Live Coding in the Time of COVID

Scott Wilson
University of Birmingham
s.d.wilson@bham.ac.uk

Konstantinos Vasilakos
Istanbul Technical University (MIAM)
konstantinos.vasilakos@gmail.com

Serkan Sevilgen
Istanbul Technical University (MIAM)
ssevilgen@gmail.com

ABSTRACT

The COVID-19 pandemic has created both challenges and opportunities for live coding ensembles. These are particularly acute for ensemble which make use of networked live coding systems, such as the Birmingham Ensemble for Electroacoustic Research (BEER), and the Istanbul Coding Ensemble (ICE). This paper looks at two local area network works developed by BEER, and how they were adapted for internet presentation during lockdown, and discusses some possible future directions for collaborative work.

1. INTRODUCTION

This paper explores some issues around the practice of Live Coding during the COVID-19 pandemic, and the problems arising from the necessity of adopting networked performance practices. Specifically, we will look at adaptations and extensions of two pieces developed by the Birmingham Ensemble for Electroacoustic Research (BEER) based on low-latency local networked systems, to wide area network practice using emerging internet audio and video communications systems, often with high-latency conditions.

2. EXISTING PIECES

The two existing pieces adapted are both implemented in SuperCollider (McCartney, 1996) and make use of the Utopia networked music library (Wilson et al. 2013).

2.1 Pea Stew

Pea Stew (2012) was developed as a collaborative network sonification piece employing live-coded interventions in the signal path. It was inspired primarily by Nicolas Collins' classic analog feedback piece *Pea Soup* (Collins, 1974), which featured a single chain with an amplitude follower-driven phase shifter at its heart. This simple approach, when correctly tuned, allowed performers to create a 'site-specific architectural raga', in which shifting resonances allow evolving pitch patterns to emerge within the performance space (Collins 2011). *Pea Stew* implements a similar process sited within a network. Following a seeding process to initiate resonance, audio is streamed between all participants in a fully-meshed network topology. Through a mixer interface implemented in SuperCollider, performers could effectively vary the topology by turning down or muting the signals of some players, with negative gains permitted to allow for phase inversion. The phase-shifting is implemented using a Fast Fourier Transformation-based process on a bin-by-bin basis. In addition, performers can intervene in the signal chain at their local node using live-coded processors. In effect this creates a communal instrument, with no single performer in control. Figure 1 shows the basic signal chain for *Pea Stew*.

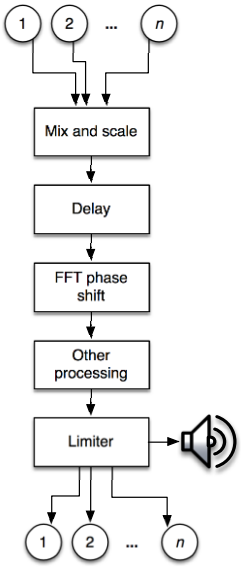

Figure 1. *Pea Stew* signal chain.

Audio networking was implemented using the *JackTrip* application (Chafe, et al. 2008). While this provided low-latency audio connections, because of its requirement that each connection be configured as an individual client-server pair, configuring networks of the type required for *Pea Stew* for varying numbers of performers on an ad hoc basis proved complicated and error-prone, especially in cases where some participants lacked experience with network music practices. For this reason, Wilson developed the UtopiaJackTrip utility class, which in combination with Utopia's peer discovery capabilities automated configuration in most cases.

A more detailed discussion of *Pea Stew* can be found in Wilson et al. (2014).

2.2 PianoCode

PianoCode (2014) was developed as a collaborative project between BEER and pianist Xenia Pestova. The central concept was to use machine listening techniques to capture aspects of improvisations by Pestova, using a 'listener' computer which shared note onset information to all participants over a local network. Data so acquired could be used as material for live-coded responses. Initial implementations developed through workshop sessions suffered from a form of 'latency' inherent to the use of coding as an 'interface': Live coding, while very flexible, is not always fast. We found that the rate of incoming data was high enough to be potentially overwhelming, and the lapse of time between input and live-coded response was often too long to feel musically effective. A solution was found through the adaptation of a classic graphical representation of music, the piano roll. Figure 2 shows the *PianoCode* 'piano roll' interface.

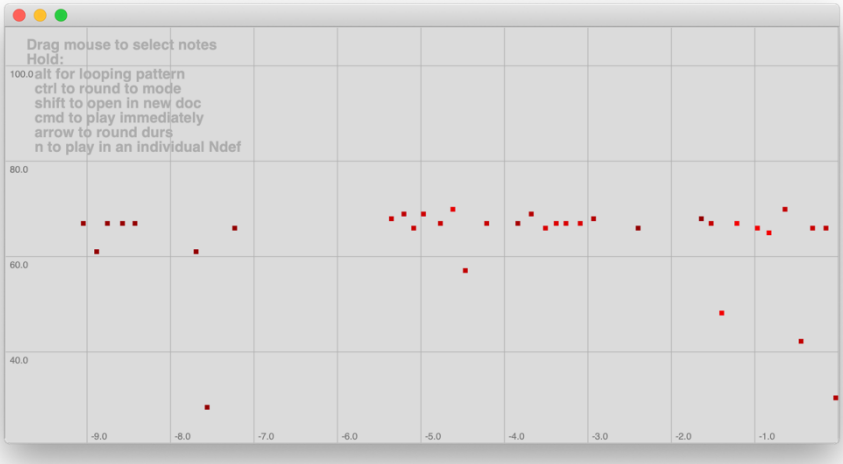

Figure 2. 'Piano roll' interface used in *PianoCode*.

This interface represents a constantly scrolling record of the last ten seconds of note onsets, with MIDI note number represented on the vertical axis and amplitude shown by brightness. Performers can click and drag to select passages and generate pre-formatted code with the selected onset sequences converted to SuperCollider patterns. Keyboard modifiers (including user-defined ones), can alter behaviour, e.g. play immediately, loop, etc.

The technology developed for *PianoCode* later formed the basis for a number of variants, including *BassCode*, *PercCode*, *DirkCode* (with composer Dirk Stromberg performing on his Phallophone electroacoustic instrument), and most recently *Symphony in Blue 2.0* (2021). This is discussed below along with a more detailed explanation of some of the technical aspects of this series of pieces in their most recent form..

3. PEA STEW (RECALIBRATED)

BEER's 10-year anniversary fell in 2021, and despite the limitations imposed by the pandemic and lockdowns, we wanted to do something special which could involve a variety of past and current members of the ensemble. BEER's 'mother' organisation, Birmingham ElectroAcoustic Sound Theatre (BEAST), was presenting their annual BEAST FEaST festival online with a theme of 'Recalibration'. Given this, adapting *Pea Stew* to a WAN/Internet performance for inclusion in the festival seemed a natural approach.

This however, presented several challenges, many of them typical of online electroacoustic music performance under lockdown conditions. The issues noted above regarding the complexity of networking remained, but with additional complications. Without direct access to university (or similar) computer networks with non-private internet addressing, most performers must deal with trying to stream audio across Network Address Translation (NAT), using home routers. (Even with centralised discovery, having at least one peer directly on the internet and not behind NAT greatly aids stability.) This can be problematic in several ways, including by adding latency, but one particularly problematic issue is to do with discovery. Without a central server to register participants and mediate communication, NAT can render performers' machines unreachable. While router configuration (e.g. configuring port forwarding or setting up a 'DMZ') can help with this, configuring and testing this is outside of many musicians' expertise. Supporting this remotely only adds to the difficulties.

A further problem is that while a number of online streaming apps do use centralised architectures for discovery and in some cases for streaming (with additional latency implications), most of these run as applications which rely on virtual audio drivers for patching. *Pea Stew* requires a relatively flexible routing configuration with input and output connections at different points in the signal chain, and feedback from each local source coming via other nodes over the network. Doing this with a single

app/virtual driver may not be possible, and running multiple instances of an app, while possible in some cases, only increases the already considerable complexity of configuration.

A solution was found through the use of the SonoBus (https://sonobus.net) audio-over-OSC application, and the VST plugin support for SuperCollider developed by Christof Ressi at IEM (https://git.iem.at/pd/vstplugin). SonoBus has proven one of the simplest to configure and easiest to use of the various streaming platforms that have become popular during lockdown. By running it as a VST hosted within SC itself, a single instance could be used and take advantage of SC's highly flexible routing and feedback facilities. The mixer functionality was removed in favour of using SonoBus' mixer interface. Performers need only install the needed components, and (after running the supplied code) join the correct SonoBus 'group'. Other configuration aspects are handled automagically in SC code.

While successful, and in some ways easier to use than the previous JackTrip approach, this online version differed in other ways. On LANs, network latency can be extremely low. In an internet performance, with performers in Birmingham, Istanbul, and Hong Kong, latency could easily be in hundreds of milliseconds. This added aspects of rhythm and delay in addition to pitched resonance. The lack of a performance space also created other opportunities. The performance documented at https://youtu.be/NlFM1D5vMTM uses Ambisonic spatialization and binaural encoding, combined with Zoom video and screen captures to create a somewhat unreal virtual performance 'space' out of disparate online elements.

4. SYMPHONY IN BLUE 2.0

Symphony in Blue 2.0 (SiB 2.0) is a live coding adaptation of Kamran Ince's composition Symphony in Blue (2012) (http://www.kamranince.com/html/compositions.php?id=189&page=). It is based on previous work of BEER developed in the *Piano Code* series of pieces (see section: 2.2, above). For this version, the Istanbul Coding Ensemble (ICE), the home live coding ensemble of the Center for Advanced Studies in Music (MIAM) at Istanbul Technical University, and Jerfi Aji (piano) from (ITU/MIAM) were joined by Wilson, who took part from Birmingham, in the U.K.

Analysis of the acoustic signals of the piano using machine listening in SuperCollider was implemented both locally and remotely, with the signal streamed to Wilson using SonoBus. The performance lasted for approximately ten minutes. We gracefully hope that it will be presented in the music/performances section of the International Conference on Live Coding (ICLC) this year. A link of the recording can be found at this link (https://youtu.be/vMXCswvoV2E).

Networking deployment has been at the forefront of live electronic/computer music utilized as a means to facilitate real time communication and distribution systems for sharing data and superimposing information to synchronize performers both locally and telematically. Primary examples of this have been demonstrated by the Hub, a live electronics group which also introduced the real time tweaking of source code of software at the same of the performance (Gresham-Lancaster, 1998; Collins and Escrivan Rincón, 2011). Other more recent examples include laptop ensembles namely, the Cybernetic Orchestra, Powerbooks Unplugged and BEER. Other examples that made use of network capabilities in a creative practice include Chris Chafe's *Ping* (Chafe, 2001) which used latency information of a network to create a multichannel work. Finally, a more recent work by Network Ensemble has also adopted networks in their creative practice (Smith and Tacchini, 2017).

4.1 Machine Listening for Live Coding over the Network

Machine listening and the use of signal processing and feature extraction has been a staple method since the very beginning of Computer music to facilitate real time dialog between acoustic instrumentalists and digital music practitioners on stage. Some early music works areRobert Rowe's *Flood Gate* and *Banff Sketches* (Rowe, 1992). Besides presenting an exhaustive overview of machine learning in computer music, some more recent examples relevant to the context of this section include the works of Dan Stowell and Nick Collins (Collins, 2015; Stowell & Plumbley, 2007). Interestingly, machine listening can be used as an interface to control a live coding system. Collins (2015) describes this approach of creating feedback loops between live input and its analysis data fed back into the control of the audio source and map these parameters using live coding as the core performance strategy:

> ...an abstract feedback loop can easily be constructed where an audio construct's output is analyzed, the analysis result used to feedback into control of the audio, and parameters of this graph (or the complete graph) made available for live coding manipulation.

As in *PianoCode*, machine listening software and networked music systems allow for the distribution of analysis data in the form of onset events. Using the Utopia platform locally the analysis data from the piano signal was shared to ICE coders via OSC messages offering pitch, amplitude, and onset time information for each detected note onset event. To simplify the wide area network aspects of the piece, the piano audio signal was streamed to Wilson over SonoBus and the analysis performed locally for his use. While this meant that there was additional latency in the analysis response 'feedback' loop, and that Wilson was possibly working with a slightly varying analysis stream, this did not appear to influence the overall interaction between the performers. (Indeed, part of the 'philosophy' of the *PianoCode* series of pieces is to embrace the limitations and inaccuracies of machine listening approaches in a pragmatic and flexible way, with mistakes in transcription contributing to the process of variation of the input's musical material.) SonoBus was proven to provide a stable platform for audio distribution with fine adjustments and minimum latency distributing the piano signal over network (see section 3).

4.2 Technical Details and Hardware used in SiB 2.0

As noted above, ICE's communication platform is implemented with Utopia (see section: 2). We devised a custom-made system that was initially developed for *PianoCode* using the Pitch and Onsets Ugens provided in SC, which implement a autorcorrelation Pitch tracker and a spectral onset detector respectively. The onset trigger is delayed slightly, as this gives time for the pitch estimate to stabilize before the data is shared. This process is run on an analysis computer, and the resulting data is passed to all connected performers via the convenient `sendAll` method of Utopia's AddressBook class.

A code snippet illustrating the analysis follows:

```
in = Mix(SoundIn.ar(~audioIn));  // Reading data from computer's audio inputs.

amp = Amplitude.kr(in); //Amplitude follower Ugen.

chain = FFT(LocalBuf(~onsetBufSize), in * 10); // Fast Fourier Transformation live analysis, storing in a local buffer.

onsets = Onsets.kr(chain, thresh);  // Onset detection Ugen.

#pitch, hasFreq = Pitch.kr(in);  // Pitch tracking Ugen.

SendReply.kr(DelayN.kr(onsets * hasFreq, delay, delay), '/onsets1', [Peak.kr(amp, onsets), pitch, hasFreq]); // Wrapped in a SendReply class of SC to send over via OSC internal messaging.
```

Once this information is received from Utopia's OSC sharing capabilities it is rendered in every user's window showing the incoming data in the form of rolling note events (see fig: 2). The data is represented in real time, so the user can select certain notes to map those with the musical parameters of their sound SynthDefs. While the data can be rendered as patterns in SC, it can be also used directly as individual values for sound generation Ugens, such as SinOsc's frequency parameter, using the user implemented function capabilities mentioned above. An example of a predefined function that renders current pitch values as drones using oscillators follows:

```
// using the 'p' key as a modifier

~funcDict[\p] = {|durs, midinotes, amps, mods, soloDefName, controlNames, nameString|

	var ampScale, durScale, out;

	// gate arg can re-trig drone

	Ndef(nameString.asSymbol, {|t_gate = 1|

		ampScale = 0.5 / amps.sum;
```

```
            durScale = (40 / durs.maxItem).floor;

            out    =    SinOsc.ar(midinotes.midicps,    mul:    amps.collect({|amp,    i|
Env.sine(durs[i]    *    durScale,    amp    *    ampScale).delay(TRand.kr(0.0,    3.0,
t_gate)).kr(gate:t_gate)}));

            Splay.ar(out.scramble) // randomly reorder and pan across stereo field

      }).play(vol:0.5);

}
```

Similarly, some other custom functions were adopted, for example, solo 'instruments' for gestural improvisation using analysis material. This proved to be a very convenient method for improvising with the acoustic performance without having to type everything from scratch and build a dialog between solo live coding turns and specific piano parts.

Machine listening in live coding can be a very fruitful approach for creating coherence between acoustic performance and electronics. In our specific case of live coding over a network, we found it to be a very efficient strategy for providing a generic framework of interaction amongst coders, which can be one of the most challenging issues in live electroacoustic performance due to the broad possibilities of digital sound generation and the varying aesthetic preferences of each performer. Although the machine listening and code generation framework went some way to reducing the difference in responsiveness of coders and acoustic performer, some adjustments were needed. As noted above, the pianist can respond more or less instantly. With live coding however, more steps may be required: typing, execution, feedback of the result, correction and appreciation of the musical outcome, debugging problems and errors, and sometimes even adjustment between first aims and embracing the unexpected artifacts that arise from tweaking code on the fly. The performers' strategy then is to deploy some pre-prepared synthesis algorithms and mapping strategies using incoming values from the listening system. The typical live coding strategy of deployment followed by cycles of modification allows for some more nimble responses.

4.3 Studio Adaptation for Online Live Coding Performances

SiB 2.0 was premiered and recorded in the studio of ITU/MIAM. For the performance of the piece the ensemble used a new system for visualization of the code executed, implemented using web-based applications and server packages in NodeJS (see below). In addition, the work was filmed, which changed the atmosphere of the performance drastically from what the performers were used to during the rehearsals.

As all the Istanbul-based performers were in the same space, the monitoring of all signals was handled via a headset monitoring system with individual mixes controlled by each performer. These were streamed using SonoBus to Wilson in Birmingham, and his signal was similarly routed to another audio channel in the studio monitoring system.

Although this was a new experience for the performers it offered a different interaction level, which was embraced by the coders and instrumentalists. Indeed, the idea of performing a live-coding piece in a studio and being able to hear every performer individually seems at first a paradox but at the same time offers some technical affordances, as opposed to traditionally hearing through a general PA system on stage. But also it poses some challenges. For example, the improvisational part of modifying algorithms on the fly is endangered by the ability of one to overfocus on the relationship of their individual sound dynamics with each performer, instead of relying on the general appreciation of the sonic outcome while improvising in a collective manner.

4.4 P5 Creating visuals in the Browser over the Network

It is common for live coding performers to project their code while improvising in front of the audience, as a means to expose virtuosity and emphasize the 'liveness' of the performance.

In this project we adopted web technologies using JavaScript and NodeJS (https://nodejs.org). Web-based applications coupled with web server packages can provide a powerful paradigm for distribution of various data amongst performers and other useful traits for collective interactions.

ICE's projection utilities are built on top of NodeJS and run in Chrome browsers visualizing each coder's script when executed in SC. The code is communicated via Utopia, to a host computer that runs the NodeJS. It then passes it as text using OSC messages to the interface of the application that is opened as a window in the browser. The idea behind this is simply to allow the audience to follow the execution of the code of each coder. As such few artistic enhancements are made to the text presentation other than adding some fade in/out and some simple animation: Once a piece of code is executed by a coder it ascends to the middle of the visualization window and after some given time it fades out. For cosmetic reasons and personalization of the environment a cube displaying the logo of the ICE is rotating inside the window. While minimal in approach, and not aspiring to the level of artistic visualisation, it acts as primer for the group to involve more appealing visualizations of code and graphics in the future, with a possibility of making use of web servers, for remote communication (see section: 5).

The technology that was used for this projection utility includes shaders (Gonzalez & Lowe, 2015) in P5JS. It is the online adaptation of the Processing language for visuals and creative coding projects.

5. FUTURE CONSIDERATIONS AND WORK

During a rehearsal or performance, members of networked ensembles often sit in the same physical room and connect to the same local area network. The COVID19 outbreak forced musicians to perform remotely, and networked musicians needed to tackle the problems of connecting with each other over the Internet. Musicians need to change their modem/router/firewall settings to open the ports for communication which can be cumbersome or unavailable if they are behind institutional firewalls like schools. Audio-sharing tools like Sonobus or JackTrip should be set up properly, and they require high-bandwidth Internet access. In his work with ICE, Sevilgen has been developing one approach to overcome those issues by sending the data and code that would produce the sound to peers and generating audio on each musicians' computer in real-time, instead of rendering audio locally and sending it to others over the Internet. The approach under development is similar to that employed in projects like the Extramuros application (Ogborn et al. 2015), but can use local installations of SC or similar platforms rather than a browser-based live coding environment. This would have the advantage of unifying communication in one network platform, and removing the need for piecemeal approachs for different data types.

The software library Remote OSC (https://github.com/serkansevilgen/remote-osc) is developed to enable easy data communication among remote performers. Musicians can use it as an intermediate layer to communicate, share data, and code with each other while keeping their music software, editors, programming languages the same. The goal is to provide a major building block for developing bespoke wide area networked music performance applications. The library handles data transmission over the internet and leaves the artistic decisions to the musicians. It has a client/server architecture to manage the traffic without the hassle of port settings on the router. Performers can send values to control synthesizer parameters or code snippets to others who use the same audio programming environment as SuperCollider. Local SuperCollider or other apps should synthesise the incoming message as audio. Since each client receives the same data, the same audio would be output on each local computer in most cases. (The use of pseudo-randomness is an obvious exception, as in Extramuros.)

Remote OSC is platform-agnostic and can work with any software as long as the message is in OSC (Open Sound Protocol) protocol. It is an open-source application written in Node.js. Remote OSC has two main components: a server application that has to be installed on a publicly available web server and a client application that needs to run on musicians' computers. The server acts as a traffic manager and distributes the incoming messages to all connected clients. The client application establishes a permanent connection to the server via Web Sockets and listens to messages coming from other performers through the server, and sends them to the backend audio application (like SuperCollider, Max/MSP, Csound, etc.). A video at https://www.youtube.com/watch?v=uP1_Jp-sfOc demonstrates an application of a networked real-time sonification. Both performers start their Remote OSC client programs and establish a permanent connection to each other via the server. Performer

1 generates data in SuperCollider and sends it as an OSC message to the Remote OSC client. Data is distributed to connected clients via the Remote OSC server. Performer 2 gets the data into SuperCollider and employs sound generation algorithms to produce sonification in real-time.

Remote OSC can be configured with many-to-many and bidirectional patterns for networked music ensembles. One-to-many, one-to-one and unidirectional setups can also work for projects with real-time sonification, live installations, etc. The OSC messages could carry text or visual data that can be consumed by a program like Processing for telematic audio/visual performances.

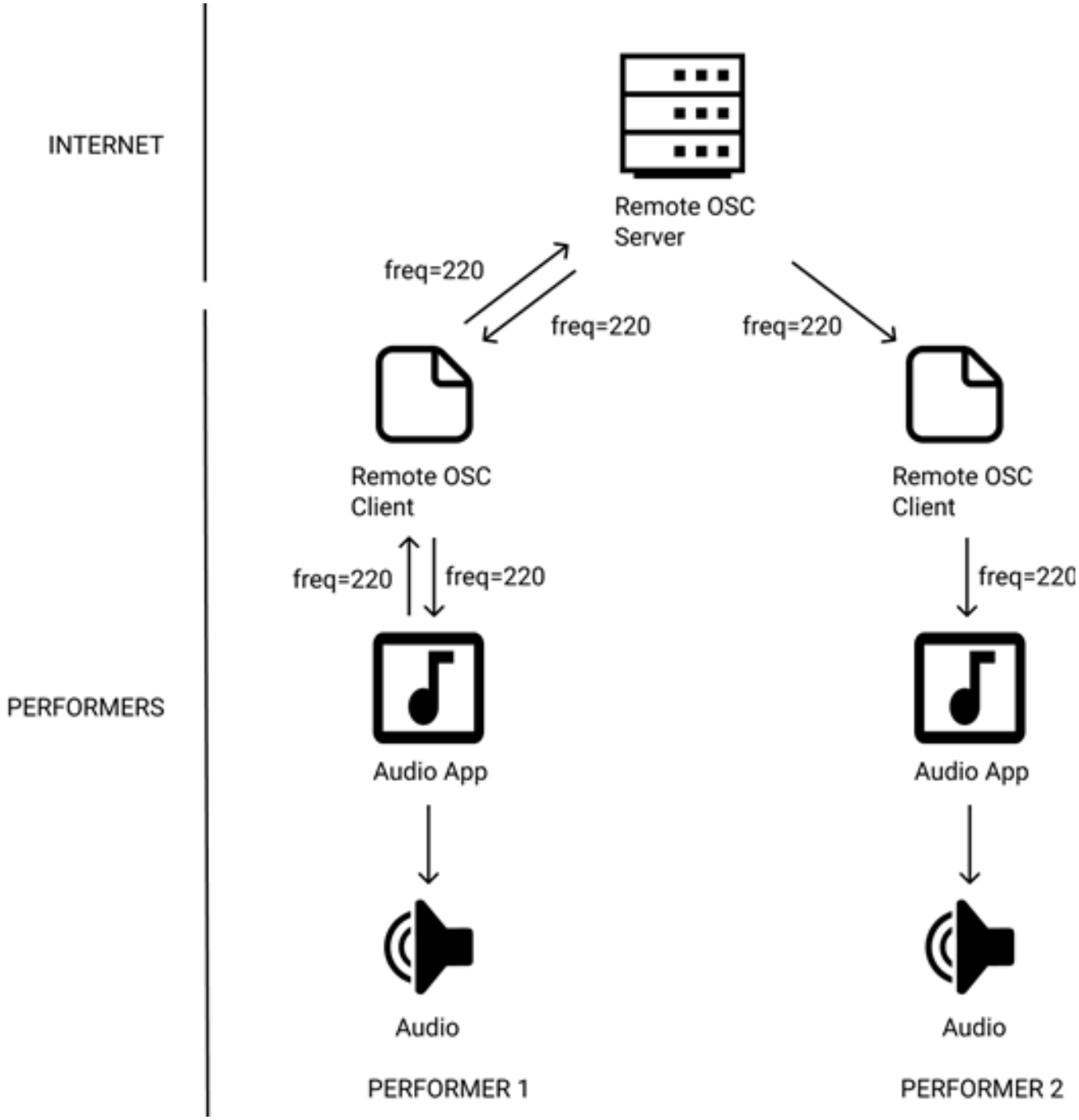

Figure 3 A typical use of Remote OSC in a networked ensemble. Performer 1 sends freq=220 data which will be received by all members

6. CONCLUSIONS

The COVID-19 pandemic and its related lockdowns have presented both challenges and opportunities to computer musicians. Particular issues surrounding network configuration, peer discovery, and latency confront performers whose practice embraces the use of network live coding systems, such as BEER and ICE. At the same time the situation has afforded us the opportunity to creatively explore solutions to these issues, which in themselves may be suggestive of further music making opportunities. It will be interesting to see how these developments continue to evolve, as we look forward to a future that while (hopefully) less constrained than over the past 18 months, continues to embrace the possibilities of telematic collaboration.

Acknowledgments

The authors wish to thank Xenia Pestova, Beast FEaST, Jerfi Aji, Onur Dagdeviren. Sound Recording/Mixing: Oğuz Öz. Studio Crew: Laçin Şahin, Deniz Ünsal, Nurgül Süleymanoğlu. Filming/video editing: Cem Onerturk. And MIAM Studio at Istanbul Technical University.

REFERENCES

Chafe, Chris et al. 2008 onwards. JackTrip. https://jacktrip.github.io/jacktrip/ Accessed 22 September 2021.

Chafe, C., 2001. Ping. [Music] Available at: <https://chrischafe.net/portfolio/ping/> [Accessed 27 Apr. 2021].

Collins, Nicolas. 2020. Improvising with Architecture -- Pea Soup and Related Work with Audio Feedback. Available online at www.nicolascollins.com/texts/peasouphistory.pdf. Accessed 22 September 2021.

Collins, N. (2015). Live Coding and Machine Listening. Proceedings of the First International Conference on Live Coding, 8. http://sro.sussex.ac.uk/id/eprint/60029/1/iclc2015-proceedings.pdf

Collins, N. and Escrivan Rincón, J. d', 2011. The Cambridge companion to electronic music. [online] Cambridge: Cambridge University Press. Available at: <http://dx.doi.org/10.1017/CCOL9780521868617> [Accessed 23 Feb. 2019].

Collins, N. (2015). Live Coding and Machine Listening. Proceedings of the First International Conference on Live Coding, 8. http://sro.sussex.ac.uk/id/eprint/60029/1/iclc2015-proceedings.pdf

Collins, N., 1974. Pea Soup. Available at: <https://www.nicolascollins.com/aboutpeasoup.htm> [Accessed 27 Apr. 2021].

Gonzalez, P., & Lowe, J. (2015). The Book of Shaders. The Book of Shaders. thebookofshaders.com

Gresham-Lancaster, S., 1998. The Aesthetics and History of the Hub: The Effects of Changing Technology on Network Computer Music. Leonardo Music Journal, [online] 8, p.39. https://doi.org/10.2307/1513398.

Lewis, G. E. (2000). Too Many Notes: Computers, Complexity and Culture in Voyager. Leonardo Music Journal, 10, 33–39. https://doi.org/10.1162/096112100570585 Smith, O. and Tacchini, F., 2017. Network Ensemble Selected Network Studies. The Strong of the Future. [online] Rizosfera. Available at: https://www.academia.edu/33394585/Network_Ensemble_Selected_Network_Studies [Accessed 27 Apr. 2021].

McCartney, J., et al. (1996-2021). SuperCollider. [Audio programming] Available at: http://supercollider.github.io/ [Accessed 28 Sep. 2021].

Ogborn, David & Tsabary, Eldad & Jarvis, Ian & Cárdenas, Alexandra & Mclean, Alex. (2015). Extramuros: making music in a browser-based, language-neutral collaborative live coding environment. 10.5281/zenodo.19349.

Rowe, R. (1992). Machine Listening and Composing with Cypher. Computer Music Journal, 16(1), 43. https://doi.org/10.2307/3680494

SCListeningHandout.pdf. (n.d.).

Stowell, D., & Plumbley, M. (2007). Adaptive Whitening for Improved Real-Time Audio Onset Detection. Proceedings of the International Computer Music Conference, 8. http://c4dm.eecs.qmul.ac.uk/papers/2007/StowellPlumbley07-icmc.pdf

Wilson, Scott, Norah Lorway, Rosalyn Coull, Konstantinos Vasilakos, and Tim Moyers. 2014. “Free as in BEER: Some Explorations into Structured Improvisation Using Networked Live-Coding Systems.” Computer Music Journal 38 (1): 54–64. https://doi.org/10.1162/comj_a_00229.

Wilson, Scott., Julian Rohrhuber, and Alberto de Campo. 2013. Utopia Network Music Library. Available online at https://github.com/muellmusik/Utopia. Accessed 22 September 2021.

Wilson, S., Collins, N., & Cottle, D. (Eds.). (2011). The SuperCollider Book. MIT Press. https://mitpress.mit.edu/books/supercollider-book