THE ICSI AUDIO LABORATORY

A tool to aid the study of audio transmission

over heterogeneous networks

Roy Chua

May 17, 1996

Contents

List of Figures

Summary

The first part of this report documents a study into the problems involved in establishing audio communication over heterogenous, and possibly lossy, networks like the Internet. In particular, it first provides a description of the problems involved in sending audio over networks. The report then goes on to describe how other researchers have attempted to study and overcome the problems and presents other proposals for dealing with the problems.

The second and third parts of this report describe the design and implementation of a new prototype tool that was created to aid the study of audio transmission over networks. This tool, the International Computer Science Institute Audio Laboratory or IAL, provides its users with a means to quickly evaluate new techniques for processing audio data. It is hoped that this tool will enable researchers to spend less time rewriting code that is irrelevant to their research and to spend more time on researching resilient coding and audio processing techniques.

Acknowledgements

First of all, I would like to thank Professor Domenico Ferrari, my research advisor, for his time, patience and advice.

I would also like to express my utmost gratitude to Dr. Andres Albanese of the International Computer Science Institute (ICSI) for his time, energy and guidance in supervising and supporting my project at the ICSI.

I want to express my appreciation to Professor Michael Luby of UCB/ICSI for teaching me about Priority Encoding Transmission and for supporting my research through the PET grant.

Many thanks must go to Dr. Rainer Storn of Siemens/ICSI. Rainer provided me with all the necessary signal processing information and code, all of which were essential to getting my project up to speed and working. His advice has been invaluable and the IAL is as much a product of his efforts as it is mine.

I thank Hartmut Chodura (Fraunhofer Institut für Graphische Datenverarbeitung/ICSI) for introducing wavelets to me and for providing me with the code to his "Chatty Box". Many of the code routines, in particular, the wavelet filter, in the ICSI Audio Laboratory (IAL) have their origins, in one form or another, in "Chatty Box".

Kudos to Steve McCanne and Van Jacobson, of Lawrence Berkeley Labs (LBL), for writing a great audio tool like VAT (Visual Audio Tool) and for making available the source code to the public [20]. Many of the Tcl/Tk [22] mechanisms in the current version of IAL are based on the code from VAT, and the IAL is currently being modified to use both the network code and audio from VAT – we hope that this will enable multicast support and provide better portability to other platforms.

I

Background

1 Introduction

In the early days of large wide-area networks like the Internet, the primary form of communication was electronic mail. Presently, with greater bandwidths, there is a move towards using the Internet for other types of communications. Despite the creation of the MBone [9, 18], the multimedia backbone, multimedia communication over the Internet is at best limited. Some of the reasons for the slow acceptance of multimedia communication include bandwidth limitations, the presence of packet losses on the Internet and high variability in traffic conditions. These conditions often result in difficult to predict loss rates and high jitter rates.

Bandwidth limitation will slowly become less of a problem as the backbone of the Internet is upgraded and users switch to higher speed offerings; for example, moving from T1 to T3 trunk lines, upgrading from a 28.8Kbps modems to 128Kbps ISDN BRI lines. However, many believe that in the near future, packet losses and jitter will still be a fact of life on the Internet. Hence, researchers continue to work on schemes that allow multimedia communication through the tolerance of losses. For example, the researchers at the International Computer Science Institute have developed one such resilient scheme, called Priority Encoding Transmission (PET). [1].

With such schemes in place, and with the judicious application of other techniques, multimedia communication over a lossy network like the Internet can become a reality. For example, consider the many "Internet Phones" or "Web Phones" that have appeared on the market in recent months. Or the increasing popularity of RealAudio™, a standard for sending unidirectional audio streams over the Internet at low (14.4Kbps or 28.8Kbps) rates. No doubt, applications like these are still plagued by many of the problems mentioned earlier, as users of these tools will attest to. However, their proliferation demonstrates the feasibility of such kinds of communication.

As mentioned earlier, there are many applications that allow two-way or multi-party (in multicast mode) communication over the Internet. If we were to focus on those that run on multiple platforms and are freely available, we would come up with, to mention some of the more popular ones, candidates like VAT (Visual Audio Tool) from LBL (Lawrence

Berkeley Labs), CU-SeeMe from Cornell University, IVS (INRIA Video Conferencing System) from INRIA (Institut National de Recherche en Informatique et en Automatique) and RAT (Reliable Audio Tool) [12] from UCL (University College London). In particular, VAT, from LBL is part of the standard set of applications that are used over the MBone.

Several of these applications have introduced different techniques that deal with some of the problems associated with audio transmission over the Internet. For example, RAT uses a piggybacking scheme, in which a low-resolution version of a previous audio packet is piggybacked on the current packet being sent, and IVS has a prototype of a scalable CODEC with an adaptive feedback mechanism. Hartmut Chodura of the Fraunhofer-IGD is working on a multi-platform digital telephone application [5] that will incorporate resilient coding (PET) as well as digital signal processing techniques that deal with the non-transport problems associated with audio communication over networks (especially long haul networks).

However, we have yet to see a definitive study of the myriad parameters that affect two-way or multi-party audio communication over the Internet. Furthermore, there does not exist a comprehensive toolkit that allows such studies to be done in an efficient manner. The ICSI Audio Laboratory (IAL) is a tool that has been designed from the ground up to fill this niche. A prototype of the IAL was designed and built at the International Computer Science Institute as part of the author's research in the PET group. Before we can embark on a detailed discussion of the design of the IAL, it is necessary to first explore the various problems involved in audio communication over networks.

2 Problems

There are different types of problems involved in achieving good two-party or multi-party audio communication over both short-haul and long-haul networks. In general, these problems are either network-based (i.e. they exist because of the underlying transport architecture) or non-network-based, like the problems with echos and feedback howling. Most of the research in audio communication over networks have focussed on attempting to combat the loss of packets during a session. However, before good audio communication can be carried out, other types of problems have to be investigated and dealt with. Most of the problems we will focus on involve mainly two-party communication. We believe that the results should extend to multi-party (more than two) communication. The following, though not an exhaustive list, are some of the more common problems that affect two-party communication over a network.

Loss of packets. When transmitting over networks like the Internet, or even over local area networks, losses will inevitably occur. Obviously, one could use a transport protocol that would retransmit whenever losses occurred. However, the use of such protocols (like TCP over IP) usually results in increased latency (on the order of round-trip times) and the lack of application control over the handling and detection of losses. It is for this reason that most audio communication software use datagram packets (for example, UDP over IP) as their preferred mode of transport. In this way, losses can be detected and dealt with if necessary, and the application can decide whether retransmission is both necessary and tractable.

Bolot studied the characteristics of end-to-end packet delays and losses in the Internet [3]; and more recently, in his joint work with Crépin and Vega Garcia [4], he dealt with an analysis of audio packet loss in the Internet. Both studies were neither long-term studies (in [3], tests lasted only 10 mins) nor comprehensive. For example, only one packet size was used in both studies (32 bytes in [3] and the IVS default fixed size of 320 samples per packet in [4]). In the second study, at most two samples were taken on one particular day for each of their two test routes. Bolot concluded that the number of packets lost was small and randomly distributed when the network load was low and the number of consecutively lost packet increase as the network becomes more loaded. These results form the basis for RAT's technique for dealing with packet loss. To do a quick validation of Bolot's studies, a few quick experiments were done at ICSI using a UDP echo client program written by the author. The results are similar to Bolot's and without further extensive test results that prove otherwise, it appears that Bolot's results are justifiable. The lack of extensive studies of audio packet losses over the Internet (and over local area networks) further highlights the need for a flexible audio tool that allows the setting of various network parameters like packet length, adding source routes to packets etc.

In any case, packet loss is a major problem in audio communication over networks and it will have to be dealt with if users are to have a better than mediocre quality of communication.

Large delays. Over long-haul links, like network connections between the west coast of the United States and countries in Europe, or between the US and Asia, long delays abound. Researchers in the field of audio communication have indicated that humans can tolerate delays of between 150-300ms in a two-party conversation. When faced with delays of more than 180ms to Europe and on the order of 800ms to some countries in Asia (for small sized packets of 32 bytes), it is clear that there is not much room left for processing overhead. In such cases, we would want to keep the size of the audio packet as small as possible, to minimize delays. Of course, when the round-trip-time (RTT) is much less, we can increase packet sizes and do more processing to create better audio quality. Therefore, a good audio application should be able to adapt to the RTT and provide the best quality given that.

Jitter in packets. Audio sampling is usually done at a fixed frequency and hence, audio packets are sent out at regular intervals (assuming the sending workstation is not too heavily loaded). However, at the receiving end, packets do not come in with fixed delays. This variation in packet arrival delays (jitter) is a further challenge that must be overcome to provide good audio quality. One way would be to slowly adapt to this jitter by introducing small, almost unnoticeable delays before playing samples from a new talk spurt (humans seldom speak continuously in normal conversation). For a continuous audio stream, this would obviously not be possible, so the rate of playback will either have to be varied slightly over time or, if delays are not too large, introduce a "delayed playback buffer" (i.e. we buffer a certain number of packets before we start playing back the stream) at the start of playing the stream and hope that this "buffer" is sufficient to counteract the jitter during the course of playing the stream. VAT incorporates some means by which to adapt to the jitter,

and a sophisticated audio tool should have this ability as well.

Audio quality and bandwidth. Telephone quality (toll-quality) speech can be obtained with a bandwidth of about 3KHz (300 Hz to 3.3 KHz). An 8KHz sampling rate (deemed sufficient by the Nyquist theorem to deal with a signal of less than 4KHz bandwidth), using 8 bits per sample (PCM), or using 12 bits per sample and using compression to get 8 bits (A-law or μ-law), usually provides toll quality speech on workstations. Using 16 bits per sample (and/or increasing the sampling rate) will provide better quality sound. However, the increase in quality comes with a price – an increase in bandwidth. The increase in bandwidth could result in either more packets being lost if the network becomes too loaded, or in having less bandwidth available for other users or applications. Of course, compression and better coding could be used to reduce the required bandwidth. Whatever the case, the bandwidth should be increased to a point where the quality of the audio, as affected by the bandwidth, loss rates and delays, is optimal (we could also take into account other applications or users who may be sharing the link).

Echos. In a two-party communication with some delay, there is always a problem with echos when using certain types of input and output devices. For example, the normal workstation configuration of an omni-directional microphone and a workstation speaker results in bad echo problems. This happens when the signal that is generated at one user's end, and played back at the other end, is captured by the far-end recording device and sent back to the user who generated that signal. This phenomenon is perceived as an "echo" of one's speech and can be extremely annoying. The greater the round-trip delay, the more disturbing the echo. To deal with this, we can employ a few different techniques, including echo suppression and echo cancelation, both of which will be discussed in greater detail in a later section. Obviously, if all users were to use headphones and good quality microphones, we would not have an echo problem. This is, however, an unlikely scenario since headphones and hand-held microphones are cumbersome and uncomfortable, to say the least.

Howling. The howling problem occurs under the same scenario as the echo problem. When gains are turned up on the recording device (microphone) and the playback device, there is the possibility that one will introduce a feedback loop that results in signals at certain frequencies building up energy. This creates a howling sound that is extremely annoying. Most users usually keep gains below the threshold at which howling occurs, but sometimes, certain loud sounds (either speech or background noise) can trigger a howling feedback loop.

Computational resources. This is usually ignored as being a factor affecting the quality of audio communication, but it is an important one, as anyone who works on real-time signal processing will know. If we are to digitally process the audio signal to obtain some desired effect (like less bandwidth through compression), we have to ensure that the problem is computationally feasible in real-time, on the average workstation. A user seldom runs a single task on a workstation and there will inevitably be contention for workstation resources like memory and CPU time. A sophisticated audio tool could watch the workstation load and reduce its coding complexity to obtain optimal audio quality given the limited resources.

3 Proposals to examine

We will next examine proposals that deal with some of the above problems. A few of these proposals have already been implemented in some of the freely available tools mentioned earlier. Furthermore, many of the commercial "Internet Phone" products will surely include one or more of these techniques. However, since these are commercial products, their source code is not freely available and their techniques for producing good quality speech communication are often guarded trade secrets.

3.1 Coding for efficiency

There are numerous different standards for voice and audio encoding available today. Most of them provide the means to achieve good quality voice encoding at various rates. Of course, for audio encoding (as opposed to just voice encoding), we will have to use different kinds of coding algorithms, or simply transmit high-bit rate signals. For the moment, we shall concern ourselves with mainly speech encoding techniques. These techniques provide the means to get better quality speech using less bandwidth, although they increase computational complexity. An audio tool should provide some means to evaluate the different encoding and to determine if they are useful.

Pulse Code Modulation - PCM

Pulse code modulation involves sampling a given signal at some fixed rate and then quantizing the signal by assigning the closest fixed quantization level for each sample. Using 8-bits, we can get up to 256 quantization levels. The more bits we use, the greater the quality (the less the quantization errors). Usually the step size between quantization levels is uniform. However, we can use techniques that allow us to use non-uniform step sizes to get better signal to noise (distortion) ratios by exploiting known signal traits. For example, the μ-law and A-law systems both amplify low-volume signals more than high-volume signals. These techniques use a sampling levels of 12 bits and compress them down to 8 bits. Standard 8-bit PCM, or μ-law/A-law signals therefore take up 64Kbps – the nominal rate for toll or telephone quality speech. The sampling rate or quantization levels may be increased if we need better quality audio (for example, CD-quality audio can be obtained at 16bit and 44.1KHz sampling rates). Variations of PCM include DPCM (Differential Pulse Code Modulation), and ADPCM (Adaptive DPCM). These are predictive coding schemes that exploit the correlation between neighboring samples of input signals to compress the signal. For toll quality speech, we need 64Kbps for PCM. With ADPCM, we can get almost toll quality speech with just 10-40Kbps of bandwidth.

Delta Modulation - DM

Delta modulation is sometimes defined as a 1-bit version of DPCM. Basically, it attempts to approximate a signal by using a series of linear segments of constant slope. It produces a staircase like output signal that tracks the sampled signal. Adaptive Delta Modulation (ADM) can obtain almost toll quality speech at bandwidths of 10-40 Kbps.

Vocoder and Linear Predictive Coding - LPC

If we only want good quality (or recognizable) speech, we can turn to vocoders and linear predictive coding (LPC). Vocoders model the speech generation process by using an excitation signal that models the way vocal cords modulate air pressure and a filter that characterizes the vocal track. LPC vocoders model the vocal tract by using a single, linear all-pole filter. The actual operations of vocoders are beyond the scope of discussion of this report and the interested reader is referred to speech processing texts for further discussions. Suffice to say that these techniques require complex algorithms and a substantial amount of computation time, but they result in good quality speech at very low bit rates, on the order of 1-15 Kbps.

3.2 Coding for resilience - Priority Encoding Transmission and Wavelets

Priority Encoding Transmission, first introduced in [1] and described briefly in [15] is an efficient Forward Error Correction scheme that deals with different levels of priority. In essence, it provides its users with the ability to set different loss probabilities on different portions of their data, allowing for the graceful degradation of a data stream. It has been demonstrated that a PET protected MPEG stream could result in less distorted pictures in the face of network losses [17]. Subsequently, a new version of the PET algorithm [2] and an MPEG CODEC was integrated into a version of VIC [19], the standard video conferencing tool used on the MBone. The resultant software [16] showed that PET protected MPEG video was less distorted and resulted in subjectively better video quality when there were network losses.

In video, there are a few coding standards which are natural for PET protection. PET works when the data to be protected has inherent priorities. For example, in MPEG, there are I, P and B frames, listed in order of importance. Similarly, in H.261, there are means to prioritize the traffic. However, the standard coding algorithms for audio do not easily lend themselves to prioritization. It is difficult, for example, in basic PCM, to decide which samples or even which bits are more important than others. One could presumably use the energy of small mini-packets and use that as a priority, but it is unclear that we will see much better quality audio. It was therefore proposed that other coding techniques be used for audio.

One such coding technique is the wavelet transform. Wavelet transforms (introductory papers:[11, 29, 31]) are mathematical functions that allow one to break data down into different frequency components and then to study each component with a resolution that is matched to its scale. They overcome the time-frequency resolution limitation of short-time Fourier transforms (i.e. taking the transform of a temporal window of a time varying signal). Given a non-stationary signal $g(t)$, the wavelet transform may be defined as[13]:

$$WT(\tau, a) = |a|^{-1/2} \int_{-\infty}^{\infty} g(t) \cdot \psi^* \left(\frac{t-\tau}{a} \right) dt$$

where ψ^* is a basis function that belongs to the two parameter family of basis functions denoted by:

$$\psi_{\tau,a}(t) = |a|^{-1/2} \psi \left(\frac{t-\tau}{a} \right)$$

and a is a *scale factor* or *dilation parameter* and τ is a *time delay*. The basis functions $\psi_{\tau,a}(t)$ are wavelets that are derived from a prototype $\psi(t)$ called the mother wavelet. Figure I.1 shows some families of wavelet functions. The Haar wavelet is the simplest wavelet, often used for educational purposes. It should be noted that the wavelet transform is actually a subset of a more versatile transform, the wavelet packet transform [7].

Wavelets (wavelet transform and wave packet transform) have been used in numerous signal processing applications [8], including fingerprint compression and data de-noising [11] . Furthermore, with fast algorithms that implement fast wavelet transforms [6], using wavelet transforms for real-time data like speech and video streams is feasible. Where audio coding is concerned, wavelets have been used successfully to get significant compression with good quality for both speech [30] and high quality monophonic CD signals [25]. For instance, discarding small wavelet coefficients for speech signals resulted in compression ratios between 1.3 and 14.3 depending on the length of the conjugate mirror filter used in the wavelet transform (the longer the filter, the better the compression, but the greater the computational complexity). In the case of [25], through the employment of wavelet best basis selection, Sinha and Twefik managed to achieve almost transparent coding of monophonic CD quality signals (16-bit PCM, sampling rate of 44.1KHz – bit rate of 705.6Kbps) at bit rates of 64-70Kbps. When dynamic dictionary encoding and psycho acoustical perceptual masking techniques were added, the rate went down to 48-66Kbps.

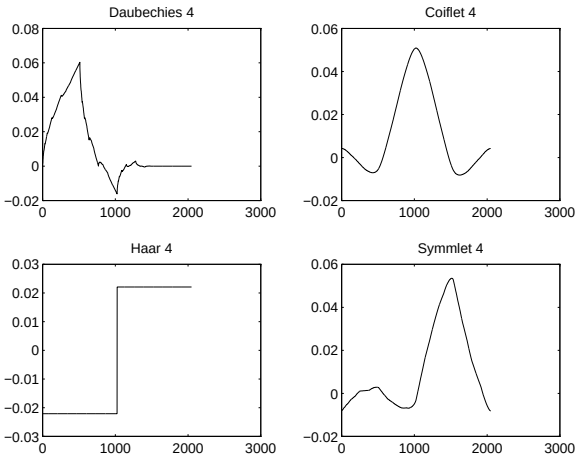

Figure I.1: Families of Wavelet Functions

The above alone would prove sufficient justification to use wavelets as our encoding function. Nevertheless, our main reason for using the wavelet transform is that the magnitude of the wavelet transform coefficients provide the necessary prioritization. The greater the magnitude of the coefficient, the greater its importance. Preliminary studies of compression through truncation of small magnitude coefficients (by Hartmut Chodura and Martin Isenburg of Fraunhofer-IGD/ICSI and Rainer Storn of Siemens/ICSI) have yielded promising results. PET can then be applied to the wavelet coefficients, putting more redundancy on the larger magnitude coefficients. As an added feature, we can also factor in psycho acoustic effects [21] into our calculation of priorities. Our aim in calculating priorities will be to achieve a prioritizing algorithm that is in direct correlation with perceived audio quality, while still working under the resource constraints of an average workstation.

It is unclear how much better this approach will be over the RAT approach of piggybacking an LPC packet on a PCM packet. We believe, though, that PET with wavelets allows a more graceful degradation when network losses occur and that the computational complexity should be no more than that required to implement LPC.

3.3 Howl Removal

Howl removal can be achieved by a variety of methods. One method, which has been used by the telephone industry, is to use a frequency shifter to shift the transmitted signal up by some small, almost imperceptible amount. This shift has to be large enough to ensure that energy cannot build up at any particular frequency. By not allowing this energy to build up, we effectively eliminate the howling effect. It would be useful to determine the effectiveness of such a technique, and to study, in frequency space, the characteristics of howling.

Fourier Analysis

Fourier analysis of an input signal is a means by which we can view the frequency components of the signal. There are packages available on workstations and on personal computers that allow users to view the frequency make-up of a time-varying signal. However, it would be useful to have an audio tool which would enable the user to perform Fourier analysis in real-time, while data were being transmitted across the network. Many different Fast Fourier Transforms (FFT) algorithms are available and they operate fast enough to enable real-time Fourier analysis.

3.4 Echo Cancelation

Echo cancelation is not an easy process. It requires a large adaptive filter which attempts to model the acoustics in the room in which the speaker is located. To cancel the echo, the output signal (going out through the loudspeaker) is processed through the adaptive filter, and the result is subtracted from the microphone input (see figure I.2). In the ideal case, the filter works correctly and the echo (signals from the loudspeaker that are picked up by the microphone) is removed.

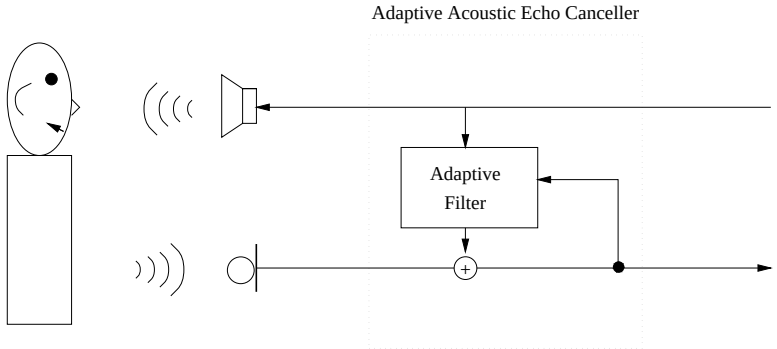

Figure I.2: Echo Cancelation Block Diagram

However, in real life, the adapting of the filter is a complicated and computationally intensive process. If successful, though, the quality of two-party communication is substantially improved. Most echo cancelation in audio conferencing is done with the aid of

high speed digital signal processors (DSP). For instance, PictureTel has an excellent echo cancelation algorithm that works on a low-cost Texas Instrument DSP chip. With workstation computation power growing exponentially though, it might be possible, in the near future, to implement a decent echo canceler purely in software. In that respect, Chodura and Storn [5] have proposed a degradable echo canceler, which adapts to the processing power available and attempts to provide the best echo cancelation given the computation constraints. A mechanism, similar to their proposal, should be included in a high-quality audio tool.

3.5 Echo Suppression

If a means to deal with echos is required, and computational resources prove not readily available, echo supression may be used instead of echo cancelation. In most echo suppression schemes, instead of varying the microphone or loudspeaker gains, the microphone or loudspeaker is completely shut off when there is an incoming or outgoing signal (priorities can be set so that the appropriate device is shut off). This in effect creates a half-duplex channel. In our proposed scheme (figure I.3), Rainer Storn (Siemens/ICSI) has suggested that instead of shutting off the signal, the gain may be modulated.

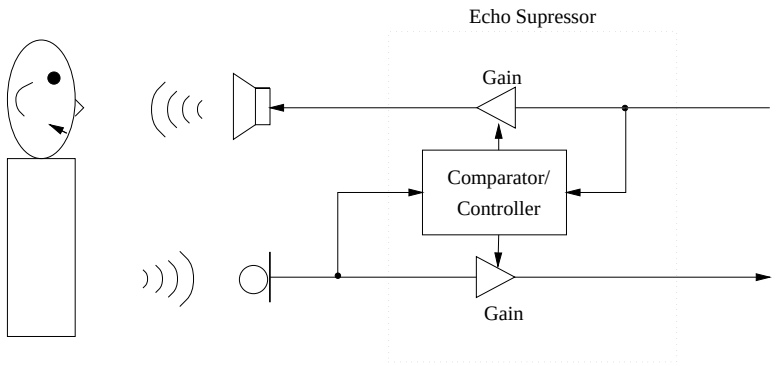

Figure I.3: Echo Suppression Block Diagram

We believe that this will result in more natural two-way communication, and prevents irritating speech clipping effects. This technique, when used with silence detection (not sending data when the input signal is below some threshold), should be able to deal with echos (though not as satisfactorily as echo cancelation) and even eliminate howling to some extent.

3.6 Adaptive Feedback Mechanisms

There have been several studies and proposals for adaptive coding and transmission algorithms that will result in better multimedia quality. For example, in [10, 14, 27], methods are proposed for producing better video and audio streams while taking into account CPU

or network loads. It would be useful to implement some form of these adaptive algorithms and see if they are useful, feasible and result in more efficient audio transmission.

Storn [26] proposes a fuzzy controller to adapt the PET-MPEG priority scheme to the network conditions. A similar approach could be taken with PET protected audio (whether through the use of wavelet encoding or other forms of encoding). If it were possible to generate output with RTP ([24]) headers, we could use RTCP to get feedback on the current state of the network and adapt our output accordingly. In any case, RTP is becoming a standard ([28]) on the MBone and within reasonable limits (assuming that it makes sense to use RTP with the audio encoding format), any audio tool (especially those that aim to support multicast) should aim to support this standard.

4 The Need for a Tool

Given the above proposals, some of which have been implemented, or are being implemented, and others which are not yet implemented, it would be useful to have a tool which enabled the quick implementation and evaluation of each of the proposals in a standardized environment. This tool should not be merely an audio communication tool, like RAT, IVS or VAT, but should allow users to quickly prototype different approaches to audio communication without dealing with irrelevant coding.

For instance, a signal processing researcher might be interested in trying out a new audio encoding format and wants to replace the current encoder with her/his new format. Or, a networks researcher might want to modify the audio packet parameters to see how changing packet sizes, or using variable packet sizes can affect packet loss rates and patterns. This tool should function effectively as a kind of audio laboratory, where users can simply plug-in new encoders, decoders, or change the network control functions, quickly and efficiently.

With the above in mind, we set about designing such a tool at the ICSI.

II

The ICSI Audio Laboratory

1 History

The ICSI Audio Laboratory or IAL is an ongoing project at the International Computer Science Institute (ICSI) in Berkeley. The aim of this project is to provide a tool for the network and signal processing researchers to quickly evaluate different approaches to audio coding.

The IAL had its origins in a program called "Chatty Box", written by Hartmut Chodura of the Fraunhofer-IGD during his stint at ICSI. Originally, the program was written to test the feasibility of wavelet encoding of audio streams. The program was subsequently modified, by this author, for robustness and to incorporate code to perform howl removal through frequency shifting. The howl removal code in "Chatty Box" was the result of Rainer Storn's research and filtering routines. Later on, extra functionality was added to this program to perform FFTs and to display a real-time FFT of the input signal. During the time that research was being done to combat the problem of howl removal in the frequency domain, we realized that it would be more efficient to have some kind of tool into which we could quickly plug in filters that we had designed and evaluate the results immediately. This would free us from the bother of attempting to integrate new code into "Chatty Box". It was at this time that the idea for developing the IAL was born.

2 Overview

The IAL consists of code written in both C/C++ and Tcl/Tk. The initial version of the IAL used slightly modified audio code and network code from "Chatty Box" (the audio and network code had been previously obtained by Hartmut from other sources and modified). At present, work is underway to port IAL to use some of the audio and network code from VAT. In addition, we are adopting the Tcl/Tk classes and mechanisms for integrating Tcl/Tk into C++ used by VAT. We believe that by using more of the code from VAT, we will be able to quickly port the IAL to the many platforms that VAT currently supports (SunOS, Solaris, Linux, etc).

The IAL was designed, from day one, to be "user-friendly". We want the researchers who use our tool, many of whom may only be familiar with writing numerical routines in C, to able to create new "filter plug-ins" for the IAL without understanding the internals of the IAL. However, the IAL should also support those who need a great deal of control over the audio transmission process. In order to make this a reality, we had to create strict high-level interfaces for the "filter plug-ins", while exposing a great deal of internal state to those who would be modifying the audio module or the network module. Figure II.1 provides a good overview of how the IAL is structured. Observe that the IAL is divided

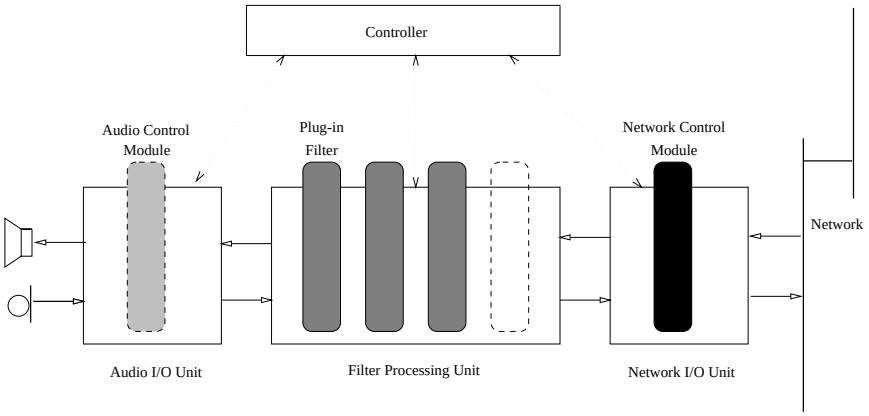

Figure II.1: IAL Block Diagram

into three different parts. The Audio I/O Unit, the Filter Processing Unit and Network I/O Unit.

The Audio I/O Unit is responsible for all audio control functions, including audio input and output, fixing the sampling rate, dealing with device problems like overflows and underflows. In addition, there is the provision to insert an Audio Control Module into the Audio I/O Unit that performs special tasks, like echo cancelation, or echo suppression. The rationale behind delegating such tasks to the Audio Control Module (ACM) is that such techniques require intimate knowledge of the audio driver and have to be closely coupled to the audio devices. For example, an echo cancelation routine would need to know when overflows or underflows occur and it must not perform cancelation, or update the filter weights when this happens.

The Filter Processing Unit consists of a bank of filter plug-ins. These plug-ins perform two major functions – encoding and decoding. The data stream, on its way from the Audio I/O Unit to the Network I/O unit, will pass through each of the plug-ins and the signal will be processed accordingly. The plug-ins in the unit are managed by the Filter Manager and the plug-ins may be attached or detached on on the fly (i.e. when the program is running). Furthermore, these plug-ins can be activated or deactivated when the program is running. The major difference between attaching/detaching and activating/deactivating is that in the former case, the user may reorder the filters or add new filters, and in the latter, the user can only choose whether to bypass the currently attached filter. Also, statis-

tics that are gathered about a filter (like time per execution) will not be retained between detachment and attachment.

The Network I/O Unit deals with the sending and reception of packets over a network. Like in the Audio I/O Unit, We also provide the ability to insert a Network Control Module (NCM) into this unit. The NCM may provide coding that deals with adding redundancy (like PET) and may additionally contain routines to do packet spreading and striping. Once again, our rationale for having such tasks done in the NCM, instead of the Filter Processing Unit, is that these tasks will need to interact closely with the network, like dealing with packet losses, etc.

To get a sense of all all the units tie in together, it would be instructive to observe the data flow across the units:

- When there is sufficient data in the audio buffer, the system generates an interrupt which wakes the audio handler within the Audio I/O Unit. The handler then copies the data into a global set of buffers and calls the central controller to tell it that data is ready. The central controller invokes the filter manager to do all the necessary encoding of the audio frame. In encoding the audio frame, the filter manager marks the packet with the unique filter-ids of the filters that the packet was processed through (to enable proper decoding at the receiving end). The controller then calls the Network I/O Unit to send the data out through the network.

- When a packet arrives, the handler code in the Network I/O Unit executes. If a Network Control Module (NCM) is present, it will be allowed to deal with the incoming packets and to call the central controller when it has built up an audio frame. Otherwise, the Network I/O Unit calls the central controller directly. The controller then calls the filter manager to do all the necessary decoding of the audio frame and then hands the data to the Audio I/O Unit which then places the data into the speaker buffer and has the audio device driver playback the data.

We will next describe the user interface structure of the IAL before moving on to describe each of the abovementioned units in greater detail.

3 User Interface

Given the modular design of the IAL, it was necessary to provide a way by which one could generate independent user interfaces that controlled each of the different modules.

Our user interface paradigm is described in figure II.2. For each unit in the IAL, there is a corresponding, independent graphical user interface (GUI) written in Tcl/Tk. All of the user interfaces in Tcl/Tk are written to interact directly with the corresponding object classes written in C++. By doing this, we limit interaction between classes and reduce code complexity.

For the Audio I/O Unit, there is an Audio Control Console (ACC), which provides both basic user interface (UI) support for simple audio functions, like choosing which input and output devices are to be used and their I/O gains, as well as added UI support for the Audio Control Module (ACM), if it is present. For more information, see section 5 (page 28), which describes the ACC and ACM in greater detail.

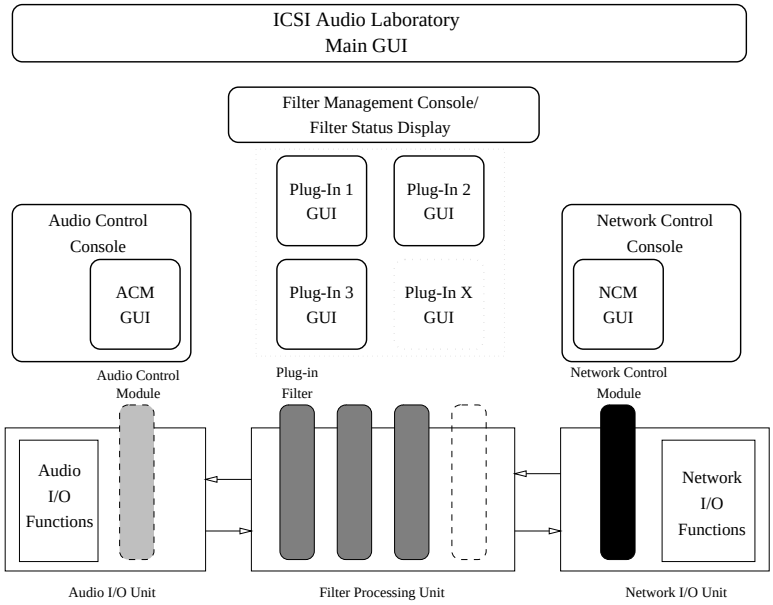

Figure II.2: IAL User Interface Design

The Filter Processing Unit has its Filter Management Console (FMC) and Filter Status Display (FSD). The FMC liases with the C++ Filter Manager Object to enable the user to attach and detach filters on the fly, while the FSD works with the Filter Manager to manage each independent filter plug-in. The FSD also controls whether a particular filter's GUI is displayed or hidden. Note that the designer of a filter need not create a GUI if she/he does not feel a need for one (or does not know enough Tcl/Tk to write one or modify a given UI template).

The Network I/O Unit's UI is similar to that of the Audio I/O Unit. The Network Unit has a Network Control Console (NCC) which also provides simple network functions (this will be expanded when the port to the VAT network code is complete) like selecting the network mode, and in addition, provides support for the Network Control Module (NCM) if it exists.

In addition to the above UIs, there is a main control interface (figure II.3) from which a user might pop-up or hide a certain unit's GUI (II.3-pointer 1) or set certain global parameters (II.3-pointer 2). Right now, the only global parameter supported is the audio frame size, measured by the number of samples per frame. Observe that the frame sizes are all powers of 2. This is due to the fact that the FFT and Wavelet Filters all require input arrays that are of such sizes.

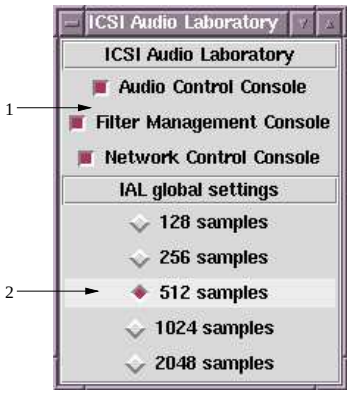

Figure II.3: IAL Main Controls Screen Shot

4 Filter Management

The Filter Processing Unit and the Filter Manager are an important part of what the IAL does. In fact, if we were to strip away the ACM and the NCM, the IAL would still fulfill one of its main goals – that of providing an easy way for researchers to efficiently evaluate new audio processing techniques (i.e. test new audio filters).

The use of Tcl/Tk in the IAL was by no means an accident. Before designing the IAL, it was decided that filter registration should be a run-time process. By having a run-time interpreter built into the IAL executable, it is possible to load new filters in at run time and attach and detach them on the fly. To add a new filter to the IAL, the user simply has to provide two object (`.o`) files which are linked into the IAL executable. One of these will be the compiled C++ filter code itself, and the other object file will contain a static string that is the Tcl/Tk code associated with the filter. This Tcl/Tk code will register the filter (the registration code will execute when IAL gets loaded up, before the UIs are built) and its corresponding user interface. The UI is optional and the user may choose not to provide one. It is highly recommended though, that each filter provide associated UI code, so that its parameters are easily accessible to the user at run-time.

The Filter Manager (FM) in the IAL consists of both a C++ class and associated Tcl routines. The Tcl routines will, at run-time, determine what filters are available to the FM. These routines will communicate with the C++ class as necessary, to allow the attachment and detachment of the filters, and to control the activation and deactivation of these filters. Obviously, when the filter manager is attaching and detaching filters (short time periods), data that is sent for processing will not be processed – i.e. it will be sent to the network without additional processing.

To interact with the filter manager, the user utilizes the Filter Management Console (FMC) user interface. The FMC provides the user with an interface to attach and detach filters, to rearrange filters, as well as control individual filter settings (through the Filter Status Display portion of the FMC). The interaction between the plug-ins, their GUIs and the Filter Management Console is depicted in figure II.4. The FSD provides fine-grained

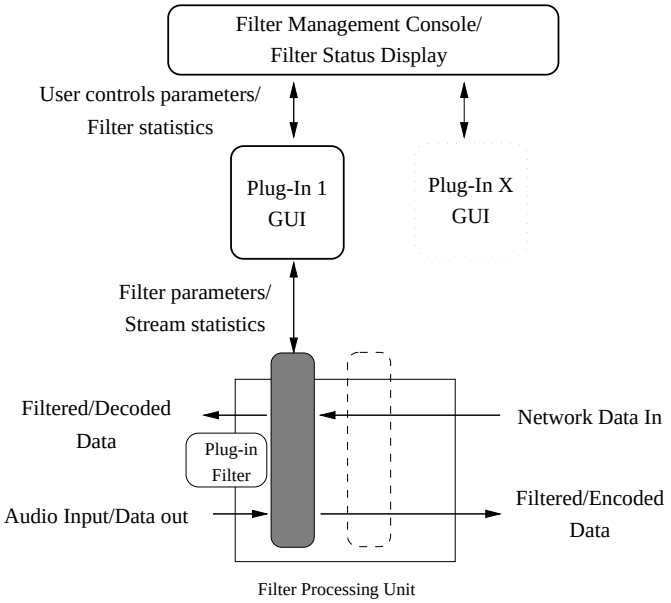

Figure II.4: Filter Management Console Block Diagram

control of each filter. By clicking checkbuttons, we can inform the Filter Manager of our preferences, and we can also call up the appropriate GUI for the filter that we are interested in.

To provide the reader with a clearer picture of the FMC, we provide an on-screen snapshot of the FMC in figure II.5.The FMC user interface can be divided into 7 sections:

① **Attach/Detach Buttons.** These buttons allow the user to move filters in between the two lists – the list of available filters, and the list of filters to be attached. Note that the filters in the "Attached Filters" list are not considered to be attached until the "Update Filter List" button is pressed.

② **List of Attached Filters.** This shows one of two things depending on the status of the "Update Filter List" button. If that button is disabled (greyed out), then this is a list of the attached filters, in correct order. Otherwise, this displays a list of filters to be attached when that button is pressed. The user can change the order of the filters by holding down the left mouse-button on a filter and then moving the filter up or down the list. Note that only one item on this list may be selected at any time.

③ **Filter Status Display.** This is the in-depth display of an individual filter and is obtained by clicking the middle mouse-button on an attached filter. It is discussed in greater detail in the next paragraph.

④ **Update Filter List Button.** This button is active if the "List of Attached Filters" has been modified in any way (i.e. if a filter was attached, detached, or if the ordering

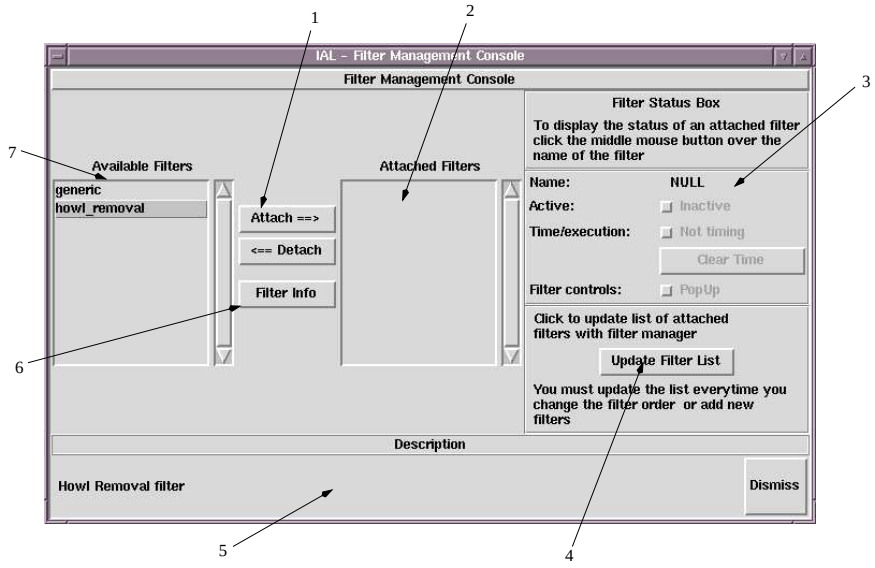

Figure II.5: Filter Management Console Screen Shot

was changed). When the button is active, clicking on it results in the detaching of all previously attached filters, and will force the Filter Manager to attach, in order, the list that is currently displayed.

⑤ **Filter Description.** This provides a short description of the filter currently selected. If there is more than one selected filter, then the information displayed here will be for the last filter that was selected.

⑥ **Filter Info Button.** Pressing this will pop-up a small window that provides a more detailed description of all the filters that are currently selected.

⑦ **List of Available Filters.** This is a list of all the filters that were loaded into the IAL and registered. A user can attach one or more of these filters by using the left mouse button (in conjunction with the shift or meta keys for extended selection, if so desired) to select the filter(s) and clicking the "Attach" button.

Figure II.6 shows both the filter information display and the filter specific UI for the Howl Removal Filter (described in the next chapter) as well as the Filter Status Display UI. The different parts of this figure are described below:

① **Filter Information Pop Up.** This is the more detailed description of the Howl Removal filter. This pop-up is the result of selecting the Howl Removal filter and then pressing the "Filter Info" button as described in the earlier list.

② **Filter Name/Activate Filter Checkbutton.** "Name" shows the name of the filter that has been selected for display in the FSD. The "Active/Inactive" checkbutton allows

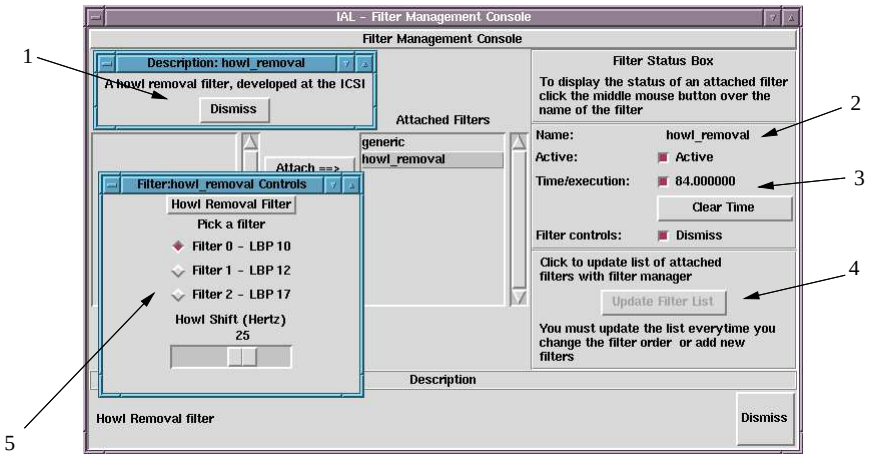

Figure II.6: Filter Status Display Screen Shot

the user to activate or deactivate a currently attached filter. A filter that is inactive will not be invoked by the filter manager during the encoding process. This is used to provide quick feedback to the user about how her/his filter affects the audio stream.

③ **Filter Timing Checkbutton.** When this button is active, the filter manager will time the execution of the filter. The time per execution, in milliseconds will be displayed here. Currently, we only time the encoding process, but are making changes to support timing for the decoding process as well.

④ **Update Filter List Button.** Note that in this screen-shot, this button is inactive. It means that the user, after attaching the list of filters (by using the attach function and registering her/his request through pressing this button), has not modified the "Attached Filters" list.

⑤ **Howl Removal Filter UI.** This is the user interface that is provided by the Howl Removal filter. By using the "Pop-Up" button on the FSD, we can control whether this window is displayed or not. Note that it contains controls specific to the Howl Removal filter.

Having finished our in-depth discussion of the Filter Manager and its associated supporting functions and UIs, we will move on to the audio and network control aspects of the IAL. At this time, we will defer the discussion of the actual plug-in filters to the next chapter.

5 Audio Unit

The Audio I/O Unit (figure II.7) provides all that is necessary to deal with the audio control portion of the IAL. This includes the audio driver that is operating system and hard-

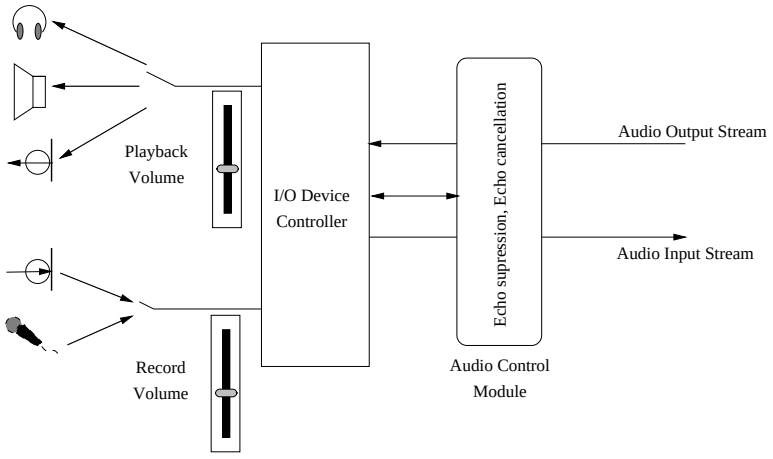

Figure II.7: Audio I/O Unit Block Diagram

ware specific, as well as a group of I/O device functions that interface with the driver. At present, we only support the Solaris/Sparc environment and the functions are specific to that environment. However, we hope that when the code completes its migration to the VAT audio drivers, we will have system independent I/O code that can interface with various drivers on various operating systems and hardware platforms.

Presently, the unit provides the ability to choose among the available playback and input devices and it also allows the user to set the record and playback volumes (figure II.8). There is also a means by which the unit can detect overflows and underflows in the audio playback and record buffers.

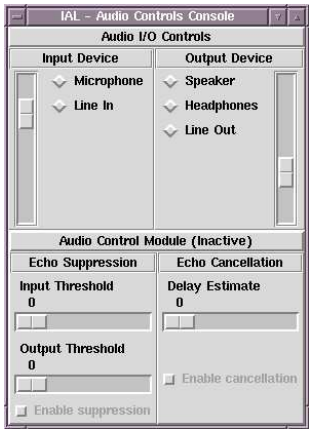

Figure II.8: Audio Control Console Screen Shot

The Audio Control Module is currently inactive. Part of the reason for this is that the research on echo suppression and acoustic echo cancelation is still in progress at the ICSI, though we believe that we will have a prototype echo suppression module for testing very soon. Also, since the migration to the VAT audio code is almost complete (at least on the Sparcstations), and since the ACM interface will probably change to adapt to the new audio code, we feel that it would be prudent to add in the ACM after the new audio code works successfully and is sufficiently tested.

There is, however, one caveat to observe in migrating the audio code to a system independent version – since routines that belong in the ACM will often need to interface intimately with the audio device driver, it may happen that the device drivers on certain platforms may not provide sufficient functionality to support the ACM. A simple solution would be to have the ACM disable itself whenever it discovered that certain functions were not available on a platform. Unfortunately, this is not a graceful

solution. Ideally, the ACM should be able to adapt to provide reduced functionality on such platforms. We feel that further research and experiments will have to be done on this before we have an adequate solution.

6 Network Unit

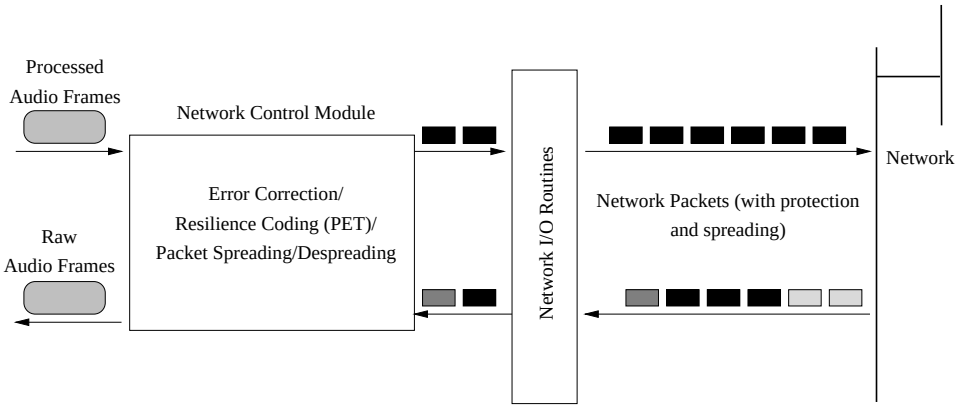

Figure II.9: Network I/O Unit Block Diagram

The Network I/O Unit (figure II.9) does for the network what the Audio I/O Unit did for the audio devices, albeit with one distinction – since we do not support different network transport protocols (for now, we deal only with UDP/IP), we do not really have the equivalent of an audio driver in the Network Unit. This situation might change in the near future, if we decide to support different transport protocols. In figure II.9, observe that the Network Control Module (NCM) converts between processed audio frames and network packets. Each audio frame may generate more than one network packet and conversely, more than one network packet may be required to generate an audio frame.

The Network Unit currently takes an audio frame and prepends a simple header to it before sending it on its way. Similarly, when a packet comes in, it strips off the header and sends it along for audio processing. Lost packets are presently detected and treated as null data, so there is no playback when that happens. In other words, the unit simply performs the most basic tasks necessary to send and receive audio over the network. The only available options are those that control the I/O mode - receive only, send only, no output (which means that the audio data is run through the filters, but we don't send the packets out) and local loopback mode (in which outgoing packets are simply diverted into the incoming data stream).

The screen display of the Network Control Console (NCC) shows the current state of the network unit (figure II.10):

① **Network Options.** This is not implemented yet, but when the network code has completed its migration to VAT's network code, which supports multicasting, these

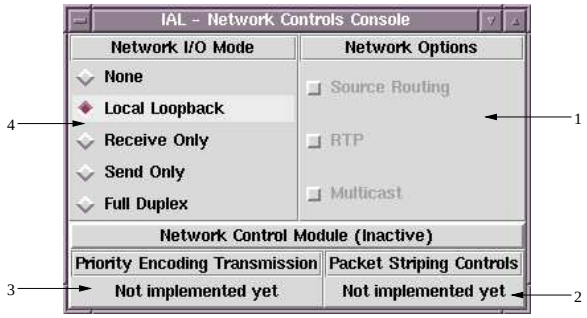

Figure II.10: Network Control Console Screen Shot

options will become available.

② **Packet Striping UI.** When the packet striping/spreading mechanisms are developed in conjunction with PET, the user interface would appear in this space.

③ **PET UI.** Similarly, the PET user interface would occupy this space.

④ **Output Mode Options.** This is, at present the only enabled panel on the NCC. It allows the user to select one of the four different network I/O modes.

As in the Audio Unit, when the network code has completed its migration to the multicast-enabled network code from VAT, work will begin on the Network Control Module (NCM). We expect that the NCM will support features like resilient coding, generation of RTP-compatible packets and also some adaptive mechanism that takes into account the RTT of the link and the network congestion. This will be discussed in greater detail in Chapter IV.

III

Filters

1 General Discussion

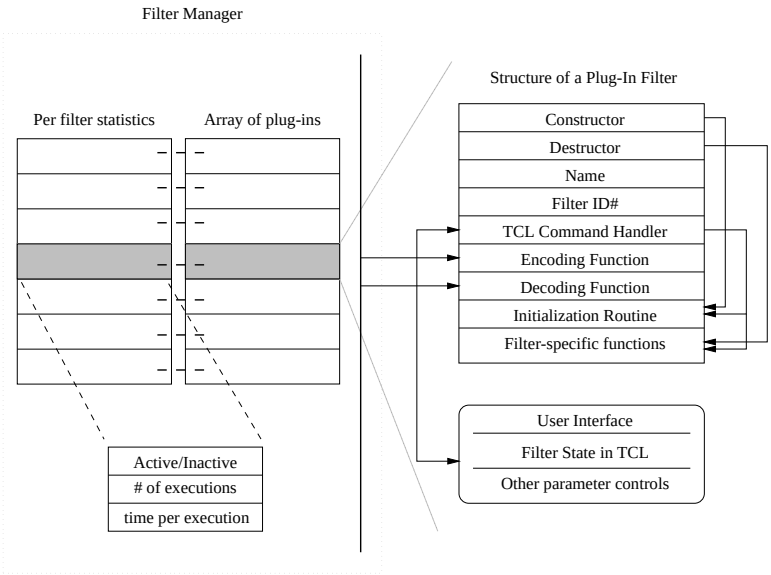

Figure III.1: Structure of a plug-in filter

Figure III.1 shows the structure of a filter, and its relation to the internal structure of the filter manager. Within the filter manager itself, there are a few fixed size arrays (limited by a fixed constant, currently set to 256). The use of fixed size arrays is believed to be reasonable since attempting to run IAL with a large number of filters will be somewhat impossible (at least in the next 2 to 3 years), even if computational power grows exponentially. One of the fixed size arrays holds the current list of attached plug-ins. Corresponding entries in

the other arrays contain the status and statistics of each filter.

At any given time, the list of currently attached filters in the filter manager is maintained both in the C++ domain as well as the Tcl/Tk domain. This is not mandatory, since we could always keep the list in one domain and have code in the other domain ask its counterpart for the list. However, it makes coding somewhat easier and is more efficient. As long as consistency is maintained across both domains, the IAL will run correctly.

To create a plug-in filter, the user will have to create a new C++ specialized filter class that inherits from the `Filter` class in the IAL distribution. The base `Filter` class dictates both the structure and the interface for this new filter. The user has to supply a name for the filter, as well as the encoding and decoding functions. The rest of the functions are optional, and the only other code that has to be written is a single line Tcl script that registers the filter at run time. Since sample filter classes are provided, the user can simply take one that fits her/his needs and modify it for the new filter. By exploiting the inheritance mechanism in C++, we allow the user to create her/his own filter specific functions and Tcl commands, while forcing the filter to conform to rigid interfaces required to make the filter manager work. In addition, the global `filter-id.h` header file will have to be updated to add this filter. The globally unique filter-id is necessary to ensure that the correct decoders are called in sequence when an encoded audio packet arrives at its destination.

When the user is done with the `.cc` and `.tcl` files, she/he simply places them in a new directory within the IAL distribution (a new directory under `IAL-FILTERS-SRC`) and adds that directory name to the list of `FILTERS` in the IAL Makefile. When the user next builds the IAL executable (by typing `make filters;make all`), the new filter will automatically be compiled and added to the IAL code. This technique has the added bonus of allowing users to distribute object files containing proprietary filter code without releasing the source files. Subsequently, when there exist different releases of IAL, with slightly different `Filter` base classes, Tcl run-time checks could be added to provide version matching.

In the current implementation of the IAL, all the encoding and decoding functions take in a pointer to an array of floats and the size of that array. The way the array is initially generated is by taking the PCM samples and dividing it by the number of quantization levels to form a float. Our reason for choosing the array of floats is to make writing digital filters easy, since the use of floats frees us from having to worry about scaling problems.

We shall now look at a few different filters that have been implemented within the IAL.

2 The Generic Filter

The Generic Filter exists simply to serve as a reference for those creating their own filters. It has code that doesn't do anything, except print comments on standard output. It also provides a simple Tk do-nothing interface to demonstrate how one can write filter-specific GUIs.

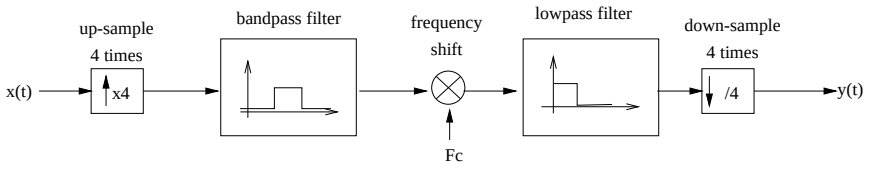

Figure III.2: The Shifting Howl Removal Filter

3 The Howl Removal Filter

The Howl Removal filter provides a means to shift an input signal by some given value, in the hope of preventing a build up of energy within some frequency bands. The shift is implemented through the use of two filters, a bandpass filter and a lowpass filter. The block diagram of the filter is given in figure III.2. Note that the filter only has an encoding function. There is no need for a decoding function – in fact, we would not want to 'undo' the shift, since this would nullify our original efforts in encoding.

As can be observed from figure III.2, the signal is first up-sampled by 4 and then passed through the bandpass filter, followed by the actual frequency shift and then through the lowpass filter, before being down-sampled by 4, to obtain the final shifted signal. The design and initial coding of the frequency shifter was done by Rainer Storn at the ICSI. In addition, the coefficients for the lowpass and bandpass filters were generated using differential evolution based optimization techniques. The aim here is to get good bandpass and lowpass filters while keeping the length of these filters relatively short and computationally tractable.

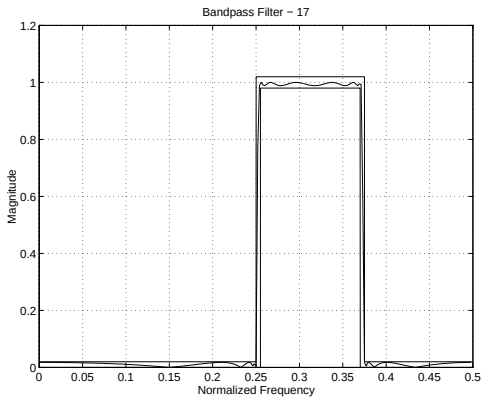

Figure III.3: The Bandpass Filter Transfer Function

This plug-in allows the user to modify the amount of the shift, in both the negative and positive directions. This can be seen from the user interface of the filter (figure II.6), in which there is a sliding scale which the user can operate. Furthermore, this plug-in allows the user to select one of a few different bandpass filters, each of a different length, to determine which bandpass filter gives the least distortion, while still being computationally feasible on an average workstation.

Having used the howl removal shifter in some experiments, we discovered that the filter was in fact, a huge burden on the CPU, and it appears that the extra volume gain in playback that we get may not be worth the amount of resources that have to be spent on the howl removal filter. However, more comprehensive and detailed studies will have to be done before more can be said about this filter.

4 The Wavelet Filter

The wavelet filter (figure III.4) is a slightly modified version of the wavelet code that was present in Hartmut Chodura's "Chatty Box". It uses the code from Numerical Recipes in C ([23]) to implement the discrete wavelet transform. Specifically, the wavelet transform as found in Numerical Recipes, uses the Daubechies family (figure III.5) of wavelets as its basis function and uses a pyramidal algorithm (involving Quadrature Mirror Filters) for generating the wavelet coefficients.

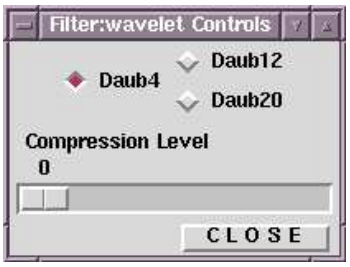

Figure III.4: Wavelet Filter Controls Screen Shot

This plug-in provides the ability to switch between the different wavelet basis, including Daub4, Daub12 and Daub20, in increasing order of computational complexity. In addition, it provides a means for testing the effectiveness of using wavelets as a means of compressing speech or audio data. The user is presented with a slider that allows her/him to set a compression level (this determines the threshold for truncation of wavelet coeffcients). Subsequently, the number of non-zero coefficients are indicated on standard output. It is usually undesirable to have information from the filter printed on the standard output device; preferably, the plug-in's user interface should have a means for providing feedback to the user. In this case, it be relatively simple to modify the filter to provide feedback on its GUI.

The block diagram of the wavelet filter is given in figure III.6. Note that there is the provision for a wavelet display, but we have not implemented one as yet. We could of course simply use the Stripchart widget that we use for the FFT Filter (see section 5). However, we are looking for other means of displaying the wavelet coefficients in a way that would allow us to locate the coefficients in a time/frequency plot and tell their magnitude simultaneously.

Presently, Rainer Storn (Siemens/ICSI), Martin Isenburg (Fraunhofer-IGD/ICSI) and the author are researching wavelets and the application of PET to wavelets. To demonstrate the useability of the IAL, we will be incorporating the results of this research into the next version of the wavelet filter. The new wavelet filter, coupled with a modifed Network Control Module that supports PET encoding, should prove to be a most interesting combination.

5 The FFT Howl Cancelation Filter

Initially, the FFT filter was simply created to view the spectrum that howls create in the frequency domain. It incorporates a real-valued, in-place Cooley-Tukey Radix-2 FFT that was ported and modified by Rainer Storn from "Real-Valued Fast Fourier Transform Algorithms" by Sorensen, H.V. et al" in ASSP-35, June 1987 (pp. 849 - 863). The block diagram of the filter is given in figure III.7, and a screen snapshot of its associated user interface is show in figure III.8 Note once again that this filter, like its earlier howl removal counterpart, only has an encoding function. In addition, note that the Hamming Window that is applied to the signal before the transform is meant to counteract aliasing and other sig-

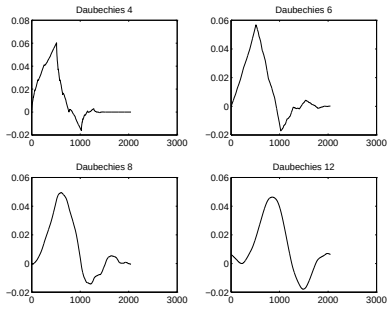

Figure III.5: The Daubechies Family of Wavelets

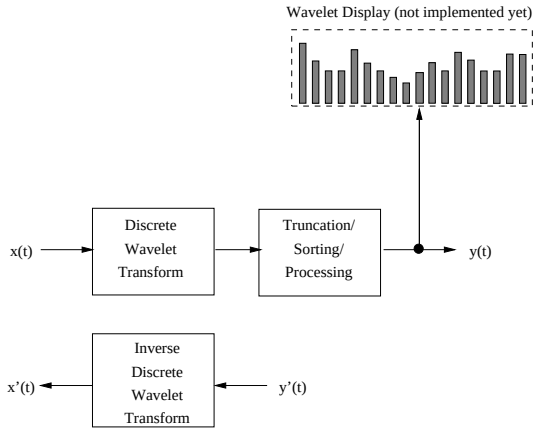

Figure III.6: The Wavelet Filter

nal artifacts that come about because we are doing a hopping FFT – i.e. we do FFTs on a packet by packet basis, instead of doing it using a sliding window, which would be more accurate, but which would take much more computation.

Once we had the FFT and Inverse FFT in place, we decided to test various novel approaches to howl removal, in the frequency domain. Our initial expectations were that the aliasing effects in the time domain that would result from modifying signals in the frequency domain, would be intolerable. However, our experiments have shown that we can do substantial modification (even implement a frequency shift operation) in the frequency domain while still maintaining speech of tolerable quality.

Our approach to the removal of howling in the frequency domain involves the use of a "zeroing window". Basically, this is a window, of some width (say 1KHz), that slides within the higher frequency bands (generally 2KHz to 4KHz, where howls usually build up), zeroing coefficients as it moves along rapidly. This prevents any kind of energy build up at any frequency band, and effectively eliminates howling. Obviously, when the mi-

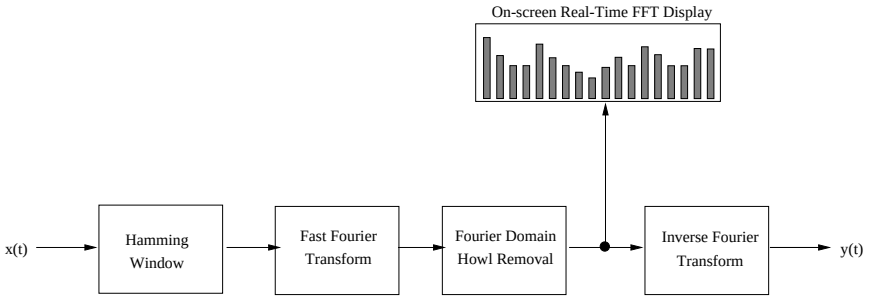

Figure III.7: The FFT Howl Cancelation Filter

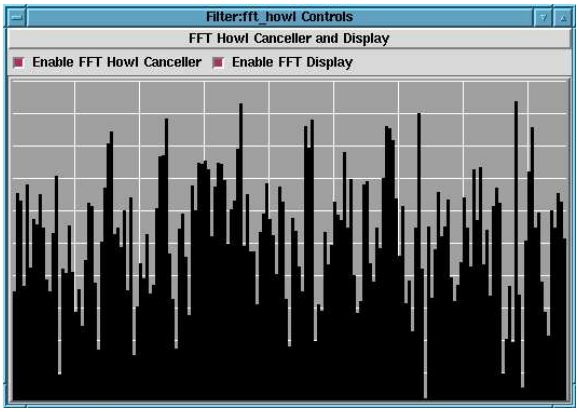

Figure III.8: FFT Howl Cancelation Filter Controls Screen Shot

crophone is brought closer and closer to the speaker, there is a distance threshold beyond which howling inevitably occurs. However, when we combined this technique with the shifting howl removal filter, we discovered that we could place the microphone so that it almost touched the workstation speaker (in local loopback mode), without generating a howl. This frequency domain howl remover proved to be much less intrusive than the frequency shifting howl removal. Furthermore, it runs around 4 times as fast the as frequency shifting howl removal. In many cases, its effect on normal speech are almost transparent. We have still to determine what the optimal settings for the different parameters (width of the window, sliding speed and so on) are, and if attenuation, as opposed to complete zeroing, will work better.

In our experiments with this filter, we also attempted to create frequency shifts in the frequency domain. If this is successful, we will have a more efficient means to do frequency shifting (compared to the earlier howl shifting). Early results appear promising, but more research has to be done before any conclusions can be drawn.

IV

Observations, Current and Future Work

1 Observations

The design and prototyping of the IAL and its associated filters form the basis for the author's Master's project. It has been demonstrated that the IAL provides an excellent framework in which to implement various filters and run experiments on these filters. It is relatively easy to create new filters and add them to the IAL filter repository. Furthermore, the filter management console allows good control over the filters and the use of Tcl/Tk with C++ has proven to be the correct choice for implementing the filter repository.

We expect work on the IAL to continue and are confident that it will prove to be successful. However, there are certain drawbacks that are present in the current version of the IAL and before the IAL will gain wide acceptance, some of these drawbacks will have to be dealt with. The more pressing problems will be described in the following section. Other problems that require extensive code changes, but which will not adversely impact the acceptance of the IAL in the short run will be dealt with in the "Future Work" section.

2 Current Work on IAL

The following is a list of work in progress at ICSI:

Port to VAT code This is the most important item on our agenda right now, and we expect that the work to integrate network and audio code from VAT into the IAL will be completed very soon. This port will enable the IAL to support multicasting, and will pave the way to porting IAL to different operating systems and hardware platforms.

Creation of Network Control Module Interface When the port to VAT code is done, we will have to work on the interface for the NCM and provide ways to integrate PET and the striping algorithm into the module. This is the next most important item

since having the NCM in place will allow us to proceed in our studies of the use of wavelets and PET in audio transmission.

Creation of Audio Control Module Interface Similarly, we will have to work on the ACM, and provide hooks for the echo suppression and echo cancelation modules, which will be implemented at some later time.

Filter constraints We are still working on ways to resolve specific requirements that some filters have by providing a means for these filters to report any constraints that they may have (for example, the wavelets and FFT filters botht require packets which have sizes that are powers of 2). In addition, we are working on means to detect conflicts and provide ways by which the user can interactively indicate how a conflict is to be resolved.

3 Future Work on IAL

The following consitutes the list of what we would like to have incorporated into the IAL in the near future. These are basically implementations of some of the more complicated proposals described earlier in this report.

Echo cancelation Much more research and development has to be done in our studies into echo cancelation before a prototype version can be built. In particular, the author believes that the development of a gracefully degradable acoustic echo canceler will not happen for some time. However, the IAL would be a good tool to use to test out potential candidates.

Echo suppression We believe that echo suppression is feasible and that it would not require too much coding to implement. When the ACM interface is fully defined, we will begin work on echo suppression and should have a working version soon after.

Adaptive feedback mechanisms It should be possible to incorporate some kind of adaptive feedback mechanism (akin to those described earlier in the report) into the NCM. Perhaps we could write an NCM that supports RTP and RTCP, and adapts the resilient coding function and the pack striping algorithms accordingly. However, more research has to be done before we embark on such a project.

Supporting different audio encoding Currently, due to the constraints imposed by the audio drivers, we only support PCM as our encoding format. However, it should be possible to write a filter that converts from the float type to PCM to some other format, like DM or ADPCM, before sending the data out on the network. In order to support such a filter, we will have to change the way the filter manager interfaces with the encoding and decoding functions in the filters. If there is a demand for this kind of support, we will look into the matter and redesign the filter manager accordingly.

Bibliography

[1] Andres Albanese, Johannes Blömer, Jeff Edmonds, and Michael Luby. Priority Encoding Transmission. *35th Annual Symposium on Foundations of Computer Science*, 1994. Also as International Computer Science Institute Technical Report TR-94-039.

[2] Johannes Blömer, Malik Kalfane, Marek Karpinski, Richard Karp, Michael Luby, and David Zuckerman. An XOR-based erasure-resilient coding scheme. Technical Report TR-95-048, International Computer Science Institute, Berkeley, CA, August 1995.

[3] Jean-Chrysostome Bolot. Characterizing end-to-end packet delay and loss in the Internet. *Journal of High-Speed Networks*, 2(3):305–323, December 1993. A preliminary version appers in Proc. ACM Sigcomm '93, San Francisco, CA, pp 289-298, September 1993.

[4] Jean-Chrysostome Bolot, Crépin Hugues, and Andres Vega Garcia. Analysis of audio packet loss in the Internet. In T. D .C. Little and R. Gusella, editors, *Network and Operating System Support for Digital Audio and Video*, volume 1018 of *Lecture Notes in Computer Science*. Springer-Verlag, 1995. Fifth International Workshop, NOSSDAV'95, Durham New Hampshire, USA, April 19-21, 1995 Proceedings.

[5] Hartmut Chodura and Rainer Storn. Audio communication for distributed collaborative systems. Preprint - technical report.

[6] Mac A. Cody. The fast wavelet transform. *Dr. Dobb's Journal*, pages 16–20, 24–28, April 1992.

[7] Mac A. Cody. The wavelet packet transform. *Dr. Dobb's Journal*, 19:44–46, 50–54, April 1994.

[8] Ronald R. Coifman, Yves Meyer, Steven Quake, and M. Victor Wickerhauser. Signal processing and compression with wave packets. Technical report, Numerical Algorithms Res. Group, Department of Math., Yale University, New Haven, CT 06520, April 5, 1990. ftp://math.yale.edu:/pub/wavelets/cmqw.tex.

[9] Hans Eriksson. Mbone: The multicast backbone. *Communications of the ACM*, 37:54–60, August 1994.

[10] Kevin Fall, Joseph Pasquale, and Steven McCanne. Workstation video playback performance with competitive process load. In T. D .C. Little and R. Gusella, editors,

Network and Operating System Support for Digital Audio and Video, volume 1018 of *Lecture Notes in Computer Science*. Springer-Verlag, 1995. Fifth International Workshop, NOSSDAV'95, Durham, New Hampshire, USA, April 1995 Proceedings.

[11] Amara Graps. An introduction to wavelets. *IEEE Computational Science and Engineering*, 2(2), Summer 1995.

[12] Vicky Hardman, Martina Angela Sasse, Mark Handley, and Anna Watson. Reliable audio for use over the Internet. In *Proceedings of INET '95*, June 27-30, 1995. Proceedings on-line at http://www.isoc.org:80/HMP/proc1.html.

[13] Simon Haykin. *Communication Systems*, chapter Appendix 3. John Wiley & Sons, Inc, third edition, 1978, 1983, 1994.

[14] K. Jeffay, D.L. Stone, T. Talley, and F. D. Smith. Adaptive, best-effort delivery of digital audio and video across packet-switched networks. In V. Rangan, editor, *Network and Operating System Support for Digital Audio and Video*, volume 712 of *Lecture Notes in Computer Science*, pages 3–14. Springer-Verlag, 1993.

[15] Bernd Lamparter, Andres Albanese, Malik Kalfane, and Michael Luby. PET - priority encoded transmission: A new, robust and efficient video broadcast technology. Technical Report TR-95-046, International Computer Science Institute, Berkeley, CA, August 1995.

[16] Bernd Lamparter and Malik Kalfane. The implementation of PET. Technical Report TR-95-047, International Computer Science Institute, Berkeley, CA, August 1995.

[17] Christian Leicher. Hierarchical encoding of MPEG sequences using Priority Encoding Transmission (PET). Technical Report TR-94-058, International Computer Science Institute, Berkeley, CA, November 1994.

[18] M. R. Macedonia and D. P. Brutzman. MBone provides audio and video across the Internet. *IEEE Computer*, 27(4):30–36, April 1994.

[19] Steven McCanne and Van Jacobson. *vic*: A flexible framework for packet video. In *Proceedings of ACM Multimedia 1995*, November 1995.

[20] Steven McCanne and Van Jacobson. Visual audio tool - source code v4a8, 1996. Available from ftp://ftp.ee.lbl.gov/conferencing/vat/alpha.

[21] Peter Noll. Wideband speech and audio coding. *IEEE Communications Magazine*, pages 34–44, November 1993.

[22] John K. Ousterhout. *Tcl and the Tk toolkit*. Addison-Wesley Publishing Company, 1994.

[23] William H. Press, Saul A. Teukolsky, William T. Vetterling, and Brian P. Flannery. *Numerical Recipes in C: The Art of Scientific Computing*, chapter 13.10, pages 591–606. Cambridge University Press, 1988-1992.

[24] Schulzrinne, Casner, Frederick, and Jacobson. RTP: A transport protocol for real-time applications. IETF Internet-Draft, November 1995. Soon to be a proposed IETF Standard.

[25] Deepen Sinha and Ahmed H. Tewfik. Low bit rate transparent audio compression using adapted wavelets. *IEEE Transactions on Signal Processing*, 41(12):3463–3479, December 1993.

[26] Rainer Storn. Modeling and optimization of PET- redundancy assignment for MPEG sequences. Technical Report TR-95-018, International Computer Science Institute, Berkeley, CA, May 1995.

[27] Terry Talley and Kevin Jeffay. A general framework for continuous media transmission control. ftp://ftp.cs.unc.edu/pub/users/jeffay/papers/ACM-MM-95.ps.Z, 1995.

[28] Ajit S. Thyagarajan, Stephen L. Casner, and Stephen E. Deering. Making the MBone real. In *Proceedings of INET '95*, June 27-30, 1995. Proceedings available through the WWW at http://www.isoc.org:80/HMP/proc1.html.

[29] Brani Vidaković and Peter Müller. Wavelets for kids: A tutorial introduction. ftp://ftp.isds.duke.edu/pub/Uers/brani/papers/wav4kids[A-B].ps.Z, 1994.

[30] Mladen Victor Wickerhauser. Acoustic signal compression with wave packets. ftp://pascal.math.yale.edu/pub/wavelets/acoustic.tex, 1989.

[31] John R. Williams and Kevin Amaratunga. Introduction to wavelets in engineering. Technical Report 92-07, Intelligent Engineering Systems Laboratory, M.I.T., October 1992. Submitted for publication to Int. J. Num. Mthds. Eng., November, 1992.