Juan-Pablo Cáceres and Chris Chafe
Center for Computer Research in Music and Acoustics (CCRMA)
Stanford University
The Knoll, 660 Lomita Dr.
Stanford, California 94305-8180, USA
{jcaceres, cc}@ccrma.stanford.edu

JackTrip/SoundWIRE Meets Server Farm

Systems for real-time, high-quality and low-latency audio over the Internet that take advantage of high-speed networks are available and have been used in the last several years for distributed concerts and other musical applications (Renaud, Carôt, and Rebelo 2007). The difficulty of setting up one of these distributed sessions is, however, still very high. Most musicians involved in such sessions have experienced the disheartening amount of time that can be lost in rehearsal, where most of the time is spent adjusting the connection rather than playing music.

Keeping delay to a minimum is one of the main goals when tuning network parameters. Delay is known to be disruptive in musical performance (Chafe and Gurevich 2004), so a sensible goal is to minimize it as much as possible. Often, there is a tradeoff with audio quality. The longer the latency, the better the audio (i.e., fewer dropouts) if facing problematic network conditions. For most users who are not familiar with the network protocols and delivery of the Internet Protocol Suite (commonly known as TCP/IP), understanding the meaning of these protocols' parameters can be daunting. (In particular, we use the User Datagram Protocol [UDP], which is part of the Internet Protocol Suite.)

We present here a server-based application that can be of use to intuitively tune these parameters using "auditory displays" (Chafe and Leistikow 2001). With it, musicians tune their network connection much like they do their instruments, using their ears. The implementation is part of the JackTrip application (Cáceres and Chafe 2009), a software program for low-latency, high-quality, and multichannel audio streaming over TCP/IP wide-area networks (WANs). JackTrip is a peer-to-peer system which can interconnect many bidirectional nodes.

The design and architecture is first geared towards implementation of this quality-of-service (QoS) evaluation method. The architecture has also been extended to provides other types of service—in particular, a central "mixing hub" to control audio in a concert where multiple locations are involved.

Computer Music Journal, 34:3, pp. 29–34, Fall 2010

QoS Evaluation Metrics

Cromer gives a good definition of QoS:

> The term *Quality of Service (QoS)* refers to statistical performance guarantees that a network system can make regarding loss, delay, throughput, and jitter (Comer 2005, p. 510).

Most of the networks available today are *best effort delivery*, i.e., they don't provide any specific level of QoS. As such, this infrastructure can be problematic because sound is unforgiving in regard to packet loss and jitter; any lost data is immediately audible. In evaluating a particular connection, we want to know "instantaneous" QoS, that is, assess its quality at any given moment. Users should be able to adjust their settings to achieve the optimal quality given the current bandwidth and congestion conditions. This should be convenient and a conscious part of setting up. It should also be monitored with regard to longer-term changes: a connection that is perfectly clean at 1 AM can become congested at 9 AM. A bad connection today can be a surprisingly good one a year from now when intermediate network upgrades are put in place, or when the user asks that their service be enhanced.

A connection is presently either tuned by trial and error, or is set automatically by an adaptive mechanism that changes the data rate depending on bandwidth availability (Qiao et al. 2008). Adaptive methods are typically found in unidirectional streaming and have a disadvantage for bidirectional high-quality audio. Latency is a parameter we want to keep constant. To accommodate changing amounts of jitter, adaptive methods can arbitrarily increase and decrease the local buffering, affecting

total latency in a way that is very disruptive for musical performance.

In this article, we describe an implementation of a tool that lets musicians tune a connection completely by ear. Parameters like buffer size, sampling rate, packet size, and packet redundancy, among others, can be adjusted using this "auditory display" mechanism.

"Pinging" the Network, Acoustically

The advantages of evaluating very fine-grained jitter and packet loss using these "auditory displays" have been previously discussed in the literature (Chafe and Leistikow 2001). The method consists of listening to a pitched sound in order to assess delay, jitter, and loss. The procedure produces a tone by recirculating audio in the network path and thus allows for fine-grained listening of the packet flow. (The granularity is determined by the sampling rate and the packet size. For example, at 48 kHz and 64 samples/packet, the granularity is 1.3 msec.) The acronym SoundWIRE describes the technique used in this project, "sound waves on the Internet from real-time echoes." In principle, it uses the Karplus-Strong plucked-string synthesis algorithm (Karplus and Strong 1983) and simply replaces string delay lines running in local host memory with network "memory."

This technique can be extended to incorporate different-sounding "auditory pings" using other physical models (Chafe, Wilson, and Walling 2002), but the underlying approach is the same. In the case of, for example, a string physical model, musicians want to tune their connection to get a sounding instrument that has the highest possible pitch (low delay) without vibrato (jitter). Users also want to minimize extraneous impulses coming from packet loss.

In the next section, we present an architecture of a server that clients can use to evaluate and tune their connection solely based on auditory feedback, much like the way guitar players tune their instruments.

Multi-Client Concurrent Server

We extended the JackTrip platform to include a system for QoS evaluation. The new architecture provides a multi-client concurrent server that can be used, among other purposes, to evaluate QoS or to function as a central hub that mixes or relays audio. Taking advantage of multi-core computers, it is possible to concurrently serve hundreds of clients with uncompressed real-time audio and processing plugins.

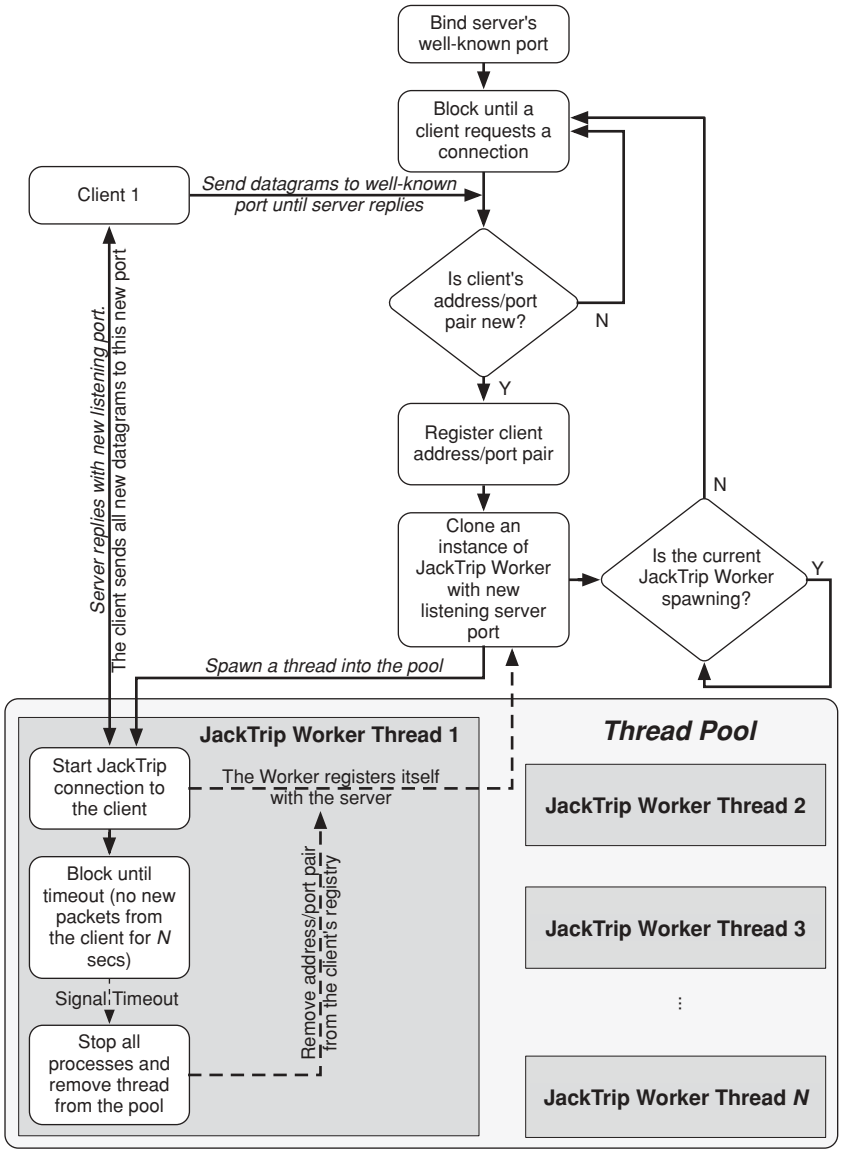
Server Architecture

The UDP is a connection-less protocol, and consequently identification of a client's IP number has to be done on a packet-per-packet basis. Several techniques for dealing with multiple clients connecting are discussed in the literature (Stevens, Fenner, and Rudoff 2003), but no standard exists as in the case of Transmission Control Protocol (TCP) servers. (See Comer [2005] or Peterson and Davie [2003] for a good description of the differences between TCP and UDP protocols.)

Our implementation relies on a "smart" client which can change to a new server port number after being assigned one for exclusive communication. This technique has the advantages of being easily adapted to multiple platforms (it works currently on both Linux and OS X, and could be ported to Windows), being lightweight, and not requiring root privileges. In turn, the server expects its clients to change connection ports.

Figure 1 describes the architecture of the system. The server listens on a well-known port for client connection requests. For every new request, the server has to check if the originating address/port pair is new. If it is, it registers it in an array of active address/port pairs and blocks the requests of new clients while this one is being processed. It then allocates a new port to communicate exclusively with this client, and informs it of the new port. The client then stops sending packets to the well-known port and starts to send them to its own assigned one. From then on, the whole JackTrip process is added to a thread pool and runs independently in its own thread. The server is freed to wait for new client requests. The thread runs until the client stops sending packets (or the server doesn't receive them) for a certain amount of time. At that point a signal is emitted, and the server deletes the

Figure 1. Multi-client concurrent server algorithm.



client IP/port pair from the active clients registry and removes the process from the thread pool. The implementation is written in C++ using the Qt libraries (Nokia Corporation 2008–2009) for networking and multithreading.

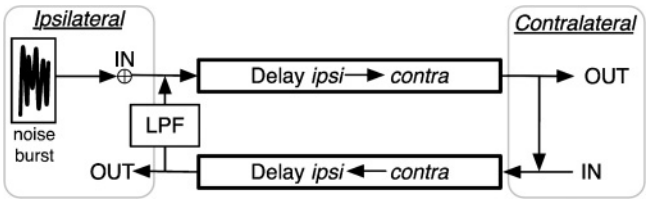
Server Applications

The server has been deployed in a couple of applications, presented in the following. Although we had these two scenarios in mind when we created the server architecture, it is also possible to imagine and design different applications using the presented work.

Quality of Service (QoS) Evaluation

Each connection between the client and the server recirculates audio and implements a Karplus-Strong string model (Karplus and Strong 1983). This configuration has been discussed in detailed

Figure 2. Karplus-Strong algorithm implemented in the network path, recirculating audio.



previously (Chafe et al. 2002). Figure 2 shows a basic implementation of the algorithm. The ipsilateral host (which in our system corresponds to the server) generates excitations (plucks or noise bursts) that are "echoed" back from the contralateral host (the client) recirculating in a loop that includes a low-pass filter (LPF).

To test a connection, a client connects against a known server IP number (e.g., that of a server at Stanford University's CCRMA). The path is sonified with this string model. As the network delay increases, the pitch of the sting will be lower. Variances in the latency will be perceived as vibrato of the string model. Packet losses are translated into impulsive types of sounds (for the case when the receiver plays zeros when it doesn't receive a packet) or into wavetable types of sounds (for the mode when the system keeps looping through the last packet received). (More details on these two modes can be found in Cáceres and Chafe [2009].)

Providing this service for intuitive and quick evaluation of connection QoS is the original intended application of this technology. By connecting to the server and "listening" to the path, users can tune their connections to its optimal settings. As mentioned earlier, there is a trade-off between latency and sound quality. In the presence of jitter, the local buffering has to be increased to avoid late packets, but at the same time we don't want to increase it too much (to avoid unnecessary latency). Doing this without a tuning tool, by trial and error, requires experience and can be frustrating for new users. If, in turn, musicians can listen and tune the connection in the same way they tune an instrument, the setup is much faster and intuitive. Again, the goal for musicians is to tune their pitch to be as high as possible (lower latency) with the smallest possible vibrato (jitter).

Our prototype server application, JamTest, is currently deployed on a 16-core server running Fedora Linux. Users can connect using a GUI-based QoS testing client on Linux or Mac OS X.

Star-Topology Connection/Mixing Hub

Mixing and managing remote connections when more than two sites are involved can be very complicated. Engineers have to deal with audio channels coming from different places (sometimes on confusingly different channels), all with different levels. They also need to make sure local audio is sent to the peer with proper gains. A solution to centrally manage these types of situations designates a master location which can mix and/or relay all the channels and send them back to the respective connected peers.

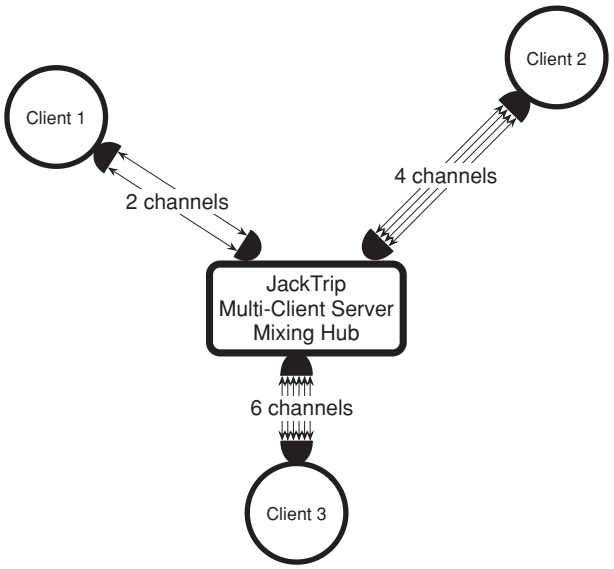
The present server implementation allows a server to dynamically connect and disconnect audio from different clients. In this case the server will act as a "hub" between several locations. Each client can have a different number of channels and different network tuning parameters. (JackTrip presently uses Jack [Davis 2009] as its audio host. This has the limitation that sampling rate and buffer are fixed at *Jack start-time* and cannot be tuned after the server has started.)

Figure 3 illustrates this for an example with three clients. The server can mix and re-route all the audio channels between the clients, hence allowing a multi-site performance with one site acting as a master relay service and/or mixer.

Conclusions and Future Work

The first decade of the 21st century evidenced a dramatic increase in the speed and reliability of high-speed networks. This increase is expected to continue. We have provided a system for musicians to tune and optimize their connections against a reference server in a way that lets them adapt to their given network situation. The server can also be used to interconnect multiple sites with arbitrary numbers of channels, and it can be a "mixing hub" that distributes audio to all the locations from a

Figure 3. Multi-client server as a hub.



central place. This system has been used successfully in numerous concerts since the summer of 2009.

Scalability in network performance is a big issue that still needs to be solved. Learning how to connect hundreds or even thousands of remote locations for a global jam session is a pending goal. Multicast at the network layer would provide a solution for a fully connected peer-to-peer mesh. Clients would select from a list of peers they want to connect with, and then send just one packet via multicast (using its underlying network layer implementation). Network routers and switches determine when a copy needs to be made. Access Grid (Daw 2005) implements this for a fixed number of audio channels; however, this infrastructure is not yet ubiquitous. Furthermore, when the number of audio channels and other settings differ among the clients, a new and consistent solution is required so that they can inter-operate.

Scaling up and distributing physical models embedded in the network path can also serve to perform "global string network symphonies," where the global network becomes the instrument itself, an instrument distributed throughout the world.

Acknowledgments

This work was carried out in cooperation with MusicianLink, Inc., and funded by National Science Foundation Award Grant No. IIP-0741278 with a sub-award to CCRMA. See the online final report (Chafe 2009). Fernando Lopez-Lezcano and Carr Wilkerson from CCRMA have provided continuous assistance in the implementation and server infrastructure setup.

References

Cáceres, J.-P., and C. Chafe. 2009. "JackTrip: Under the Hood of an Engine for Network Audio." *Proceedings of International Computer Music Conference*. San Francisco, California: International Computer Music Association, pp. 509–512.

Chafe, C. 2009. "STTR Phase I: MusicianLink Platform for Audio Collaboration and Real-Time Distributed Audio Processing." Technical report, Center for Computer Research in Music and Acoustics (CCRMA), Stanford University. Available at http://ccrma.stanford.edu/~cc/pub/pdf/qosServer-nsfFinalReport.pdf. Accessed 1 April 2010. Final Report, Technical Research Summary, National Science Foundation Small Business Technology Transfer Research Award No. IIP-0741278.

Chafe, C., and M. Gurevich. 2004. "Network Time Delay and Ensemble Accuracy: Effects of Latency, Asymmetry." *Proceedings of the AES 117th Convention*. New York: Audio Engineering Society p. 6208.

Chafe, C., and R. Leistikow. 2001. "Levels of Temporal Resolution in Sonification of Network Performance." *Proceedings of the Seventh International Conference on Auditory Display (ICAD 2001)*. Helsinki: Laboratory of Acoustics and Audio Signal Processing and the Telecommunications Software and Multimedia Laboratory, Helsinki University of Technology, pp. 50–55.

Chafe, C., S. Wilson, and D. Walling. 2002. "Physical Model Synthesis with Application to Internet Acoustics." *Proceedings of the IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP '02)*, volume 4. Los Alamitos, California: Institute of Electrical and Electronics Engineers, pp. 4056–4059.

Comer, D. E. 2005. *Internetworking with TCP/IP, Vol 1*. Upper Saddle River, New Jersey: Prentice Hall, 5th edition.

Davis, P. 2009. "JACK: Connecting a World of Audio." Available at http://jackaudio.org/. Last accessed November 2009.

Daw, M. 2005. "Advanced Collaboration with the Access Grid." *Ariadne* 42. Available at http://www.ariadne.ac.uk/issue42/daw/intro.html. Last accessed November 2009.

Karplus, K., and A. Strong. 1983. "Digital Synthesis of Plucked-String and Drum Timbres." *Computer Music Journal* 7(2):43–55.

Nokia Corporation. 2008–2009. "Qt Software." Available at www.qtsoftware.com/. Last accessed November 2009.

Peterson, L. L., and B. S. Davie. 2003. *Computer Networks: A Systems Approach*. San Francisco, California: Morgan Kaufmann, 3rd edition.

Qiao, Z., R. Venkatasubramanian, L. Sun, and E. Ifeachor. 2008. "A New Buffer Algorithm for Speech Quality Improvement in VoIP Systems." *Wireless Personal Communications* 45(2):189–207.

Renaud, A. B., A. Carôt, and P. Rebelo. 2007. "Networked Music Performance: State of the Art." *Proceedings of the AES 30th International Conference*. New York: Audio Engineering Society.

Stevens, W. R., B. Fenner, and A. M. Rudoff. 2003. *Unix Network Programming, Volume 1: The Sockets Networking API*. Boston, Massachusetts: Addison-Wesley Professional, 3rd edition.