Design and Implementation of a Multicast Audio Conferencing Tool for a Collaborative Computing Framework*

Sarah E. Chodrow *Michael D. Hirsch* *Injong Rhee* *Shun Yan Cheung*
Department of Mathematics and Computer Science
Atlanta, GA 30322
{sal,hirsch,rhee,cheung}@mathcs.emory.edu
Phone: (404) 727-7580

Abstract

We present the design of CCFAudio, a collaborative audio tool, and report some experiences with its usage. The tool is specifically designed for collaboration, supporting very high audio quality, non-Mbone multicast, mixing, and hands-free operation.

1 Introduction

High speed wide area networking and multimedia computing have made it possible to transmit and process large volumes of multimedia data, a prerequisite for supporting Computer Supported Collaborative Work (CSCW) applications. CSCW uses networked computers to support cooperation between two or more people jointly performing a task or solving a problem. In a collaborative session, users employ many different applications that process and transmit, in real-time, video, audio and other types of data.

The "Collaborative Computing Framework" (CCF) is a multidisciplinary research effort at Emory University that aims to develop a CSCW system for natural sciences research. An increasingly common research methodology involves the simultaneous use of data management, computation, interaction, and graphical display/manipulation. Experiments and data analyses have come to depend heavily on the computation as well as upon collaboration that combines expertise from several different sub-disciplines at multiple, geographically distributed, locations. The overall goal of the CCF project is to evolve a distributed computing and collaboration framework that integrally supports human audio/visual communication, efficient and fast computational transforms, and distributed data management facilities.

*Research supported in part by NSF grant ASC-9527186.

Human audio communication plays a central role in a collaborative session because it is typically the stream that carries the critical component for group discussion [1]. CCFAudio tool is not "yet another Internet phone" but a collaboration tool developed in close consultation with experimental scientists. The functionalities provided by the CCFAudio tool include very high audio quality; multi-party non-Mbone communication including mixing of multiple simultaneous speakers; "hands free" operation; adaptability to high and low speed links, including a client/server mode for very low bandwidth connections; and the ability to record (play) an ongoing (previous) conversation. Although it is impossible to guarantee any quality of service in the current network architecture, works in audio transmission [2, 3, 4, 5] have demonstrated empirically that audio conferencing can be carried over the Internet under normal network conditions. Experiments with CCFAudio indicate that the tool can achieve audio quality comparable to long distance telephone calls but with slightly higher delays.

The paper is organized as follows. In Section 2 we discuss related work in network audio. We present the design of our tool in Section 3 and conclude the paper with a discussion in Section 4.

2 Related Work

In [2] Casner and Deering reported experiences with the first packet audio experiment over the Internet using "vat" [6]. Subsequently, many other audio conferencing tools have appeared, including freely available tools such as SpeakFreely [7] and LinuxPhone [8]. These and other audio tools are intended primarily as "Internet phones" lacking many features necessary to support collaboration. Two applications with similar functionality to CCFAudio are vat and Free Phone [5]. Both of these use the Mbone for multicasting packets whereas we use our own UDP IP-multicast. This avoids congesting the Mbone and enables users without Mbone access to collaborate[1]. Free Phone is the more complete tool of the two, including such features as a phonebook and the ability to "call" someone.

The audio tool in [4] attempts to improve audio quality by re-sending the same audio sample in two consecutive packets but in different encodings. This repair scheme has been generalized in [5] where an audio packet can contain several audio samples in different speech codings. CCFAudio currently has no such repair mechanism, but does have hooks to implement the scheme in [4]. However, we have found by experiment that even with 20% packet loss, conversation is quite understandable via CCFAudio.

3 The CCFAudio Tool

CCFAudio is an audio conferencing tool with additional features tailored for use in a collaborative session. Considerable effort was spent to achieve audio quality comparable to telephone (conference) calls. CCFAudio tool has evolved into *the* means of communication among members of the CCF team. A common use of CCFAudio is debugging other CSCW applications under development by describing to one another what each is seeing on the local display.

CCFAudio consists of two audio subsystems and a graphical user interface (GUI) subsystem, each running as a separate thread or threads. The first subsystem, "send-audio", packetizes voice samples from a source and transmits them to other tools. The second subsystem, "receive-audio", receives packetized voice samples from multiple sources, mixes the samples and plays the result. The GUI subsystem provides a point and click interface for controlling the audio parameters as well as graphical feedback showing who is in the conference and who is speaking at any time.

3.1 The Send-Audio Subsystem

Figure 1 shows the send-audio subsystem. There is only one thread, saudio, in send-audio, that repeatedly reads a fixed amount of data from the audio source and stores it into a buffer. If the sample contains voice data then it is converted to the user-specified format (e.g., GSM) and multicast to all members in the session. Implementing

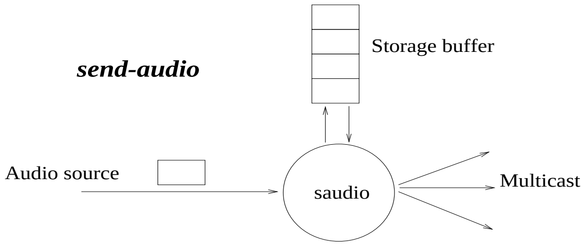

Figure 1: Send-Audio Subsystem

this method naively results in poor audio quality. To conserve bandwidth, CCFAudio transmits only when the user is speaking, while avoiding dropouts when the user is too quiet to trigger voice detection. A number of ad hoc measures have been implemented to this end.

- A sample with volume less than v is considered silent; otherwise it is treated as voice. In both cases the sample is buffered.
- If the current sample contains voice and the previous sample did not, then up to k_1 buffered samples that have not been transmitted are sent prior to sending the current sample.
- Up to k_2 samples are sent after the last voice sample.

We have found empirically that the latter two measures can improve audio quality significantly. However, the parameters v, k_1 and k_2 are highly sensitive to the articulation of individual speaker and the audio hardware used. CCFAudio therefore allows a user to change these values dynamically using its graphical interface. Feedback from the recipients is most helpful in determining the best values.

Send-audio can also accept input from an audio file. In this case, the saudio thread will read and transmit data at the data rate given in the header information of the audio file.

3.2 The Receive-Audio Subsystem

Human hearing is very sensitive to interruption of the audio stream, either as gaps in the stream, or as dropouts. Frequent interruption of the stream can make a conversation unintelligible. It is therefore very important to ensure that the audio output device does not become overloaded due to overflow or idle due to underflow. (Overflow occurs when data is written to the audio output device faster

[1] We discovered, to our surprise, that many natural scientists do not have access to the Mbone.

than it can drain, and underflow is the reverse condition.) These will result in dropouts and gaps, respectively. Another cause of dropouts is packet loss, which can be alleviated by sending a voice clip redundantly [4, 5], though CCFAudio does not yet do this. Audio packets arriving out of sequence can be re-ordered, but only if no subsequent audio packets has been played out. Natural interactive conversation becomes difficult when round trip delay exceeds a certain value, so audio packets should be played out as soon as possible without triggering underflow or overflow. Because of this, late arrivals can be considered lost.

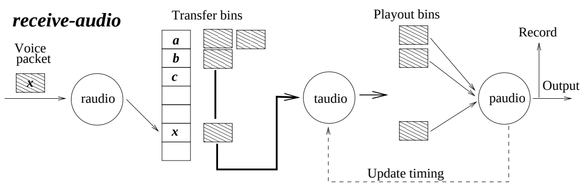

Figure 2: Receive-Audio Subsystem

Figure 2 shows the design of the receive-audio subsystem. It consists of three threads. The raudio (receive audio) thread receives packets of voice data from the other users, converts them–if necessary–to a uniform audio encoding and deposits them into their corresponding transfer bins (one bin per collaborator). The taudio (transfer audio) thread periodically removes the sample at the head from each *playable* bin and transfers it to the playout bin. Samples from a transfer bin are marked playable if the number of samples exceeds a given threshold τ. This playout delay is used to reduce jitter. The paudio (play audio) thread mixes the samples in the playout bin and sends the result to the output device. This thread can also record the conversation in an output file which can be re-played later by the send-audio subsystem.

The taudio thread also determines the time t_{end} when the output device will become idle and feeds this value back to the taudio thread. The taudio thread will wait until just prior to t_{end} before transferring the next batch of samples from all playable bins. By delaying until the last moment the transfer of packets to the play bin, packet loss is reduced to a minimum (since a late packet is considered lost). Users can reduce packet loss further by increasing τ. When τ is increased, the round trip delay is artificially increased and packets from remote users are allowed additional time before they are scheduled for playout. Currently, a single value τ is defined for all users and for later releases, independent values will be used.

When recording a conversation, some care must be taken in recording silence. Silent periods are an important part of the dynamics of a conversation, yet recording long silences wastes resources and is usually unnecessary. We represent silence in the recording logarithmically as follows. When everyone is silent for 1 (2^0) second, an "empty" voice packet containing all zero values is written. If silence continues for another 2 (2^1) seconds, a second empty packet is output. A third packet is written if silence continues for another 4 (2^2) seconds, and so on. Up to 5 empty packets can be written consecutively. When send-audio replays a file, it will pause one second when an empty packet is read (the empty packet is *not* transmitted).

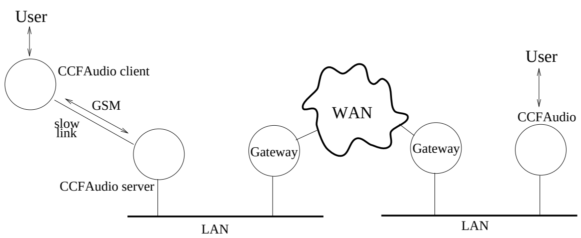

Figure 3: Audio conference using CCFAudio over a slow link

3.3 The GUI Subsystem

Users can control a number of parameters in the send audio and receive audio subsystem through the graphic user interface, namely, the input and output volume; the audio source and drain; the encoding method (GSM, μlaw, etc.); the threshold amplitude v for voice detection; and the parameters k_1, k_2 and τ. The setting of parameters v, k_1, k_2 and τ are especially important for ensuring high quality conversation. In a typical scenario, participants will discuss the quality of the reception so that the proper adjustments can be made. Once the optimal settings are found, users can communicate without further mouse/keyboard interaction with CCFAudio. The settings can be saved in a startup file and will be reused when CCFAudio is restarted.

The GUI subsystem also provides a small window listing the conference participants alongside small red LED-like buttons. When voice data from a participant is played the corresponding LED turns green, providing rapid visual feedback as to who is talking. There is also a single click facility for muting/unmuting one's speaker, microphone, or other collaborators. (See Figure 4.)

Figure 4: The CCFaudio GUI.

A separate program acts as a "ringer" interface to CCFAudio, so that collaborators can initiate conferences and invite other participants without prearrangement.

3.4 CCFAudio Client/Server

CCFAudio can be started in three different modes. Operation in the normal mode has been described in the previous section. The client and server modes are intended for use over a low bandwidth link such as a modem connection as given in Figure 3. Such a connection is able to support only a few GSM encoded audio streams, so for such a the user to participate in an audio conference, they must minimize the number of audio streams carried over the low speed link. The CCFAudio operating in the server mode will perform as specified in the previous section, except that it will encode the resulting audio data into GSM and send it to the corresponding CCFAudio client. Hence, multiple streams may be combined into a single mixed audio stream in the server. The client CCFAudio will play out audio packets received from the server. It also encodes voice from a user in GSM coding and transmits it to the server for distribution.

4 Discussion

We have used CCFAudio extensively and successfully as a collaboration tool to support not only software development, but natural science research (specifically, among a group of chemists on the Emory campus.) Our execution environment is heterogeneous–we currently have working CCFAudio executables for Solaris 5.4 and 5.5.1, IRIX 6.2 (all threaded kernels), and IRIX 5.3 (which does not support threading.) Development for other operating systems and audio hardware is underway.

Within the confines of the CS department LAN, and between the CS LAN and the Chemistry LAN, we experience no packet loss, no noticeable delay, and audio quality comparable to telephony. Experiments with off-campus users have suffered from moderate packet loss (15%-30% observed from Atlanta to the remote sites–Aiken, S.C., and Toronto, Ontario; 1%-3% in the opposite direction) due to the overloading of the Atlanta Internet routers. (`ping` displays similar packet losses.) Even under the worst level of packet loss, the audio quality was tolerable.

Other audiotools offering similar functionality (vat, Free Phone) demonstrate certain shortcomings in implementation, especially with respect to audio quality. They also suffer from the fact that multicast is done over the Mbone. In our experiments with Free Phone, the audio quality, even at the highest bandwidth, is always slightly distorted. This may be caused by timing issues alleviated by the multi-threading of sender and receiver subsystems in CCFAudio. In addition, manipulating Free Phone's GUI causes dropouts during playback, which again can be alleviated by multithreading.

References

[1] E. M. Schooler, "Conferencing and collaborative computing," *ACM Multimedia Systems*, vol. 4, pp. 210–225, Oct 1996.

[2] S. Casner and S. Deering, "First IETF Internet audioast," in *ACM SIGCOMM Computer Communication Review*, pp. 92–97, ACM, July 1992.

[3] H. Schulzrinne, "Voice communication across the Internet: A network voice terminal," Tech. Rep. TR 92-50, Dept. of Computer Science, U. Massachusetts, July 1992.

[4] V. Hardman, A. Sasse, M. Handley, and A. Watson, "Reliable audio for use over the internet," in *Proceedings of INET'95*, pp. 171–178, June 1995.

[5] J.-C. Bolot and A. Vega-García, "Control mechanisms for packet audio in the internet," in *Proceedings of INFOCOM*, pp. 232–239, March 1996.

[6] V. Jacobson and S. MacCanne, "vat manual pages, LBL, UC Berkeley, CA."

[7] J. Walker, "http://www.fourmilab.ch/."

[8] M. H. Parke, "http://www.lightlink.com/mhp/lpp."