frontiers in Digital Humanities

TECHNOLOGY REPORT
published: 26 November 2018
doi: 10.3389/fdigh.2018.00027

I am Streaming in a Room

*Chris Chafe**

Center for Computer Research in Music and Acoustics, Stanford University, Stanford, CA, United States

Internet Acoustics is the study of sound traveling through the Internet, treating it as an acoustical medium just like air or water. Real-time streaming of sound, something commonplace nowadays, can be exploited for its own "physics" of propagation. In a digitally-connected telecommunication world, rooms of the kind which will be described enclose remotely collaborating musicians in their own reverberated sound. The ambience which results is the product of an acoustical loop which creates room-like resonances created between internet endpoints which recirculate sound echoes on the paths between them. These are synthesized acoustical spaces engineered to resemble actual rooms and distinct from other kinds of online rooms where "room" is used metaphorically for gatherings of users participating in teleconference or chat applications. The present article describes room-like internet reverberation for local area and wide area networking, respectively named LAIR and WAIR. Aspects of the medium, algorithms used and initial musical experiments are detailed. To support these topics, the article also presents a theory of operation for jacktrip, the low-latency internet streaming software which was modified for the project.

Keywords: network music performance, internet acoustics, jacktrip, internet reverberation, Schroeder-style reverberation, freeverb, echo construction

OPEN ACCESS

Edited by:
Mark Brian Sandler,
Queen Mary University of London,
United Kingdom

Reviewed by:
Charalampos Saitis,
Technische Universität Berlin,
Germany
Franziska Schroeder,
Queen's University Belfast, United Kingdom

***Correspondence:**
Chris Chafe
cc@ccrma.stanford.edu

Specialty section:
This article was submitted to Digital Musicology, a section of the journal Frontiers in Digital Humanities

Received: *02 March 2018*
Accepted: *08 November 2018*
Published: *26 November 2018*

Citation:
Chafe C (2018) I am Streaming in a Room. Front. Digit. Humanit. 5:27. doi: 10.3389/fdigh.2018.00027

1. INTRODUCTION

Internet reverberation requires at least two hosts to create an acoustical loop. Closer endpoints—in terms of network audio round-trip time (RTT)—are associated with the sound of smaller-sized rooms. Auditorium-sized reverberation results from longer distances, for example, between continents.

Multiple rooms may coexist and opening acoustical portals between them is a matter of interconnecting the audio streams of different rooms. Each new participating endpoint joins the acoustical space by becoming a node in an interconnected mesh. All sounds entering the mesh are reverberated by the mesh.

Echoes often plague voice and music telecommunications systems. Squelching annoying feedback with echo cancellation algorithms becomes necessary when delays are long enough and echoes are loud enough to be perceptible. It's the same with real rooms. Depending on its intended application, a listening space may need acoustical treatment to dampen wall reflections or conversely loudspeakers may be used to enhance the direct sound of the sound being listened to. Reverberation which is completely appropriate for a choir singing in a cathedral may obscure intelligibility of someone speaking to the audience. Management and manipulation of room resonances is an age-old tool in creation of good sounding spaces for music and speech. Passive modifications use curtains, acoustical absorbers and diffusers and are generally subtractive. Active electronic systems use real-time digital signal processing (DSP) and are generally additive.

The technique presented here, a form of echo construction, does the opposite of echo cancellation. Software is used to create "internet walls" which are additive in nature. To see how

it works, let's think of sound propagating in any medium. A sound source emits a sound (in air, it's a pressure disturbance and on the internet, it's a stream of packets of non-silent audio data) which for illustration's sake we can simplify by imagining as an impulse like a balloon pop. The balloon's impulsive pressure disturbance expands outward as an increasing sphere until it's energy is entirely dissipated. But what if it hits a wall along the way? The impulse bounces off the wall and creates an impulse reflection. For simplicity, we'll observe the reflection from the point of view of the source (balloon) position. Out to the wall and back again at the medium's speed of sound. A second wall inserted right behind the source position will create a train of echoes. Any sound emitted will create a diminishing series of copies of itself—a bounce of a bounce of a bounce, until fully dissipated.

The DSP version of this is the filtered delay loop (FDL) which is an infinite impulse response (IIR) "unit reverberator" first mentioned by Moorer (1979). It's a comb filter modified to have a low-pass filter in its feedback loop with which "The purpose of placing a filter in the loop is to simulate the effect of the attenuation of the higher frequencies by the air." As shown in **Figure 1**, it's a feedback circuit in which the time taken around the loop determines the base frequency of the repetition. Inserting the attenuating low-pass filter causes the recurring train of echoes to die out and completes the approximation of what happens in air between two parallel walls.

Internet echo construction uses FDLs. Taking the place of two walls, the two hosts of a streaming connection act as reflectors which transmit back what they receive. The FDL low-pass filter is inserted somewhere in the acoustic loop. For example, this could be at one or even both of the hosts' loopback algorithms. Sounds can be emitted into the loop from either host, as if from either side of a room.

Real rooms have complex geometries with many walls, reflection paths and resonances. We'll see how internet reverberation can be made similarly complex by cloning acoustic loops to create banks of them and adjusting the banks to approach the necessary density and variety of resonances.

The final section of this article is devoted to a theory of operation which explains and differentiates in detail the three modes of the peer-to-peer streaming system "jacktrip" (Cáceres and Chafe, 2010a). These are its standard two-way connection mode, its hub and spoke mode, and lastly, its mode supporting internet reverberation over wide area networking.

2. BACKGROUND

Early study of internet acoustics at CCRMA required the development of a system for low-latency, uncompressed audio streaming over IP. That software evolved into jacktrip and is shared today as an open-source application widely used for jamming, rehearsing and concerts. Similar systems are discussed in a comprehensive review of network music performance technologies in Rottondi et al. (2016). The present project revisits jacktrip's original use as part of an experiment to treat acoustical loops in the internet as sound-producing objects. This idea relates to certain methods for physical modeling sound synthesis the earliest of which is the Karplus-Strong plucked string, an efficient computer algorithm consisting of delay lines and loop filters (Karplus and Strong, 1983). The KS string became ubiquitous in computer music with memorable compositions (for example, David Jaffe's *Silicon Valley Breakdown*) and numerous extensions to the technique (Jaffe and Smith, 1983; Sullivan, 1990; Smith, 1993).

A KS-like algorithm entered the realm of internet acoustics through experimentation between two hosts (Chafe et al., 2002) and by 2003 had produced a distributed algorithm for "plucking the internet." The algorithm's delay memory was no longer local

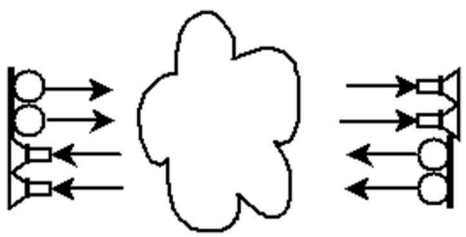

FIGURE 2 | Depiction of a single full-duplex (SFD) jacktrip connection streaming stereo uncompressed audio bi-directionally between two network hosts. Software on both sides packetizes incoming local audio and sends it to the cloud with minimal delay, likewise playing back audio from received packets with minimal delay.

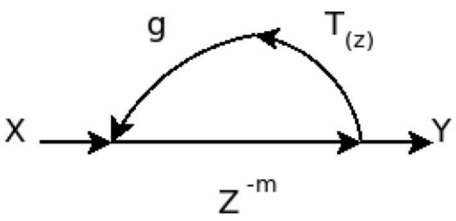

FIGURE 1 | A filtered delay loop (FDL) which is an infinite impulse response (IIR) "unit reverberator." The output signal Y is the result of mixing the input signal X with feedback from T(*z*) which has been delayed by *m* samples and multiplied by gain *g*.

FIGURE 3 | KS-like algorithm in which audio recirculates between two hosts. The SFD software, **Figure 2**, has been modified on both sides to include audio loopback and a loop filter (not shown). Aka "SoundWIRE" for Sound Waves on the Internet from Real-time Echoes, this circuit can be excited or "plucked" to sound like a guitar string.

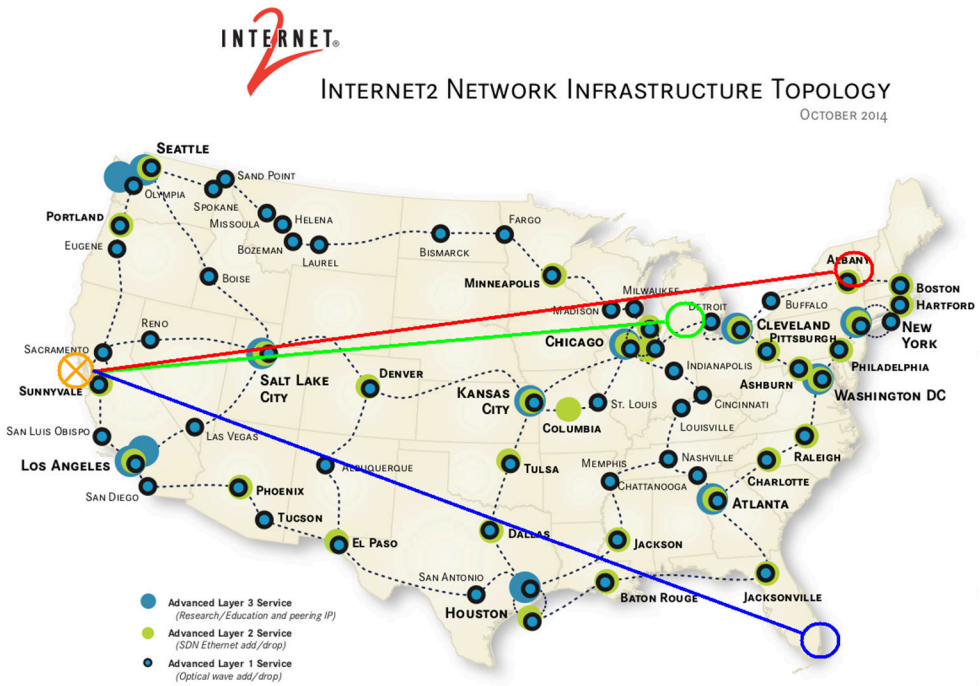

FIGURE 4 | NPAPW 2016 demonstration of WAIR system. The red, green and blue lines indicate individual freeverb "rooms" which are cross-patched at the summing site in California. Musicians in New York, Michigan and Florida could hear each other in a conjoined acoustical space.

computer memory (as in the original KS string) but the time of flight across an internet path. Pitch frequency of a recirculating "pluck" excitation (which could be had by simply tapping a microphone on either side) was a direct result of the path's RTT, **Figure 3**. And since the pitch fluctuates as RTT varies, it was conceived of as a very sensitive means for sonifying network quality of service (QoS) (Chafe and Leistikow, 2001).

Vibrating guitar strings and echoing parallel walls can both be modeled with FDLs. For a KS-like guitar string, the loop filter is tuned to be "ringy" (high Q, with strong resonance). The parallel wall case is the opposite, typically a very damped (low Q, weakly resonant) loop. Once an internet implementation of the guitar string had proved that the requisite time delay could be obtained using the network, it was natural to contemplate implementation of internet reverberators using well-known, FDL-based reverberation techniques (Chafe, 2003). Banks of FDLs would mimic the complexity of room geometry. Because each FDL element requires a separate channel, the idea would capitalize on jacktrip's support of large numbers of synchronized, parallel audio channels (extreme tests of streaming capacity have hit hundreds of channels).

Internet reverberation was demonstrated a decade after having been described in concept. The period saw changes to jacktrip's architecture, adoption of an updated reverberator algorithm, "freeverb" (Smith, 2010), and incorporation of a new DSP programming language (Faust[1]) for coding freeverb and its FDL banks. In 2013, a LAIR system was implemented consisting of three simultaneous freeverb "rooms" running on Stanford's campus-wide network. Participants at three endpoints were interconnected through LAIR portals (Chafe and Granzow, 2013). In 2016, the system was improved for wide area networking and demonstrated with four endpoints in a nation-wide WAIR mesh.

LAIR and WAIR are similar in many ways and from here on the remainder of the article will present details pertaining to the more recent WAIR system.

3. WAIR

3.1. N-way Mesh of Reverberators

In the four-endpoint WAIR shown in **Figure 4**, a central server is in California and three clients are located at points along the East Coast. Each of the clients sets up its own two-way freeverb circuit with the central server. The server runs in jacktrip's multiclient (audio hub) mode (Cáceres and Chafe, 2010b), a persistent process which listens for incoming client connections. **Figure 5** shows mode's hub and spoke design. Clients (or "spokes") connect at will and on connection begin bi-directional streaming with the server. The hub's automatic system for management of connections was designed as an improvement over manually maintaining many SFDs from a central point.

The central hub's own audio source can be distributed to all clients and the clients' streams can remain independent from one another. In many situations, however, it's desirable that audio streams be cross-patched to allow clients to

[1]http://faust.grame.fr/

hear one another. Cross-patching happens at the server either manually, with a jack patching application (for example, qjackctl[2]), or programmatically through APIs offered by the host's audio service, such as the jack audio connection kit.[3]

WAIR does the latter by incorporating a portal connection procedure capable of rewiring itself when it senses clients connecting or disconnecting. The idea is analogous to opening doors between newly-created rooms. When a new client's freeverb is instantiated, the portal algorithm makes the signal connections required for sharing audio between the new freeverb and all existing freeverbs. The result is an audio mesh made of dynamic nodes where each node has its own audio perspective on the total scene. The impression for a given WAIR participant is that they're in a room of their own and from that room they can hear slightly more distant sounds in all other currently running rooms. Tuning the cross-talk between freeverbs is sensitive. The system will go into self-feedback if the portal gain parameter is too great.

3.2. Freeverb in Jacktrip—Implementing an Internet Reverberator

An implementation of the freeverb reverberator is included as a library function in distributions of the Faust DSP language. Freeverb is a high-quality example from a class of reverberators with elements and structures proposed by Schroeder (1962) and further described in Moorer (1979). Freeverb's well-known antecedents from the 80's are exemplified by JCRev and NRev[4]. Across this class of reverberators, there are differences in quality due to the number of elements and channels used, their sequential arrangement and the exact parameter tunings applied.

Freeverb's parameters offer control of damping, room size and dry/wet mix. Freeverb's 16 FDLs, **Figure 6**, have delay lengths tuned to time delays ranging from approximately 25 ms to 37 ms. Like all Schroeder-style reverberators, coincident fundamental resonances (and their harmonics) are avoided by ensuring that delay times are mutually prime. For use in the WAIR system, freeverb has been modified so that its bank of FDL elements are the product of looping audio between two network endpoints. The substitution of network delay starts with subtracting 1100 samples from the original delay lengths (an equivalent of 23 ms of round trip delay). In cases where more delay is desired than the network path provides, an additional amount can be added back in at the client host (by command line specification).

Freeverb is shown in **Figure 7** with two DSP blocks, recirculating FDLs and inline APs. For the WAIR implementation these blocks are installed in the client using jacktrip's DSP plug-in architecture (Cáceres and Chafe, 2010a). "ProcessPlugin" modules are programmed in Faust. The language is especially well-suited for generating complex multichannel circuits which can be emitted as C++ and then compiled into the ProcessPlugin format.

The WAIR system is a combination of hub mode, WAIR servers, WAIR clients, portals and plugins. **Figure 8** illustrates the components involved in creating a room with the client in Florida and hub in California. Additional clients in Michigan and New York function like the one in Florida (components omitted in the figure). The portals which are cross-patched in California allow the 3 WAIR rooms to hear each other. A "DCB" plugin is installed on each server in the signal path to its portal. This plugin computes a DC-blocking filter and applies the portal's gain factor.

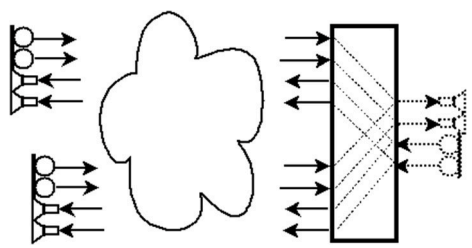

FIGURE 5 | jacktrip's multiclient (audio hub) mode. The main server (on the left) spawns multiple dedicated servers for an arbitrary number of clients (on the right). Audio input and output cross-patching happens at the main server, if needed.

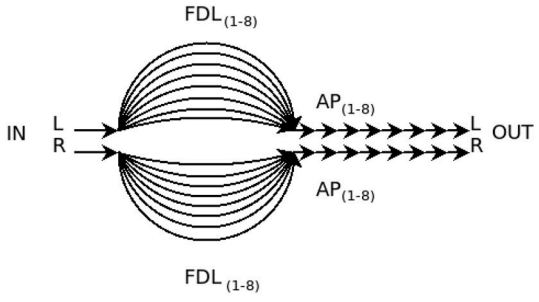

FIGURE 6 | Freeverb's 2 × 8 FDLs and 2 × 8 All-Pass delays.

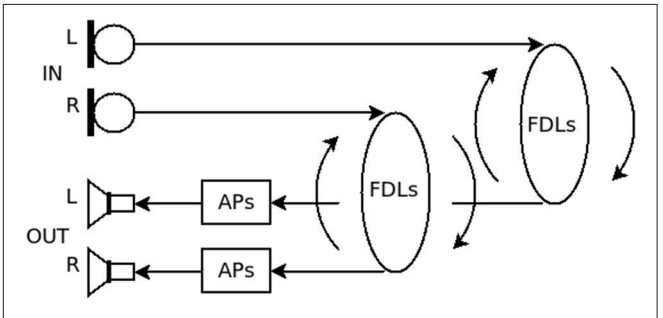

FIGURE 7 | Freeverb depicted as blocks.

[2]https://qjackctl.sourceforge.io/
[3]http://www.jackaudio.org/
[4]https://ccrma.stanford.edu/software/stk/

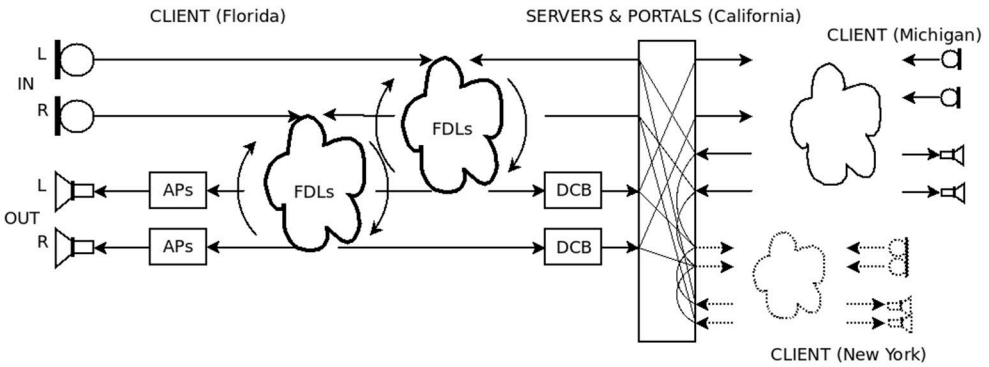

FIGURE 8 | A 3-room WAIR system with hub in CA and clients in FL, MI, and NY.

3.3. Implementing WAIR in Jacktrip

3.3.1. Theory of Operation

The following theory of operation begins with describing the way jacktrip's two original modes operate. The SFD jacktrip connection mode is documented first because it is a component of hub mode. The latter system spawns multiple instances of the former. More information on both modes can be found in Cáceres and Chafe (2010a) and Cáceres and Chafe (2010b), respectively. The detailed sequences presented here have not been documented elsewhere and while making for somewhat tedious reading, are needed to prepare the presentation of WAIR mode and its various extensions to hub mode.

3.3.1.1. jacktrip single full-duplex connection (SFD) session

Standard jacktrip operation sets up a single full-duplex audio streaming between two hosts, a server and a client. The only actual difference between server and client is the server's need to have a public IP address and its initial function of listening for incoming connections. Otherwise, they consist of identical sequences of setting up network-related and audio-related processes. The order of events is transcribed below. It is initiated and ended by underlined commands issued by the hosts' operators (either human or script). All other steps proceed automatically.

Before launching a jacktrip job, the host's audio service needs to be running and accepting clients so the job can connect to the local audio system. An example of an audio server which allows dynamic connections from jacktrip is the jack audio connection kit mentioned above.

3.3.1.2. SFD sequence

server host — (begins session, listens for any client)
A jacktrip server application is started on a host by issuing the command **jacktrip -s**
The application instantiates a jacktrip instance with mode set to SERVER

step 1 — The jacktrip instance checks if the application's intended network ports are already in use, sets up the desired audio interface and installs an audio callback in the already running audio server

2s server only — The instance creates a temporary UDP receiver socket and starts listening for incoming datagrams
The application prints **Waiting for Connection From a Client...**

client host — (waits for server, streams to server)
A client application is started on a host by issuing the command **jacktrip -c <server>** which requires the server's IP address or name
The application instantiates a jacktrip instance with mode set to CLIENT

step 1 (same as server)

2c client only — The jacktrip instance sets the peer address for its network receiving and sending processes

step 3 — The instance forks the receive process which binds its socket to the receive port, sets up its ring buffers, sets real-time priority and starts listening
The application prints **Waiting for Peer...**

step 4 — The instance forks the send process which binds its socket to the client host, sets up its ring buffers, sets real-time priority and starts transmitting

server host — (waits for client, streams to client)
When a datagram is received by the jacktrip instance, the incoming packet's IP address is identified as the client and the temporary socket is deleted

step 3 (same as client)

step 4 (same as client)

both hosts	(start audio, verify incoming stream, run indefinitely)
step 5	The audio process starts
step 6	The application waits in its event loop
step 7	When the receive socket receives its first incoming datagram, it checks the packet's audio settings The application prints **Received Connection from Peer!**
step 8	Outgoing and incoming datagrams continue to stream, the receive process keeps track of timing between incoming datagrams and if they stall out, the application prints **UDP waiting too long (more than 30 ms)...** **UDP waiting too long (more than 30 ms)...**
both hosts	(end session) The application is stopped by issuing a **<ctrl>c** command

The handshake between server and client relies on connections to known UDP port numbers, (the pre-determined default is 4464). If agreed upon ahead of launching both jobs, a port offset can be specified with **-o <offset>** added to the above commands, for example, **jacktrip -s -o10** starts a server on port 4474 which is reached by **jacktrip -c <server> -o10**.

The similarity of server and client makes it possible to connect two clients together (in -c mode) if both have public IP addresses. Furthermore, it's fine for a server to start after its client (or, for instance, to stop and restart one side while the other stays running).

3.3.1.3. jacktrip hub mode

For a server in hub mode whose job is to tend to multiple client spokes, the story starts with understanding how ephemeral ports work. Also called dynamic ports, these are unique temporary ports provided by the hub server in response to connection requests initiated by hub mode clients. Each ephemeral port is associated with an automatically spawned SFD server. All initiation requests are sent to the hub's common listening port. Existing SFDs persist while the hub server tends to new clients wishing to establish connections.

3.3.1.4. hub and spoke sequence

hub server host	(begins session, listens for any client, spawns JackTripWorkers as needed) A jacktrip hub server application is started on a host by issuing the command **jacktrip -S** The application instantiates a UdpMasterListener The UdpMasterListener instance opens a TCP socket on the standard port and begins a loop listening for connections
step 6	(same as SFD)
	The application prints **JackTrip HUB SERVER: TCP Server Listening in Port = 4464** **JackTrip HUB SERVER: Waiting for client connections...** ==========
client host	(initiate connection) A client application is started on a host by issuing the command **jacktrip -C <server>** which requires the server's IP address or name
step 1	(same as SFD) The application instantiates a jacktrip instance with mode set to CLIENTTOPINGSERVER.
2C client only	The jacktrip instance connects to the server's TCP port and sends its UDP receive socket port number
hub server host	(advertises ephemeral port) When a connection is made to the UdpMasterListener instance TCP socket, the incoming packet's payload contains the UDP port which the client wants to use. The application prints **JackTrip HUB SERVER: Client Connection Received!** **JackTrip HUB SERVER: Client Connect Received from Address : <client>** **JackTrip HUB SERVER: Reading UDP port from Client...** **JackTrip HUB SERVER: Client UDP Port is = 4464** and sends its ephemeral port to the client
client host	(receives the port, closes the TCP connection and continues as SFD client)
steps 3-8	(same as SFD)
hub server host	(spawns a dedicated SFD server, continues the loop) The UdpMasterListener spawns a new JackTripWorker listening in SERVERPINGSERVER mode, adds it to the JackTripWorker thread pool and starts it. The application prints **JackTrip HUB SERVER: Client TCP Connection Closed!** **JackTrip HUB SERVER: Spawning JackTripWorker...** **JackTrip HUB SERVER: Starting JackTripWorker...** **JackTripWorker: PeerNumChannels = <chans>**

steps 1, 2s, 3-5 (same as SFD)
The UdpMasterListener increments the number of running JackTripWorker threads and the application prints
JackTrip HUB SERVER: Total Running Threads: <threads>
==========

steps 7, 8 (same as SFD)
The UdpMasterListener loop continues and the application prints
JackTrip HUB SERVER: Waiting for client connections...
==========

client host (end session)
The application is stopped by issuing a **<ctrl>c** command

hub server host (releases a dead session)
If the client stream stalls out for too long, its server ends the session and the JackTripWorker ID and port are freed for future use. The application prints
UDP WAITED MORE THAN 30 seconds. Stopping JackTrip...
JackTrip Processes STOPPED!
_____ _____
JackTrip ID = <ID> released from the THREAD POOL
_____ _____

hub server host (end session)
The application is stopped by issuing a **<ctrl>c** command

3.3.2. Modifications to Implement WAIR Mode

A third jacktrip mode was added to implement WAIR. The following are the specific modifications made to jacktrip for this purpose. Parameters, members and methods are listed for the classes affected. (jacktrip at the time of these modifications was version 1.1 and was obtained from the project's repository in early 2018 [5])

WAIR mode extends hub mode and is invoked by adding the **-w** argument. Specify **-wS** or **-wC** to start, respectively, either a WAIR server or a WAIR client. The new mode adds a parameter member to the Settings class (**mWAIR**).

Two additional parameters, also new members of the Settings class, can be set by command line: **-N** (to set **mClientAddCombLen** for the addition of extra delay to the FDLs), and **-H** (to set **mClientRoomSize** which overrides the default value of Freeverb's room size).

A fourth new parameter member, **mNumNetRevChans**, is set internally to a fixed value of 16 channels. This specifies the number of FDLs in each WAIR connection, each of which requires a separate network audio channel.

In the Settings class method startJackTrip, jacktrip instances created for clients have two ProcessPlugins appended, **ap8x2** (for Freeverb's stereo series of 8 all-pass delays per audio input channel) and **Stk16** (to create the OnePole filters for the 16 FDLs, extend their lengths to prime relationships, and apply either **-N** or **-H** modifications).

The UdpMasterListener class has a new method, connectMesh, with which the hub server manages audio connections between spawned WAIR rooms. When a new room goes live, its audio is cross-patched into other rooms with connectMesh(true) and when it is released it is deleted from the mesh with connectMesh(false). The cross-patching functionality is borrowed from JMess, an application for storing and restoring jack patches.[6]

Spawned servers belong to the JackTripWorker class. In WAIR mode these are given unique IDs (with names like "WAIR0, WAIR1") for cross-patching by connectMesh. These servers have one ProcessPlugin appended, **dcblock2gain** (for DC blocking between WAIR rooms and setting the gain between WAIR rooms).

The AudioInterface class manages audio signal buffers for network and audio input/output, and signal processing. Sizes have been adjusted to accommodate the extra network audio channels and two buffers have been added to handle intermediate stages of the signal processing plugins.

As always, an audio callback function will be installed in the already running audio server. The following specifies the normal callback tasks and then provides details on the extensive modifications necessary for WAIR's audio callback function.

3.3.2.1. SFD and hub modes audio callback

The original audio callback comprises 4 steps.

audio input — local audio input is transferred from the audio server (**inBuffer**)

net output — computeProcessFromNetwork calls receiveNetworkPacket (**mOutputPacket** → **outBuffer**)

net input — computeProcessToNetwork calls sendNetworkPacket (**inBuffer** → **mInputPacket**)

audio output — local audio output is transferred to the audio server (**outBuffer**)

3.3.2.2. WAIR mode audio callback

audio input (same as above)

[5]https://github.com/jcacerec/jacktrip

[6]https://github.com/jcacerec/jmess-jack

net output	computeProcessFromNetwork calls receiveNetworkPacket (16 ch **mOutputPacket** → 16 ch **mNetInBuffer**)
client DSP	client computes 16 ch **Stk16** ProcessPlugin (**mNetInBuffer** → **mInProcessBuffer** → **Stk16** → **mOutProcessBuffer**)
server DSP	server is a 16 ch straight wire (**mNetInBuffer** → **mOutProcessBuffer**)
net input	computeProcessToNetwork calls sendNetworkPacket after a 2 ch to 16 ch fan-out and mix ((2 ch **inBuffer** + 16 ch **mOutProcessBuffer**) → 16 ch **mInputPacket**)
client DSP	client fans in 16 ch to 2 ch and computes **ap8x2** ProcessPlugin (16 ch **mNetInBuffer** → 2 ch **mAPInBuffer** → **ap8x2** → **outBuffer**)
server DSP	server fans in 16 ch to 2 ch and computes **dcblock2gain** ProcessPlugin (16 ch **mNetInBuffer** → 2 ch **mAPInBuffer** → **dcblock2gain** → **outBuffer**)
audio output	(same as above)

4. CONCLUSION

The occasion for the first public demonstration of WAIR was the 2016 meeting of the Network Performing Arts Production Workshop. Four endpoints were connected in a nation-wide mesh as discussed above, with musicians at the New World Symphony Concert Hall in Miami, the University of Michigan, Ann Arbor and Rensselaer Polytechnic Institute, Troy. The WAIR server was running at CCRMA, Stanford University.

The approximately 20 minute improvisation which was performed[7] featured carillon bells (in a studio), saxophone, daxaphone, fretless electric guitar and cello. Studio engineers in the audience at the New World Symphony reported that the acoustical result heard over a stereo PA system in the symphony hall was attractive and complemented the hall's own sound. Performers were able to hear others well and said they experienced a "composite hall." During sound check we found that the portal gain parameter interacted with the freeverb room size parameter and could lead to self-oscillation (feedback) with either value being too great.

WAIR mode is included in an upcoming release of jacktrip with the hope that others may be interested in experimenting with its possibilities. Additionally, the release includes a new command line argument **-V** which turns on "verbose" mode and prints exactly the step numbers detailed in the sequences for all modes above.

There have been multiple motivations for providing this informantion. First, it is hoped that WAIR mode has been sufficiently documented as a concept, and secondly, that its jacktrip source code modifications can be more easily followed. Lastly, the detailed sequences should provide a more precise understanding of jacktrip execution order, something which has been somewhat difficult to grasp heretofore and which is needed if the system is to be ported to other languages and systems in the future.

AUTHOR'S NOTE

The title "I am Streaming in a Room" is a play on "I am Sitting in a Room," a wonderful work by Alvin Lucier hereby appropriated and unwittingly twisted. I couldn't help myself.

AUTHOR CONTRIBUTIONS

The author confirms being the sole contributor of this work and has approved it for publication.

ACKNOWLEDGMENTS

Several musicians, led by Rob Hamilton and John Granzow, have been involved in demonstrating WAIR. My deepest gratitude to colleague and collaborator Juan-Pablo Cáceres who continues to support jacktrip. Thanks also to reviewers for very constructive and detailed suggestions.

[7] https://purl.stanford.edu/ty997vz5847

REFERENCES

Cáceres, J-P., and Chafe, C. (2010a). Jacktrip: under the hood of an engine for network audio. *J. New Music Res.* 39, 183–187. doi: 10.1080/09298215.2010.481361

Cáceres, J-P., and Chafe, C. (2010b). Jacktrip/soundwire meets server farm. *Comput. Music J.* 34, 29–34. doi: 10.1162/COM_a_00001

Chafe, C. (2003). "Distributed internet reverberation for audio collaboration," in *Audio Engineering Society Conference: 24th International Conference: Multichannel Audio, The New Reality* (Banff, AB: Audio Engineering Society).

Chafe, C., and Granzow, J. (2013). "Internet rooms from internet audio," in *Proceedings of Meetings on Acoustics*, Vol. 19 (Montreal, QC: Acoustical Society of America), 1–6.

Chafe, C., and Leistikow, R. (2001). "Levels of temporal resolution in sonification of network performance," in *Proceedings of the 7th International Conference on Auditory Display (ICAD2001)* (Helsinki), 50–55.

Chafe, C., Wilson, S., and Walling, D. (2002). "Physical model synthesis with application to internet acoustics," in *Proceedings 2002 International Conference on Acoustics, Speech and Signal Processing* (Orlando, FL: IEEE), IV–4056–IV–4059.

Jaffe, D., and Smith, J. (1983). Extensions of the karplus-strong plucked-string algorithm. *Comput. Music J.* 7, 56–69.

Karplus, K., and Strong, A. (1983). Digital synthesis of plucked string and drum timbres. *Comput. Music J.* 7, 43–55.

Moorer, J. (1979). About this reverberation business. *Comput. Music J.* 3, 13–28.

Rottondi, C., Chafe, C., Allocchio, C., and Sarti, A. (2016). An overview on networked music performance technologies. *IEEE Access.* 4, 8823–8843. doi: 10.1109/ACCESS.2016.2628440

Schroeder, M. (1962). Natural sounding artificial reverberation. *J. Audio Eng. Soc.* 10, 219–223.

Smith, J. (1993). "Efficient synthesis of stringed musical instruments," in *Proceedings of the International Computer Music Conference*, 64–71.

Smith, J. (2010). *Physical Audio Signal Processing : For Virtual Musical Instruments and Audio Effects.* W3K Publication.

Sullivan, C. (1990). Extending the karplus-strong algorithm to synthesize electric guitar timbres with distortion and feedback. *Comput. Music J.* 14, 26–37.

Conflict of Interest Statement: The author declares that the research was conducted in the absence of any commercial or financial relationships that could be construed as a potential conflict of interest.