A SIMPLIFIED APPROACH TO HIGH QUALITY MUSIC AND SOUND OVER IP

Chris Chafe, Scott Wilson, Randal Leistikow, Dave Chisholm, Gary Scavone

Center for Computer Research in Music and Acoustics (CCRMA)
Stanford University
soundwire@ccrma.stanford.edu

ABSTRACT

Present systems for streaming digital audio between devices connected by internet have been limited by a number of compromises. Because of restricted bandwidth and "best effort" delivery, signal compression of one form or another is typical. Buffering of audio data which is needed to safeguard against delivery uncertainties can cause signal delays of seconds. Audio is in general an unforgiving test of networking, e.g., one data packet arriving too late and we hear it. Trade-offs of signal quality have been necessary to avoid this basic fact and until now, have vied against serious musical uses.

Beginning in late 1998, audio applications specifically designed for next-generation networks were initiated that could meet the stringent requirements of professional-quality music streaming. A related experiment was begun to explore the use of audio as a network measurement tool. SoundWIRE (sound waves over the internet from real-time echoes) creates a sonar-like ping to display to the ear qualities of bidirectional connections.

Recent experiments have achieved coast-to-coast sustained audio connections whose round trip times are within a factor of 2 of the speed of light. Full-duplex speech over these connections feels comfortable and in an IIR recirculating form that creates echoes like SoundWIRE, users can experience singing into a transcontinental echo chamber.

Three simplifications to audio streaming are suggested in this paper: Compression has been eliminated to reduce delay and enhance signal-quality. TCP/IP is used in unidirectional flows for its delivery guarantees and thereby eliminating the need for application software to correct transmission errors. QoS puts bounds on latency and jitter affecting long-haul bidirectional flows.

1. INTRODUCTION

Internet routes good enough for experimenting with transcontinental audio already exist and will likely soon become commonly accessible to music creators and media industries. We have tested applications for remote recording, distributed performance and SoundWIRE network evaluation. This report describes our underlying engine, various applications for unidirectional and bidirectional streaming, and the networks employed in recent testing.

Professional-quality audio can be characterized as requiring:

1. uncompressed linear sampling (or at least, fast, lossless signal compression if compression is necessary)
2. multiple channels
3. CD quality resolution or better
4. very near real-time bidirectional throughput for interactive applications

The goals of our research have been to adhere to these criteria and particularly to explore the latter, in which the hope is to approach as close as possible the basic speed limit of the physical network.

Transmission speed of real-time data is restricted by the speed of light (or about 70% of the speed of light, in the case of copper). The theoretical round trip time (RTT) across the USA and back is approximately 40msec. Our experiments with very good networks have achieved RTT as low as 75msec. Because compression introduces encode / decode processing delay we have avoided it – but, neither has it been necessary – we are routinely able to transmit 10 channels worth of 24bit, 96kHz. audio signals.

Along with our new professional audio applications we have developed SoundWIRE, a utility which provides an intuitive way of evaluating transaction delay and delay constancy. Its final form is an enhanced "ping" that uses actual sound reflection between two hosts. A musical tone, such as a guitar pluck, can be created by recirculating a digital acoustic signal, turning an internet connection itself into a vibrating acoustic medium. Network delay between these reflections substitutes for the string of the guitar creating a tone whose stability represents perfectly regular service and whose pitch represents transmission latency. The ear's ability to discern minute differences makes this an unforgiving test of network reliability.

1.1. A short history of a short history

Experiments of the past 15 months have ranged from local area network (LAN) tests, including deployment on campus internets, to long-haul wide area networks (WAN). Some of these will be discussed in detail, below.

Briefly described, an end-to-end audio application involves two host computers (a server and a client) connected by one or many network segments. Each segment is typically shared by other computers (connected to it by network hubs and switches) who contribute their own data traffic in competition with the audio. Routers are dedicated computers that tie together segments and take care of relaying traffic between them according to destination. Routers with quality of service (QoS) capabilities can give priority service to designated traffic, but this feature is still experimental and can introduce undesirable side-effects of overhead and policy management requirements. The need for of QoS is avoided by providing "over-provisioned" networks that are able to meet the real-time requirements of the applications running across them. Problems begin here though, because the definition of "real time" depends on the task. File transfer and email applications have a very different notion of real time than, say, telesurgery. In fact, the upper bound for delay even differs from one type of audio application to another.

The Internet Protocol (IP) which provides data routing within and across physical networks does not itself guarantee reception of data at the listening end. As a result, differing protocols built on top of IP (see [1] for a good description) have been chosen to deal with this, depending on the type of audio application. Applications have been of four types: LAN-unidirectional, WAN-unidirectional, LAN-bidirectional and WAN-bidirectional.

UDP (user datagram protocol) is a low-overhead mechanism with no delivery guarantees, fast but unreliable. It can be extended by the application designer to compensate for its susceptibility to transmission errors over best effort networks. These inherent uncertainties can be overcome by instead choosing *TCP* (transmission control protocol) to guarantee reliable data transport. TCP incorporates an acknowledgement scheme and will retransmit corrupted or missing data, also correcting any out-of-order arrivals. It is a simple to way to add the necessary audio delivery guarantees to an application, but introduces its own buffering and processing delay. *QoS mechanisms* provide a way to increase reliability and reduce latency by putting the burden of flawless delivery on the network itself. QoS router capabilities, as will be shown, provide a way to meet the requirements for low-overhead, fast transport that is less susceptible to errors and that approaches the theoretical limit of transmission speed.

The first public WAN experiment was a unidirectional concertcast from McGill University to New York University, September 26, 1999. The team combined UDP (their application's primary protocol) with a TCP-based error compensation technique that could reliably retrieve missing data. The signal was delayed a number of seconds by data buffering and compression processing (Dolby AC3 compression was used)[2][3]. Long delays are no difficulty in such a case, however, because delay is not perceived when listening to or recording a concert provided the application is entirely unidirectional. The McGill system was employed again in July, 2000 between Montreal and Tokyo for the INET2000 Conference.

In February, 2000 our team began to listen to round trip reflections around the laboratory as musical tones using a TCP-based SoundWIRE method. Interestingly, evaluating RTT in this way provides an intuitive impression of "network distance" by converting times to pitches. A second version, which we call *Soundping*, converts ICMP ping (the usual Internet Control Message Protocol utility) statistics directly to sound and allows listeners to tap the outer reaches of the Internet without needing to create bidirectional flows.

Our subsequent work involved TCP-based unidirectional musical events, followed by TCP and UDP bidirectional demos. As the project has evolved, so has its code base. The following gives an overview of the present system.

2. THE STREAMING ENGINE

The streaming engine is intended to be highly flexible and support many different applications. To this end, we have designed the engine using a core stream class and plugins that fall into one of four categories: input, output, processing, and input mixing. Each of the input and output plugins runs in its own thread and makes calls into the stream class to submit or receive data.

The stream class handles any number of plugins of each type provided that they are all expecting and providing the same type of data.

2.1. Input

An input plugin spends most of the time blocked against its data source. Each time the plugin reads a buffer of data it calls the containing stream class to post the buffer. It then returns to block against the data source again.

The input plugins we have currently implemented are a digital audio input plugin for reading audio data from a sound card, network plugins for streams using the UDP and TCP protocols, and a passthru plugin (actually implemented as an input plugin and an output plugin to forward data from the output of one stream to the input of another on a local machine). Future possibilities for input plugins are midi, RS-422 machine control, generic serial signals, or any of the many available external controllers.

2.2. Output

An output plugin is very similar to an input plugin except that it may choose to block either against the stream class waiting for data or against its data sink. The threaded implementation permits each plugin to implement this in the most appropriate way. For instance, in the case of the digital audio output plugin, we never want to cause underruns to the sound output hardware so we advise the stream to never block the plugin but instead to return a buffer of zeros in the case that there is not a full buffer of data available. In the case of the network on the other hand, we can afford to block waiting for data because, especially in the TCP case, the system is so much more elastic than the hardware sound output case.

2.3. Processing

The processing plugins provide a way to act on each buffer of samples as it passes through the system. In implementing a network waveguide (SoundWIRE), we use this functionality to filter the signal as it passes through the stream. This could also be used to apply reverb or other audio processing to the signal. We could also use this point to encode or decode the signal with error correction codes and/or compression algorithms. The plugins will always be applied in the order they are installed, therefore it would be conceivable to apply reverb, compression, and error correction coding to a signal in that order on the transmitting side and then to recover the signal by applying the decoding of the error correction and compression in the opposite order on the receiving side.

2.4. Coordination and tuning

The stream class itself is implemented as a simple arbitrator between all the plugins. In our current implementation there is a separate circular buffer for each output plugin. Each buffer of received data is passed through the plugins and then a copy is written to each of the output circular buffers.

The case of many input plugins is much trickier because the procedure for combining the input signal is different depending on what kind of signal is being streamed. The audio case is currently implemented with an additional circular buffer that handles many write pointers. Each input plugin adds its data into the input buffer and the summed signal is then forwarded on through the processing and into the output buffers. In the future we can implement mixing support for midi and other data streams that demand it.

In order to adapt the streaming application to the type of stream, the network quality, and the tradeoff between latency and delivery guarantees we can easily adjust several parameters:

1. packet size for network transmission
2. fragment size for sound i/o read and write operations
3. size of the circular buffers
4. threshold for filling a circular buffer before the first read operations are permitted on it
5. TCP retransmit, fast acknowledgement schemes and other options
6. thread priority (though, so far this has been unnecessary)

2.5. Configuration Examples

The simplest application we can create using the streaming engine is a loopback that reads from the sound card and writes directly back out to the sound card again as shown in Figure 1.

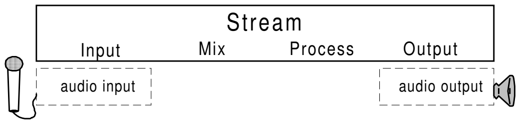

Figure 1: *Loopback case.*

Figure 2 illustrates adding a reverberation processing plugin to make a simple live reverb processor.

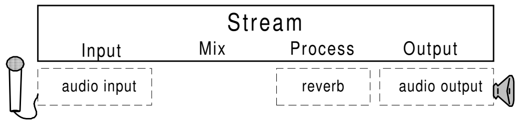

Figure 2: *Reverb case.*

Figure 3 shows the streaming engine as a means to pass audio into and out of devices via network hardware. In this advanced case, multiple hosts are connected via internet as a 2D network waveguide mesh with audio input and output taps at each node. In this topology the network simulates a drum head or a plate reverberator.

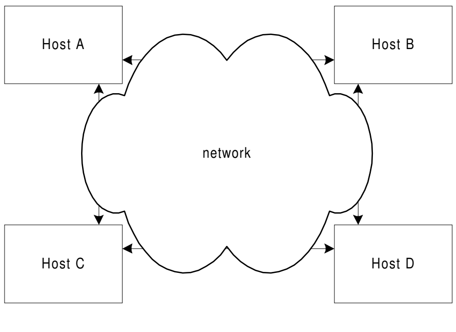

Figure 3: *Multiple waveguide case.*

Unidirectional streaming is achieved by first creating a server stream process on the receiving machine and then connecting to it with a transmit stream process on the client machine (Figure 4).

Bidirectional streaming is achieved by running two streams (both sides of the unidirectional case) on both machines as shown in Figure 5.

Figure 6 demonstrates a 1D network waveguide with audio input and output taps on both ends created by recirculating a stream in a bidirectional loop.

2.6. Implementation

The streaming engine is implemented in C++ using the CommonC++ (cross-platform Windows / Unix) library [4] for thread and socket support. Synthesis Tool Kit (STK) C++ classes [5] provide numerous processing elements and were used for the first streaming engine prototype. Graphical user interface classes are provided by the Qt C++ library [6].

3. OBSERVATIONS AND APPLICATIONS

General observations:

1. Success is governed by careful tuning of all parameters mentioned above and an appropriate choice of protocol which, as mentioned above depends on the type of application. If TCP, its retransmit algorithms and buffering parameters can be tweaked for low-latency, small packet throughput.
2. The physical layer permits an extremely large amount of real-time audio. Once things are well set up in terms of system tuning, one can usually increase the number of simultaneous channels dramatically.
3. QoS has tremendous impact on cutting through traffic situations, especially those likely to be encountered in WAN paths.

Four recent project milestones illustrate the evolution and application of our work.

3.1. LAN teleconcert, Ambisonics B-format audio, TCP/IP

A "teleconcert" on May 24, 2000 constituted our first professional-quality audio streaming in a public musical event. Two Stanford University concert locations (approximately 1km. apart) were linked via internet to transmit live performance from one stage to the other, one direction at a time. Spatial characteristics of each location (one indoors, one outdoors) were transmitted using Ambisonic microphone techniques. The 4-channel Ambisonics B-format can be reconstructed at the receiver into n-channels of surround sound. In the concert a ring of 5 loudspeakers was placed at one site and 8 at the other (a plugin was written to do B-format decode).

The teleconcert was routed over the Stanford University core backbone. Each concert site was adjacent to a fast-ethernet switch, and these were connected to a core router. Two other core routers were in the path. Core backbone capacity is rated at 2.5Gbps and is highly overprovisioned. Typical cross-campus RTT's are on the order of one millisecond or less. We used the TCP streaming engine and saw no difficulty with the increased overhead since it simply added to the delays that were necessary for audio buffering (approximately a 1 sec. aggregate delay with this early version of the engine).

Our network for the May concert was "plug and pray," in the sense that no attempts were made to use special connectivity or other guarantees, only the existing infrastructure. The two host

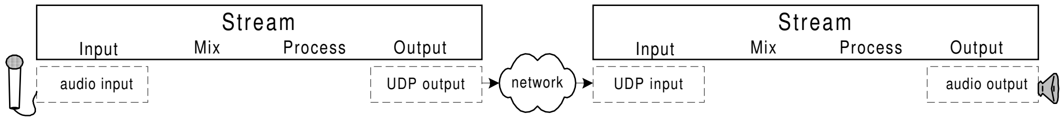

Figure 4: *Unidirectional case.*

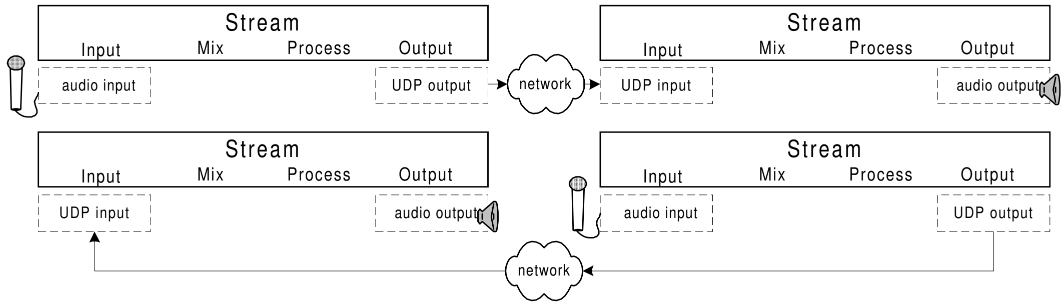

Figure 5: *Bidirectional case.*

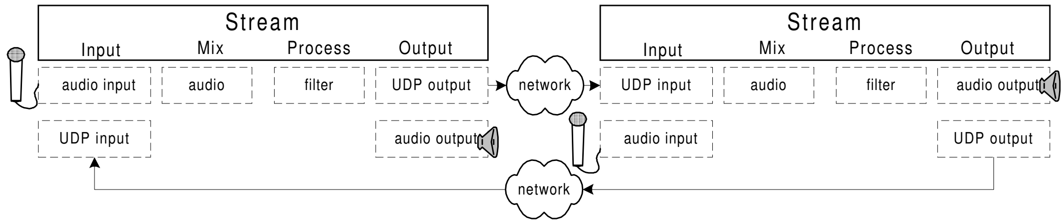

Figure 6: *Waveguide case (SoundWIRE).*

computers, besides serving the audio, kept open "chat" windows for operator messages to be passed back and forth during the concert. Audio transmissions during rehearsal and concert were faultless.

3.2. LAN remote recording, 10-ch, TCP/IP

At the Banff Centre for the Arts Summer Arts Festival, 10-channel recordings were made by transmitting audio over the Centre's internet, at a distance of about 0.5km. A number of continental premieres were recorded including Krzysztof Penderecki's Sextet, July 14, 2000 and his Credo, featuring over 200 musicians, July 15th. Again, the streaming engine was TCP and aggregate delay approximately 1 sec.

3.3. LAN bidirectional, trumpet duo, TCP/IP

The following month, we connected headphones and mics for two trumpeters located in the rooms used for the previous recordings. A TCP-based, 1-channel, bidirectional application was tuned to provide delays of about 200msec. The musicians were initially mystified by trying to perform in such a situation (especially with no visual cue for starting together). It only became possible to avoid recursive tempo slowing when one player agreed to play behind the other.

The recordings and the trumpet duo across the Banff Centre network involved a mix of network segments of differing capacities. Like the Stanford test, existing networks were used. One slower 10Mbps segment was in the path but had no adverse effect. Since there were no competing flows on it during the concert, we had access to its full capacity (crucial, given our 7Mbps 10-channel recording stream of 16bit / 44.1kHz. audio). For bidirectional work, we fell far short of the ($<$ 5msec.) RTT available. Either another protocol, better TCP tuning, or our latest engine would have gotten us closer to the LAN's very low latency.

3.4. WAN bidirectional, SoundWIRE, UDP & QoS

In October, 2000, using next-generation networks, 2-channel data was tested bidirectionally between CCRMA and facilities at Cisco, North Carolina and at Internet2 headquarters in New York (approx. 9000km. round trip). RTT on the order of 75msec. was achieved and sustained. Mics and headphones were connected together and compared to a telephone connection also open between the same rooms. We felt the network RTT was nearly as good as the telephone's and conversation seemed comfortable. The audio quality was, of course, much better.

SoundWIRE was tested over the same connection, implemented by an IIR circuit in which audio recirculates in a loop as described above. With the mics adding excitations to the loop, one had the impression of speaking or singing into a large echo chamber.

Our bidirectional transcontinental audio connection was carried over the Internet2 Abilene backbone. Separate QoS testing segments were created at entry points so that we could listen to the effectiveness of QoS as it battled artificial traffic congestion.

Test segments consisted of a pair of interconnected routers running a QoS configuration plus load generators to inject traffic. Router configurations provided a means to enable or disable expedited forwarding (EF) of the audio traffic. The incoming router marked our audio traffic for EF according to source and destination addresses and the outgoing router (connected to Abilene) prioritized its processing of the marked traffic with respect to the artificial traffic (which only traveled on the test segment).

4. CONCLUSIONS

Internet technology has matured to where it can support high-quality audio. Reciprocally, we believe audio may become important for internet engineering, particularly with regard to evaluating QoS, which has become an important goal in engineering for a whole range of interactive applications. Working with audio in this way, one quickly trains their ear to hear characteristics of a connection.

Commonly available connectivity such as fast-ethernet components in PC's on institution-sized internets (small "i") provide plenty of bandwidth for local work involving real-time concertizing and recording. The bidirectional experiments described in this paper indicate that next-generation backbones, especially those with QoS capabilities, can extend these possibilities across the worldwide Internet (capital "I").

Most large institutions are already wired with networks that can transport audio. We have had satisfactory results with 10Mbps segments, even when running them near 100% of capacity. Data rates as high as 23Mbps can be seen in our hungriest experiments (10 channels of 24bit / 96kHz. unidirectional), but that's relatively modest. Large-scale studios will design for transport of hundreds of channels around a studio complex. Given the phenomenal capacity of today's LAN networking gear is well within reach: 1Gbps equipment is replacing fast-ethernet (100Mbps) in new installations and 10Gbps is in early deployment. Analog or special-purpose digital audio wiring in campuses and studios could quickly become a thing of the past.

At the September, 2000 AES conference another McGill University concert was transmitted across WAN to demonstrate remote surround sound recording at the University of Southern California. The high-resolution 24bit / 96kHz. 10-channel signal again proves the capacity of next-generation networks for professional audio. And that, "...data compression is a temporary aberration that will be swept away with time." [7]

Delay is the final issue that remains. Once it can be pushed down to its theoretical limit, it will be interesting to see what musical possibilities can be made of truly interactive connections. In the immediate future we are designing experiments using QoS techniques that will put known bounds on latency and jitter (our present experiment did not provide such guarantees). These experiments can only be conducted on testbed networks whose routers are adapted to the purpose. It is too early to know much about the terms under which QoS will become generally available but, on the technical side, we have seen that EF techniques are effective for real-time bidirectional audio even under heavy traffic.

There is little doubt that dedicated IP / audio devices will be forthcoming and will replace the PC's used in present experiments. In fact, our team is completing a stand-alone "network audio appliance" whose processor runs embedded IP protocols directly connecting fast-ethernet with a hardware digital audio interface. We intend it to run wireless, as well.

5. ACKNOWLEDGEMENTS

The present work was funded by National Science Foundation Grant No. ANI-9977246 and Stanford University's Internet2 Application Development Fund. We appreciate the dedication and insights of the following individuals who have made technical contributions to work reported here: Fernando Lopez-Lezcano, Jay Kadis, Joe Anderson, Wayne Sung, Ron Roberts, Don Pelton, Michael Gröger, Pablo Mochcovsky, April Kintzel, Graham Lindgren, Ben Teitelbaum, Stanislav Shalunov, and John Moore. Analog Devices and M-Audio generously provided hardware used in the project.

6. REFERENCES

[1] Comer, D., Internetworking with TCP / IP, vol. 1, Principles, Protocols and Architectures, Prentice-Hall, New Jersey, 2000.

[2] Xu, A. et al., "Real Time Streaming of Multi-channel Audio Data over Internet", Journal of the Audio Engineering Society, Vol. 48, Number 7/8, July/August 2000.

[3] `http://www.cim.mcgill.ca/~jer/projects/aes/`

[4] `http://cplusplus.sourceforge.net/`

[5] `http://www-ccrma.stanford.edu/CCRMA/Software/STK/`

[6] `http://www.trolltech.com/`

[7] Moorer, A., Heyser Memorial Lecture to Audio Engineering Society, `http://www.aes.org/technical/Heyser.html`