.1.2 Browser Threads Simulation

A simple program to simulate the browser threads generating the four different types of IVDS/WWW data was also developed. From the command line, the user can specify the number of active browser threads, the number of AV packets, the size of each AV packet, and the type of data in the AV packets. For each browser thread, the browser simulation program spawns a new process that constructs the requested number of AV packets creating the headers based on the data-type. The AV packets are stored in an array for each browser thread. The browser thread process makes a request to the known master UNIX domain socket of the EMM. When the EMM grants a new dedicated UNIX domain socket to the browser thread process, the latter starts sending the AV packets to the EMM through the new socket.

5.2 Tests and Results

One strong feature of the LANalyzer is its high quality timer. Each packet received by the LANalyzer is time-stamped with μsec accuracy. The Browser Server Network Interface can estimate the packet arrival time for each Decoder Box. By checking this with the actual inter-packet arrival times at the LANalyzer we can assess the correctness of the error prevention algorithm. All of the timings at the Browser Server Network Interface software were performed using the `gettimeofday()` function which has a precision of 1 μsec.

5.2.1 Screen Painting Times

The first test was to determine the average time required to paint a standard 640-by-480 pixel Web page with no video or audio. Tests were performed for both 8-bit and 24-bit color serving a different number of Decoder Boxes at the same time. Table 5.1 shows the results from the program for 8-bit and 24-bit color whereas Figures 5.2 and 5.3 give graphical presentations of these results. The times include the UNIX domain socket data transfer, AV packet scheduling, Ethernet frame construction, and Ethernet frame transmissions.

Table 5.1: Screen Painting Times for 8-bit and 24-bit Color Data

Active Browsers	8-bit Color Times (sec)	24-bit Color Times (sec)
1	5.11	11.43
2	8.24	21.03
4	14.40	40.12
8	27.41	77.30
10	33.82	97.13
12	40.20	116.77
16	55.06	158.85
20	64.69	
24	77.47	

The total time for packet distribution increases linearly as the number of active browsers increases.

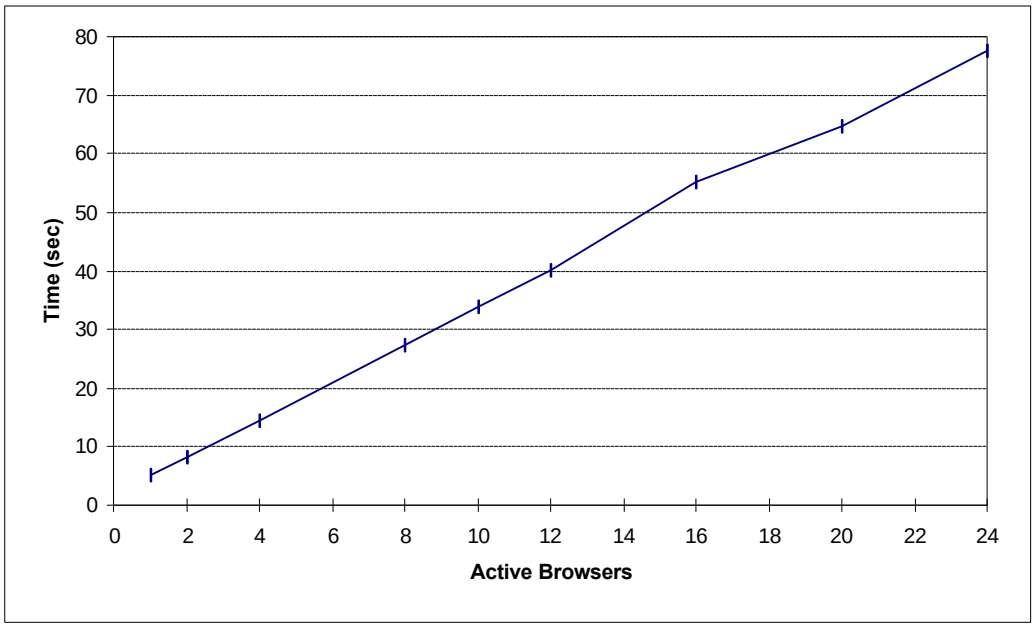

Figure 5.2: Screen painting times for 8-bit color data.

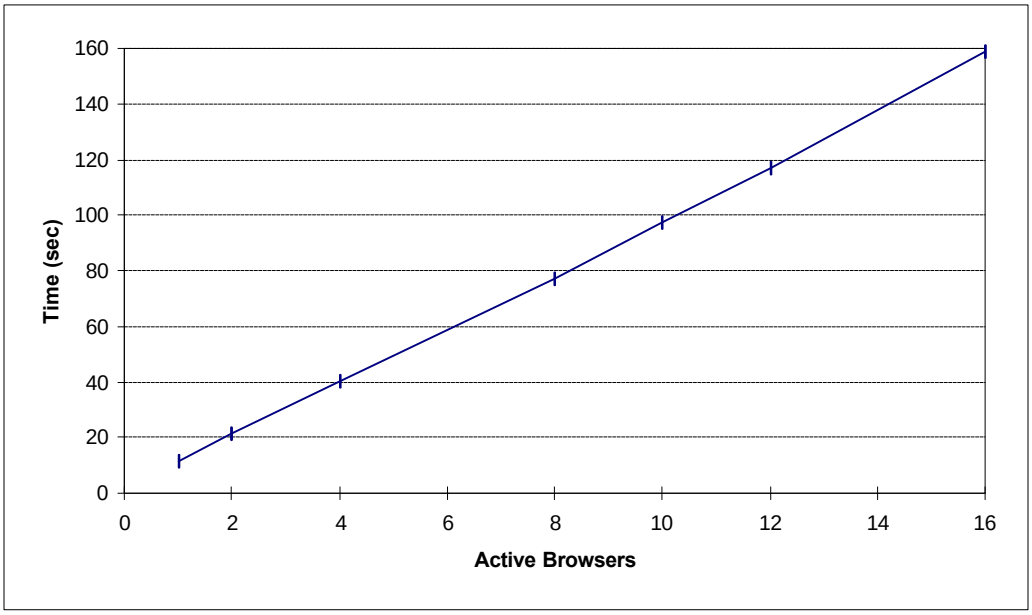

Figure 5.3: Screen painting times for 24-bit color data.

As expected the times for 24-bit color data are three times greater than those for 8-bit color data. Based on this we can expect the time for distributing the AV packets necessary to

display images of 640-by-480 pixels, 8 bits per pixel, at 256 active browsers would be 880 seconds or about 14.6 minutes. For 24-bit color data, it would take about 42 minutes. These times are definitely prohibitive, but this conservative estimation assumes that data for all 256 users will make a request at the same time. Also, performance is limited by the workstation's performance and the 10-Mbps Ethernet used for testing.

5.2.2 Program Statistics

For the next test the following statistics were collected for the cases of 8-bit video, 24-bit video, and audio.

- Total Time: the total time required to receive the AV packets from the browser threads and transmit them to the appropriate Decoder Box.
- UNIX Time: the time to transfer the AV packets from the browser threads to the Network Interface through the UNIX domain sockets (per browser).
- Ether Time: the time to schedule the AV packets, construct and transmit the Ethernet frames (per browser).
- Mean Error: the mean error of the late packets, where error is the actual transmission time minus the latest acceptable video/audio transmission time for each Decoder Box.
- Std Dev Error: the standard deviation of the late packet errors, as above.
- Pcntg Error: the percentage of the number of the late packets with respect to the total number of transmitted packets.

5.2.2.1 Video Statistics

In both types of video the Network Interface is sending enough AV packets to paint a 640-by-480 pixel screen. In order for the NTSC screen to be refreshed 30 times per second, the video memory transfer rate was 13.5 MHz as required by the NTSC encoder. Since the only transmitting node is the Browser Server, and hence there can be no collisions in the network, each AV packet was constructed using the maximum possible Ethernet frame data field size. So, each 8-bit video AV packet carried 1491 pixels, while each 24-bit video AV packet carried 497 pixels.

A 640-by-480 pixel, 8 bits per pixel video image requires 207 AV packets, while a 640-by-480 pixel, 24 bits per pixel video image requires 619 AV packets. Table 5.2 presents a summary of the statistics for 8-bit color video data. Figure 5.4 displays graphs of the mean errors and the standard deviation of these errors versus the number of active browsers.

Table 5.2: Statistics for 8-bit Color Video Data

Active Browsers	Total Time	Unix Time	Ether Time	Mean Error	Std Dev Error	Pcntg Error
1	5.32	2.916	0.417	0.070	0.008	97.58
2	8.10	2.748	0.434	0.137	0.022	97.58
4	14.52	2.783	0.443	0.292	0.038	97.58
8	29.46	3.119	0.463	0.652	0.091	97.58
10	33.28	2.834	0.498	0.744	0.106	97.58
12	37.91	2.741	0.522	0.854	0.137	97.58
16	52.26	2.908	0.531	1.200	0.192	97.58
20	64.49	2.856	0.499	1.495	0.234	97.58
24	77.66	2.869	0.487	1.814	0.267	97.58

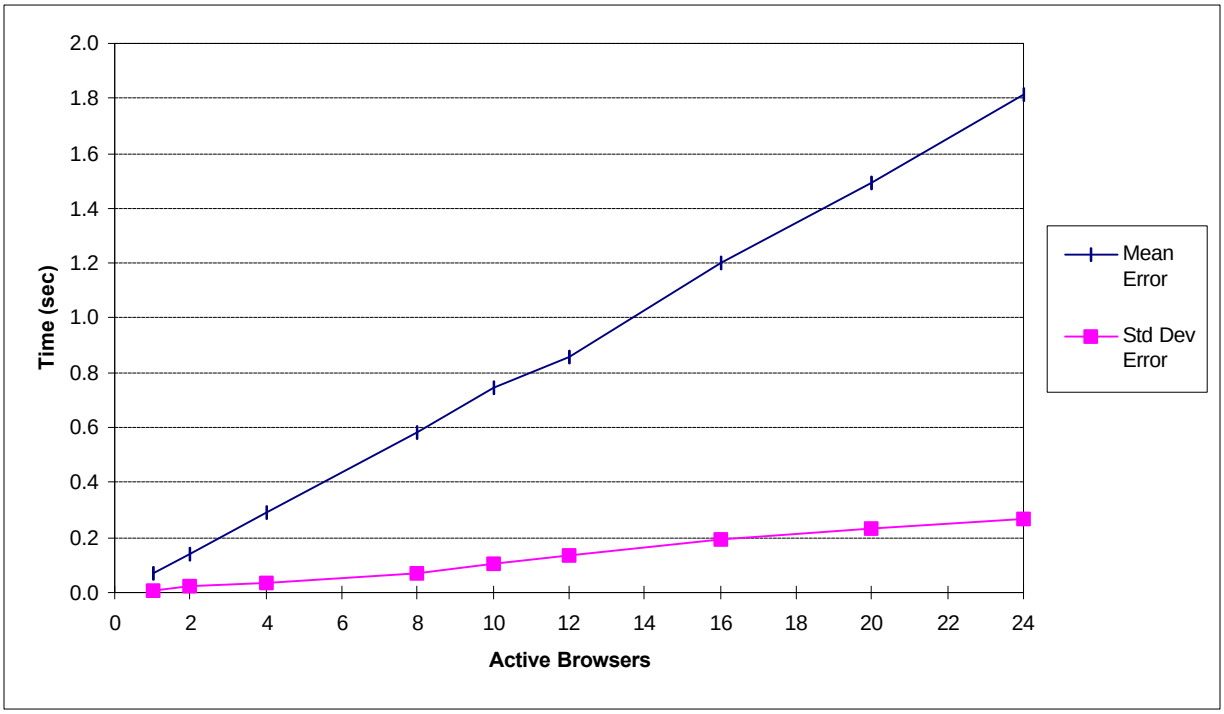

Figure 5.4: Error statistics for 8-bit color video data.

A summary of the statistics for 24-bit color video data is presented in Table 5.3. Figure 5.5 displays graphs of the mean errors and the standard deviation of the errors versus the number of active browsers.

Table 5.3: Statistics for 24-bit Color Video Data

Active Browsers	Total Time	Unix Time	Ether Time	Mean Error	Std Dev Error	Pcntg Error
1	11.69	8.509	1.181	0.346	0.330	96.12
2	20.43	8.074	1.177	0.723	0.622	95.99
4	41.10	8.620	1.224	1.560	1.309	95.90
8	77.13	8.289	1.331	3.053	2.503	95.87
10	100.19	8.749	1.315	4.008	3.301	95.93
12	115.02	8.372	1.408	4.559	3.697	95.83
16	156.33	8.599	1.301	6.407	5.204	95.86

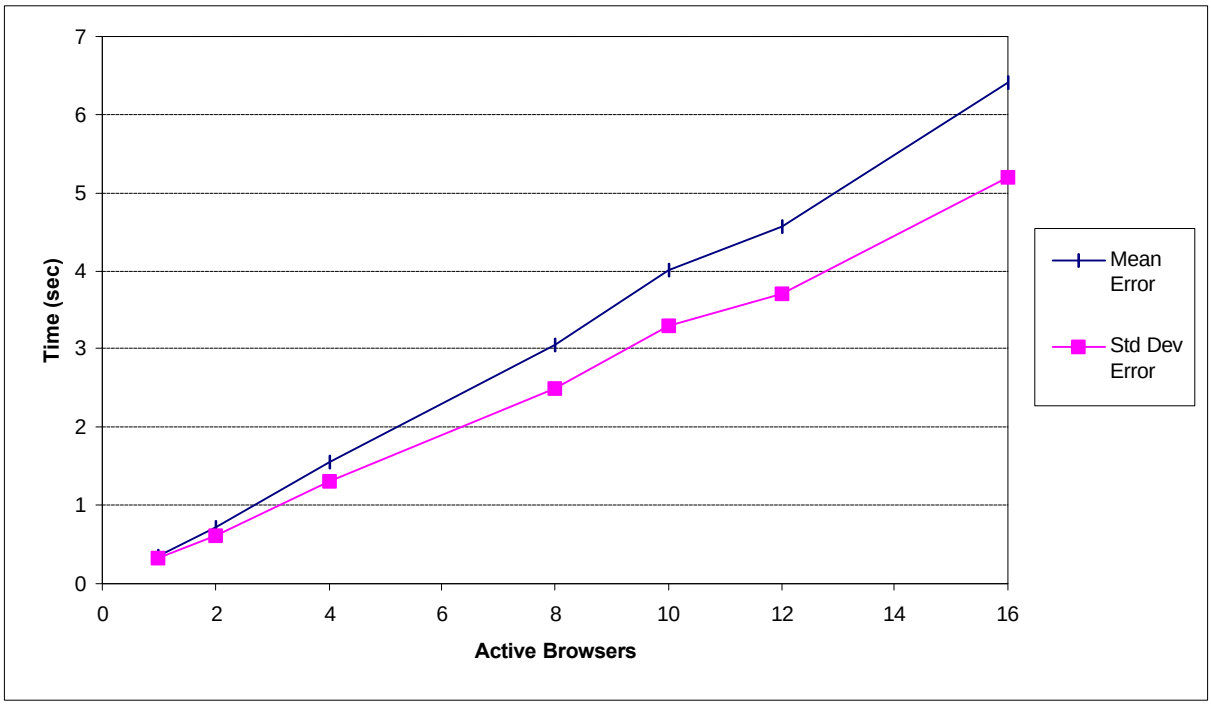

Figure 5.5: Error statistics for 24-bit color video data.

It is obvious from the percentages of late packets (Pcntg Error) that both 8-bit video and 24-bit video will not perform well in the test system. As expected, the UNIX transfer time is the most time consuming operation of the EMM, since it involves copying the AV packet. As with

the total time, the errors and standard deviation of errors increase linearly with the number of active browsers.

5.2.2.2 Audio Statistics

To test the audio performance of the Network Interface software module, a 447 KB audio sample was used. Each audio AV packet contained 745 16-bit audio samples, hence the Network Interface was sending 600 AV packets to each Decoder Box. The audio memory transfer rate was 22.2 KHz. Table 5.4 shows results and Figure 5.6 displays a graph of the mean errors and standard deviations of these errors versus the number of active browsers.

Table 5.4: Statistics for Audio Data

Active Browsers	Total Time	Unix Time	Ether Time	Mean Error	Std Dev Error	Pcntg Error
1	10.96	7.815	1.139	0.419	0.218	17.67
2	21.14	8.490	1.157	0.463	0.752	5.90
4	40.78	8.581	1.187	1.083	1.454	6.17
8	76.27	8.212	1.351	1.670	2.519	6.89
10	95.13	8.236	1.237	1.648	2.941	8.62
12	112.23	8.156	1.270	1.206	2.907	13.19

Like the total video data distribution times, the total audio video distribution times increase linearly with the number of active browsers. But unlike video data distribution, the mean errors and standard deviations of errors for audio data do not increase linearly with the number of active browsers. Instead their values start decreasing for more than 10 active browsers. This can be attributed to the fact that the error prevention algorithm will immediately stop sending data to a Decoder Box if the top entry of the LPQ contains an entry with continuing audio data type.

Figure 5.7 presents the total times versus the number of active browsers. Another interesting point for the audio data is how the percentage of late packets is affected by the number of active browsers. This is presented in Figure 5.8.

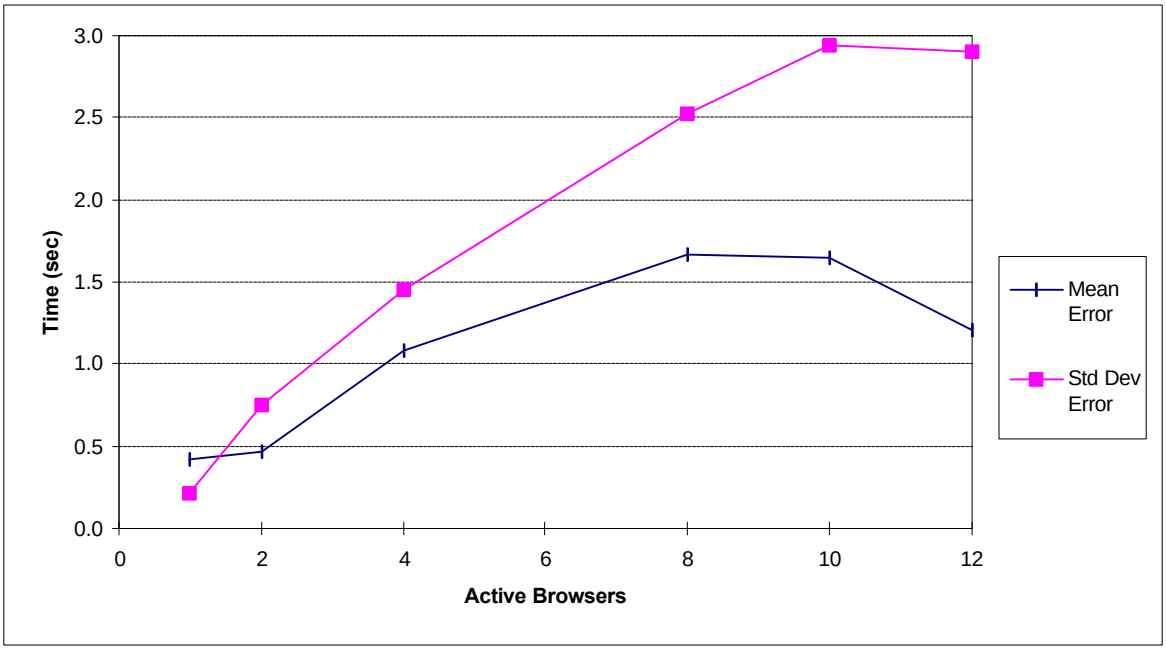

Figure 5.6: Error statistics for audio data.

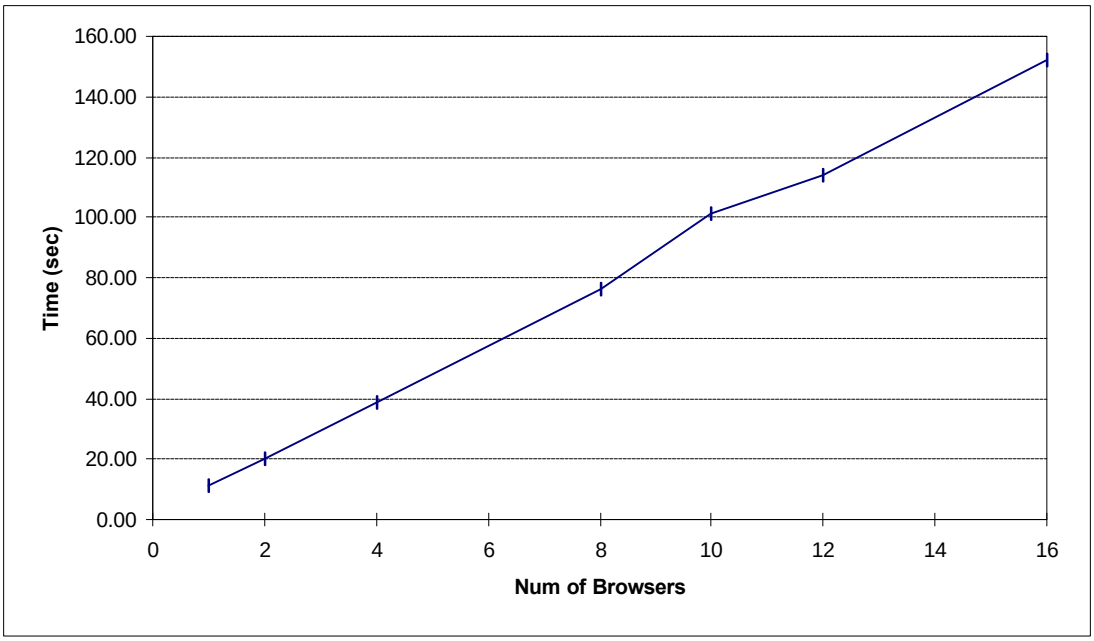

Figure 5.7: Audio transmission times.

The percentage of late audio packets is very low compared with that of late video packets because the audio memory is bigger and the audio transfer rate is much slower than for video. Nevertheless, it also becomes prohibitive when the number of active browsers is more than 10.

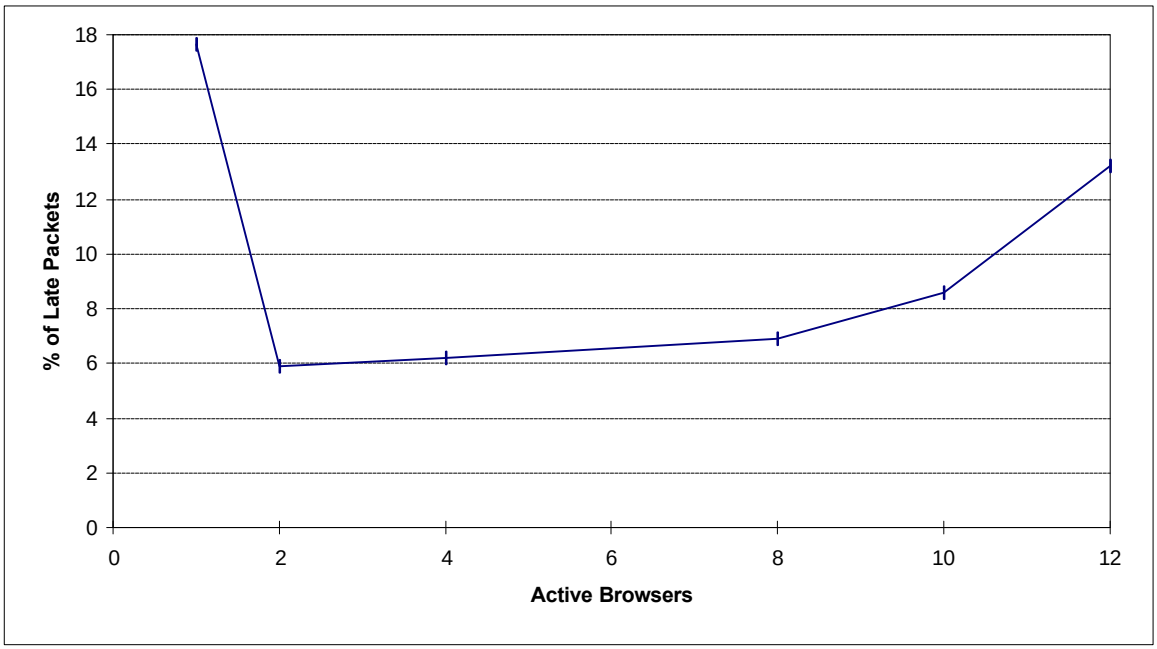

Figure 5.8: Percentage of late audio packets.

5.2.3 LANalyzer Results

As discussed in Section 5.1.1, the LANalyzer keeps a time-stamp for each frame received. From this we can get the inter-packet arrival time and the relative arrival time for all received packets. Table 5.5 shows a comparison of the LANalyzer's relative arrival times versus the program's Ethernet transmission times. Figure 5.9 presents the difference between the two measurements. Unfortunately, the LANalyzer's memory capabilities limits the tests to at most 8 browsers.

Table 5.5: LANalyzer vs. Software Times

Active Browsers	Software Times (sec)	LANalyzer Times (sec)
1	5.11	3.19
2	8.24	6.54
4	14.40	13.32
8	27.41	27.03

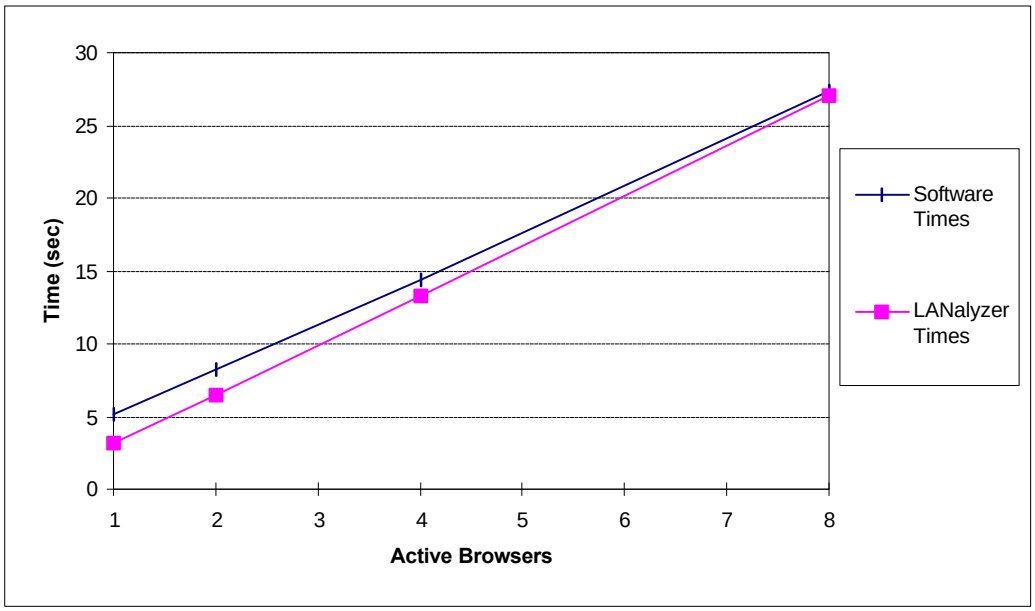

Figure 5.9: LANalyzer vs. program measurements.

5.2.4 Scheduling vs. No Scheduling Times

To see the effect of the error prevention algorithm on the overall performance of the Browser Server Network Interface I ran the tests for 8-bit video, 24-bit video, and audio data bypassing the AV packet scheduling. Each AV packet is transmitted by the EIM as soon as it is received by the EMM through the UNIX domain socket. Note that for the no scheduling case there is no guarantee that the packets will be neither lost nor arrive late. Table 5.6 summarizes the results and the difference between scheduled and non-scheduled transmissions for 8-bit video and Figure 5.10 displays a graph of these results.

Table 5.6: 8-bit Video Scheduling and No Scheduling Times

Active Browsers	8-bit Times Scheduling	8-bit Times No-Scheduling	8-bit Difference
1	5.112	5.282	-0.171
2	8.239	8.163	0.075
4	14.396	14.783	-0.387
8	27.415	27.071	0.343
12	40.201	40.013	0.188
16	55.059	51.109	3.950

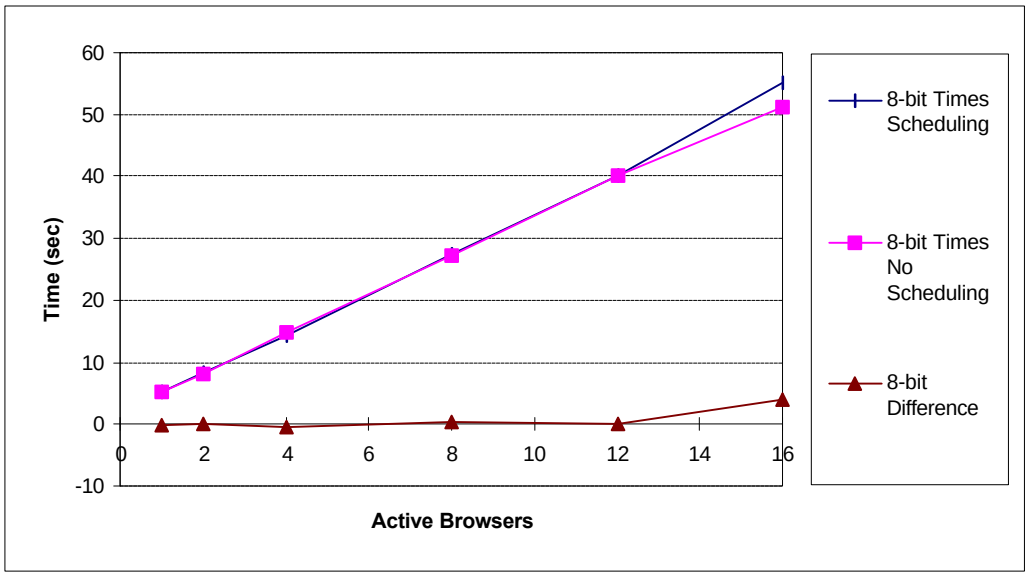

Figure 5.10: 8-bit video scheduling and no scheduling times.

Table 5.7 shows the performance difference for scheduled and unscheduled 24-bit video data. A graphical representation of these results is shown in Figure 5.11. The 447-KB audio sample was used again to test the difference in audio performance for scheduled and unscheduled transmissions. The results are summarized in Table 5.8 and displayed in Figure 5.12.

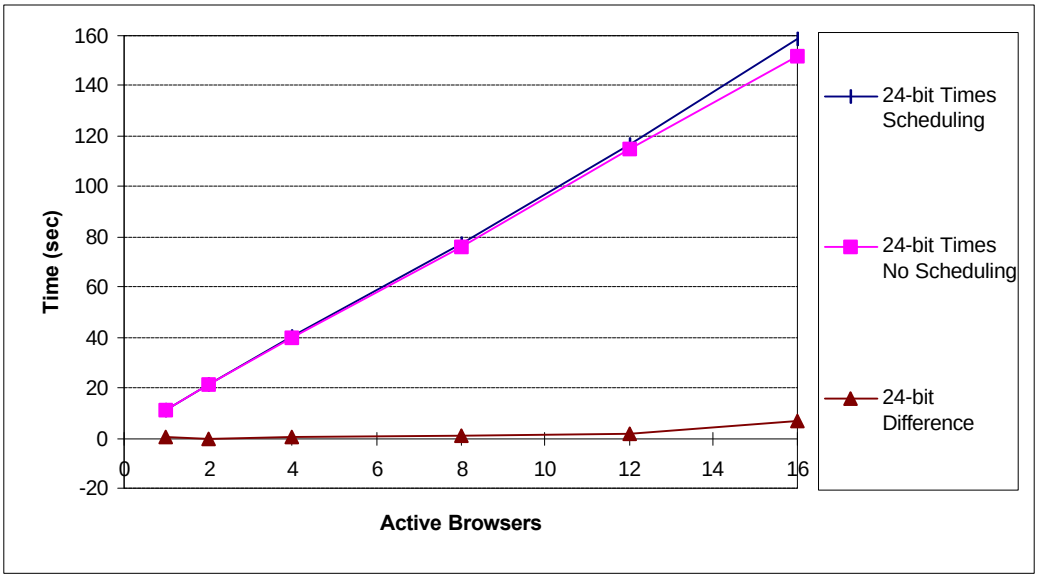

Figure 5.11: 24-bit video scheduling and no scheduling times.

Table 5.7: 24-bit Video Scheduling and No Scheduling Times

Active Browsers	24-bit Times Scheduling	24-bit Times No-Scheduling	24-bit Difference
1	11.428	11.222	0.206
2	21.028	21.602	-0.574
4	40.124	39.773	0.351
8	77.301	76.178	1.123
12	116.770	114.855	1.915
16	158.846	151.862	6.984

Table 5.8: Audio Scheduling and No Scheduling Times

Active Browsers	Audio Times Scheduling	Audio Times No Scheduling	Audio Difference
1	10.972	10.959	0.014
2	19.922	19.884	0.038
4	38.564	38.834	-0.270
8	76.048	74.658	1.390
12	113.994	112.253	1.741
16	151.859	146.511	5.348

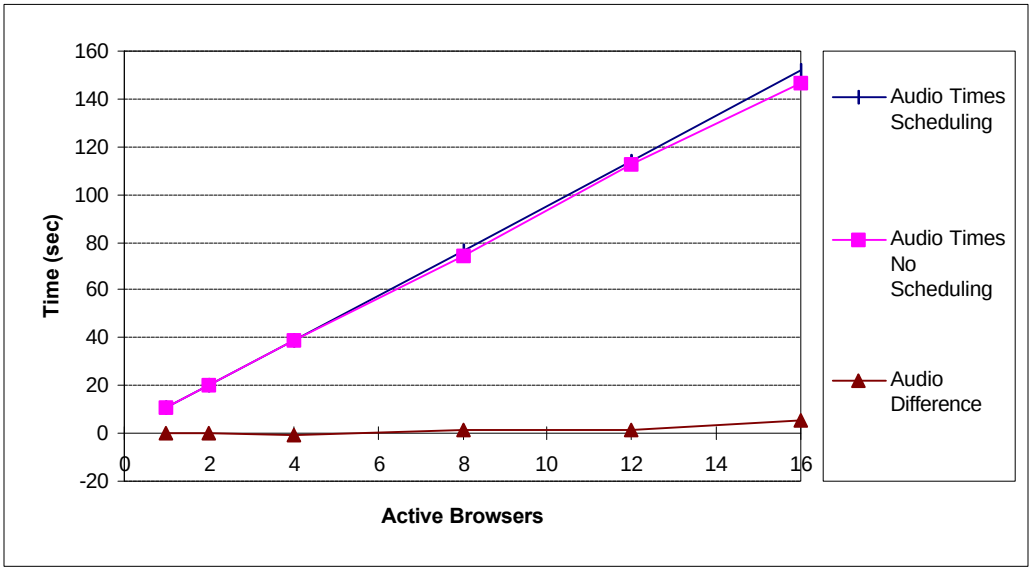

Figure 5.12: Audio scheduling and no scheduling times.

The results indicate that the effect of the error prevention algorithm is minor on the overall transmission time. The delays occur mostly when the Network Interface cannot send any packets because the Decoder Boxes' memories are full.

5.2.5 Effect of Time Slot Constants

Two constants in the EMM play an important role in the performance of the Network Interface. The TIMEPERBROWSER defines how many AV packets will be received through the UNIX socket for each active browser. The TIMEPERDECODER is the time slot that determines how many AV packets will be transmitted to a Decoder Box every time. Tables 5.9, 5.10, and 5.11 summarize the performance effects of these constants for 8-bit video, 24-bit video, and audio data transmissions. For the 8-bit video case, tests were performed with 8 active browsers. For 24-bit video and audio data the tests were performed with 4 active browsers. Although the total times were not greatly affected by the different time slot values, the means and standard deviations of the transmission errors were influenced as seen in Figures 5.13 and 5.14 for 8-bit and 24-bit video, respectively.

Table 5.9: Effect of Time Slots on 8-bit Video Scheduling

TimePer Browser (packets)	TimePer Decoder (msec)	Total Time	Unix Time	Ether Time	Mean Error	Std Dev Error	Pcntg Error
5	100	28.048	2.952	0.478	0.617	0.086	97.585
10	100	27.445	2.908	0.460	1.205	0.184	95.169
50	100	27.201	2.781	0.454	2.391	2.286	89.855
100	100	28.535	2.961	0.515	2.088	3.365	89.855
150	100	28.251	2.965	0.486	1.737	3.088	89.855
200	100	28.366	2.974	0.499	1.408	2.861	89.855
250	100	28.817	3.029	0.486	1.339	3.043	89.855
50	5	27.689	2.809	0.442	0.320	1.107	98.671
50	10	29.105	3.007	0.444	0.611	1.567	97.585
50	50	27.726	2.871	0.459	2.407	2.328	89.855
50	75	27.377	2.813	0.460	2.413	2.314	89.855
50	100	27.960	2.877	0.460	2.461	2.365	89.855

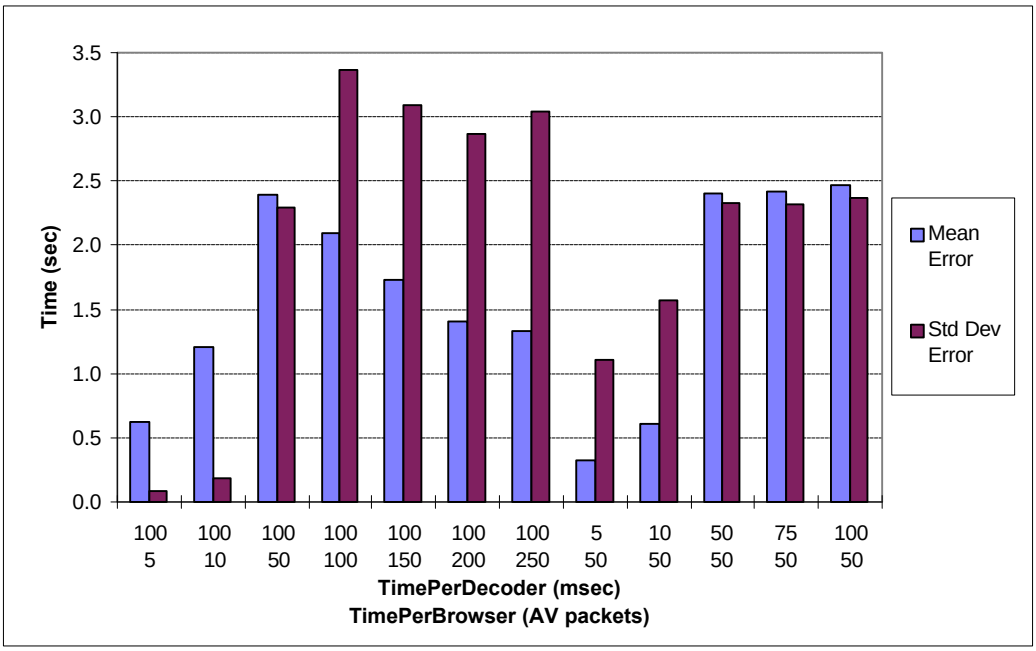

Figure 5.13: Effect of time slots on transmission errors for 8-bit video.

Table 5.10: Effect of Time Slots on 24-bit Video Scheduling

TimePer Browser (packets)	TimePer Decoder (msec)	Total Time	Unix Time	Ether Time	Mean Error	Std Dev Error	Pcntg Error
5	100	44.271	9.370	1.330	0.331	0.020	99.192
50	100	40.322	8.617	1.276	2.954	0.601	91.963
100	100	42.102	8.874	1.323	3.017	2.533	91.680
200	100	41.811	8.824	1.288	2.575	3.819	91.842
400	100	41.372	8.673	1.257	2.032	3.731	91.680
700	100	43.617	9.267	1.289	1.564	3.900	91.842
50	10	42.125	8.784	1.280	0.306	0.820	99.273
50	50	42.352	8.897	1.317	1.574	1.363	96.042
50	100	41.245	8.841	1.284	2.999	0.710	92.003
50	150	40.277	8.510	1.365	2.964	0.467	91.923
50	200	41.819	9.018	1.285	3.084	0.531	91.923
50	300	41.004	8.810	1.263	3.014	0.522	91.923
50	400	40.678	8.615	1.209	3.011	0.371	91.923

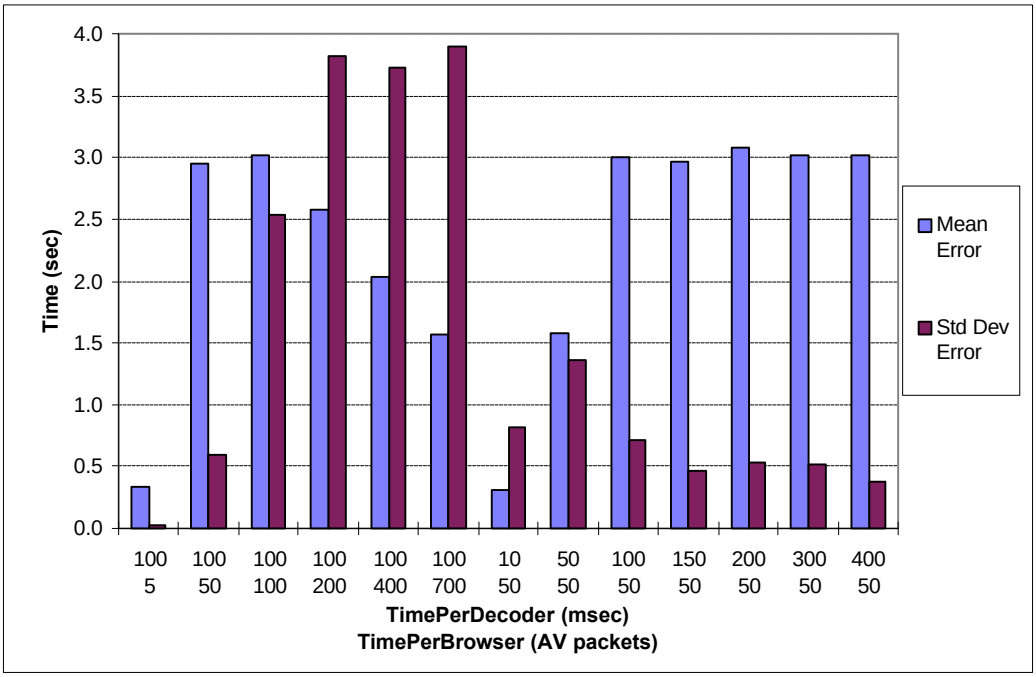

Figure 5.14: Effect of time slots on transmission errors for 24-bit video.

Table 5.11: Effect of Time Slots on Audio Scheduling

TimePer Browser (packets)	TimePer Decoder (msec)	Total Time	Unix Time	Ether Time	Mean Error	Std Dev Error	Pcntg Error
5	100	39.985	8.331	1.253	0.307	0.016	98.958
10	100	37.744	7.823	1.246	0.576	0.040	97.917
50	100	39.076	8.115	1.190	2.733	1.176	59.125
100	100	38.944	8.117	1.260	0.964	3.082	7.042
200	100	38.829	8.089	1.266	0.790	4.865	8.333
400	100	38.018	7.884	1.295	0.530	4.580	7.167
600	100	39.182	8.203	1.242	0.579	4.000	6.292
100	10	38.517	7.926	1.232	2.721	2.253	2.125
100	50	39.001	8.130	1.221	1.126	2.958	4.917
100	100	38.853	8.087	1.262	0.744	3.070	7.708
100	150	40.093	8.390	1.366	0.962	3.308	7.083
100	200	38.894	8.141	1.226	4.394	2.666	43.667
100	400	35.871	7.864	1.512	5.831	0.009	83.333

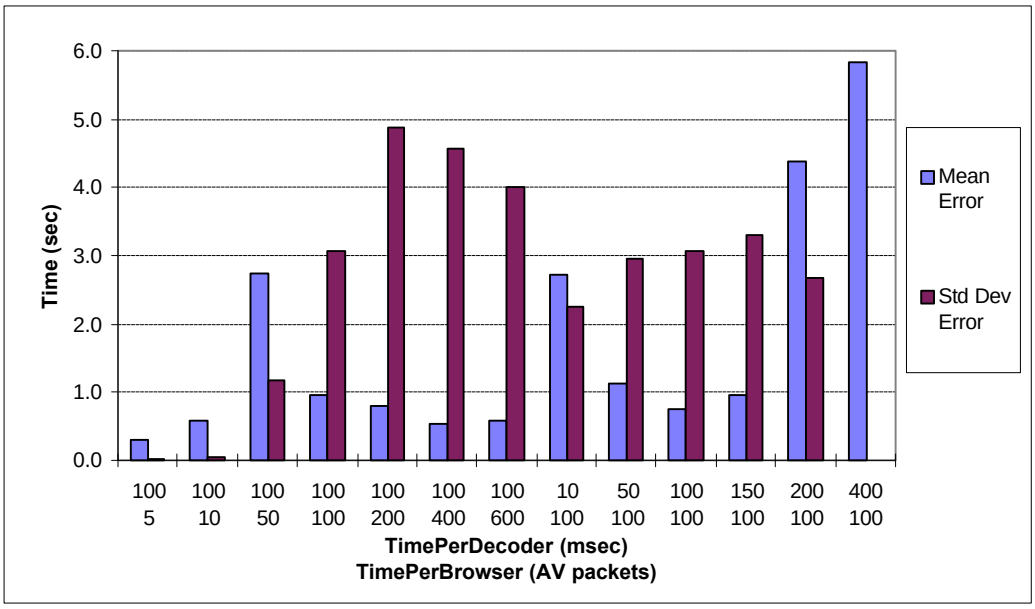

Figure 5.15: Effect of time slots on transmission errors for audio data.

The variation of the two time-slot constants has the same effect on both 8-bit and 24-bit video transmission. Audio data, however, responds differently to variations in the time-slot constants. The best values for all three data types are presented in Table 5.12.

Table 5.12: Best Time Slots Values

	TimePerBrowser (AV packets)	TimePerDecoder (msec)
8-bit Video	50	5
24-bit Video	5	100
Audio	5	100

The time slots variations had a great effect on the percentage of late packets (Pcntg Error) for the audio data transmissions, as shown in Figure 5.16. As seen from the graph the best performance is obtained with a TimePerBrowser of 100 audio AV packets, and a TimePerDecoder of 10 msec.

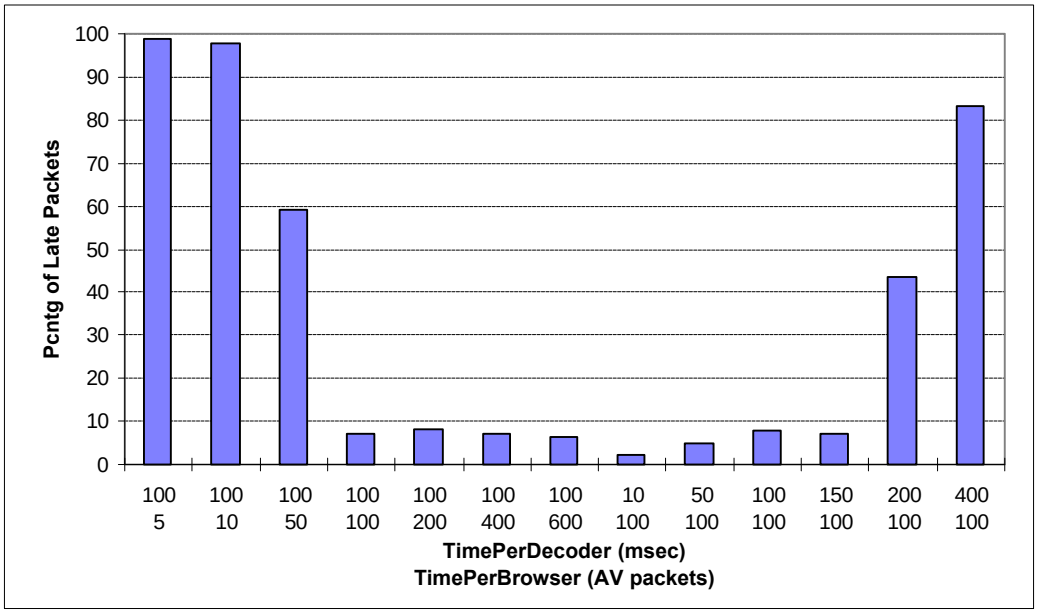

Figure 5.16: Effect of time slots on the percentage of late packets for audio data.

5.3 Summary

Testing of the Network Interface of the IVDS/WWW Browser system was performed only on the Browser Server end since the Decoder Box was not manufactured. Results from a dedicated intranet were obtained only for 10-Mbps operation. Based on these results we can infer that both 8-bit and 24-bit video operation is not possible at a 30 frames/sec rate. Static Web pages can be supported for multiple active browsers. Audio operation performs relatively well with a few gaps when the workload is high.

Chapter 6 - CONCLUSIONS

6.1 Problem Statement

Virginia Tech's Center for Wireless Telecommnications and Interactive Rresponse Services, Inc., have developed the IVDS system to deliver data services to the general public. One of these services is Web browsing where the user uses a remote control unit as input and a TV set as output. All processing is performed at a central Browser Server which serves up to 256 users. At the user's site a Decoder Box acts as a client to display the Web pages on the TV set. This thesis presented the networking requirements and a possible solution for the network interface between the Browser Server and the Decoder Box of the IVDS WWW Browser service.

6.2 Solution

At the Browser Server two software modules, the Ethernet Management Module (EMM) and the Ethernet Interface Module (EIM), were developed. The EMM receives data from the browser threads that correspond to the active Decoder Boxes. The EMM acts as a server with the browser threads as clients which pass the AV packets to the EMM through UNIX domain sockets. The EMM stores the AV packets in a data structure for each Decoder Box. An "error prevention algorithm" was developed to schedule the transmission of the AV packets. The algorithm makes use of the fact that the Decoder Boxes have fixed memory sizes and memory transfer rates, since they are completely hardware based. When an AV packet is scheduled for transmission the EIM constructs the Ethernet frame and transmits it to the correct Decoder Box using the dedicated Fast Ethernet NIC.

At the client end a Fast Ethernet physical layer chip-set was designed to serve as the network interface for the Decoder Box. The chip-set is based on 100BASE-TX network technology, although with minor modifications it can also serve 100BASE-FX networks. The chips were selected and the circuit was designed in such a way that it can support both 10-Mbps and 100-Mbps operation. The network interface communicates with the rest of the Decoder Box through the Medium Indepentent Interface. This allows even more flexibility in terms of selecting and using a Fast Ethernet technology.

Given today's communication networks that exist for the general public, the best deployment option of the IVDS/WWW system would be in a cable TV network with the use of cable modems at both ends. The latest cable modems offer downstream bandwidth of up to 40 Mbps, but it is anticipated that it will soon exceed 100 Mbps. Another deployment option uses standard telephone lines and ADSL modems. ADSL technology offers a maximum of 9 Mbps downstream bandwidth.

6.3 Results

The Browser Server Network Interface ensures that no packet is lost due to memory overflow at the Decoder Box. At the same time it attempts to send the packets on time, so that no gap appears either on video or audio. As seen from the tests, the 30 frames/sec NTSC rate is impossible to achieve. Static Web pages are transmitted in reasonable time. In the case of 8-bit pixels when two users make a request at exactly the same time the Web pages will be displayed on their TV sets in about 8 seconds. In the extreme case of 16 users making a request at the same time, the Web pages will arrive in about 55 seconds. Some gaps in the audio are to be expected for the audio transmission when the workload is high. When 12 users request an audio clip of size about 500 KB the percentage of late audio packets reaches the prohibitive value of 13%.

The Decoder Box Network Interface could not be tested since the Decoder Box board had not been built. Nevertheless, the selected Fast Ethernet chip-set has been extensively used in many other commercial Fast Ethernet applications, such as NICs and repeaters. This is an affirmation that the chip-set would perform within its specifications, especially given the fact that we are only using its reception functionality.

6.4 Future Work

There are many ways to expand the IVDS/WWW Browser Network Interface. Future plans for the IVDS system include moving the server processing from a central high-end computer to a distributed environment consisting of several low-end computers. The EMM has been designed in such a way so that it can support this change with minor modifications, namely changing the UNIX domain protocol to an INET protocol (TCP/IP or UDP/IP).

Fiber-To-The-Home (FTTH) networks will eventually be installed. FTTH will make the IVDS/WWW Browsing system even more appealing since there will not be the need for intermediate networks and modems (cable modems or ADSL modems). The Decoder Box Network Interface will be able to connect to fiber-optic cables using the 100BASE-FX interface. The selected chip-set can support this network technology by swapping the DP83223 chip (100BASE-TX transceiver) with a 100BASE-FX transceiver.

References

[3Com96] 3Com Corp, 1996. *100BASE-T Fast Ethernet: A High Speed Technology for Cost-Effective Scaling of 10BASE-T Networks*, <http://www.3com.com/0files/strategy/fasteth.html>.

[Bel91] Bellamy, J. 1991. *Digital Telephony*, Wiley, New York.

[BeG92] Bertsekas, D. P. and Gallager, R. 1992. *Data Networks*, Prentice-Hall, Englewood Cliffs, NJ.

[Cic95] Ciciora, W. S. 1995. *Cable Television in the United States: An Overview*, Cable Television Labs publication.

[CaTV97] CATV CyberLab WWW pages, 1997, <http://www.catv.org>, Broadband Bob.

[CLC96] Coopers & Lybrand Consulting, 1996. "Winners & Losers in the Evolution of the Internet & the World Wide Web", CLC Telecom and Media Group, N.Y., and <http://www.colybrand.com/industry/infocom/mediapr.html>.

[CLR90] Cormen, T., Leiserson, C., and Rivest, R. 1990. *Introduction to Algorithms*, MIT Press, Cambridge, MA and McGraw-Hill, New York.

[CMC95] Cox, N., Manley, C. and Chea, F. 1995. *LAN Times Guide to Multimedia Networking*, McGraw-Hill, New York.

[Con95] Connors, J. 1995. "The Evolution of Cable Television to Interactive Communications Service Provider", < http://www.tezcat.com/~chicago/modem/sun.html>.

[Com91] Comer, D. E. 1991. *Internetworking with TCP/IP, Volume I: Principles, Protocols, and Architecture*, Prentice-Hall, Englewood Cliffs, NJ.

[CoS93] Comer, D. E. and Stevens, D. L. 1993. *Internetworking with TCP/IP, Volume III: Client-Server Programming and Applications*, Prentice-Hall, Englewood Cliffs, NJ.

[DaN96] David, T. and Norris, C. 1996 "CableTV WWW Browser with Wireless User Access", <http://fiddle.ee.vt.edu/courses/ee4984/Projects1996/david_norris/david_norris.html>

[Dav97] David, T. 1997. “Internet for the Masses”, *WWW: Beyond the Basics*, Prentice-Hall, Englewood Cliffs, N.J. and <http://ei.cs.vt.edu/~wwwbtb/book/chap23/>.

[Day96] Day, R. 1996. “The Great PC/TV Debate”, *OEM*, CMP Media.

[Erg96] Ergen, F. 1996. “Effects of Interface Format, Feedback Style, and System Lag on the Usability of Hand-Held Internet Controllers.” Master’s Thesis, Virginia Polytechnic Institute and State University, Blacksburg, VA.

[Exc88] Excelan, Inc. 1988. *LANalyzer EX5000 Series Network Analyzers*, Excelan Inc., San Jose, CA.

[Fin96] Finkelstein, R. “Network computers -- The birth of a new industry”, <http://www.links.inter.net/art/index.html>, Links Technology Corp.

[Fra96] Franks, S. 1996. “IVDS System Channel Simulator and Repeater Design.” Master's Thesis, Virginia Polytechnic Institute and State University, Blacksburg, VA.

[Gai96] Gaida, K. 1996. “Interactive TV will shape the European information highway”, *BYTE*, January 1996, McGraw-Hill, New York.

[Gbu96] Gburzynski, P. 1996. *Protocol Design for Local and Metropolitan Area Networks*, Prentice-Hall, Englewood Cliffs, NJ.

[Gil95] Gillett, S. E. 1995. *Connecting Homes to the Internet: An Engineering Cost Model of Cable vs. ISDN*, MIT Laboratory for Computer Science Technical Report 654.

[Gre96] Green, H. J. 1996. “IVDS Consumer Control Unit Evolution and Bar Code Interface Design.” Master’s Thesis, Virginia Polytechnic Institute and State University, Blacksburg, VA.

[Hal96] Halfhill, T. R. “Break the Bandwidth Barrier”, *BYTE*, September 1996, McGraw-Hill, New York.

[Hal97] Halfhill, T. R. “Cheaper Computing, Part I”, *BYTE*, April 1997, McGraw-Hill, New York.

[Haw97] Hawes, A. 1997. “Designing an IVDS World Wide Web Browser Hardware Architecture.” Master’s Thesis, Virginia Polytechnic Institute and State University, Blacksburg, VA.

[Hug97] DirectPC WWW pages, June 1997, <http://www.direcpc.com/>, Hughes Network Systems.

[Hyb97] Hybrid Networks, Inc. WWW pages, July 1997, <http://www.hybrid.com/>, Hybrid Networks.

[IEEE90] Institute of Electrical and Electronics Engineers 1990. *Carrier Sense Multiple Access with Collision Detection (CSMA/CD) Access Method and Physical Layer Specifications.* ANSI/IEEE Std 802.3.

[IEEE95] Institute of Electrical and Electronics Engineers 1995. *Local and Metropolitan Area Networks-Supplement - Media Access Control (MAC) Parameters, Physical Layer, Medium Attachment Units and Repeater for 100Mb/s Operation, Type 100BASE-T (Clauses 21-30).* ANSI/IEEE Std 802.3u.

[KeR88] Kernighan, B. W. and Ritchie, D. M. 1988. *The C Programming Language*, Prentice-Hall, Englewood Cliffs, NJ.

[Mot97] Motorola 1997. “Motorola's CopperGold Transceiver Delivers: More, Better, Faster”, <http://www.mot.com/SPS/MCTG/MDAD/adsl/adsl_whitepaper.html>.

[New97] Newman, J. “ADSL: Putting a Charge Into Your Copper Wire”, *Network Computing*, CMP Media, May 1997.

[NSC97a] National Semiconductor Corp. 1997, *DP83223 - TWISTER High Speed Networking Transceiver Device*, National Semiconductor Press.

[NSC97b] National Semiconductor Corp. 1997, *DP83840A - 10/100 Mb/s Ethernet Physical Layer*, National Semiconductor Press.

[PEI97] Pulse Engineering, Inc. 1997, *PE-68515 10/100BASE-T Transformer Module*, Pulse Engineering, San Diego, CA.

[Sta93] Stallings, W. 1993. *Local and Metropolitan Area Networks*, Macmillan, New York.

[Ste90] Stevens, W. R. 1990. *UNIX Network Programming*, Prentice-Hall, Englewood Cliffs, NJ.

[Ste92] Stevens, W. R. 1992. *Advanced Programming in the UNIX Environment*, Addison-Wesley, Reading, MA.

[Ste94] Stevens, W. R. 1994. *TCP/IP Illustrated, Volume 1 - The Protocols*, Addison-Wesley, Reading, MA.

[Ste96] Stevens, W. R. 1996. *TCP/IP Illustrated, Volume 3 - TCP for Transactions, HTTP, NNTP, and the UNIX Domain Protocols*, Addison-Wesley, Reading, MA.

[Str96] Strom, D. June 1996. “Breaking the Internet Speed Barrier”, *Windows Sources*, Ziff-Davis Publishing.

[Zak96] Zakon, H. 1996. “Hobbes' Internet Timeline v2.5”, The MITRE Corporation, <http://info.isoc.org/guest/zakon/Internet/History/HIT.html>.

Appendix A : Software Code

IVDS/WWW Browser Server - Networking Interface

Appendix B : Timing Results

Note: All the results can be obtained upon request.

Results for the effect of the number of active browsers on time with AV packet scheduling

For 8-bit video

esnd	-i	ec2	-b	5	-d	50000		
browser	-b	X	-n	207	-s	1491	-t	1

For 24-bit video

esnd	-i	ec2	-b	50	-d	50000		
browser	-b	X	-n	619	-s	497	-t	0

For Audio

esnd	-i	ec2	-b	50	-d	50000		
browser	-b	X	-n	600	-s	745	-t	2

Active Browsers	8-bit Time (sec)	24-bit Time (sec)	Audio Time (sec)
1	5.11	11.43	10.97
2	8.24	21.03	19.92
4	14.40	40.12	38.56
8	27.41	77.30	76.05
10	33.82	97.13	101.27
12	40.20	116.77	113.99
16	55.06	158.85	151.86
20	64.69		
24	77.47		

8-bit video	24-bit video	Audio
1 browser	1 browser	1 browser
5.094696	11.26384	10.93804
5.122889	11.27732	10.9622
5.092358	12.04249	10.93989
5.128125	11.28098	11.04853
5.120425	11.27488	10.9718
5.1116986	**11.4279**	**10.97209**
2 browsers	2 browsers	2 browsers

Appendix B: Timing Results cont'd

8.012375	20.50306	19.97356
8.111996	20.04166	20.01149
8.682885	21.84444	19.95631
8.8752	21.78034	19.74572
7.98824	21.26212	19.96883
8.170238	20.7367	19.87421
8.021033	**21.02805**	**19.92169**
8.04678	4 browsers	4 browsers
8.23859338	41.03875	37.75161
	41.24779	37.90817
4 browsers	41.23968	38.07147
13.95099	41.27585	38.08717
14.358686	38.83669	39.58926
14.432588	39.09733	39.85385
14.428619	39.13781	39.86899
13.997568	39.11991	39.8901
14.3615	**40.12423**	37.73472
14.358083	8 browsers	37.89826
14.398404	76.03926	38.05982
14.276179	70.7181	38.05646
14.712751	75.37088	**38.56416**
14.711145	75.71163	8 browsers
14.762416	75.47725	75.14304
14.3957441	75.76986	75.35501
8 browsers	76.20724	75.26307
24.937042	76.8706	75.46613
26.864016	76.58301	75.27648
27.193398	79.88258	75.53897
27.288879	79.71439	75.31491
27.334971	79.51336	75.34869
27.456976	79.56067	76.63803
27.451102	79.59637	76.63226
27.459934	79.98302	76.87987
25.655439	79.81784	77.21095
27.841131	**77.301**	76.65541
28.142278	10 browsers	76.64753
28.055179	96.9583	76.72073
28.166238	97.60096	76.67936
28.239958	97.92485	**76.04815**
28.28087	97.74887	10 browsers
28.266547	97.31728	101.044
27.4146224	96.62922	101.4769

Appendix B: Timing Results cont'd

10 browsers	96.93082	101.5002
31.30557	96.66419	101.5769
35.187894	96.70035	101.6633
35.006776	96.76772	101.5957
35.187292	96.31943	101.6576
35.003335	97.09244	101.8524
35.134927	97.11378	101.8529
35.328435	96.87604	101.7709
35.279641	96.9538	101.0486
35.288376	97.32087	100.5745
35.354717	97.25749	100.7439
28.766365	97.10808	100.8735
32.523453	97.11876	100.7884
32.652186	97.13921	101.6003
33.223328	96.60928	100.7326
33.330829	96.88126	100.7586
33.621741	96.95366	101.3485
33.538321	97.09027	100.981
33.532113	97.7691	**101.272**
33.600228	97.6682	12 browsers
33.584342	97.22149	114.9363
33.8224935	97.254	115.3307
12 browsers	97.34959	115.5165
35.939736	97.473	115.4982
40.88335	**97.12708**	114.3807
41.002827	12 browsers	115.4614
40.832937	113.7073	114.5032
41.328301	114.2789	114.5651
41.327864	113.46	114.5388
41.110041	113.6616	114.5294
41.262702	114.0075	114.6303
41.139694	114.2549	114.6884
41.227311	114.1119	112.3409
41.215885	114.596	113.1968
41.298265	114.1189	112.742
34.925435	114.4054	112.8146
39.656074	113.3512	113.8357
39.715262	111.7554	113.4695
40.308312	117.8483	113.1042
40.307341	118.6783	113.0576
40.117254	118.8192	113.1002
40.092705	119.0657	113.602

40.208707	119.0528	112.9945
40.121426	118.8927	113.0244
40.266116	119.5261	**113.9942**
40.266896	118.9208	16 browsers
40.263564	118.9587	150.9737
40.2007502	119.0073	151.4354
16 browsers	119.1855	152.6314
41.424337	119.1695	152.1862
53.958362	118.2011	152.4663
54.811134	119.1267	151.8112
55.842885	118.5337	151.8747
56.42507	118.5582	151.6714
56.449639	116.7498	151.76
56.892967	116.7824	151.7326
56.982344	116.837	151.4348
56.929073	118.5134	151.4475
56.689158	116.8508	151.4186
56.692395	116.8687	151.41
56.744734	116.9141	151.4346
56.911295	116.9528	151.408
56.97735	**116.7701**	151.6545
56.824553	16 browsers	151.155
56.887968	157.4135	151.9913
49.652494	158.4499	152.9038
53.158493	159.323	152.0361
53.247725	158.9128	152.3288
55.294384	159.5563	152.2314
55.144826	159.011	152.2745
55.204813	158.676	152.1578
55.054413	159.03	152.1338
55.212539	159.1256	151.7598
55.230513	158.9535	151.8972
55.212806	158.708	152.0307
55.326143	158.794	152.2393
55.316993	158.7565	151.8242
55.354261	158.8273	151.7647
55.342844	159.0694	**151.8587**
55.321037	158.9259	
55.361964	**158.8458**	
55.0587348		
20 browsers	8-bit video	
48.41209	24 browsers	

63.65436	66.161362
63.32387	66.260167
64.257371	76.21228
64.162298	76.572835
65.51536	77.598809
65.622624	77.571111
65.844811	77.629875
65.565783	78.729764
66.006845	78.777292
65.759938	78.321249
66.12979	78.401184
65.84606	78.37781
66.145063	78.495096
66.047443	78.568831
65.915348	79.16011
65.929535	79.201463
65.947511	79.105496
66.253583	79.19063
66.034016	79.198711
53.21855	79.239465
59.111765	79.238398
64.658785	79.235963
65.226528	79.10667
65.220733	78.828997
65.384732	**77.465982**
65.353967	
65.622279	
65.520656	
65.93635	
65.909588	
65.924424	
66.051899	
65.860869	
65.9686	
66.022364	
66.017011	
66.238618	
66.044097	
66.063009	
64.6932131	

Statistics for 8-bit video packet transmissions

Active Browsers	Total Time	Unix Time	Ether Time	Mean Error	Std Dev Error	Pcntg Error
1	5.32	2.916	0.417	0.070	0.008	97.5845
2	8.10	2.748	0.434	0.137	0.022	97.5845
4	14.52	2.783	0.443	0.292	0.038	97.5845
8	26.64	2.759	0.481	0.584	0.072	97.5845
10	33.28	2.834	0.498	0.744	0.106	97.5845
12	37.91	2.741	0.522	0.854	0.137	97.5845
16	52.26	2.908	0.531	1.200	0.192	97.5845
20	64.49	2.856	0.499	1.495	0.234	97.5845
24	77.66	2.869	0.487	1.814	0.267	97.5845

Total Time	Unix Time	Ether Time	Mean Error	Variance	Std Deviation	Pcntg Error
esnd	-i	ec2	-b	5	-d	50000
browser	-b	1	-n	207	-s	1491 -t 1
5.143768	2.727627	0.415755	0.065725	0.000067	0.008185	97.5845
5.111661	2.694203	0.419035	0.064894	0.000043	0.006557	97.5845
5.436329	3.031047	0.421123	0.073013	0.000053	0.00728	97.5845
5.8189	3.421212	0.414065	0.082402	0.000079	0.008888	97.5845
5.110878	2.707169	0.414073	0.065232	0.000044	0.006633	97.5845
5.324307	**2.916252**	**0.41681**	**0.070253**	**0.0000572**	**0.007509**	**97.5845**
browser	-b	2	-n	207	-s	1491 -t 1
7.966105	2.702232	0.412635	0.135064	0.000405	0.020125	97.5845
8.076672	2.706511	0.444582	0.134791	0.000617	0.024839	97.5845
8.021275	2.68934	0.413976	0.136383	0.000283	0.016823	97.5845
8.023705	2.699243	0.47664	0.136227	0.000499	0.022338	97.5845
8.28745	2.700044	0.417376	0.142789	0.000485	0.022023	97.5845
8.39645	3.037431	0.44423	0.142886	0.000709	0.026627	97.5845
7.977023	2.711453	0.416722	0.135238	0.000404	0.0201	97.5845
8.068208	2.738553	0.449115	0.135897	0.000505	0.022472	97.5845
8.102111	**2.748101**	**0.43441**	**0.137409**	**0.0004884**	**0.021918**	**97.5845**
browser	-b	4	-n	207	-s	1491 -t 1
14.31803	2.709903	0.412562	0.290598	0.001804	0.042474	97.5845
14.68723	3.037904	0.487395	0.296868	0.000504	0.02245	97.5845
14.68278	2.686269	0.436929	0.29527	0.000961	0.031	97.5845
14.72458	2.689404	0.442129	0.294578	0.001323	0.036373	97.5845
13.9335	2.693162	0.415574	0.280472	0.001774	0.042119	97.5845

14.21186	2.69041	0.451895	0.285459	0.000887	0.029783	97.5845
14.27754	2.706836	0.451333	0.285535	0.001329	0.036455	97.5845
14.29737	2.699545	0.451192	0.283021	0.001862	0.043151	97.5845
14.45642	2.98037	0.42196	0.29289	0.003123	0.055884	97.5845
14.86729	2.69701	0.439032	0.302128	0.000925	0.030414	97.5845
14.84096	2.777378	0.450349	0.301422	0.001277	0.035735	97.5845
14.93662	3.029331	0.45033	0.29924	0.002303	0.04799	97.5845
14.51951	**2.783127**	**0.442557**	**0.29229**	**0.001506**	**0.037819**	**97.5845**
browser	-b	8	-n	207	-s	1491	-t	1
24.92365	2.736272	0.430738	0.548282	0.022541	0.150137	97.5845
26.90738	2.698827	0.475865	0.592772	0.000887	0.029783	97.5845
26.85187	2.719513	0.467014	0.590495	0.001304	0.036111	97.5845
26.92653	2.722759	0.481267	0.589609	0.001721	0.041485	97.5845
26.92343	2.72745	0.47034	0.589343	0.002218	0.047096	97.5845
26.99258	2.71418	0.473275	0.588745	0.002786	0.052783	97.5845
26.942	2.718808	0.474605	0.588158	0.003398	0.058292	97.5845
26.96139	2.710727	0.473205	0.587616	0.004079	0.063867	97.5845
23.70038	2.715459	0.431137	0.517116	0.039509	0.198769	97.5845
26.7196	3.036914	0.493734	0.589552	0.004528	0.06729	97.5845
26.92348	2.701739	0.501741	0.596419	0.001912	0.043726	97.5845
26.99841	2.72337	0.496502	0.594443	0.002874	0.05361	97.5845
27.0399	2.709194	0.505435	0.593672	0.003381	0.058146	97.5845
27.1689	3.033084	0.500143	0.596754	0.003837	0.061944	97.5845
27.14452	2.70427	0.510372	0.588023	0.008179	0.090438	97.5845
27.18688	2.763796	0.512399	0.587393	0.009116	0.095478	97.5845
26.64443	**2.758523**	**0.481111**	**0.58365**	**0.0070169**	**0.07181**	**97.5845**
browser	-b	10	-n	207	-s	1491	-t	1
28.64428	2.734453	0.428439	0.636875	0.074331	0.272637	97.5845
32.75424	3.021396	0.492756	0.731657	0.022305	0.149349	97.5845
33.55395	2.707197	0.519719	0.752585	0.007127	0.084422	97.5845
33.53313	3.15454	0.513507	0.753952	0.006192	0.078689	97.5845
33.72395	2.728849	0.522806	0.757323	0.005374	0.073308	97.5845
33.78764	2.708574	0.524755	0.755031	0.007272	0.085276	97.5845
33.85607	2.709018	0.537896	0.754176	0.008109	0.09005	97.5845
33.86342	3.313451	0.524021	0.753774	0.009259	0.096224	97.5845
33.90497	2.695606	0.523182	0.753223	0.010294	0.101459	97.5845
33.86923	2.734635	0.526279	0.752548	0.011367	0.106616	97.5845
30.38301	2.737297	0.439219	0.679919	0.053009	0.230237	97.5845
33.08967	3.056655	0.442469	0.742024	0.016043	0.126661	97.5845
33.47723	2.704959	0.486532	0.754965	0.005074	0.071232	97.5845
33.81668	2.727975	0.49642	0.759942	0.003613	0.060108	97.5845
33.87734	2.708514	0.51491	0.758947	0.004145	0.064382	97.5845

33.84427	2.720089	0.495053	0.756772	0.005661	0.07524	97.5845		
33.86394	2.692263	0.487777	0.756448	0.006463	0.080393	97.5845		
33.86179	2.719822	0.49394	0.755754	0.007277	0.085305	97.5845		
33.98404	3.076515	0.496533	0.755048	0.008314	0.091181	97.5845		
33.91876	3.036824	0.497941	0.754431	0.009433	0.097124	97.5845		
33.28038	**2.834432**	**0.498208**	**0.74377**	**0.0140331**	**0.105995**	**97.5845**		
browser	-b	12	-n	207	-s	1491	-t	1
31.4703	2.688189	0.422866	0.705208	0.116992	0.342041	97.5845		
37.26515	2.699869	0.502836	0.838338	0.034109	0.184686	97.5845		
37.8647	2.709237	0.520787	0.857271	0.018588	0.136338	97.5845		
38.20574	2.704232	0.540976	0.866564	0.013472	0.116069	97.5845		
38.5055	2.701863	0.532065	0.874704	0.00842	0.091761	97.5845		
38.47419	2.750079	0.528858	0.872373	0.009823	0.099111	97.5845		
38.61233	2.705977	0.526721	0.873745	0.009569	0.097821	97.5845		
38.78001	2.74417	0.536368	0.870339	0.013726	0.117158	97.5845		
38.77886	2.718994	0.539293	0.868315	0.01649	0.128413	97.5845		
38.83179	2.692395	0.539389	0.867788	0.017691	0.133008	97.5845		
38.81659	2.79	0.542952	0.867132	0.019008	0.13787	97.5845		
38.84182	3.097876	0.549309	0.866414	0.020603	0.143537	97.5845		
31.84625	2.698519	0.426686	0.714434	0.112615	0.335582	97.5845		
37.33655	2.731751	0.497263	0.84684	0.027724	0.166505	97.5845		
38.10499	2.695408	0.514112	0.863522	0.014056	0.118558	97.5845		
38.56584	2.688601	0.519632	0.87436	0.007209	0.084906	97.5845		
38.45098	2.705498	0.52271	0.874147	0.007159	0.084611	97.5845		
38.49065	2.676574	0.529135	0.87521	0.006755	0.082189	97.5845		
38.70672	2.689721	0.529475	0.875331	0.008103	0.090017	97.5845		
38.71792	2.700281	0.531913	0.873307	0.010264	0.101311	97.5845		
38.81258	3.018805	0.541935	0.871052	0.012893	0.113547	97.5845		
38.74289	2.718449	0.531867	0.870772	0.0141	0.118743	97.5845		
38.78241	2.729645	0.558672	0.869571	0.015323	0.123786	97.5845		
38.78229	2.723843	0.534052	0.869593	0.016741	0.129387	97.5845		
37.90779	**2.740832**	**0.521661**	**0.85443**	**0.0229764**	**0.13654**	**97.5845**		
browser	-b	16	-n	207	-s	1491	-t	1
42.37671	2.704966	0.416928	0.969639	0.274266	0.523704	97.5845		
53.2555	2.713466	0.511637	1.232491	0.036621	0.191366	97.5845		
53.86341	3.049832	0.528304	1.251471	0.020004	0.141435	97.5845		
53.91754	3.009157	0.517936	1.250944	0.020415	0.142881	97.5845		
53.84412	2.962497	0.524535	1.25019	0.020978	0.144838	97.5845		
53.80725	3.436644	0.522357	1.247709	0.022489	0.149963	97.5845		
54.55771	3.011341	0.538114	1.255651	0.016416	0.128125	97.5845		
54.4751	2.685828	0.529858	1.259441	0.015738	0.125451	97.5845		
54.47768	2.750188	0.531255	1.260676	0.016471	0.128339	97.5845		

54.47598	2.740871	0.529827	1.260091	0.017902	0.133798	97.5845
54.60531	2.727385	0.530591	1.259495	0.01925	0.138744	97.5845
54.61774	3.605371	0.583781	1.248011	0.028815	0.16975	97.5845
54.63545	3.133572	0.546685	1.247573	0.03104	0.176182	97.5845
54.64351	2.721562	0.577037	1.244905	0.035118	0.187398	97.5845
54.66282	2.702232	0.546443	1.240842	0.045403	0.21308	97.5845
54.67088	3.395414	0.543232	1.240004	0.048034	0.219167	97.5845
40.2061	3.032877	0.424206	0.915633	0.250752	0.500751	97.5845
49.00521	2.734096	0.514499	1.132305	0.083588	0.289116	97.5845
50.20366	3.089052	0.519314	1.15576	0.060453	0.245872	97.5845
50.98751	2.689045	0.522902	1.178909	0.035787	0.189175	97.5845
50.93674	2.681338	0.52933	1.178222	0.036209	0.190287	97.5845
51.95346	2.691715	0.543861	1.201452	0.016196	0.127264	97.5845
52.01449	2.712164	0.548012	1.200809	0.017086	0.130713	97.5845
51.91682	3.029218	0.546366	1.196576	0.019874	0.140975	97.5845
51.95761	2.724184	0.550437	1.195932	0.020915	0.14462	97.5845
52.2147	3.047274	0.541949	1.193101	0.028076	0.167559	97.5845
52.14887	2.700155	0.53072	1.192506	0.029639	0.17216	97.5845
52.15685	2.718961	0.52776	1.192059	0.031279	0.176859	97.5845
52.39936	2.707238	0.558246	1.191507	0.032972	0.181582	97.5845
52.45895	3.394711	0.582689	1.190225	0.034885	0.186775	97.5845
52.26925	3.08276	0.53275	1.189954	0.036901	0.192096	97.5845
52.53972	2.686802	0.551219	1.189706	0.038826	0.197043	97.5845
52.258	**2.908497**	**0.531337**	**1.200431**	**0.0450749**	**0.192096**	**97.5845**
browser	-b	20	-n	207	-s	1491	-t	1
56.14146	3.424801	0.423245	1.306216	0.296164	0.54421	97.5845
61.99587	2.708644	0.470065	1.444702	0.14694	0.383328	97.5845
66.07656	2.706711	0.475014	1.535866	0.031894	0.178589	97.5845
65.73405	3.430714	0.481215	1.535053	0.032448	0.180133	97.5845
65.73379	2.748886	0.490127	1.531506	0.033029	0.181739	97.5845
65.67464	2.722084	0.492844	1.52971	0.033806	0.183864	97.5845
65.60885	3.060644	0.495351	1.528987	0.034691	0.186255	97.5845
66.04713	2.726612	0.510535	1.541	0.021941	0.148125	97.5845
66.00751	2.752519	0.522513	1.540099	0.023246	0.152466	97.5845
66.21962	2.710052	0.532927	1.539365	0.024516	0.156576	97.5845
66.38262	3.041078	0.543102	1.538498	0.026064	0.161443	97.5845
66.14537	3.021945	0.508353	1.538622	0.027752	0.166589	97.5845
66.2865	2.7054	0.497541	1.53832	0.029325	0.171245	97.5845
66.41262	2.736517	0.519418	1.527279	0.051557	0.227062	97.5845
66.32597	2.726541	0.52577	1.52655	0.053818	0.231987	97.5845
66.41219	2.70766	0.534895	1.525779	0.056154	0.236968	97.5845
66.41517	3.407659	0.527206	1.525187	0.058959	0.242815	97.5845

66.50209	2.733938	0.519389	1.52481	0.061359	0.247707	97.5845
66.5552	2.72312	0.518726	1.524268	0.063928	0.25284	97.5845
66.51925	2.747664	0.511871	1.523876	0.066566	0.258004	97.5845
56.52577	2.846697	0.426447	1.315283	0.234834	0.484597	97.5845
57.45691	2.70921	0.517315	1.33816	0.220341	0.469405	97.5845
63.90908	2.728223	0.516278	1.487695	0.046203	0.214949	97.5845
64.21821	2.714829	0.537843	1.48879	0.04342	0.208375	97.5845
63.73923	2.713122	0.543678	1.488075	0.043844	0.20939	97.5845
63.80308	3.078728	0.548327	1.487346	0.044427	0.210777	97.5845
64.38379	2.729066	0.545687	1.486871	0.045039	0.212224	97.5845
64.41513	2.69896	0.542289	1.504576	0.028954	0.170159	97.5845
64.36177	3.424138	0.514855	1.495642	0.034997	0.187075	97.5845
64.42894	2.697177	0.510319	1.502923	0.027868	0.166937	97.5845
65.01814	3.074103	0.519695	1.501036	0.03316	0.182099	97.5845
64.94988	2.704651	0.522276	1.50038	0.034796	0.186537	97.5845
64.19012	2.742183	0.443797	1.492899	0.046514	0.215671	97.5845
64.25421	2.732146	0.455418	1.492097	0.048401	0.220002	97.5845
64.30699	2.715644	0.450711	1.486558	0.059048	0.242998	97.5845
64.25123	2.737269	0.446068	1.486176	0.0612	0.247386	97.5845
63.72952	2.732479	0.438808	1.485531	0.063393	0.25178	97.5845
64.36819	2.7266	0.456894	1.484827	0.065703	0.256326	97.5845
63.78902	2.746528	0.449055	1.484175	0.068102	0.260964	97.5845
64.40621	3.448227	0.465739	1.483348	0.071041	0.266535	97.5845
64.49255	**2.856079**	**0.49879**	**1.495452**	**0.0623861**	**0.233903**	**97.5845**
browser	-b	24	-n	207	-s	1491	-t	1
68.77081	2.754903	0.422982	1.614336	0.373871	0.61145	97.5845
70.60491	3.14097	0.432937	1.657278	0.316372	0.56247	97.5845
74.24446	2.718652	0.450474	1.745504	0.178387	0.422359	97.5845
76.86133	2.701045	0.471723	1.813834	0.074991	0.273845	97.5845
77.94624	2.708577	0.484793	1.840442	0.038303	0.195712	97.5845
78.06296	3.776839	0.483635	1.839337	0.039523	0.198804	97.5845
78.03475	2.702951	0.485744	1.83864	0.040509	0.201268	97.5845
78.24456	2.704415	0.481806	1.836754	0.042723	0.206695	97.5845
78.21955	3.06081	0.485274	1.833989	0.044658	0.211324	97.5845
78.6142	2.725697	0.515214	1.845413	0.030306	0.174086	97.5845
78.88121	3.128901	0.496873	1.845288	0.032304	0.179733	97.5845
78.87508	2.72963	0.494922	1.844804	0.034095	0.184648	97.5845
79.00227	2.695665	0.495884	1.844264	0.035913	0.189507	97.5845
78.76538	3.077099	0.497117	1.843648	0.038063	0.195097	97.5845
78.87643	2.72075	0.535761	1.842161	0.039907	0.199767	97.5845
78.93706	2.723956	0.52017	1.841948	0.041909	0.204717	97.5845
78.72642	3.026493	0.488413	1.841969	0.044277	0.210421	97.5845

78.78878	2.75258	0.495367	1.841225	0.046657	0.216002	97.5845
78.77961	2.711036	0.493788	1.842574	0.048268	0.2197	97.5845
78.91152	3.032051	0.505573	1.843382	0.051022	0.22588	97.5845
79.01588	2.738053	0.502385	1.815399	0.100682	0.317304	97.5845
78.95171	3.087807	0.490509	1.8131	0.108624	0.329582	97.5845
78.79815	2.717193	0.471642	1.811073	0.116783	0.341735	97.5845
78.83782	2.721478	0.480081	1.810308	0.120212	0.346716	97.5845
77.6563	**2.869065**	**0.486794**	**1.814445**	**0.0849316**	**0.267451**	**97.5845**

Results for the effect of time-slots on audio packet transmissions

TimePer Browser	TimePerDecoder (msec)	Total Time	Unix Time	Ether Time	Mean Error	Std Dev Error	Pcntg Error
5	100	39.985	8.331	1.253	0.307	0.016	98.958
10	100	37.744	7.823	1.246	0.576	0.040	97.917
50	100	39.076	8.115	1.190	2.733	1.176	59.125
100	100	38.944	8.117	1.260	0.964	3.082	7.042
200	100	38.829	8.089	1.266	0.790	4.865	8.333
400	100	38.018	7.884	1.295	0.530	4.580	7.167
600	100	39.182	8.203	1.242	0.579	4.000	6.292
100	10	38.517	7.926	1.232	2.721	2.253	2.125
100	50	39.001	8.130	1.221	1.126	2.958	4.917
100	100	38.853	8.087	1.262	0.744	3.070	7.708
100	150	40.093	8.390	1.366	0.962	3.308	7.083
100	200	38.894	8.141	1.226	4.394	2.666	43.667
100	400	35.871	7.864	1.512	5.831	0.009	83.333

	Total Time	Unix Time	Ether Time	Mean Error	Variance	Std Deviation	Pcntg Error
browser	-b	4	-n	600	-s	745	-t 2
esnd	-i	ec2	-b	5	-d	100000	
	39.59109	7.858658	1.226924	0.305532	0.000756	0.027495	98.3333
	40.0398	7.847962	1.2566	0.307824	0.000139	0.01179	99.1667
	40.12668	8.780849	1.264569	0.307654	0.000161	0.012689	99.1667
	40.18174	8.835673	1.264306	0.307608	0.000176	0.013266	99.1667
	39.98482	**8.330786**	**1.2531**	**0.307155**	**0.000308**	**0.01631**	**98.95835**
esnd	-i	ec2	-b	10	-d	100000	
	36.87185	7.861053	1.20687	0.568012	0.005632	0.075047	96.6667

	37.8749	7.80697	1.259122	0.57847	0.000756	0.027495	98.3333
	38.06046	7.807168	1.256656	0.578345	0.000831	0.028827	98.3333
	38.168	7.816045	1.261986	0.578104	0.000893	0.029883	98.3333
	37.7438	**7.822809**	**1.246159**	**0.575733**	**0.002028**	**0.040313**	**97.91665**
esnd	-i	ec2	-b	50	-d	100000	
	39.03774	7.840091	1.145859	2.596875	1.447506	1.203123	62.3333
	39.02909	7.835207	1.169647	2.857758	1.676607	1.294839	62
	39.08869	7.833807	1.224875	2.562201	1.415458	1.18973	59.6667
	39.14919	8.951008	1.218748	2.913515	1.034365	1.017037	52.5
	39.07618	**8.115028**	**1.189782**	**2.732587**	**1.393484**	**1.176182**	**59.125**
esnd	-i	ec2	-b	100	-d	100000	
	38.48512	8.85191	1.128774	0.388508	11.61014	3.407366	12
	39.03061	7.907982	1.362134	1.22905	7.99133	2.826894	3.8333
	39.08782	7.869846	1.266411	0.436605	11.2122	3.348462	9.8333
	39.17351	7.837656	1.281931	1.800895	7.53694	2.745349	2.5
	38.94426	**8.116849**	**1.259813**	**0.963765**	**9.587654**	**3.082018**	**7.04165**
esnd	-i	ec2	-b	200	-d	100000	
	38.13166	7.826977	0.946041	0.25477	26.41755	5.139801	17.5
	39.00349	7.830036	1.39179	1.36029	19.95025	4.46657	3.3333
	39.04271	8.872611	1.329944	1.208933	24.14808	4.914069	3
	39.13974	7.828251	1.395247	0.337411	24.3887	4.938491	9.5
	38.8294	**8.089469**	**1.265756**	**0.790351**	**23.72615**	**4.864733**	**8.333325**
esnd	-i	ec2	-b	400	-d	100000	
	37.40503	7.844291	1.042065	0.338344	25.67249	5.066802	11.3333
	38.15613	7.977453	1.384245	0.146619	8.569885	2.927437	9.3333
	38.222	7.872575	1.284904	1.027907	18.80339	4.336287	2.1667
	38.28812	7.840499	1.466852	0.605747	35.8791	5.989917	5.8333
	38.01782	**7.883705**	**1.294517**	**0.529654**	**22.23122**	**4.580111**	**7.16665**
esnd	-i	ec2	-b	600	-d	100000	
	38.13009	7.979274	0.919375	0.261171	39.92279	6.318449	11.6667
	39.48843	7.948532	1.394311	0.008893	0.000092	0.009592	9.5
	39.51624	8.970324	1.340138	0.740728	10.59442	3.254907	1.8333
	39.59522	7.912356	1.313736	1.303606	41.19556	6.418377	2.1667
	39.18249	**8.202622**	**1.24189**	**0.5786**	**22.92822**	**4.000331**	**6.291675**
esnd	-i	ec2	-b	100	-d	10000	
	38.43019	7.897273	1.204712	1.36122	7.233267	2.689473	4
	38.5564	7.883252	1.218269	3.938494	3.006553	1.733941	1.1667
	38.53122	7.921128	1.261915	3.562463	4.586658	2.141648	1.1667
	38.5486	8.004293	1.242883	2.020376	5.984114	2.446245	2.1667
	38.5166	**7.926487**	**1.231945**	**2.720638**	**5.202648**	**2.252827**	**2.125025**
esnd	-i	ec2	-b	100	-d	50000	
	38.55914	8.984407	1.031319	0.593626	10.60584	3.256661	7.8333

	39.07131	7.847415	1.242994	1.229161	8.002293	2.828832	3.8333
	39.19514	7.836358	1.283992	0.755308	9.926469	3.15063	5.6667
	39.17721	7.853126	1.327373	1.925839	6.73634	2.595446	2.3333
	39.0007	**8.130327**	**1.22142**	**1.125984**	**8.817737**	**2.957892**	**4.91665**
esnd	-i	ec2	-b	100	-d	100000	
	38.38065	8.879659	1.133568	0.381041	10.45613	3.233594	12.1667
	38.86412	7.820124	1.360161	1.173361	7.486949	2.736229	4
	39.07447	7.868247	1.26368	0.420321	10.16419	3.188133	10.1667
	39.09368	7.778917	1.290287	1.001284	9.746546	3.121946	4.5
	38.85323	**8.086737**	**1.261924**	**0.744002**	**9.463454**	**3.069975**	**7.70835**
esnd	-i	ec2	-b	100	-d	150000	
	39.86166	8.956791	1.093249	0.769377	11.34134	3.36769	6.8333
	40.10169	7.877747	1.160033	1.284478	10.01076	3.163978	4
	40.10895	8.905641	1.265435	0.316651	12.63452	3.554508	14.1667
	40.30019	7.817986	1.946601	1.475991	9.907334	3.147592	3.3333
	40.09312	**8.389541**	**1.36633**	**0.961624**	**10.97349**	**3.308442**	**7.083325**
esnd	-i	ec2	-b	100	-d	200000	
	38.08968	7.780122	1.027572	2.839742	5.437692	2.331886	68.1667
	39.04285	8.000494	1.402262	3.766145	6.718619	2.59203	53.8333
	39.14109	7.951802	1.420313	5.531256	8.860176	2.976605	33.6667
	39.30268	8.832082	1.053522	5.440381	7.641499	2.764326	19
	38.89408	**8.141125**	**1.225917**	**4.394381**	**7.164497**	**2.666212**	**43.66668**
esnd	-i	ec2	-b	100	-d	400000	
	33.6428	7.878126	0.989755	5.832992	0.000072	0.008485	83.3333
	35.09202	7.824056	1.681326	5.829561	0.000037	0.006083	83.3333
	36.64092	7.888618	1.687613	5.829736	0.00009	0.009487	83.3333
	38.10905	7.866444	1.68908	5.829879	0.000172	0.013115	83.3333
	35.8712	**7.864311**	**1.511944**	**5.830542**	**9.28E-05**	**0.009292**	**83.3333**

Appendix C : Decoder Box

Appendix C.1. DP83840A Pin Connections Description

Appendix C.2. Decoder Box Schematic

VITA

Theodoros P. David was born in Nicosia, Cyprus, 1971. He received a Fulbright scholarship to study at Washington University in St. Louis. He graduated from Washington U. in 1994, earning a *B.Sc. in Electrical Engineering* and a *B.Sc. in Computer Science*. After working for a year as a Systems Analyst at Card Tech Ltd., he continued with graduate studies at Virginia Polytechnic Institute and State University (Virginia Tech). He is a member of *Tau Beta Pi* (National Engineering Honor Society) and *Eta Kappa Nu* (International Electrical Engineering Honor Society), as well as the *IEEE* (Institute of Electronics and Electrical Engineers).

He will be graduating with a *M.Sc. in Electrical Engineering* (Computer Engineering option) in September 1997. He will be working as a Software Engineer for NCR in Columbia, South Carolina beginning October 1997.