Experimental System for Latency-Based Networked Composition Support and Performance

Ari Liloia

February 24, 2025

Abstract

This report documents the design, implementation and testing of a prototype experimental latency-based system (ELBS) for networked musical performance. The system creatively incorporates a potentially disruptive amount of latency as a feature of performance and an opportunity for novel musical experiences. Specifically, the system introduces a model of performance influenced by technical restrictions (remote users subject to significant latency interacting through personal computers) and aesthetic characteristics associated with performance (in-time creation of complex music whose coherence can be attributed to ensemble coordination and individual musicianship). A further goal was to include four features not previously integrated in a single networked musical performance system: To allow the sequential relaying and interpretation of musical instructions, to create different versions of ensemble musical output for different end users, to sonify / artfully represent network characteristics, and to allow further editing of performances out of time. The complete prototype is a web application with a virtual machine-hosted server and a browser-based client application, which can be made accessible to users via an independently distributed URL. The report details further specifics of the playing paradigm, the prototype implementation, and the test session at which the system was tested to demonstrate its basic performance functionality. A composition derived from the edited output of the first test session was also created.

1 Introduction

Asynchronous or “out of time” musical collaboration via file sharing is commonplace. However, there are very few compelling systems for quasi-real time collaboration, other than emulations of being present in the same physical space in real time, an experience that is not possible to replicate perfectly due to communication latency. Some degree of latency is inevitable in networked systems, and is usually considered a problem whose effects on emulating real-time communication should be mitigated. Less explored is the idea of embracing latency as a feature of a networked application, and developing composition and performance techniques that incorporate delay.

Broadly, the goal of the proposed system is to facilitate non-realistic musical performance over a network. In the service of facilitating performance, the ELBS prototype has the following characteristics: The system gives users the impression that they are playing “live”, emulating or mediating characteristics associated with liveness when necessary. It supports the collaborative, simultaneous and in-time creation of music. Henceforth, “user” refers to a human operating a system.

In the service of facilitating performance that deviates from conventional models of performance without shifting musical agency away from users, the ELBS prototype has the following characteristics: The controls of the digital instruments provided to users are such that there is a low bar for entry, but skill can be developed such that complex music can be produced and virtuosity is possible. The number of dimensions of salience are reduced due to technical constraints and aesthetic concerns. While the system pursues secondary goals of facilitating novel interactions between users and with the network itself, the primary goal of enabling the creation of coherent music that displays complexity, in its dimensions of salience, that can be attributed to user activity, is not compromised.

We introduce several other goals that have not yet been explored within the same application: (1) to facilitate both performance and composition support, (2) to explore cascades of musical responsibility, (3) to display users’ musical activity to each other, (4) to produce distinct but equally valid musical outputs for different users, and (5) to sonify and artfully represent network characteristics.

To show that the prototype is functional and is nominally operable as a performance system, I presented a usable prototype of the system to the members of my thesis committee at an informal testing session. Between the session and my thesis defense, I used the record of the performance as a starting point for a piece of music composed out of time.

2 Background

2.1 Primer on Networking

The following is a review of some of the basic principles and considerations of networking.

2.1.1 Packet-Switched Networks

In "packet-switched" networks, data streams are segmented into datagrams or *packets*. Packets are transmitted, routed, and received independently on network links connected by packet switches, simulating a permanent and direct connection between end users. [1, Chapter 1.3, 2.2]. Packets contain a "payload", a segment of the data stream, and a "header" with data identifying its source, destination, and other details necessary for reconstructing data after successful transmission.

Transport-layer protocols, or sets of rules that operate on the level of endpoints for communication, or *sockets*, facilitate the partitioning, transmission and reception of packets. Protocols are also responsible for recovering from packet losses. A packet's payload and header size depend on the protocol used. Payload size is also limited by the connection's maximum transmission unit (MTU), the largest amount of data that can be transmitted as one packet. [1, Chapter 7.7]. Transmitting a data frame in larger and fewer packets lowers the total packet processing overhead.

Generally, network performance is evaluated based on the amount of data that can be transmitted per unit time (bandwidth / throughput) and the amount of time it takes for a packet to travel between endpoints (latency). Delay is inherent to networked communication; packets cannot be transmitted and received over a network link instantaneously.

2.1.2 Types of Latency

Sources of delay include transmission, propagation, and queuing delay. Transmission delay, or the time required to "place" a data frame onto a network link, depends on the amount of data and the bandwidth of the link. Propagation delay, or the time required for a packet to travel across a network link, depends on the medium of propagation and the length of the link itself. Queuing delay, or the time a packet is stored at a switch before transmission, depends on the congestion control mechanisms enacted by the router and the transport-level protocol. [2, Chapter 1.5]. While transmission and queuing delay can scale up with distance due to intermediate switches (also called store-and-forward delay), propagation delay increases significantly and reliably with distance.

Due to unpredictable congestion and delays, the transmission time between end users for a packet can vary. Variation in packet delay is called *jitter*; it typically increases with physical distance between network endpoints, as longer routes can expose data to more variables that can affect packet timing.

2.1.3 TCP Packet Loss Recovery

Packets may be dropped, duplicated or delayed substantially due to network errors. Some protocols implement mechanisms that recover automatically from these issues. One such protocol, TCP, implements an acknowledgement mechanism that detects and triggers the retransmission of a dropped packet. TCP endpoints (senders and receivers of segmented data) monitor a sequence number, which increases with the payload size of each received packet. When a packet is successfully received, the receiver sends an acknowledgement (ACK message) with the next expected sequence number back to the sender, communicating that the data up to that sequence number has been successfully received. If the sender receives an unexpected sequence number or an ACK message is not received within the retransmission timeout (RTO), TCP protocol on the sender side assumes the packet has been dropped and triggers retransmission.

Thus, given `TT` as the best-case transmission time between sender and receiver, the worst-case transmission time given one dropped packet is `TT + RTO`. [3]. The minimum value of `RTO` set by the TCP standard is one second [4]. A dropped packet can also be detected by the TCP receiver. If a packet is dropped, the next packet sent to the receiver `P` after the first will have an incorrect sequence number. This triggers the receiver to send a NACK message back to the sender, triggering the transmission of the data associated with the lost packet. The worst-case transmission time given one dropped packet detected by the receiver is then `(3 * TT) + P`. `P` depends on user activity and implementation details.

Another protocol, UDP, is best-effort and does not offer a reliable in-order stream of data. In a nutshell, TCP is appropriate when reliability of transmission is prioritized over worst-case latency, and vice versa for UDP.

TCP headers are 20 bytes minimum [5].

2.2 Early and Influential Networked Music Systems

Throughout literature on networked music, two groups are consistently cited as influential in their use of networked computers in music creation and performance: the League of Automatic Music Composers (LAMC) and the Hub.

The LAMC was active between 1978 and 1982. LAMC members wrote programs they called *sub-compositions* to process and generate musical control data like melodic structures or pitch information. These programs were written on single-board KIM computers [6] which, while affordable and accessible, did not enable standard networking. The group resourcefully exploited the computers' parallel ports and interrupt lines to carry out rudimentary networking, exchanging control data between machines. Control data output by the network was converted to sound through direct digital synthesis [7] or used to control synthesizers. The goal of the music produced by the LAMC was to manifest network activity, rather than individual creative expression [8].

The Hub was active between 1985 and 1995. The group utilized a client-server system in which musical control data could be written to or read from common memory. Before performances, the group agreed on a protocol for sharing control data; performers then programmed their own instruments to communicate accordingly. The Hub is credited with introducing "distributed" musical performance, in which musicians are separated by physical distance significant enough that conventional methods for collaboration and coordination are no longer possible, or require substantial technical support. A 1987 performance saw two groups performing concurrently at separate locations in the same city, with their respective hubs linked via a modem over a telephone line. The groups created distinct musical outputs based on the same shared data.

Both the LAMC and the Hub both created unpredictable and occasionally chaotic music. While the former hoped to attribute the novelty of the music to the activity of the network, the latter hoped that it arose from the new forms of human interaction the system enabled. [8] [9, Chapter 2.2.1].

In the initial "performances" carried out by the LAMC, the system was set up and allowed to generate music autonomously. Later performances saw them adjusting parameters of the system in real time, as they found it "more fun to perform along with the network" [8]. John Bischoff, a former member of the LAMC, described the activities of the group as "network music" [10]. Chris Brown, a former member of the Hub, used the phrase "Computer network music" to refer to musical processes that "[aim] to reveal the voice of the system itself, the sound of the network instrument", positioning the network itself as the primary instrument and source of musical agency, in contrast with traditional emphasis on the virtuosity of the individual in conventional performance [8].

While the Hub and LAMC are frequently cited as progenitors of the subfield of networked music now called "networked music performance" [11, 12, 13, 9], neither phrase used to describe their activities mentions performance. In 2001, John Lazzaro and John Wawrzynek used the phrase "networked musical performance" (NMP) to describe "a group of musicians, located at different physical locations, [interacting] over a network to perform as they would if located in the same room" [14]. This phrase now refers to a much wider class of systems, as new ideas about what constitutes performance have developed.

2.3 Theories of Liveness and Performance

Musical performance has long been associated with "liveness". Liveness is often defined by its contrast to what is *mediatized*, or simulated, shaped or altered by technology to some extent for aesthetic reasons. The value of liveness in musical performance, as defined by an absence of mediatization, is attributed to unique feelings that music that has been broadcast or recorded cannot replicate [15, 16]. Mediatization can be contrasted with *mediation*. Musical instruments, by design, mediate music creation. Complex musical patterns are associated with gestures, which then become specific and measurable goals that can be directly pursued and maintained through clearly defined actions [17]. Traditional acoustic musical instruments are recognized for their capacity to produce complex music and the skill required for their use.

The liveness of an event is commonly associated with its spatial proximity to and temporal simultaneity with an audience. During live musical performance, music is being created "in-time", or such that it is perceived in full during its creation[18]. In musical scenarios that do not accommodate a separate audience, such as a "jam" session of simultaneous collaborative musical activity, performers are effectively each other's audience. Here, liveness can also be associated with temporal simultaneity of performers.

Early discussions of liveness in the age of recorded music acknowledged that some conventions of liveness did not account for emerging musical experiences facilitated by technology. For example, [14] describes a system that connects remotely located and visually isolated keyboard players by transmitting players' gestures and reproducing them at each other's instruments; musicians interact simultaneously but are physically separated.

In an contemporary exploration of liveness in modern music, Paul Sanden proposes an audience-centered definition of performance. He suggests that performance is characterized by particular forms of mediatization, rather than their absence. From the perspective of an audience, "corporeality, spontaneity, interaction, temporality, and spatial proximity all contribute to the experience of liveness associated with ... performance" [19, Chapter 1]. These characteristics are interconnected; they can be mediatized to varying extents or completely absent without compromising an audience's ability to experience an event as live. For example, audiences' appreciation of live musical performance is associated with their appreciation of the physical effort required for virtuosic instrumental performance [20]. If a performer convincingly mimics gestures associated with playing an instrument while accompanied by audio playback, an audience may perceive the event as a live musical performance. The liveness of an event is determined by its audience's belief in its liveness, regardless of the agency of the "performer" over the sound being produced.

In a survey of the field of performance studies, Marvin Carlson proposes a performer-centered theory of performance. He suggests that performance is associated with engagement with the rules of a performance space (sonic, physical, virtual), an appreciation of the skill required to follow said rules, and an awareness of some standard for performances [21]. Thus, both musical and interpersonal skills are required for performance, and the musical output of the ensemble can be attributed back to performer activity. While rules and standards vary between music systems, Carlson proposes that systems for in-time music creation in which these skills are utilized continuously and simultaneously can be said to enable musical performance.

Consider that as a system deviates from traditional models of performance, different or fewer skills may be required to perform music. For example, consider again the system proposed in [14]: Introducing artificial delay between when gestures are produced and transmitted could potentially go unnoticed by users, or attributed to slow user reactions rather than system design. Users could also be presented with novel digital instruments that introduce new musical conventions and simplify interactions. Key presses could trigger pre-composed musical segments, These alterations could reduce the possibility of chaotic musical output by reducing the skill required to engage with other users or the level of engagement needed to produce music [22]. Alternatively, they could require musicians to develop new skills and accommodate a different but no less demanding performance paradigm.

Carlson argues that performance stems from the agency of human participants, and their interaction with instruments, the environment, and each other. In a contemporary discussion of liveness in performance, Di Scipio proposes a more distributed perspective of agency in performance. He defines performance as the manner in which a *system* of human and mechanical resources and the *site* at which it is deployed combine to create a self-sustaining dynamic "performance ecosystem", both spatially, as encouraged by the site, and temporally. A performance ecosystem achieves its goals by sensing and acting on its environment, making it "self-observing" [23]. The network of interactions between system and site that cannot be attributed to musicians' agency alone are part of performance. This viewpoint could be seen to extend Carlson's. While both perspectives address the importance of dynamic social interaction in the performance process, Di Scipio proposes that music can emerge from other kinds of interaction. For example, user activity can be "circulated and modulated" [24] through a series of feedback mechanisms incorporated into the performance site. If a user is aware of how their musical gestures are "transformed" while propagating through the site, music produced can be called a result of the interaction between system (users) and site.

Henceforth, unless used when referencing Di Scipio's theory of performance, "system" refers to the instrument that facilitates interaction between users, and not the users themselves, e.g. performance systems enable musicians to perform. Carlson's theory of performance suggests that in a performance system that transforms user input through feedback networks, informed and skilled interaction with the feedback network itself is necessary for musical activity with the system to be considered performance. If this hypothetical performance system is collaborative, skilled and informed interaction with other users is also necessary. All three of these complementary perspectives - Sanden's primarily audience-oriented perspective, Carlson's primarily performer-oriented perspective, and Di Scipio's consideration of the interplay between system and site - are relevant to and provide context for contemporary discussions of traditional (co-located and instrumental) and networked musical performance.

2.4 Goals of Traditional (Co-Located, Instrumental) Musical Ensembles

Most existing music performance systems, traditional or novel, share the goal of producing coherent and complex music in-time. Music is *complex* if it displays some degree of "dimensional salience" [25], such that the features that are important to the music (e.g. rhythms in percussive music) are coherent. Music produced by an ensemble is *coherent* if it displays some degree of order along its dimensions of salience [26].

In traditional musical performance, liveness is experienced directly with minimal mediatization (Sanden). Performers engage with well-established rules and standards of musical performance, and can appreciate each other doing so (Carlson). Interplay between system and site is subtle but still present (Di Scipio).

2.4.1 How is Coordination Achieved?

In traditional ensembles, coherence is primarily the result of coordination between musicians. Coherence is achieved via *joint action*, in which two or more musicians coordinate their actions to bring about a change in their environment [26].

Two skills that enable musical joint action are anticipatory auditory imagery and prioritized integrative attention. Anticipating sounds or movements helps musicians achieve intermediate goals that support musical cohesion. "Imagery" here can also include the interaction of other sensory information, such as visuals, with audio [27]. For example, musicians can play drums "on beat" with a metronome ticking at regular intervals by anticipating its sound and noting its movement. In the absence of direct visual contact, musicians often use cues from their peripheral vision [28]. Other visual methods for musical synchronization include eye contact [29] and breathing [30]. This has also been described as "synchronization by anticipation" [31].

Prioritized integrative attention refers to the division of a musician's attention between their own actions and those of others. The former is prioritized over the latter to the extent that the overall sound of the ensemble is maintained. This skill enables musicians to adapt their actions based on others' behavior [32].

[33] defines "situational visibility" as the design space that encompasses the physical arrangement of the performance space and the placement of visualizations within that space, with respect to how these affect visibility among participants. This idea applies to both traditional and novel performance scenarios. Consider how this reflects Di Scipio's idea of the self-observing performance ecosystem, as discussed in 2.3: The performance site is an environment, virtual or physical, from which musicians rely on feedback to make decisions and adjust their behavior.

Many genres of music demand a level of musical synchronization between performers further than just simultaneity / synchronicity in periods of musical interaction, such as rhythmic or dynamic synchronization. In traditional co-located instrumental ensembles, this is accomplished via joint action, which is supported by continuous and near-immediate multi-sensory feedback. Musicians in close proximity can observe each other's movements and connect each other's gestures to sounds they produce [22], in part because delay between one musician's action and other musicians' perception of said action is small enough to not be considered disruptive to musical activities. Sound travels at roughly 343 m/s, and the delay experienced by members of co-located ensembles can range from 5-10 milliseconds in a small room to 80-100 milliseconds on opposite sides of a concert hall [34].

Additionally, members of traditional instrumental ensembles can be assumed to have a basic understanding of the sounds each musician's instrument can produce. This helps them to identify who is making what sound, which contributes to ensemble cohesion [35]. Coordination can also take place via visual cues agreed upon beforehand, such as a conductor [36] or, as in semi-improvisational "game pieces", visual cues that encourage imitating [37] or interpreting [38] other musicians' actions in different ways.

2.4.2 How is Complexity Achieved?

The relationship between control complexity and complexity of music produced, as well as the tradeoff between the former and the learning curve required to operate it, are well documented in instrument design literature [39, 40, 41]. The ideal musical instrument controls present a low barrier for entry, but skill with the instrument can be developed such that there is no upper limit to virtuosity or musical output complexity [42, 43]. In [43], Jordà introduces "efficiency" as a meaningful goal for musical instruments, defining it as the ratio of musical output complexity and "performer freedom" to control input complexity. Performer freedom can be understood as the amount of meaningful control performers can exert over the output of the instrument, such that they are playing music rather than playing "with" music [44]. While an instrument should facilitate the performance of coherent and complex music, it should also support the production of incoherent music, such that coherence is mostly a result of users' virtuosity, as opposed to a built-in "safety net" or a lack of performer freedom.

The use of familiar physical instruments sets a baseline for musical complexity in traditional ensembles; it can be assumed that creating music, individually or as a group, requires skill. This contrasts with "soundtoys", a well-established class of electronic musical "things" designed for play and exploration, rather than composition. They typically have simple controls and can only be used to create simple music [45].

2.5 Networked Music

2.5.1 Effects of Latency on Musical Interaction over Networks

While auditory latency experienced by members of traditional ensembles is usually less than 100 milliseconds, latency in networked music tools can be much higher. As noted in 2.1, certain sources of latency scale up significantly with distance between users. Estimates of latency between end users within the continental United States given in previous NMP literature are on the order of tens of milliseconds [46, 47]. [13] gives 5 milliseconds of propagation delay per 1000 kilometers between end users and 10 milliseconds of queuing delay as average values. As discussed in 2.1.3, one packet loss can raise this transmission time to the order of seconds. While multiple retransmissions of the same packet are uncommon, atypical network conditions or client-side issues may lead to packet loss or unacceptable delays. In networked music literature, latency below 25 milliseconds is generally considered unobtrusive, and musicians connected over a two-way audio stream can play with up to 75 milliseconds of latency without needing to actively manage their auditory feedback [13]. Above this range, issues with rhythmic synchronization, such as gradual deceleration, start to occur [48], and interaction may feel "lacking in musicality" [34].

At least three kinds of rhythmic timing are affected by latency: *pacing*, the shared tempo of the performance, *regularity*, timing within individual parts, and *coordination*, or timing between parts (henceforth called *rhythmic coordination* to differentiate from earlier discussions in the context of general musical coordination)

Jitter (discussed in 2.1) has the potential to be more disruptive than latency on rhythmic synchronization, as users may need to re-synchronize their playing every time the latency changes [13].

2.5.2 Activities Called "Networked Musical Performance"

The term "networked musical performance" has endured as the name of an established subfield of networked musical tools. It now accommodates a wider range of tools and perspectives on liveness and performance than the early definition given in [14]. It is commonly abbreviated to NMP, which can be used interchangeably to refer to a system and the event it facilitates. Additionally, frameworks that have attempted to categorize the entire field of networked music have identified subsets of the entire field classifiable as performance tools.

An early framework for networked music tools by Àlvaro Barbosa, widely cited in networked music literature [49, 31, 13, 9, 50], proposed a distinction between *synchronous* and *asynchronous* collaborative music systems [51]. Systems are asynchronous if the period of musical interaction between users is not simultaneous. It is given that in these systems, music is created asynchronously; participants do not contribute to and hear the complete and combined musical output at the same time. Barbosa describes systems as synchronous if the period of musical interaction between all users is simultaneous. This is the fundamental requirement for any further degree of synchronicity in collaborative music systems, and the only degree of synchronicity that can be assumed when a system is described as synchronous. This bare minimum level of synchronicity is considered a precondition for 'in-time" musical interaction [31]. This is not true of asynchronous music creation, so "synchronous" is sometimes used to imply in-time music creation, and in turn, performance.

We see the tendency to associate synchronicity with performance and in-time creation, and an emphasis on performance in the context of in-time creation despite the vagueness of the term, reflected in contemporary NMP. A 2023 survey by Miriam Iorwerth posits that NMP has "become accepted to mean any type of musical interaction that requires a computer network". However, "synchronous NMP" (defined by Iorwerth as "all participants playing at the same time as one another and interacting musically in real time") is a more popular area of study [13, Chapter 1]. In a 2016 survey of wireless networked music performance systems, Leonardo Gabrielli and Stefano Squartini define NMP as "the practice of conducting real-time musical interaction over a computer network" [9, Chapter 2.1], but they only cover "synchronous approaches". This observation supports Sanden's audience-focused theory of performance. Given that truly "real-time", instantaneous interaction over a network is not possible, similarities in these characterizations suggest that anything that mediatizes temporal simultaneity effectively can be called an NMP system.

As mentioned in 2.4.1, musical performance often requires further levels of synchronization between performers, which are made possible by continuous adaptation to multi-sensory feedback from the rest of the ensemble. Consider how this can be disrupted in systems with remotely located participants. As discussed in 2.1, latency scales up with physical distance between network endpoints. In systems that accommodate arbitrarily displaced users, it cannot be assumed that the latency present is low enough to avoid disrupting certain kinds of musical coordination. The rate at which musical joint action can occur is slowed as latency increases. As described in 2.5.1, longer and more unpredictable delays have been known to make this kind of musical coordination more difficult.

Given the challenge of distributing less accessible hardware like virtual reality headsets or bespoke musical instruments, it is practical to assume that distributed users communicate using personal computers, and rely on digital interfaces for communication. While users may be able to connect and coordinate via separate applications during a period of musical activity, it is inconvenient to switch between applications during in-time music creation.

As discussed in 2.3, liveness and performance are often associated with the impression of temporal simultaneity and spatial proximity. The complete lack of spatial proximity and the issues with ensuring temporal simultaneity make it challenging to create something that "feels" live. Given these issues, fully remote and simultaneous in-time music creation has become a popular set of technical restrictions to apply in networked music, given the goal of facilitating performance. In [51], Barbosa describes this subset of networked music tools as "remote music performance" systems (RMPs). There is precedent in contemporary surveys of NMP to say that RMP is its only focus [52]. There is also precedent for works that place themselves in the NMP tradition to give implicit [53] or explicit [54, 30] definitions of NMP that encompass only RMP.

2.5.3 Activities Not Called "Networked Musical Performance"

Golo Föllmer's 2002 survey of "net music" categorizes works based on three key characteristics of network-influenced remote music production systems: *interplay with network characteristics*, or the extent of the structure and activity of the network itself on musical output, *interactivity / openness*, or the extent of exploration and interactivity possible with the system, and *complexity / flexibility*, or the extent of musical complexity and variability possible with the system [55]. Systems for fully remote and simultaneous in-time music creation can embody these traits to varying degrees.

Broader studies of networked music that define a "performance" category of tools place systems in this group based on how well they conform to Carlson's standards for performances, as outlined in 2.3 It is reasonable to assume that a networked performance tool might pursue multiple goals, and that more technical restrictions may require more mediation or introduce playing paradigms in which more mediatization is acceptable. There is precedent in networked music tools for limiting musical exploration in the service of encouraging more complex group dynamics and social interaction [22]. However, there is precedent for not classifying systems as performance tools based on the extent to which musical interaction between users and the system is restricted or simulated in service of goals less related to complexity/flexibility. For example, in [51], Barbosa differentiates between RMPs and "shared sonic environments", which accommodate users without prior musical experience, manifest network characteristics in the sound produced, and facilitate new kinds of social interactions. In [55], Föllmer similarly introduces a "Performance" category, composed of tools designed to emulate live performance digitally. Other categories identified by Föllmer include "Soundtoys", which, as described in 2.4.2, limit user control and interaction to the extent that no musical skill is required for operation, and "Algorithm and Installation", where the music produced is more a reflection of characteristics of the network than user activity, or interaction is facilitated through music, rather than in the service of creating music.

Given their pertinence across literature, we generally apply Carlson's criteria for liveness when evaluating how and if novel music systems enable performance. In networked musical performance, technological mediation enhances rather than diminishes the sense of liveness (Sanden), to the extent that skill is still required to take part in performance (Carlson). In some networked performance systems, attributes of the network used to facilitate the performance itself are manifested in the music, subtly or directly. While the coherence and some of the complexity of the music can still be attributed to user activity (Carlson), the music produced is influenced by user interaction with and via the system (Di Scipio). Systems that accommodate issues with temporal delay and a lack of visual feedback by modifying the interaction paradigm to something not resembling traditional music performance must ensure that users themselves remain the locus of musical agency [23].

2.6 Networked Music Performance

2.6.1 Realistic and Non-Realistic Networked Musical Performance

Two prevailing approaches exist to networked musical performance.

The first approach, sometimes called the "realistic interaction approach" [9] or "realistic NMP" [56], focuses on simulating the conditions of co-located instrumental performance to the closest extent possible. Systems like these could appeal to groups of musicians whose desire to create music as they would in a traditional setting is impeded by the distance between them [47]. Recall Föllmer's characteristics of net music systems outlined in 2.5.2: Realistic interaction systems, by design, mitigate or disguise network characteristics that disrupt conventional performance. Interactivity / openness is a goal inasmuch as these systems emulate the kind of interactivity and openness present in traditional co-located non-networked settings. Notable recent works in this vein, such as JackTrip Virtual Studio (8.3), Lola [57] and Ultragrid [58] connect musicians through video and audio links that offer round-trip latency on the order of tens of milliseconds, enabling near real-time interaction. Video links provide visual feedback and anticipatory auditory imagery, and the use of conventional musical instruments implies a passing understanding of each other's abilities and the skill required to fulfill genre-specific musical responsibilities. These systems include specifications for optimal connection quality [59, 58] or depend partially on the quality of end users' audio and video streaming capabilities [57].

The second general approach does not simulate traditional musical performance. Surveys of NMP characterize work in this vein as "non-realistic NMP" [52]. Early surveys of works for remotely located users described these as *distributed* music systems [50]. More detailed discussion of specific systems in this vein can be found in Section 8. Systems like these might appeal to groups of musicians without the resources necessary to carry out successful realistic NMP performances, such as acoustic instruments or high-bandwidth connections. They might not rely on custom servers, and thus would use one of the transport-layer protocols considered acceptable for use over the Internet. Considering Föllmer's characteristics of net music systems outlined in 2.5.3, this direction might also appeal to groups of musicians who are interested in novel forms of musical social interaction, experimentation with sounds that reflect the network and its qualities, or both. The Hub and LAMC, as described in 2.2, were driven by similar motivations. While such a tool may serve multiple goals, musical complexity must be prioritized so as not to compromise a system's classification as a performing instrument.

Two important considerations in non-realistic NMP systems are (1) how latency is managed in musical communication, and (2), how coordination is encouraged without spatial proximity and continuous, intuitive multi-sensory feedback from other musicians. As we see in the following sections 2.6.2 and 2.6.3, both of these issues may require simplifying certain aspects of the musical or interaction paradigm. In departing from a conventional musical paradigm to accommodate issues with latency and user interaction, performers themselves must maintain their musical agency, such that the complexity and coherence of musical output can be attributed to user activity.

2.6.2 Non-Realistic NMP: Changing the Music-Making Paradigm to Manage the Effect of Latency on Auditory Coordination

When bandwidth is limited and continuously streaming audio between users is impractical, sending control data like MIDI data (8.4), representations of gestures with physical instruments ([14], 8.4, 8.6), or other application-specific information (8.1, 8.2) are viable alternatives. In systems with non-realistic playing paradigms, this can also mean reducing the number of dimensions of salience (described in 2.4.1) of the music produced with the system. This does not entail reducing the complexity of the music; simplifying some musical aspects can allow greater complexity in others. For example, as discussed in 2.5.1, pacing is one of several musical timings that can be affected by latency. Enforcing a set tempo supports users' pacing while allowing them to focus more on regularity and rhythmic coordination.

Systems can also restrict their focus to musical styles less dependent on precise timing, such as texture-based experimental music (8.1, 8.4), such that users can ignore delays and play without adjusting their auditory feedback. This would be acceptable so long as the coherence and complexity reflected in the complete music can be attributed to user activity, such that musical intention is mediated, rather than mediatized. A mediatization-based approach might involve restricting control over a significant portion of the complete music or entire dimensions of salience by automating or pre-composing specific musical elements.

Latency compensation techniques for rhythmic synchronization often consider a hierarchical "conductor-player"-like dynamic, in which one party leads with musical cues and the other follows. We introduce the convention of *upstream* and *downstream* roles to describe this playing relationship; downstream roles receive and follow musical cues from upstream roles to fulfill their *rudimentary* musical responsibilities in a "chain" or "cascade" of responsibility. In this context, rudimentary refers to the fundamental musical responsibilities assigned as part of a hierarchical system. We distinguish these from other actions because they represent the core responsibilities in the upstream-downstream dynamic. Non-rudimentary actions include embellishments, improvisations, or responses that add musical complexity but are not critical for maintaining cohesion. Upstream and downstream to describe the flow of rudimentary musical responsibility, not the flow of information. While these approaches can limit spontaneous jamming and other non-hierarchical interactions, in practice, various methods are combined to allow for a broader range of musical interactions.

For example, in the master-slave approach (MSA) put forward by [9], "slave" users synchronize with delayed musical output from a "master" user. Synchronization and coherence are achieved only at the slave's side, as masters also perceive the delayed output of all slaves. An arbitrary amount of latency can be accommodated, assuming that masters can adjust their auditory feedback and avoid the influence of the reconstructed output of slave players.

An approach originally proposed in [60] and described in [9] as the fake time approach (FTA) maintains rhythmic synchronization for all users. Control data for future events is sent in advance, such that it is received before it is scheduled to be executed. A system establishes a set tempo, then adds artificial latency such that the total time between when an action takes place and when it is perceived by other users is a multiple of one musical cycle (e.g. beats, measures) based on said tempo. Thus, rhythmic synchronization is maintained in the combined outputs of all participants. This approach requires clients to maintain synchronized local clocks, such that a consistent timeline is established and accidental scheduling of events long before or after their intended execution time is avoided. Systems that apply this method are described in 8.2, 8.4, and 8.6. The fake time approach is not explicitly hierarchical, but informal influence patterns have the potential to emerge between users with shifted timelines such that upstream/downstream relationships develop.

[61] proposes a similar setup, the “mutual anticipated session” (MAS). Performers’ musical contributions, called “precedent musical performance”, are delayed by the “precedent time”, which exceeds the estimated worst-case latency and is quantized to subdivisions of a measure, or “beats”. Precedent times can vary between pairs of performers. This delay causes performers to hear each other’s overlapping contributions while staying on beat in a “canon-like” style. This approach synchronizes rhythms while creating different time-shifted and rhythmically coherent versions of the music for different listeners and performers. Varying delays between performers can reveal the potential for new musical interactions and opportunities for users to coordinate and develop said skills.

Delaying all musical cues by one measure reduces the rate at which joint action occurs, and may feel like playing along to a recording [31]. While constraints (non-instantaneous message delivery) and precautions (adding delays to allow transmission of data) make immediate action in response to events impossible, faster information exchange is preferable. The ideal precedent time is long enough to consider constraints and precautions but short enough that interactions feel smooth and continuous, in the absence of visual feedback or mediatizations of spontaneity.

Note that the approaches explored in this section assume that all users perceive the musical output of all other users; individual user output is not “muted” if it is incoherent. If concrete upstream-downstream relationships were established, coherence could be maintained by muting the outputs of all users not directly upstream, as their output does not necessarily need to be perceived for upstream users to fulfill their rudimentary musical responsibilities. For example, the master-slave approach could be improved by muting the outputs of slave players to master players. Making this change to a simple setup like the MSA, in which only one “link” exists in the “chain” of musical responsibility (between master role and slave role), masters would be unaware of the music being produced by slaves. This idea could be explored more compellingly in a system with more complicated upstream-downstream relationships, or one that provides other ways to assess downstream user activity, like a visual representation of musical activity.

2.6.3 Providing Visual Feedback to Compensate for Visual Isolation and Unfamiliarity with Novel Instruments

Latency notwithstanding, in systems with visually isolated participants using digital instruments, musical joint action is inhibited by a lack of visual feedback from other users [30] and unfamiliarity with each other’s instruments [33]. In networked music literature, these issues are associated with a lack of “awareness”, or the ability to perceive and understand the actions of other actors within a collaborative environment. This is a similar concept to situational visibility (described in Section 2.4.1); where situational visibility focuses on visual feedback, awareness refers to a broader ability to understand and process information about others’ actions. In systems with displaced participants, the relationship between situational visibility and awareness is important because users are visually isolated. Physical visual cues can no longer be relied on.

In task-oriented settings, visual artifacts that help users understand a problem and its constraints while carrying it out can promote awareness [62], so some systems provide users with task-specific visualizations that change according to performance conditions. For example, there is precedent in co-located electronic music ensembles to visualize musical activity such that users can differentiate their contributions [63, 64]. This allows users to assess each other’s musical activity, such that they can self-correct if their playing breaks convention [35].

Collaborative context notwithstanding, individual musicians can benefit from visualizations of their own musical activity. Synchronized multimodal feedback has been shown to positively affect musical experiences [65], and responsiveness is considered a favorable trait of digital instruments [44].

A common task-specific visual artifact is a shared representation of a virtual performance “space”, populated by avatars representing users. Shared mental models of a performance space can help musicians anticipate and coordinate their actions [66]. “Abstract and iconic” [65] representations of users are preferred, as realistic representations can imply a more natural and realistic level of interaction than possible given technical constraints. New social dynamics can be explored in the service of music creation if, in addition to representing sounds, the positions of artifacts also influence the sound produced by the ensemble. This has precedent in systems for musical exploration with co-located participants, in which users’ positions and groupings in the physical installation space affect the music produced [67, 68]. Some installations provide users with physical avatars on a “tabletop” space [69], or with simple digital avatars in GUIs, which users may be responsible for monitoring to fulfill their musical responsibilities [68].

The ideas in this section have also been explored in virtual reality (VR) research. Virtual reality, as defined by Jonathan Steuer in 1992, is any "simulated environment in which one experiences telepresence" [70], where "telepresence" is the sensation of one's presence in a remote location, rather than one's actual physical space [71]. While VR today is commonly associated with the use of "interactive computer simulations that sense the participant's position and actions and replace or augment the feedback to one or more senses" to achieve the same effect [72], Steuer's definition also includes the installations for co-located participants discussed in this section. These are worth consideration because they suggest techniques for creating shared and immersive experiences without replacing or augmenting every part of the experience.

2.6.4 Non-Realistic NMP: Visual Feedback with Respect to Latency-Accommodating Playing Paradigm

The techniques for encouraging awareness described in 2.6.3 are mostly implemented in co-located ensembles, in which network latency is not significant enough to disrupt the prompt visualization of musical activity.

A side effect of the fake time approach introduced in 2.6.2 is that as latency increases, the rate at which joint action can occur is lowered, and musical interactions can shift from feeling continuous to "turn-based". This can make users feel they are playing over a delayed recording, rather than "live" with others [31]. A common solution to this problem is to allow continuous interaction with a locally rendered interface that provides immediate visual feedback synchronized with musical gestures, such that interaction with the rest of the ensemble is mediatized by interaction with the interface. Said interface must be rendered locally, as latency prevents it from being updated continuously. Thus, user-GUI and perceived user-ensemble interaction can remain continuous as the actual rate of musical joint action becomes more and more discrete (8.2, 8.5), evoking telepresence in the virtual space.

Systems utilizing the fake time approach that present shared representations of user activity could improve user experience by providing them with immediate visual feedback to their own gestures, while other users "see" / other clients display the same visual cues delayed.

Research on tools for co-located participants that utilize miniature physical avatars has noted that situational visibility is impeded by crowding within the virtual space [73, 74]. A networked implementation with a similar approach would require a limit on the number of avatars present in the space (implemented in 8.1) or some way of scaling up the virtual space such that clarity is maintained.

2.7 Motivation for New Work

In sections 2.6.2 and 2.6.3, techniques for adapting conventions of performance to a non-realistic remote networked performance system were discussed. As explored in 2.6.1, some non-realistic NMP work is motivated by the desire to explore novel interactions with other users and with the network itself. In this subsection, we address some of the unique possibilities afforded by networked performance, which can reinforce design decisions made in the interest of facilitating the in-time creation of coherent and complex music. It would be novel to explore all of these possibilities at once.

2.7.1 Allowing "Out Of Time" Editing of Performances

There is precedent for networked performance tools that allow for out of time reordering of material generated during play (8.5, 8.2). These tools only provide audio stems; more flexible and detailed control data, such as MIDI, would be more suitable for those interested in more advanced editing and reworking of material. In [51], Barbosa introduced the "composition support system" as a class of networked music tool, instances of which aim to support in some way the out of time production of music. A compelling and novel direction for a networked music performance tool would be to allow users to download records of musical sessions as MIDI files, such that they can be used as starting points for compositions created out of time.

2.7.2 Cascades of Rudimentary Musical Responsibility

As discussed in Section 2.6.2, some latency compensation techniques for rhythmic synchronization consider a hierarchical "master-slave" or "conductor-player" dynamic, in which one party leads with musical cues and the other follows. Defining rudimentary musical responsibilities helps musicians navigate new performance contexts.

This idea can be extended further by introducing "cascades" of upstream-downstream relationships between classes of users, such that they have different roles in gradually shaping the music produced by the ensemble. This idea has been explored before in some co-located game pieces (previously mentioned in Section 2.4.1), in which previously agreed-upon visual cues prompt co-located instrumentalists to change their playing, instructing them to "follow" other participants musically or "match" their playing in different ways. Musical agency is distributed among musicians to promote musical exploration. Individual rudimentary musical responsibilities may be simplified, as following a designated lead performer is simpler than coordinating with multiple ensemble members simultaneously. This is a result of the "game piece" approach to musical exploration; it is not done in the service of maintaining coherence or lowering the amount of skill required to participate. A compelling and novel direction for a networked music performance tool would be to assign roles to users in the same way.

2.7.3 Visualizing Musical Activity

The benefits of visualizing user activity and the concept of situational visibility discussed earlier in Section 2.6.3.

In the aforementioned game pieces, participants are co-located and perceive the combined output of all musicians. They can then be influenced by musicians to whom they are not explicitly "connected" through an upstream-downstream relationship, by taking note of the activity of the full ensemble, as well as how their own musical activity is "circulated and modulated" [24] by users further downstream. Their non-rudimentary musical activities (as described in Section 2.6.2) are enriched and informed by their perception of the ensemble, with which they are co-located. Musicians can take note of each other's playing sensibilities and gauge emergent standards for following upstream users.

It would be novel to experiment with this idea in a networked setting by presenting a shared visual representation of all users' activity. Users could carry out their own rudimentary musical responsibilities using information presented via this representation, and they could perceive other users doing the same. Thus, their non-rudimentary musical activity could be influenced by other users, as it would be in a co-located ensemble.

The implementation must address the challenges outlined earlier in Section 2.6.4. While the shared representation cannot be updated continuously, continuous interaction with this representation should mediatize continuous interaction with the rest of the ensemble. User experience could be improved by providing immediate audiovisual feedback to their own gestures, such that other clients display (other users "see", or are shown) the same visual cues delayed.

2.7.4 Bi-Located Patterns

Co-located instrumental performers experience their collective musical output almost identically and in real time. In remote music systems, variations between inter-user latencies can result in each musician hearing a slightly different version of the collective output, such that musical activity is not disrupted. Early work on distributed music systems noted this occurring unintentionally and described its effect as "semi-synchronous" musical performance. More recent work suggests that this is compelling and can be exploited musically; [75] proposes a system in which "bi-located" rhythmic patterns are perceived differently by different end users, based on inter-user latency. The MAS framework can be utilized to similar effect, given variation between precedent times.

2.7.5 Sonification / Artful Representation of Network Characteristics

As discussed in 2.5.2, musicians may be drawn to non-realistic NMP or network-based music tools by an interest in manifesting sonically or *sonifying* network characteristics. Interest has grown in embracing network latencies and disruptions as "crucial" characteristics of the network medium worth embracing [76, 77]. There is precedent for installations [78] and sound design tools / network soundtoys [79] that use repeated measurements of network latency to modify the parameters of sounds produced with Karplus-Strong synthesis [80]. These tools are more suited to sound design and exploration than music creation. [75] proposes using differences in inter-user latencies to modulate the parameters of spatial diffusion algorithms. Few networked music tools explore this further, notably not those that also allow for out of time editing of material generated during sessions. Given a MAS-like system, it would be novel to scale up precedent times based on estimates of inter-user latencies.

3 Goals

Broadly, the goal of the proposed system is to facilitate non-realistic musical performance over a network, and explore the five ideas explored in 2.7 that have yet to be simultaneously explored in one application.

Based on a literature review of what constitutes networked musical performance, both in the established field of NMP, a term whose scope now includes systems that facilitate any kind of networked musical interaction, and in surveys of the broader field of networked music, a performance system should have the following features: It should give users the impression that they are playing "live", mediatizing features commonly associated with liveness when necessary due to technical restrictions. It should support the collaborative, simultaneous and in-time creation of music. The controls of the instruments provided to users should be such that there is a low bar for entry, but skill can be developed such that complex music can be produced and virtuosity is possible. Based on further review of what constitutes non-realistic performance specifically, further goals are as follows: The number of dimensions of salience may be reduced due to technical constraints or aesthetic concerns. While the system may pursue secondary goals of facilitating novel interactions between users and with the network itself, they should not compromise the primary goal of enabling the creation of coherent music that displays complexity, in its dimensions of salience, that can be attributed to user activity.

We assume that users are physically displaced, and therefore, visually and sonically isolated. Thus we introduce the goal of accommodating an arbitrary transmission latency between clients, and the restriction of relying only on communication through the digital interfaces provided by the system.

The system's design is motivated by the preceding technical restrictions and guiding principles, as well as the goal of exploring the five ideas given in 2.7. The system is designed to work best when users are visually and sonically isolated and operate one client each; we call this the "ideal" setup for the system.

3.1 Playing and Interaction Paradigm for Proposed Application

3.1.1 Timing and Scheduling

An advance scheduling system is implemented and a set tempo is enforced throughout the ensemble as part of a MAS-like fake time approach to incorporating latency into the playing paradigm as a musical strategy.

The scheduling system allows clients to *schedule* future events locally and on other clients by sending *messages* that are *timestamped* with an execution time, ensuring precise timing. The scheduling system also requires clients to maintain and synchronize their local clocks, establishing a consistent timeline such that events are not accidentally scheduled to take place long before or after their intended execution time.

The system also enforces a tempo across all clients. The tempo itself can be changed but is always present. Tempo is calculated as a mapping from time to *beats*, which are incremented at a regular rate according to the tempo.

The specifics of scheduling and tempo are discussed further in the Requirements (4). We introduce the details most relevant to the playing and interaction paradigm here first to make it easier to discuss related concepts.

3.1.2 Drumming

The dimensions of salience of the music produced using the system are solely rhythmic. Users are presented with a drum-like digital instrument, which plays back percussive sounds in response to key presses. The action of playing a hand drum translates especially well to a computer keyboard-based digital instrument, both produce sound immediately and offer a natural rebound when "struck". The widespread appeal of drums, especially in a collaborative "drum circle" setting, can be attributed to their accessibility for beginners and potential for advanced musicianship, making them an ideal starting point for a digital instrument.

Henceforth, *drum hit* refers to the effect of a user tapping their keyboard to "play" a virtual drum, which can vary in its intended effect and interpretation by other users. Drum hit information is transmitted between clients as control data, rather than a real-time audio stream, to conserve bandwidth.

3.1.3 Client Classes and Rudimentary Musical Responsibilities

Generally, we address the second goal of exploring cascades of responsibility by introducing multiple classes of clients with different rudimentary musical responsibilities. Each class has a role in shaping the music produced by the system.

The application implements three client classes, *Composer*, *Performer* and *Listener*. Each class has a different role in creating music and a different relationship to the three kinds of rhythmic timing (pacing, regularity and rhythmic coordination) described in Section 2.5.1.

The rudimentary responsibilities of each role are as follows: Composers "play drums" by tapping their computer keyboards. The composer role requires an awareness of pacing and regularity in that playing along with the metronome encourages coherence downstream. Composers can "connect" themselves to performers, who follow along with their drums.

Performers only follow along with drums from the performers to which they are connected. Because of this relationship, we refer to the drums played by composers as *instructional* drums, and to performers' drums as *response* drums. Multiple composers can be connected to the same performer. The performer's only rudimentary musical responsibility is to play along with its connected composers' instructional drums, also "on beat". The performer role requires an awareness of rhythmic coordination, to the extent that performers follow along with what is played by composers. Performers also have agency in how closely they choose to follow composers' drums.

Listeners do not play drums; they "perceive" the combined and overlapping response drums from all performers. They do not perceive composer drums, only performer output. Because they can hear the overlapping output of all performers, they are aware of other users' relationships to and ability to carry out rhythmic timing. Listeners are the end of the "life cycle" of a beat. Despite the presence of a "listener" class, each musical perspective is legitimate and coherent, irrespective of the number of contributors.

Thus, music is created in-time (as it is perceived) and collaboratively, and the rhythmic complexity of the musical output of the system can be attributed to user activity. This also accomplishes the goal of exploring upstream-downstream cascades, as described in Section 2.7.2, as each class has a distinct but complex role in shaping the music produced by the system.

3.1.4 Interactive Local Representation of Other Clients and Musical Activity

All users are presented with (all clients display) an interactive shared representation of the entire ensemble, in which other clients (users) and their musical activity are represented as icons in two-dimensional space. Users can continuously interact with or be influenced by this representation, such that interaction with the rest of the ensemble feels continuous. Henceforth we refer to this component of the UI as the *grid*.

Clients are represented with simple avatars in the grid. They are differentiable by client class, and users can identify their client's avatar. Avatar positions are quantized to grid points, such that they are separated by small integer numbers of grid squares.

3.1.5 Drum Delay

Drum hits are represented on the grid by shapes that move between avatars. *Drum delay* refers to the time it takes for a drum hit's visual artifact to move from an upstream to a downstream client. This is also the time between when a downstream client displays an upstream client's drum hit, and the time at which said downstream client is expected to act on it or passively experiences its effect, i.e. how long the drum is delayed. Because client avatars are placed on grid coordinates, and because drum hits move one grid square per beat, drum delays are quantized to beats. Thus, if composers play "on beat", the combined outputs of all connected composers will be rhythmically coherent and beat-aligned at the performer. Likewise, if performers play "on beat", the combined outputs of all performers' response drums at a listener will be beat-aligned and rhythmically coherent, producing canon-like rhythmic music that preserves performers' rhythmic regularity around beats.

Based on the differences in drum delay between themselves and all other performers, perceives a unique and rhythmically coherent version of the combined output of all performers. This accomplishes the goal of exploring bi-located patterns, as described in Section 2.7.4.

The musical paradigm of upstream and downstream roles separated by quantized drum delays is reflected on the grid as follows:

Composers can draw lines, or *connections*, between themselves and other performers on the grid. Connections are made up of individual horizontal or vertical line segments that span grid squares. Connections can vary in length and direction and can be drawn at any time. Composers' drums / instructional drums are displayed as small nodes that traverse drawn connections, moving from composer to performer avatar at one grid square per beat. Henceforth we refer to the visual artifacts that represent composer drum hits as *traversing hits*. The drum delay between a composer and performer is the length of the connection drawn between them. Composers can be connected to multiple performers, but only one connection can exist between a composer-performer pair at a time. Connections also cannot intersect with other nodes (besides the ones at their endpoints) or other connections.

The movement of traversing hits along connections acts as a scrolling score for performers, who play drums in response as the traversing hits reach their avatars. Performer drum hits are displayed as rotated squares that propagate outward from their avatars at a rate of one grid unit per beat. Henceforth we refer to the visual artifacts that represent performer drum hits as *propagating hits*.

As propagating hits expand outward, they "collide" with listener avatars. When a listener "sees" a propagating hit (when the listener client displays a propagating hit) colliding with its avatar on its grid, it hears a drum being played (it plays back a drum sound). Thus, the drum delay between a performer and listener is the Manhattan distance between the two avatars on the grid. Listeners can move around on the grid by clicking and dragging their avatars, changing the drum delay between themselves and all performers.

All clients display musical contributions on the grid as traversing and propagating hits. When a composer or performer plays a drum, the resulting traversing or propagating hit is displayed in all clients, not just those downstream. Thus, musicians can make informed musical decisions based on perceived user activity, even if following along with certain users is not part of their rudimentary musical responsibilities. For example, while performers are not upstream or downstream from each other, they can adjust their playing to balance or counteract each other's activity levels. If users can see each other's musical contributions, they can better understand each other's playing styles and assess how well they are fulfilling their rudimentary musical responsibilities. This fulfills the goal put forward in Section 2.7.3

Allowing further editing of performances out of time, as described in Section 2.7.1, is accomplished as follows: A record of the performances, as perceived by listeners, are recorded during a session. At the end of a session, all users are given the option to download files containing performance data from the perspectives of listeners, such as when drum hits and tempo changes occurred. These files can then be edited or used as starting points for compositions developed out of time.

The final goal of the application is to sonify and artfully represent network characteristics. This is incorporated by making drum delays proportional to crude estimates of latency between clients. We crudely estimate the network latency between pairs of clients, then place their avatars within on the grid such that the distance between any two avatars is approximately proportional to the sum of their corresponding clients' end-to-end latency. Each point is then quantized to a grid point, such that the final drum delays are quantized to integer beat multiples. Thus, the configuration of avatars on the grid reflects the network "distance" between end users.

3.2 Hiding Transmission Delay

The drum delay bears similarity to the precedent time introduced by [61] in that beat-quantized delays preserve rhythmic coherence when multiple clients' output is combined. However, the precedent time includes the transmission time, while the drum delay does not. To accommodate transmission latency, we "delay" drum hit messages by some additional "hidden delay" $H \geq$ `MAX_NET_DELAY`, where `MAX_NET_DELAY` is an estimate of the worst-case end-to-end latency between clients. Recall that clients can schedule events to take place on other clients by timestamping messages with an intended execution time. If a client sends a message at time t, if its timestamp is at least $t + H$, its successful reception can be assumed to take place before its execution time.

As shown in Sections 3.2.1 and 3.2.2, this strategy can be used to effectively "shift" the performer's timeline forward, such that transmission latency from composer to performer is hidden from the performer and the listener. Introducing a hidden delay does not require changing what has been previously described, only specifying its implementation requirements.

3.2.1 Naive Strategy for Implementing Hidden Delay

Consider an example setup with one of each client type, in which the drum delay between composer and performer is D_1, and the drum delay between performer and listener is D_2. As mentioned previously, drum delays are quantized to beats, so D_1 and D_2 are both integer multiples of the duration of one beat at the current tempo of the ensemble B. When the composer plays a drum at time 0, the composer client immediately displays a traversing hit above its corresponding avatar, which begins to move one grid unit per beat immediately, such that it reaches the performer avatar at time D_1. At the same time (t = 0) the composer client transmits a message scheduling a traversing hit to be displayed at time H, such that the performer sees it reach its own avatar at time $H + D_1$. It also transmits a message that schedules a traversing hit to appear on the listener's grid at time $2H$, such that the listener sees it reach the performer avatar at time $2H + D_1$.

At time H, from the performer's perspective, a traversing hit appears at the composer's avatar and begins to move immediately, at a rate of one grid unit per beat. At time $H + D_1$, the performer sees the traversing hit reach its own avatar and plays a drum in response, timing their drum hit as closely as they can to the arrival of the traversing hit. When the performer plays a drum at time $H + D_1$, from the performer's perspective, a propagating hit appears and begins to move immediately, at a rate of one grid unit per beat. At the same time, the performer transmits a message that schedules a propagating hit to appear on the listener's grid at time $2H + D_1$, such that the listener sees it reach its own avatar at time $2H + D_1 + D_2$.

In accordance with the goal of displaying individual user activity to all other users, we also schedule a propagating hit to appear in the composer grid at $2H + D_1$. This is the closest time to D_1, when the composer sees its own traversing hit reach the performer's avatar, that the performer's response can be displayed to the composer. Showing the hit "late" is preferable to showing it even later or not at all, given the goal of displaying all users' musical activity.

At time $2H + D_1$, from the listener's perspective, multiple events occur simultaneously: The traversing hit from the composer reaches the performer's avatar and disappears. At the same time, a propagating hit appears at the performer's avatar and begins to move outward at a rate of one grid unit per beat. The listener sees the propagating hit intersect with its own avatar at time $2H + D_1 + D_2$. Shifting downstream client timelines forward by an arbitrary H conceals transmission delays; downstream users cannot tell when upstream users actually played their drum hits.

Consider now how the same series of events might appear to an additional two clients present in the system, another performer and another composer, who are not connected to each other or to any other clients. There are multiple ways in which composers / performers could perceive the activity of other composers / performers, as well as performers / composers to which they are not connected. Given that the grid exists in part to allow users to assess each other's timing sensibilities, and given that the relative simultaneity of traversing hits reaching composers and propagating hits leaving composers indicates the timing accuracy of the latter, we say that the non-connected composer and performer see the drum hits of the connected composer and performer on the same delay as the listener. Thus, the performer's response time is accurately displayed to all users.

3.2.2 Naive Strategy Example with Graphs

Consider an example configuration of five client avatars on the grid, labeled 1-5. Avatars are colored corresponding to their class - composers are red, performers are blue, listeners are green - and numbered with their ID. One of the composers (1) is connected to one of the performers (2). Their avatars and that of a listener (3) are arranged in a line, such that the "life cycle" of a beat is shown by the movement of a traversing hit left to right from composer to performer, then a propagating hit from performer to listener. Another composer (4) and performer (5) are horizontal only with each other and not connected to any avatars. Avatars 1-5 are placed at (x,y) grid coordinates (0,1), (8,1), (11,1), (4,0) and (8,0) respectively.

This setup covers all possible user perspectives in the system. If composer (4) were connected to performer (5) and playing drums, the same distortions would be applied to the "other" perspective.

The connection between this composer and performer is eight grid units long, and the performer and listener are three grid units apart. Thus, $D_1 = 8B$ and $D_2 = 3B$, where B is the duration of one beat in time units.

Henceforth we refer to the avatars labeled 1-5 as the connected composer, connected performer, listener, non-connected composer, and non-connected performer, respectively. In the naive solution, the hidden delay H can have any value greater than or equal to `MAX_NET_DELAY`, Because all other delays in the naive solution are small integer beat multiples, we set $H = 4B$ so that delays can be uniformly represented in the figures that follow. In the naive solution, H is not required to be an integer beat multiple.

This configuration of avatars, as it would appear on the grid during play, is shown in Figure 1. Figures 2, 3, 4, 5 and 6 show the distance of a drum hit from the connected composer (measured in grid squares) as its influence propagates through the ensemble over time (measured in beats), as outlined in 3.2.1, as perceived by all clients in the example setup. On the figures, red lines represent the paths of traversing hits from the connected composer, and blue lines represent the paths of propagating hits from the connected performer.

There are multiple issues with simply shifting downstream client timelines forward. We address these issues in Sections 3.2.3, 3.2.4, 3.2.5, and 3.2.6.

3.2.3 Visualizing Connected Performer Response Time to Connected Composers and Non-Connected Performers

While the flow of musical responsibility is one-directional, the flow of information throughout the system is not. We want performer activity to be visible to composers and inform their musical decision-making. More specifically, we wish to visualize how well performers' playing engages with the established rules and standards of performers, such that they can appreciate each other doing so. Performers are responsible for playing "on beat" while following along with composer drums, responsibilities previously referred to in Section 2.5.1 as regularity and rhythmic coordination.

In sections 3.2.1 and 3.2.2, we saw that the naive timeline-shifting strategy could be used to hide the delay between a connected composer and performer (labeled 1 and 2, respectively, in Figure 1) to all users, except for the connected composer (1) itself, as shown in 2. This is because the connected performer's drum hit message needs an additional H to be transmitted back to the connected composer.

Composers should be able to assess rhythmic coordination based on the relative simultaneity of a traversing hit reaching a performer and a propagating hit emanating from it. Thus, the proposed modification to the naive solution should accurately display performers' response times to the composers to which they are connected.

3.2.4 Accommodating "Non-Ideal" Scenarios to Broaden Demonstration Capabilities: Quantizing H to Beats

It is also important to consider that this is an in-development prototype. To be marketable and to encourage wide use, it should accommodate non-ideal scenarios (besides the "non-ideal" scenario of lots of latency, which the delay-based playing paradigm already addresses), so that it can be demonstrated in a variety of situations.

Requiring three remotely located users to operate the application is inconvenient for demonstration. Potential users may be hesitant to gather a group solely to test the system, so the system should allow the operation of multiple clients on the same device. Additionally, at music technology exhibitions or conferences, demonstrating the system on multiple co-located machines could better showcase the system's collaborative potential than a demonstration on one device.

In previous sections, we described an "ideal" scenario in which physically displaced users operate one client each. Users / clients were visually and sonically isolated outside of the interfaces provided by the system. The settings described in this section are "non-ideal" in that the overlapping output of multiple instances of the client application can be heard.

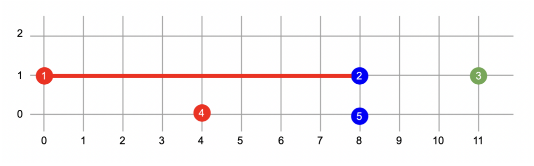


Figure 1: An example section of the grid, containing a listener and a connected and non-connected pair of composers and performers.

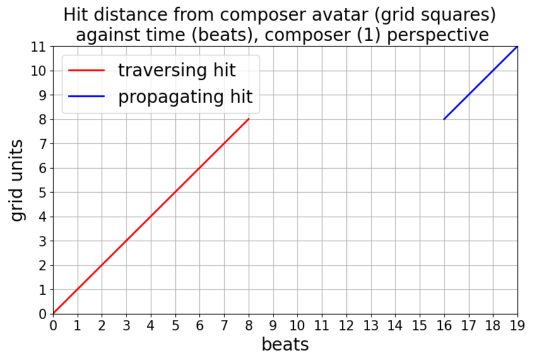


Figure 2: Hit distance from connected composer avatar (grid squares) against time (beats), connected composer (1) perspective

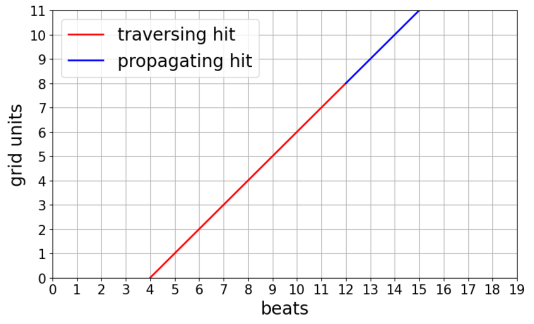


Figure 3: Hit distance from connected composer avatar (grid squares) against time (beats), connected performer (2) perspective

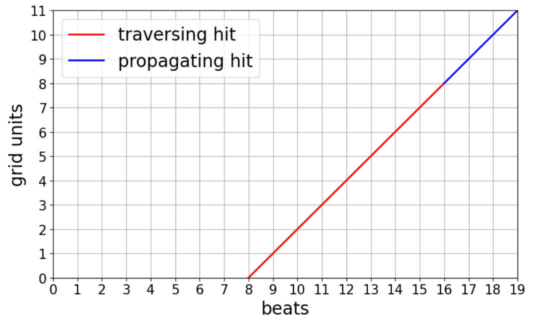


Figure 4: Hit distance from connected composer avatar (grid squares) against time (beats), listener (3) perspective

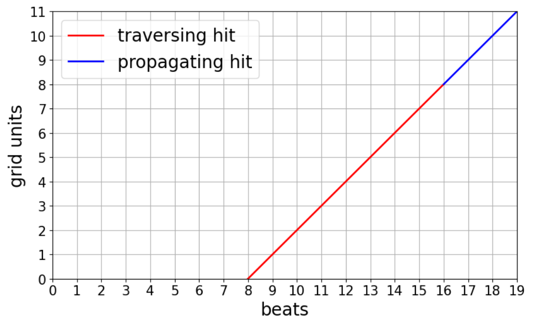


Figure 5: Hit distance from connected composer avatar (grid squares) against time (beats), non-connected composer (4) perspective

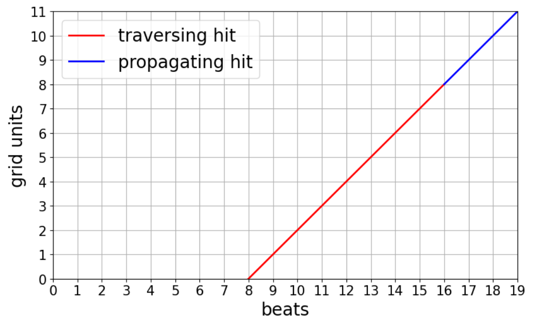


Figure 6: Hit distance from connected composer avatar (grid squares) against time (beats), non-connected performer (5) perspective

In service of accommodating these “non-ideal” scenarios, we introduce the following requirements for a modification of the naive solution: The system must preserve rhythmic coherence when the outputs of multiple clients producing sound simultaneously are overlapped. The system must synchronize the intersection of propagating hits with listener avatars across clients.

To preserve rhythmic coherence when the outputs of multiple clients producing sound simultaneously are overlapped, we quantize H to an integer multiple of the duration of one beat, such that $H mod \texttt{MAX\_NET\_DELAY} = 0$ and $H = xB$, where B is the duration of one beat and x is H in beat units. In the ideal scenario, users / clients are sonically isolated. If the output of multiple clients is perceptible at once and H is not beat-aligned, overlapping contributions will be rhythmically incoherent.

This introduces another issue with the hidden delay. Recall that the proposed application also accommodates tempo changes. The duration of one beat depends on the ensemble's tempo, so H

must also change if the tempo changes. H and x could be changed every time the tempo is changed by rounding `MAX_NET_DELAY` up to the nearest integer multiple of B. Alternatively, x could be held constant, and the maximum tempo could be restricted such that the duration of $xB = H$ is always greater than `MAX_NET_DELAY`. Using this approach, H changes every time the tempo is changed, but x does not. We take this approach because a consistent number of beats could be used as a reference point in visualizations.

3.2.5 Accommodating "Non-Ideal" Scenarios to Broaden Demonstration Capabilities: Synchronizing Propagating Hit Intersection and Listener Avatar Intersection Across Clients

We also want to ensure that clients display propagating hits intersecting with listener avatars at the same time. This is done with a specific demonstration setup in mind, which could be compelling when carried out at an expo or conference.

Consider a scenario in which individual clients are set up on separate machines, which are distributed throughout a space. Each client is being operated by one user. Machines running listener clients play audio back through their speakers, and machines running performer and composer clients play audio back through headphones. The grid and UI are visible on all machines. While a session is taking place, an audience present in the space can see the grid on all machines, and can hear the combined output of all performers at the listener clients.

If the intersection of propagating hits with listener avatars is synchronized across clients, audience members could observe any client's grid to see how performer output is combined at the listeners, while simultaneously experiencing the output of all listeners. In this way, the ideal scenario for the application could be demonstrated in non-ideal scenarios, in a more engaging way than if all users / clients were isolated. Because the hidden delay is quantized to beats, the complete output of the system remains rhythmically coherent, even if machines running composers or performers play audio back through speakers, instead of headphones.

3.2.6 Locally Rendered Grid Responsiveness to User Activity

Immediate responsiveness to user input is a favorable trait of instruments. Thus, when a composer or performer plays a drum, we want a visual artifact to appear immediately. To mediatize continuous interaction between users, we want this artifact to appear in the locally rendered grid immediately, before the reception of the corresponding message has been confirmed by the server, or before `MAX_NET_DELAY` has passed and the message's successful reception can be safely assumed. Two of the conventions that have already been introduced for displaying drum hits - that they move one grid unit per beat, and that the intersection of propagating hits and listeners is synchronized across clients - make this problematic, as shown by the following example:

Consider a configuration in which a performer and listener are separated by one grid square (drum delay of one beat between them). Assume that $H =$ `MAX_NET_DELAY` and $H = 4B$ time units, or $x = 4$ beats. When the performer plays a drum, a propagating hit appears and begins to move one grid square per beat, such that the performer sees it intersect with the listener's avatar one beat after it is played, before the listener has played the beat or even received the corresponding message. This issue is similar to the one outlined in Section 3.2.3 in that clients display the intersection of their own drums with downstream clients avatars before their response can be transmitted back, or before the message can be assumed to have been received. Both issues could be solved by loosening the requirement that drum hits always move one grid square per beat after appearing.

The next modification we make to the naive solution is that in the locally rendered grid, drum hits do not start to move one grid unit per beat until at least x beats after they are triggered. After a performer plays a drum, they see a drum hit appear immediately in their locally rendered grid. This is before x beats have passed and before the message can be assumed to have been received by the server. However, the drum hit will move a rate of $1/(x+1)$ grid units per second, such that all clients display the hit one grid unit away from the performer avatar after $x+1$ beats have passed. Thus the intersection of propagating hits and listeners can be synchronized across clients, even if listeners are just one grid unit away from performers, by distorting the distances between them.

We use a similar strategy to solve the issue outlined in 3.2.3, in which an extra hidden delay must be accommodated to display the performer's response to the composer. After a composer plays a drum, they see a drum hit appear immediately in their locally rendered grid, before x beats have passed and before the message can be assumed to have been received. However, the drum hit moves $1/(2x+1)$ grid units per second, such that all clients display the hit one grid unit away from the composer avatar after $2x+1$ beats have passed. In short, the distance between a composer and performer appears longer to the composer than to it does to other clients, including any performers to which they are connected. Performers see a shorter path, so they play earlier than composers "expect", such that their drum hit messages can be assumed to have been successfully delivered before they need to be displayed. This distance-distorting technique and the quantization of the hidden delay completes the non-naive approach to the hidden delay problem.

3.2.7 Complete Hidden Delay Strategy with Example

To further clarify the complete, non-naive approach to disguising delay, we re-examine the example configuration used to explain the naive strategy in Section 3.2.2, shown again here in Figure 7 for convenience. Again, we refer to the avatars labeled 1-5 as the connected composer, connected performer, listener, non-connected composer, and non-connected performer, respectively. We note again that this setup covers all possible user perspectives in the system.

Avatars are colored corresponding to their class and numbered with their ID. The connected composer, connected performer, and the listener are arranged in a line, such that the "life cycle" of a beat is shown by drum hits moving from left to right over time. We quantize H for the reasons outlined in Section 3.2.4, and set $x = 4$ beats arbitrarily for the example.

Recall that if composer (4) were connected to performer (5) and playing drums, the same distance distortions would be applied to the "other" perspective. We see in Figure 8 that when the connected composer plays a drum, the corresponding traversing hit is displayed moving slowly across one grid unit over the course of $2x+1$ beats. If composer (4) were connected to performer (5) and playing drums, it would be possible for composer (1) to display composer (4)'s drums on the same delay as the listener and other performers, as shown in figures 10 and 12. No distance distortion needs to take place for other composers' drum hits; composer and performer clients would only display their own drum hits slowly accelerating.

Consider that a user operating a composer client would see their own drum hits moving at $1/(2x+1)$ grid units per beat for the first $2x+1$ beats after playing a drum, while all other composers' drum hits are displayed moving at a constant speed. Users might believe they are experiencing lag if their visualizations move dramatically slower than other clients of the same class. To combat this, the composer client instead displays other composers' drum hits as shown in 11. This way, the difference between the initial speeds of traversing hits of composers, as displayed in the composer client, is more consistent. We do not implement a similar technique in the performer client, as the difference in speed between a performer's own propagating hits and other performers' propagating hits is less dramatic.

3.3 Completion Criteria

I first implemented the system consisting of the interfaces described in 3.1. There is precedent for evaluating musical prototypes by conducting informal test sessions with novice users, followed by surveys on the prototype's nominal functionality and ease of use [68, 81, 82]. To show that the prototype functions as a performance system, I tested it during two informal group sessions. First, I held an optional session outside of the Music and Technology seminar for interested participants, in which I introduced the system, gave a short demonstration, then invited attendees to participate. After this session, I polled participants on their experience and incorporated some of their feedback into a new version of the system. Between the first session and my thesis defense, I used the record of the performance as a starting point for a piece of music composed out of time. I also used the record of the performance as a starting point for a piece of music, and presented the finished piece to show how the system's results can be utilized.

To summarize, the complete thesis project consisted of the creation of a system that implemented the playing paradigm given in 3.1 (1), holding a test session with said system to generate material and incorporate feedback into the final system (2), the completion and presentation of a piece of music using the material from said session (3), and the complete thesis document (4).

The system can be evaluated on its completion according to the specifications in 3.1. The presentation and test session can be evaluated based on the nominal operability of the application and

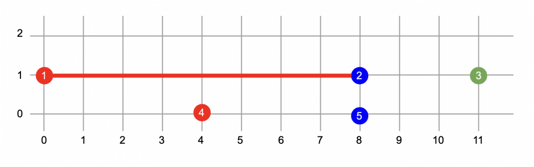


Figure 7: An example section of the grid, containing a listener and a connected and non-connected pair of composers and performers. Identical to Fig. 1 but reproduced here for convenience.

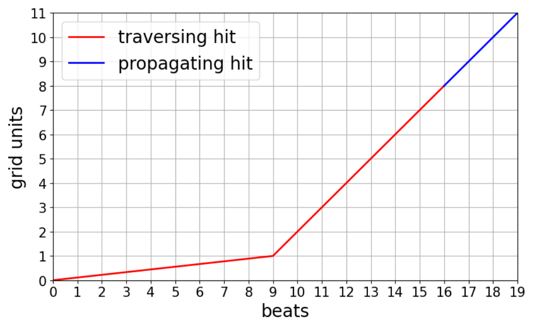


Figure 8: Hit distance from connected composer avatar (grid squares) against time (beats), connected composer (1) perspective

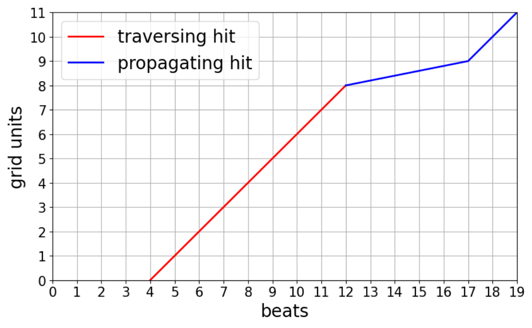


Figure 9: Hit distance from connected composer avatar (grid squares) against time (beats), connected performer (2) perspective

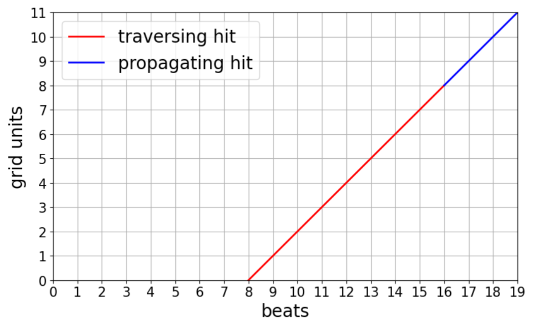


Figure 10: Hit distance from connected composer avatar (grid squares) against time (beats), listener (3) perspective

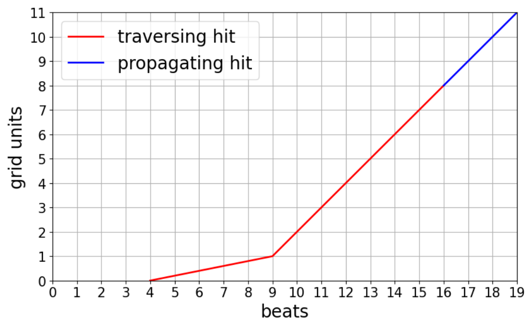


Figure 11: Hit distance from connected composer avatar (grid squares) against time (beats), non-connected composer (4) perspective

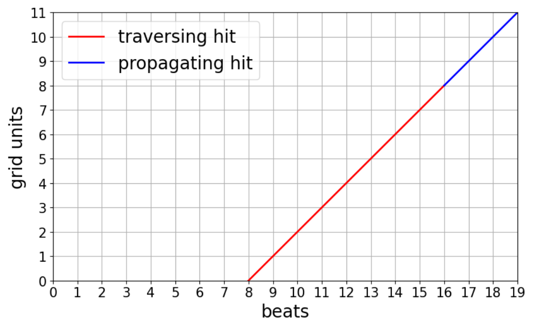


Figure 12: Hit distance from connected composer avatar (grid squares) against time (beats), non-connected performer (5) perspective

the satisfaction of the preceding specifications. The music generated during the test session can be evaluated on whether or not it incorporated the material generated during the test session.

4 Requirements

This section outlines the software requirements and additional UI components necessary to implement the system described in 3.1 Previously described features give rise to the requirements outlined in this section.

4.1 Architecture

There is precedent in other networked music tools to implement a “hub-and-spoke” network topology, in which information from end users is routed through and stored on a central server (8.1, 8.6, 8.4, 8.3, 8.2).

A front end for client operations and a back end for server functionality must also be developed. The server is then responsible for managing and updating the state of all connected clients, and should allow clients to connect through an Internet browser using an independently distributed link. Connections must remain stable, allow two-way data transfer, and maintain association with their initiating clients. The client application must implement an interactive user interface and functionality for audio playback.

4.2 Communication and Timing

A standardized format must be established for bidirectional control data transmission between client and server, such that control data can be routed between clients. The implementation must tolerate the occasional loss of commonly transmitted control data, such that ensemble performance is not significantly affected.

In keeping with the fake time approach, control data for future events is sent in advance, such that it is received before it is scheduled to be executed. This requires clients to maintain synchronized local clocks, such that a consistent timeline is established and accidental scheduling of events long before or after their intended execution time is avoided. The chosen communication format and timing convention must support scheduling future events.

A mapping between time and beats based on tempo must also be implemented to maintain the beat-based musical and visual conventions introduced in previous sections. The server is responsible for keeping track of the ensemble tempo and maintaining the fields necessary to support a mapping from beat to time. The server is also responsible for establishing and maintaining a consistent reference clock to which clients synchronize, and for initiating a clock synchronization procedure between itself and clients. The server must track whether or not individual clients are synchronized, and individual clients should maintain their synchronization status.

Additionally, an upper bound for end-to-end communication must be defined, and the application cannot depend on messages arriving faster than in this worst-case scenario.

4.3 Setup

Clients are responsible for joining the ensemble before a session takes place. For security reasons, we require users to enter a valid username and password before joining an ensemble; the latter can be distributed via alternate channels prior to a session. The server must keep track of the correct password.

As part of this process, each client’s transmission time to the server is roughly estimated. A client is *validated* once a valid username and password have been entered, its clock is synchronized with that of the server, a mapping from time to beats based on an initial tempo has been established, client-side audio playback has been initialized, and the server-client transmission time has been roughly estimated.

Thus, the client is responsible for measuring and reporting its transmission time. The server is responsible for maintaining and changing information about connected clients, including their usernames, whether or not they have entered correct passwords, their individual connections to the server, and their transmission time estimates.

Recall that at least one of each client type (composer, performer, listener) is necessary to carry out a session. When at least one of each client type has been validated, all validated clients display a voting button to initiate the session. If a new client joins or leaves the ensemble, the voting process resets. If a client leaves the ensemble and there are no longer enough clients of each type to carry out a session, the client must hide the option to confirm the ensemble. The server is responsible for assessing when there are enough validated clients of each class to begin a session. The client and server must implement a method for notifying clients when one of each type has successfully joined, and present the option to confirm the existing configuration of the ensemble to clients.

When all validated clients have voted, the estimates of client-server transmission time are used to generate estimates of end-to-end latency. These are scaled and used to generate a two-dimensional configuration of points on a grid whose relative distances reflect the relative physical or network-constrained “distance” between players. These points are then used as the positions of avatars on the grid. This step can only be carried out once all validated clients have voted to start the session. The server is responsible for carrying out this process and reporting a configuration of avatars on the grid back to clients. Clients that are still at intermediate setup stages are shut out of the session before this step begins.

4.4 Session Activities

After the positions of avatars on the grid have been generated, validated clients display a UI containing the complete grid, a chat box, a window with the current tempo of the ensemble, volume sliders for different sources of drums, and an indicator of whether the ensemble is started or stopped. The composer UI differs in that the tempo and start/stop artifacts are interactive. Any composer can start/stop or change the tempo from an initial value set by the server before or after voting is complete. Once any composer initiates this, the session is started and play begins. After this point, clients display their own and others’ activity in the grid listener “perspectives” are logged by the server, and beats begin to increase incrementally with time based on the tempo. Composers can play drums and draw connections between themselves and performers, performers can play along with connected composers, and listeners can move their avatars. Different client classes and users’ own avatars should be distinguishable within the grid.

Clients are responsible for displaying the grid and other responsive UI elements that allow these interactions and others described in 3.1. All clients must implement functionality for displaying connections drawn by composers, traversing hits, propagating hits, avatars, and other UI elements necessary for play. After a tempo change, traversing and propagating hits should continue to move at a rate of one grid square per beat. Composers must implement functionality for establishing and deleting connections, playing instructional drums, and sending tempo change, start and stop messages. Performers must implement functionality for receiving instructional drums and playing response drums. Performers should also implement a scrolling score to combine instructional drums from all connected players; while the movement of traversing hits on the grid can function as a scrolling score, the size of traversing hits, their changes in direction along drawn connections, and the amount of other information being communicated to the user via the grid make this difficult, necessitating a separate, more streamlined scrolling score. Listeners must implement functionality for receiving and playing back response drums, and for moving to different positions on the grid when clicked and dragged.

The server must keep track of the state of the session (started, stopped, in progress) and avatar positions on the grid, and broadcast it to clients for consistency throughout the ensemble. All connected clients must implement functionality for updating their own locally rendered grid given a listener position change. All clients must also implement the distance distortion mechanic described earlier, such that delay is disguised, the timeliness of performers’ response drums is visualized on the grid grid, which is kept as consistent as possible across clients at all times. It is also necessary to limit the number of connected clients and the size of the grid, such that the user interface does not become overwhelmingly cluttered.

4.5 Conflict Resolution and Verification

The server is responsible for verifying that a connection drawn by a composer or a new position chosen by a listener do not conflict with other grid elements. The server is responsible for maintaining a record of occupied and unoccupied positions on the grid, such that when a composer draws a connection or a listener moves their position, this record can be referenced to determine if the new position does not conflict with any other visual artifacts on the grid. If the action is valid, this record is updated and is used as a reference for any future actions, before it is guaranteed that the action that changed the record has propagated to the rest of the ensemble. Invalid actions trigger error messages that explain why the action was rejected.

To prevent listeners from being moved too quickly, a listener position change is followed by a *cooldown* period, during which the listener cannot be moved a second time. The cooldown period must be greater than `MAX_NET_DELAY`, such that it can be reasonably assumed that the position change message has been broadcast to all clients by the time the period is complete. Tempo changes and connections drawn between composers and performers are followed by a similar cooldown period; after a tempo change message has been received by the server from a composer, all composers' tempo wheels are locked for some cooldown time. This also prevents rapid tempo and listener position changes from disrupting playing.

4.6 Shutting Down

Any composer has the ability to stop the ensemble, given that a cooldown period following previous stop/start commands has passed. When the ensemble is stopped, the grid stops displaying user activity and drum hits no longer trigger audio playback on the client. When the server receives a stop instruction from a composer, records of the performance are used to generate performance files, which are made available to download after being generated. The server is then responsible for implementing a logging system for performers' response drums (as perceived by listeners) and for implementing a process by which this information can be disseminated, either after being converted to a format that lends itself to further editing, or after providing users with some resource to convert to a new format.

The server must implement functionality for stopping a session if a client prematurely disconnects, and for delivering a record of the session up until its interruption to clients. The client UI must communicate to users that the session has been interrupted or stopped.

5 Implementation

A note on naming: The Global Drum Circle (GDC), a distributed web-based system to enable drum circle performances across the Internet that I developed collaboratively with my academic advisor, Professor Roger Dannenberg, was used in software that formed the basis for this new system, called "ELBS" (experimental latency-based system), hence identifiers and filenames in the implementation often use "GDC" instead of "ELBS." Sections 5.1.1, 5.1.2, 5.2, and 5.5.3 through 5.5.7 refer to implementation details that are unchanged or only slightly adapted from GDC. Other aspects of GDC are described in Section 8.6.

5.1 Client-Server Implementation

The application implements a hub-and-spoke network topology. To maximize accessibility, the client application is a browser-based interface. One user is responsible for setting up a server and distributing a link to access a client. During testing, the server was set up on a Google Cloud virtual machine with a public IP address. Multiple client instances can easily be opened in separate browser tabs.

5.1.1 Back End

The back end implementation uses WebSocket, a communications protocol that enables full-duplex transmission between a client and server via a single TCP connection [83], to establish and maintain connections between clients and the server. Using a tool intended for facilitating persistent connections is more appropriate than keeping a GET request open beyond the typical request-response cycle of traditional HTTP. The back end is implemented in Java; at the time the project was started, I had more experience writing applications in Java than in other languages. There also existed an easily extensible base code for a Java-based WebSocket and HTTP server, Joe Walnes' Webbit [84]. A possible alternative would have been to use JavaScript and node.js to write a server. Node.js is a collection of independently managed libraries, which undergo frequent and unpredictable changes, leading to an unstable foundation for code development. I decided to use Webbit rather than node.js to avoid frequent updates and versioning complexities. The server implementation is composed of a main class `Gdc.java` responsible for establishing an HTTP server, and an extension of the base WebbitServer implementation `GdcWebSockets.java` that implements the ELBS server interface using the O2lite protocol over websockets.

5.1.2 Front End

The project's front end is written in p5.js, a free and open-source JavaScript library for graphics programming [85]. p5.js allows the embedding of "canvases" within web pages and offers many drawing primitives (circle, rectangle, line, etc.), but has a very limited audio API and no support for interacting with a back end. There is some precedent for utilizing p5js in networked music tools with GUIs [75]. Because the project is a prototype, it is appropriate to decouple the back end implementations of graphics and audio scheduling to allow for re-implementations in various languages and frameworks without being constrained by the details of the prototype implementation.

An alternative to p5.js would be to use ReactJs, an open-source JavaScript library for building user interfaces [86]. React's unique component-based development system makes it difficult to express components in terms of simple primitives. Because the images that need to be displayed are very specific to how the prototype operates, it makes more sense to draw from primitives than to try to use auxiliary tools like Framer Motion and React Reveal to animate elements of the page. React also utilizes unidirectional data flow, such that child components are not able to update data from a parent component; it seemed unwise to introduce this restriction when developing a prototype I knew would have to contain mutually dependent components that reacted in different ways to user input.

The client implementation consists of a directory `js/` containing several JavaScript files, `o2ws.js`, an implementation of O2lite over websockets, `scheduler.js`, which implements the scheduling algorithm, `key_positions.js`, which contains the keys whose general positions on the left and right side of the keyboard are the same across ANSI, ISO, and JIS standards, and `elbs_client.js, elbs_ui.js, grid.js` and `sketch.js`, which register handlers, keep track of state, and render the user interface using the p5js library. The p5js library itself is also included in the client implementation. Each client class is linked to a directory that includes a `user_unique.js` file containing handlers and object classes exclusive to said class, to replicate as little code as possible while minimizing the need to check the client class, as well as a file `index.html` that references the above files in `js/`.

5.1.3 Upper Bound for End-To-End Communication

It is necessary to set an upper bound for end-to-end communication. The application cannot depend on messages arriving before this time.

Recall from 2.1 that the worst-case transmission time from sender to receiver, given one dropped packet detected by the sender, is `TT + RTO`, and that `RTO` $\geq$ 1000 ms. Given a dropped packet detected by the received, the worst-case transmission time is `(3 * TT) + P`, where `P` is the time between the transmission of the dropped message and the subsequent message. These are shown in Figures 13 and 14, respectively. While an upper bound cannot be placed on `RTO`, one can be set for `P`, such that the worst-case transmission time between client and server is reduced by forcing the receiver to acknowledge a dropped packet: A timer on the client side triggers the transmission of a "no operation" (NOP) message, some `P'` ms after every message containing data is sent. However, we set `P'` $>$ `2 * TT` to accommodate the transmission time of an acknowledgment message and avoid transmitting a NOP message prematurely.

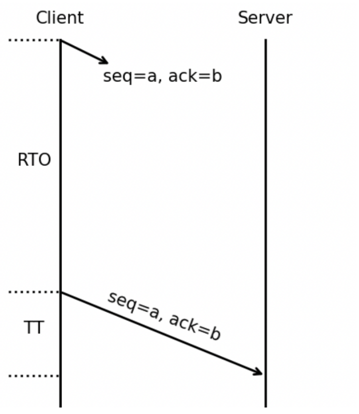


Figure 13: A timing diagram showing a packet loss detected by a TCP sender.

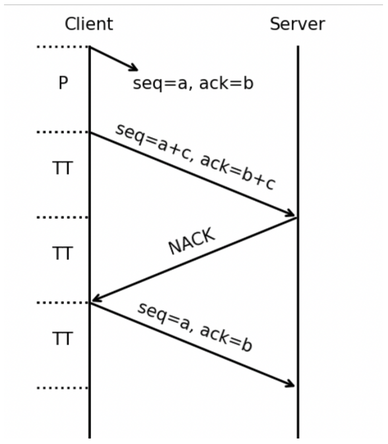


Figure 14: A timing diagram showing a packet loss detected by a TCP receiver.

If the sender transmits another message before `P’` has passed, the timer is reset. Thus, the worst-case client-server transmission time, assuming one dropped packet, is `P’ + TT`, where `P < P’`. It follows that the estimated worst-case transmission time between a client with transmission time $\mathtt{TT}_1$ to the server and another client with transmission time $\mathtt{TT}_2$ to server, assuming one dropped packet, can be reduced to $\mathtt{P' + 3TT_1 + TT_2}$. This is shown in Figure 15. We account for one packet loss in our chosen value for the end-to-end transmission time; there is theoretically no upper bound given the possibility of repeated packet losses, but this is unlikely.

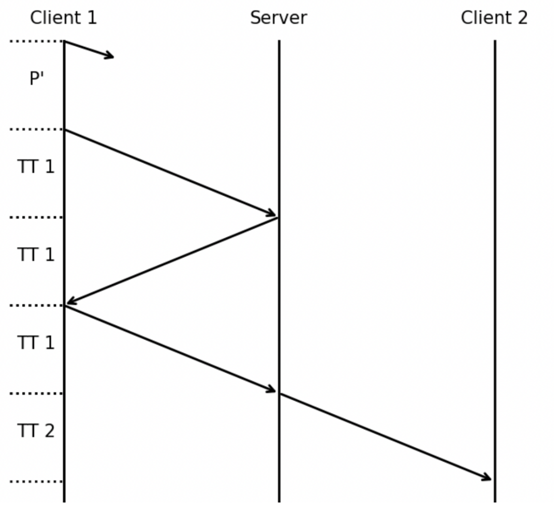


Figure 15: A timing diagram showing end-to-end transmission between end users, given one lost packet detected by the server.

Setting an upper bound then requires an estimate of the worst-case one-way transmission time between clients. It is necessary for other aspects of the implementation to make rough estimates of the one-way transmission times of clients (described further in 5.3.2). These values are on the order of milliseconds, so it would be naive to reference them as part of an estimate of the worst-case delay. Google Cloud’s virtual machine performance dashboard provides a number of useful statistics; at time of writing, the highest median round-trip transmission time is roughly 300 milliseconds between an external endpoint and Google Cloud region and 600 milliseconds between two Google Cloud regions. The highest packet loss rate between regions is roughly 0.1 percent [87]. Note that the values given by [87] do not take into consideration application-specific sources of latency, some of which are referenced in 5.3.2. To acknowledge other sources of latency possibly absent from the statistics in [87], we use the worst-case round-trip time of 300 milliseconds between an external endpoint and Google Cloud region as the worst-case one-way transmission time such that $\mathtt{TT_1 = TT_2 =}$ 300 milliseconds. We set $P' = 3 * TT = 900$ ms, which is shorter than the minimum RTO.

We then set `MAX_NET_DELAY` $= 2100$ ms, but because this is still a rough estimate, the implementation ensures that play is not disrupted even if messages are received late.

5.2 Communication and Timing

5.2.1 O2lite Messages

O2 is a network protocol for music control that supports clock synchronization. O2lite is a subset of O2 that operates over WebSockets and allows applications to connect to an O2 host that can relay messages to an ensemble. The implementation uses O2lite to facilitate communication between client and server. Java and JavaScript implementations of O2lite are utilized on the server and client side, respectively.

In O2lite, a “message” is an address pattern representing a designated operation or function, a string of characters representing parameter types (“type string”), and a set of values representing parameters with the types specified in the type string. A “service” is a named server or client that receives and acts on O2lite messages. A collection of collaborating processes is called an “ensemble”; all components belong to an ensemble and communication is only supported between members of the ensemble. Messages are delivered to and handled by services within an ensemble, and a handler must be installed for every type of message a service receives. The ELBS application implements handlers for all message types.

All O2lite messages are timestamped such that their associated functions can be called according to their timestamp [88]. The ELBS application implements its own scheduling system, so all O2lite messages are timestamped with 0.0. Thus, associated functions are called immediately and the custom scheduling system supersedes O2lite’s.

5.2.2 Clock Synchronization

O2lite also implements algorithms for synchronizing internal clocks between processes. One process' clock is designated a "reference" to which all other processes' clocks are synchronized. Time is counted in double-precision seconds since start. In the ELBS application, the server's clock is designated as the reference clock and used to synchronize clients. The O2 clock synchronization algorithm was re-implemented in Java so that the existing JavaScript implementation of O2lite could be used as is.

5.2.3 Tempo and Beats

Tempo is calculated and enforced as a mapping from time to beats, which are incremented at a regular rate according to the tempo and the *time offset*, or the time at which the tempo was changed. This value in beats is called the *beat offset*. Both values are maintained by the server. The ensemble always starts from beat = `0.0` and time = `0.0`. Beats are mapped to time via the following equation:

$$Beat = BeatOffset + Tempo * (Time - TimeOffset) \tag{1}$$

Where `Tempo`, `Time`, `Beat`, `BeatOffset` and `TimeOffset` are maintained by the server and kept consistent throughout the ensemble. Other values monitored by the server are `TempoEpoch`, which prevents tempo change conflicts, and `StartTime`, the start time of the ensemble; both are also broadcast to clients.

Note that if one were to send timing commands of the form "start now at 100 BPM,", jitter could cause all clients to operate somewhat out of synchronization. To avoid this problem, we send *mappings* from real time to beat time where mappings are determined by `Tempo`, `BeatOffset` and `TimeOffset`. Since all clocks are synchronized and mappings are time-invariant, it follows that the locally calculated beat is consistent across all clients. Mappings are transmitted in advance with a timestamp and updated synchronously across all clients.

The ELBS application supports tempo changes between nonzero positive values, as well as stopping (decreasing tempo to zero) and starting (increasing tempo from zero). Tempo changes are synchronized across clients. When a client initiates a tempo change from `Tempo` to `NewTempo` at time `TempoChangeTime` in seconds and `TempoChangeBeat` in beats, the *timemap* (the new tempo represented in beats per second, as well as the time and beat offset the tempo change was initiated) is sent to the server, along with the `TempoEpoch` of the client.

If the value of `TempoEpoch` sent with the tempo change message is the same as the value on the server, the server increments `TempoEpoch`. In the new timemap, `MAX_NET_DELAY`/2 (an appropriate estimate of the maximum net latency between server and client) are added to the time offset to ensure synchronization between clients. The current beat is then

$$Beat = BeatOffset + Tempo * (TempoChangeTime + (\texttt{MAX\_NET\_DELAY}/2) - TimeOffset) \tag{2}$$

and the new timemap values are calculated with

$$\begin{align} BeatOffset &= Beat \tag{3}\\ Tempo &= NewTempo \tag{4}\\ TimeOffset &= TempoChangeTime + (\texttt{MAX\_NET\_DELAY}/2) \tag{5}\\ TempoEpoch &= TempoEpoch + 1 \tag{6} \end{align}$$

The new timemap and `TempoEpoch` are then broadcast to all clients. A function `sched_set_timemap()` is called to then effectively change the rate at which beats progress with time.

5.2.4 Beat-Based Scheduling

The ELBS application implements its own scheduling system; all messages are timestamped with 0.0 so that associated functions are called immediately.

Events scheduled on the client side are represented as "pending events" which are composed of the beat at which an event should take place, the function to call at said beat, and the parameters to said function. An empty queue of pending events is initialized when the timemap is initially received from the server. Pending events are internally executed at the specified times and cleared from the pending events queue once executed. Functions also exist for converting beats to time, and vice versa.

5.3 Setup

5.3.1 Server Setup and Initialization

Setting up the server requires installing Java and Maven on a virtual machine with a public IP address, transferring the project from GitHub to said machine, creating a .txt file with the correct password to join the ensemble, and compiling and running the Maven application to generate a link to the role selection page, which can be sent manually to interested users. The password is created during the setup process to avoid storing the password with source code. A shell script `runscript.sh` is also available in the Git repository to carry out most of this process on a Google Cloud virtual machine. So far, I have used an e2-micro instance, which, without any additional tiers enabled, has 1 Gbps ingress bandwidth and 7 Gbps egress bandwidth [89].

The server implements an enumerated type `SessionState` to manage the state of the session. The possible values are `SETUP` (still waiting for a sufficient number of clients to connect), `ACTIVE` (clients are actively participating in composition and performance), `GENERATINGMIDIS` (server is generating MIDI files to be sent to connected users), and `COMPLETE` (the session has been stopped and is complete).

To keep track of the information associated with each connected client, a `ConnectionInfo` class is implemented with a constructor that initializes an ID number for the object. The `ConnectionInfo` also contains an enumerated type `SetupState` with possible values `INIT` (client has not yet been validated), `VALIDATED` (client's local clock is synchronized to reference clock and a valid username and password have been entered), and `VOTED` (client has voted to begin the session after an appropriate number of clients of each type have joined). It is initially set to state `INIT`. The `ConnectionInfo` also keeps track of whether or not a client's clock has been synchronized, valid credentials have been entered, and a maximum round-trip time between the client and server has been determined out of recorded values. It also keeps track of the client's class (one of `COMPOSER, PERFORMER, LISTENER`). This can be used to determine whether or not at least one of each client type has been validated by the server. The server also implements mappings `ConnectionsMap` between Websocket connections and `ConnectionInfo` objects, and `handlersMap` between message addresses and handlers. `TempoEpoch` and the timemap values `Tempo`, `BeatOffset` and `TimeOffset` are set to zero. `StartTime` is set to the system time.

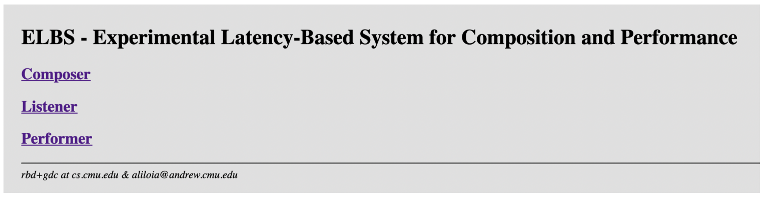
The server also implements fields `IDCounter` and `TempoEpoch` that are initially set to 0. When a new client connects with the server, a new `ConnectionInfo` object is initialized with ID `IDCounter`, a mapping `<WebSocketConnection, ConnectionInfo>` pair is inserted into `ConnectionsMap`, and `IDCounter` is incremented.

5.3.2 Client Initialization

Clients implement a state machine to manage the steps of setup. The possible states are `INIT` (audio has not yet been set up), `SYNCING` (audio has been set up but local clock has not been synchronized to server clock), `READY` (audio has been set up, clock is synchronized, tempo on client side is zero), and `PLAYING` (same conditions as ready, but tempo on client side is greater than zero).

Clients implement boolean variables to track their own setup steps (`valid_credentials_entered`, `timemap_set`, `client_voted`, `client_validated_by_server`) and to track the state of the entire ensemble and determine which UI elements to display (`voting_over_grid_initialized`).

Clients implement a class `UserInfoTable` to store information about each client connected to the ensemble, including their position on the grid (in grid coordinates and window coordinates - the latter is referenced when drawing), user name, ID, and user type (composer, performer, listener). User type and user name are stored in arrays and are accessed with a mapping between IDs and array indices. The `UserInfoTable` also maintains counters for the number of connected clients of each type, which are maintained and displayed in the UI only during setup. Client-side audio playback is implemented via a JavaScript AudioContext. The AudioContext begins in the `suspended` state. `audioContext.resume()` is called during the client initialization process, which switches the state to `running`. The AudioContext state is polled to determine if this has taken place. Once the AudioContext state is `running`, the client enters the state `SYNCING`. Upon connection to the server, `clock_synchronized, timemap_set, valid_credentials_entered, vote_cast = false`. New users are prompted with the option to choose between client types (Figure 16), which, as previously mentioned, are implemented as different web pages, such that multiple clients can be opened at once.


ELBS - Experimental Latency-Based System for Composition and Performance

Composer

Listener

Performer

rbd+gdc at cs.cmu.edu & aliloia@andrew.cmu.edu

Figure 16: The client type selection screen displayed to users after initial connection to the server.

Users select a client class by clicking on a class name, after which different client types display the same setup screen in different colors. To give users a sense of the state of the setup process, when a client first connects, its `UserInfoTable` is updated with the number of clients of each type already in the state `VALIDATED`. This information is displayed via the UI during setup only. (Figure 17).

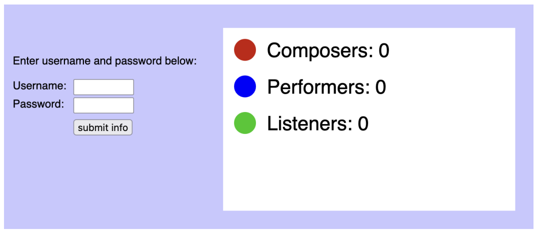


Figure 17: The composer client's UI for entering a username and password. Background color is set using the corresponding class color on the table.

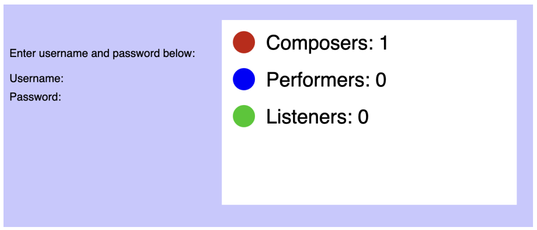


Figure 18: The composer client's UI after a username and password have been entered, but before there are enough validated clients to begin a session.

When this page is displayed, the server assigns an ID number to the client, and the O2lite clock synchronization procedure begins. The server sends the timemap to the client after their clocks have been synchronized. The UI components that enable username and password entry are also displayed during this step. When a user enters a name and password, the client transmits it to the server, which determines their validity. If they are valid, the username and password entry sections are hidden (Figure 18).

5.3.3 Estimating TTs

The process of estimating client-server transmission times is then initiated.

Accurately measuring the transmission time during setup is challenging because it can vary over short or long periods of time [1, Chapter 9.14], [90]. Averages from numerous measurements are not necessarily stable or accurate. However, transmission time is influenced by factors that tend to remain stable (physical distance, network infrastructure, protocol overhead) and factors that can be expected to fluctuate (available network bandwidth, network congestion, whether or not packets have been dropped). We can assume that variation between measurements is due to fluctuating factors, and that the minimum transmission time measured over a given time results from factors that remain stable during that time.

Transmission times are estimated using round-trip O2lite message times in milliseconds, using universal time. During solo testing of the application with multiple clients open in multiple tabs, the average setup time for an entire ensemble of 3 to 10 people was roughly 20 seconds. A pattern by which to fan out messages and compare their transmission and reception time was developed that took 20 seconds. Given $\texttt{N} = 10$, $\texttt{S} = 20$, and $\texttt{F} = 2$, `N` measurements are captured, with `S` milliseconds between the first two, and the time gap between measurements increasing in each iteration by a factor of `F`. With these values, the process of taking one estimate is roughly 20 seconds. Once `N` measurements are taken, the minimum round-trip transmission measured is halved, transmitted to the server, stored in the client's `ConnectionInfo` object, and associated with it for the rest of the session.

While we use multiple O2lite messages to include the latency and overhead produced by the application itself in the estimate, the `ping` command could alternatively be used to measure the round-trip time [91].

When the server determines that a client's clock has been synchronized, it has entered a valid username and password, and a minimum transmission time has been established, all clients' `UserInfoTables` are updated with the number of validated connected clients. The client is also notified that it has been validated.

5.4 Client Voting

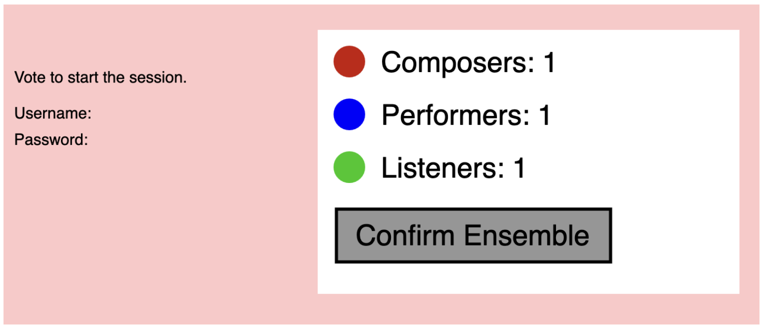
To support different configurations of client types with more than 3 clients, the `ACTIVE` state is not entered automatically once enough clients of each type have joined.

When the client determines that `timemap_set, client_validated_by_server = true` and from the `UserInfoTable` that at least one client of each type has connected, it is presented with the option to vote to begin the session in the existing ensemble state.

When the vote is cast, `vote_cast = true` and `SetupState = VOTED` on the client and associated server-side state machines.

If a new client is validated or disconnects during the voting process, voting resets. `vote_cast = false` and `SetupState = VALIDATED` on the client and associated server-side state machines. If, after a valid client disconnects, there are still enough players of each type to begin a session, the vote submission UI is displayed.

Once all validated clients have confirmed to start the session, the server begins generating a configuration of avatar positions and `SessionState = ACTIVE` is set.


Vote to start the session.

Username:

Password:

● Composers: 1

● Performers: 1

● Listeners: 1

Figure 19: The composer client's UI after enough clients of each type have been validated.

5.4.1 Converting Transmission Times to Drum Delays

The transmission time estimates taken in the previous step are used to generate the configuration of avatars presented on the grid. Thus, the distance between client avatars on the grid is roughly proportional to the sum of their roughly estimated transmission times to the server.

The server implements a class `GridState` that carries out this process. `GridState` implements a class `Node` with information about each validated client's ID and transmission time, as well as information about their position in two-dimensional space and the force exerted on them in x and y. Nodes are initially placed in a circle with radius R around the origin of this two-dimensional space. We model a spring connecting each pair of clients in our simulation, whose rest length is proportional to the sum of the clients' transmission times. Force in x and y exerted on each node by the springs is calculated iteratively and nodes are displaced accordingly. `MAX_ITERATIONS` $= 1000$ iterations of this process are carried out; during testing, a wide range of configurations safely converged within this number of iterations. To prevent divergence, forces are only calculated when the distance between nodes is greater than or equal to 1 unit of physical distance [92]. To prevent an initial configuration where the distance between any two nodes is less than one distance unit, we scale up the transmission times associated with each node by some scalar value P such that all transmission times are greater

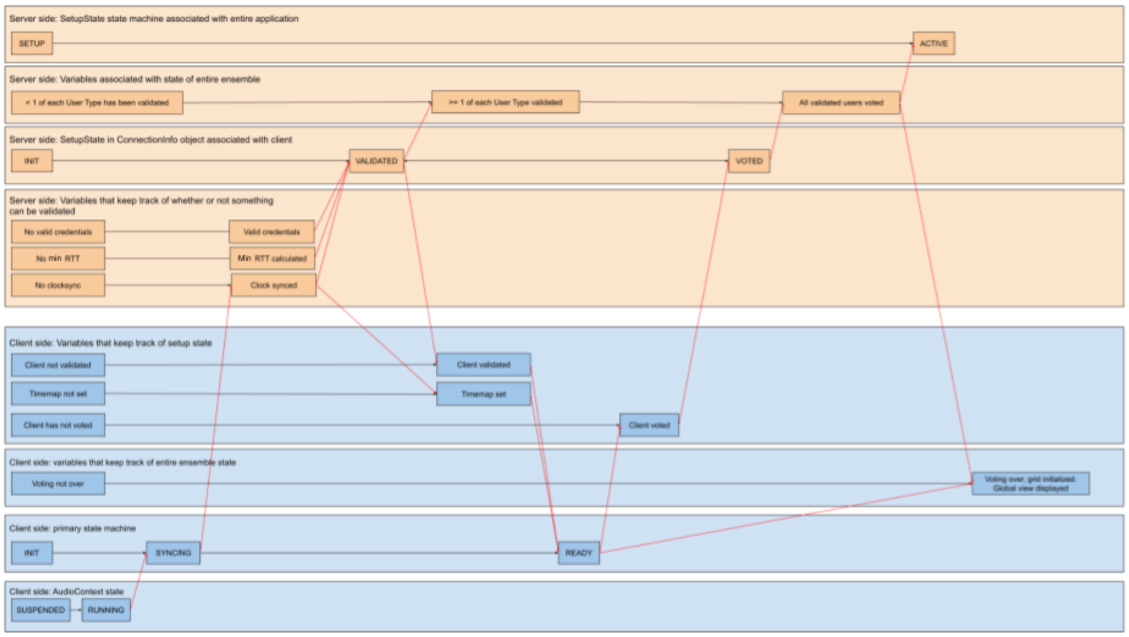

Server side: SetupState state machine associated with entire application
SETUP
ACTIVE
Server side: Variables associated with state of entire ensemble
< 1 of each User Type has been validated
>= 1 of each User Type validated
All validated users voted
Server side: SetupState in ConnectionInfo object associated with client
INIT
VALIDATED
VOTED
Server side: Variables that keep track of whether or not something can be validated
No valid credentials
Valid credentials
No min RTT
Min RTT calculated
No clocksync
Clock synced
Client side: Variables that keep track of setup state
Client not validated
Client validated
Timemap not set
Timemap set
Client has not voted
Client voted
Client side: variables that keep track of entire ensemble state
Voting not over
Voting over, grid initialized. Global view displayed
Client side: primary state machine
INIT
SYNCING
READY
Client side: AudioContext state
SUSPENDED
RUNNING

Figure 20: A diagram illustrating the state changes on the client and server side. Black arrows denote possible transitions between states, red arrows indicate the preconditions for moving between states associated with the other states shown in the diagram.

than 1. We also calculate the radius of the circle $R = 1/(2 * \sin(\pi/N)$, where N is the number of nodes, such that all nodes are separated by at least one distance unit in the initial configuration. After `MAX_ITERATIONS`, a configuration of nodes whose relative distances are relatively proportional to the sum of their transmission times scaled by P can be safely assumed to have been reached, at least to some approximation. Note that positions are restricted to the plane, so in general, it is impossible to find positions such that all pairwise distances match the pairwise transmission times.

Each point is then quantized to a grid in random order by rounding their positions, in window coordinates, to that of the nearest grid coordinate, such that each node is separated by small integer numbers of grid squares. A square bounding box for the avatars is calculated and the final configuration of nodes is set against a square grid with dimension `GRID_DIM` by `GRID_DIM`. In the implementation, `GRID_DIM` $= 16$. The grid size is intentionally limited because, like in physical ensembles, larger groups make coordination and situational visibility more difficult. Additionally, informal tests with users with a range of familiarity with computer-based music software and video games indicated that this size was optimal for exploration and operation. Larger grids necessitated smaller icons, which made it difficult for composers to point and click. Smaller grids made exploration feel limited. This choice also has musical significance; a propagating or traversing hit on one side of the grid moves across it in four measures.

Given a conflict between two grid positions, all points at a distance $n = 1$ from the nearest point are examined for conflicts. If an intermediate configuration of positions prevents any users from fulfilling their responsibilities, e.g. a composer surrounded by other composers, the issue is resolved by relocating the affected node to the closest (in Manhattan distance) open coordinate.

Grid coordinates are referenced using positive integers, with (0,0) being the top left corner. Node positions in grid coordinates are stored as `gc_x` and `gc_y` in each client's `ConnectionInfo` object. Avatars' grid coordinates are broadcast to all clients.

At the end of this process, rough estimates of network delay have been converted to temporal distance within the music produced by the ensemble. A visualization of the ensemble that conveys these distances intuitively has also been generated and broadcast to clients. After this process is complete, validated clients are presented with the interactive grid containing their own avatars and other task-specific visual artifacts.

5.4.2 Hidden Delay: Distorting Distances on Grid

It is important for other users to perceive each other's timing sensibilities, especially how promptly and accurately performers respond to composer drum hits. Users can be made aware of how well other participants are meeting their assigned tasks. However, messages need time to propagate through the system, thus requiring at least `MAX_NET_DELAY` between when a drum is played and when it can be scheduled to appear on other clients.

As explained and justified previously in section 3.2, we introduce a hidden delay $H = xB$ to hide transmission delay. H is measured in time units unless otherwise noted, x is an integer, and B is the duration of one beat at the ensemble tempo. This is expressed visually by lengthening the composer-performer distance as seen by composers and the performer-listener distance as seen by performers, relative to the distances displayed in other clients. Composer and performer drum hits appear above their avatars. They are initially large and move slowly; as they traverse or propagate, they shrink and accelerate, reaching a constant speed of one grid unit per beat when one unit away from the avatar of the client that played it Composer clients "see" their own drum hits move one grid square away from their avatar over the course of $2x + 1$ beats. Performer clients "see" their own drum hits move one grid square away from their avatar over the course of $x + 1$ beats. This movement emulates that of an object viewed from above as it rolls down a slope.

Thus, a composer's drum hit message has x beats to be broadcast, whose response drums have a further x beats to be broadcast, such that all musically upstream and downstream users perceive each other's propagating and traversing hits in the same way on the grid (with performers being x beats ahead). Performers see a shorter path, so they play earlier than composers "expect", such that their drum hit messages can be assumed to have been successfully delivered to both listeners and performers before they need to be displayed. Thus, if performers are closely following composers' instructional drums, the timeliness of their responses are visible on the grid as a propagating hit appearing immediately after a composer's traversing hit reaches them. The relative simultaneity of traversing hits reaching composers and propagating hits leaving composers indicate the timing accuracy of the latter. At the same time, the GUI is immediately responsive to user input, so users can interact continuously with the GUI even while H is well above 100 ms. Users are given the impression that they are playing "live", and an arbitrary latency can be accommodated.

Given a different method of displaying drum hits, H could be rounded up to the nearest B whenever necessary; in this case, x would be recalculated every time the tempo is changed. Thus, the duration of the acceleration process in beats for drum hit visuals would also have to change, which would require the visuals to change position suddenly or move backward. This is addressed by setting fixed values for x and the maximum tempo, ensuring that x beats always last longer than 2.1 seconds at the ensemble's highest possible tempo. In the implementation, $x = 4$ is hard-coded; the maximum tempo is then $114 \approx 60 * x/2.1 = 114.28$

Alternatively, the maximum tempo could be set slightly higher. While tolerating late arrivals without significantly disrupting performance is not an explicit goal of the system, it is a favorable trait of networked music systems. As mentioned in Section 5.6.1, drum hit messages that arrive before their downstream play time can still appear in a manner that does not disrupt play. Setting the maximum tempo slightly higher and increasing the likelihood of messages arriving late could show that this goal has been achieved.

Allowing a higher tempo could also allow for a faster rate of musical joint action. Although users can interact continuously and constantly with a locally rendered representation of the ensemble via the grid, there is always a 2.1-second delay (minimum) between a client's drum hit and when it is displayed in other clients.

I decided not to implement this, as the grid was implemented to prevent interactions from feeling sequential rather than continuous, and while 2.1 seconds is still a conservative estimate, data frame transmission time has no theoretical upper limit. Like `GRID_DIM` = 16, $x = 4$ is chosen in part due to its musical significance, as it fits evenly into measures. $x = 2$ would restrict play to low tempos, while $x = 8$ would require high speeds to avoid long delays.

The server maintains the maximum tempo in a variable `MAX_TEMPO`. The minimum tempo `MIN_TEMPO = 40` is hard-coded; it is considered the minimum tempo at which subdivision interonset intervals exist or change meaningfully [93].

5.4.3 Limiting Drum Hits Sent Per Second, Number of Connected Clients

Batching drum hit messages for group transmission could reduce network congestion during sessions. Transmitting a data frame in larger and fewer packets lowers the total overhead from processing each packet. I did not implement this to avoid overemphasizing grouped messages; individual hits that arrive late have less of an effect on play than delayed batches. Instead, a timer is implemented on the client to limit the frequency of drum hit messages. The minimum interval between drum hit messages was set to 20 ms, considered the shortest rhythmically meaningful interonset interval. This also prevents the grid from becoming cluttered further, as closely timed hits are indistinguishable. I also found during tests of the application that if drum hits were not limited, sending hits as quickly as possible slowed down the client application and disrupted the progression of the local clock.

Bandwidth limitations must be considered when determining how many clients can connect. Given that `GRID_DIM` = 16 and 256 (x,y) grid coordinates exist, we allow 256 client connections at once; the server also prevents more than 256 clients from joining the same session. This is reasonable given bandwidth expectations. Drum hit messages, which are roughly four bytes each, are expected to make up most of the network traffic during a session. Drum hit messages can be sent every 20 milliseconds, so 200 bytes could be transmitted by each client per second. Given that 256 clients total could join the ensemble, each client would need to accommodate 256 * 200 Bps = 409.6 kbps of bandwidth. Likewise, the server would need 256 * 256 * 200 Bps = 104.8 Mbps of bandwidth. So far, I have used an Google Cloud e2-micro instance to host the server, which have 1 Gbps ingress bandwidth and 7 Gbps egress bandwidth (as described in Section 5.3.1), which are well above the estimated maximum bandwidth necessary for the system.

While system supports up to 256 clients, it works best with fewer connections, as free spaces on the grid are necessary for musical activity to take place: Without any free spaces, connections cannot be drawn, listeners cannot be moved, and the entire user interface becomes overly cluttered such that individual user activity becomes difficult to monitor. Thus, most sessions are expected to involve tens of clients, maximum.

5.5 General UI Components

After a configuration of avatars has been generated, validated clients are presented with the grid and avatars, as a "chat" for non-musical communication, a window with the current tempo, volume sliders for different drum sources, an indicator of whether the ensemble is started or stopped, and a metronome. Clickable UI elements implement a `contains()` method that indicates whether or not a user's mouse is inside the element.

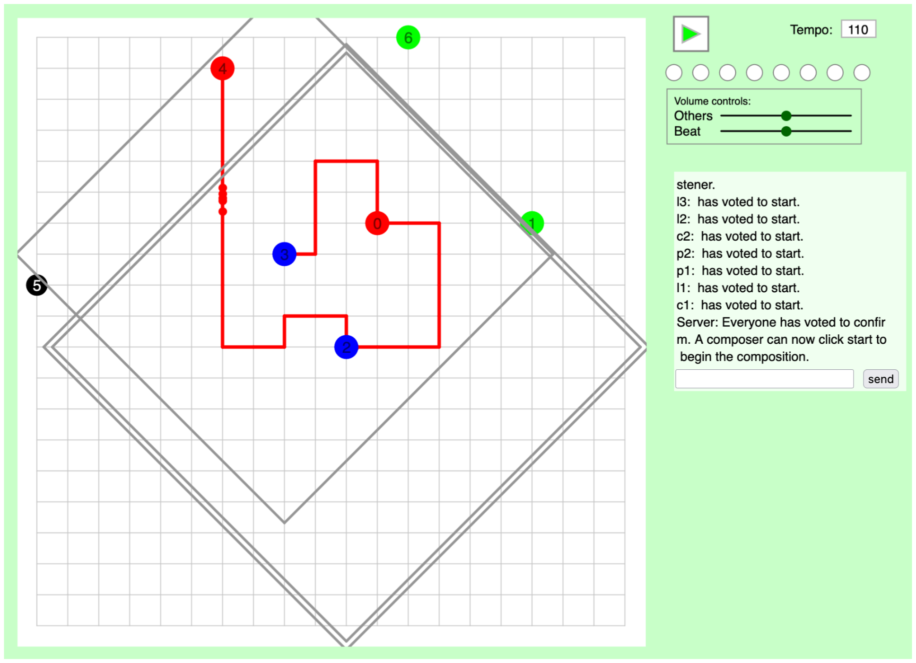
5.5.1 Grid

Clients implement a class `Grid` with the grid functionality described in previous sections, including nodes and composer-performer connections. Functions `draw_to_grid_coord` and `grid_to_draw_coord` convert between grid and window coordinates, where (0, 0) is the top left grid coordinate. From here, "player" refers to the user whose vantage point is being used to describe the application.

`x` and `y` refer to horizontal and vertical grid coordinates.

5.5.2 Nodes / Avatars

Clients are represented with simple circular avatars, or nodes, on the grid. Nodes are colored corresponding to their class; composers are red, performers are blue, listeners are green. All nodes are also numbered with their client's ID. Clients display their own nodes as black. Nodes are initialized with their corresponding player's ID, which is used to reference corresponding fields in the `UserInfoTable` to avoid replicating state. The composer and listener clients implement classes `ComposerNode` and `ListenerNode`, which, respectively, provide appropriate responsiveness and functionality to players of said type. A class `FakeNode` is implemented to represent non-player clients, as well as performers, which are not responsive on the grid.



```
stener.
l3:  has voted to start.
l2:  has voted to start.
c2:  has voted to start.
p2:  has voted to start.
p1:  has voted to start.
l1:  has voted to start.
c1:  has voted to start.
Server: Everyone has voted to confir
m. A composer can now click start to
 begin the composition.
```

Figure 21: The complete UI for a session that includes two composers (IDs 0, 4), two performers (IDs 2, 3), and three listeners (IDs 1, 5, 6), seen from the perspective of listener 5. The grid, chat box, beat lights, tempo wheel, and volume sliders are all shown. Traversing hits are moving from composer 4 to performer 2, and propagating hits are propagating outward from performers 2 and 3.

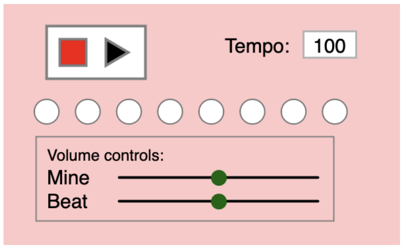


Figure 22: The composer UI, not including the grid or chat box, when the ensemble is stopped.

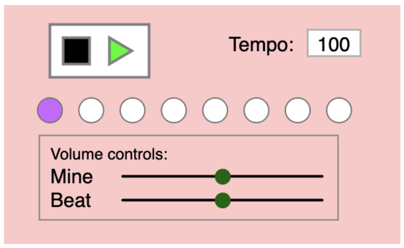


Figure 23: The composer UI, not including the grid or chat box, when the ensemble has started.

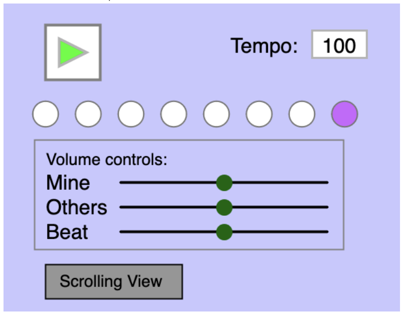


Figure 24: The performer UI, not including the grid or chat box, when the ensemble has started.

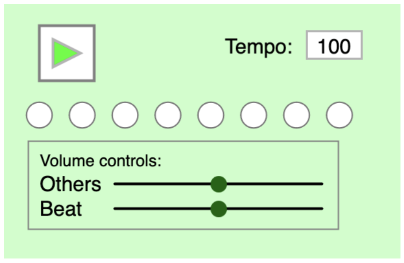


Figure 25: The listener UI, not including the grid or chat box, when the ensemble has started.

5.5.3 Tempo Indicator

All clients display a box with the current ensemble tempo in beats per second. In the composer UI, this element is responsive and can be clicked and dragged to adjust the tempo. Tempo change messages are sent with the beat b at which a composer initiated the change and the value of `Tempo_Epoch` monitored by the composer. The server confirms that the `Tempo_Epoch` sent by the client and maintained by the server are equal and broadcasts a message scheduling a tempo change to take place at beat $b+t$, where t in beat units is greater than `MAX_NET_DELAY`.

Between the reception and action time of a tempo change message, the tempo box UI element is made unresponsive and the number is changed to the upcoming tempo. This prevents conflicting tempo change messages from being transmitted. The initial tempo is set to 78 bpm, which is halfway through the available tempo range.

5.5.4 Stop / Start

We introduce *stopped* and *started* to refer to how the tempo and session state are expressed to users. The ensemble is started when the current beat is increasing with time and stopped when it is not.

When the grid is first displayed, all client interfaces reflect that the ensemble is stopped. Performers and listeners see a green triangle or red square indicating that the ensemble is started or stopped. The composer client interface displays an interactive element with both symbols, either of which can be clicked to send the corresponding message to the server. Tempo can be changed before a session is started. Stop/start messages are sent with the beat b at which a composer initiated the change and the value of `Tempo_Epoch` monitored by the composer. As with the tempo wheel, the server checks that the `Tempo_Epoch` sent by the client is the same as the one on the server. If so, the server broadcasts a message to all clients scheduling a tempo change to take place x beats in the future, where x is set such that the duration of x beats is greater than `MAX_NET_DELAY`. The stop message handler implemented on the server initiates the process of generating MIDI files from logs.

5.5.5 Mixer

All clients implement gain sliders corresponding to the groups of drums it can "hear": Composer clients play back their own drums, performers play back their own drums and instructional drums from connected composers, and listeners play back drums from all performers. All clients also present a metronome that "ticks" once per beat. Users can click and drag these sliders left or right to turn the volume of each type of drum up or down.

5.5.6 Metronome / Beat Lights

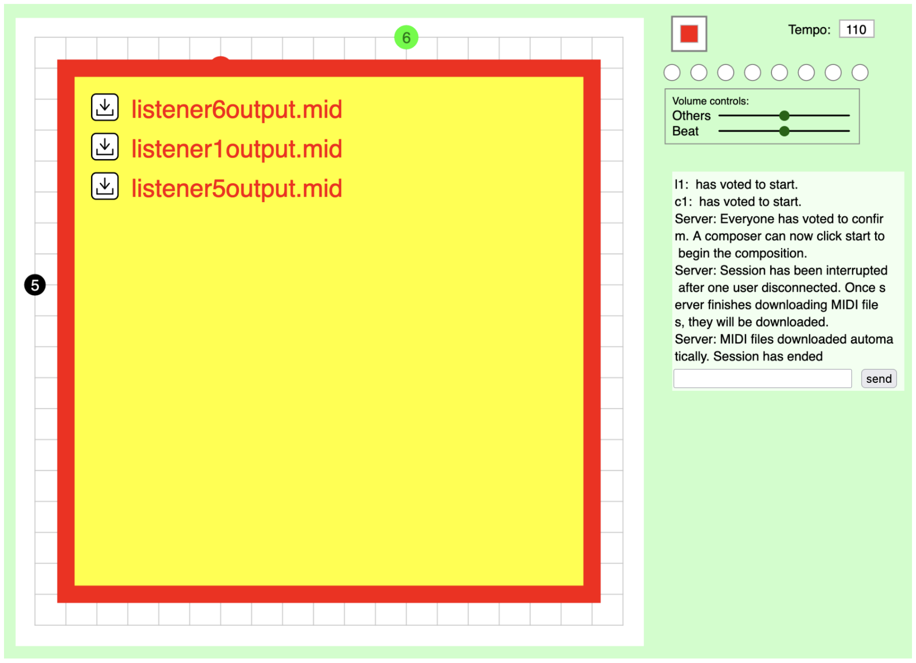
All clients display a horizontal arrangement of eight "beat lights", circles that "blink" on and off to indicate the eighth note in a bar corresponding to the current beat. The beat lights blink in sync with a metronome playing a kick drum, helping users gauge if their playing is on beat. The metronome and beat lights are inactive when the ensemble is stopped.

5.5.7 Chat Window

All users have access to a chat window, text entry field, and "send" button, with which messages can be sent between clients during play. Chats are displayed with the sender's username next to the sent message. Users are shown the chat window at all times, but can only send messages after their client is validated during setup. Clicking the text entry box enables typing, such that key presses do not trigger drum hits. (The process by which key presses trigger drum hits is described further in sections 5.6.2 and 5.7.1.) When the send button is pressed, the message is sent to the server and broadcast to clients. Key presses resume triggering drum hits until the text field is clicked again. The server also sends role-specific instructions to clients via the chat box during setup.

5.5.8 Interruption Window

After play has been stopped, the interruption window covers the grid to express that normal play is no longer possible. It is used to display messages about why the ensemble has stopped (a client has disconnected or a composer has stopped the ensemble) and the status of the MIDI files being generated by the server. When the MIDI file generation process is complete, download buttons for each file appear in the window, as shown in Figure 26.



listener6output.mid
listener1output.mid
listener5output.mid

l1: has voted to start.
c1: has voted to start.
Server: Everyone has voted to confir
m. A composer can now click start to
begin the composition.
Server: Session has been interrupted
after one user disconnected. Once s
erver finishes downloading MIDI file
s, they will be downloaded.
Server: MIDI files downloaded automa
tically. Session has ended

Figure 26: MIDI files shown as available for download after a composer has stopped a session.

5.6 Composer Functionality and UI

5.6.1 Composer-Performer Connections

Composers can draw lines between themselves and performers by pressing the mouse over their own node on the grid, dragging to draw a path along grid lines, and releasing the mouse over a performer node to complete a path. The path length determines the drum delay between the composer and performer. These paths cannot intersect with themselves, other paths, or other nodes on the screen. When a composer clicks a connection it has drawn, it is deleted.

This functionality is implemented via two different classes, `DrawingConnection` and `DrawnConnection`, representing connections that are in the process of being drawn and connections that have been drawn, respectively. All clients also implement a class `DrawnConnectionDisplay` to show connections that cannot be clicked on, for if the start point of the connection is not the client itself. All three classes are implemented as lists of `DrawingConnectionJoint` objects, which contain the endpoints of lines that make up connections. This class also implements a subclass `TraversingHit` that monitors the beat at which a drum was played by a composer and displays a circle at the appropriate point on the connection. A list of `TraversingHits` is maintained and culled as traversing hits reach their destination.

Additionally, the client implementation is such that only the beat at which a drum was played, or the beat at which a beat is scheduled to be played back, is needed to display a traversing hit. The drum index does not affect how the hit is displayed.

Drum hits are displayed based on their play time, such that if they arrive late, they appear part of the way through their path and continue their normal visual progression from that point. Thus, play is not significantly disrupted if drum hit messages occasionally arrive late.

`DrawingConnections` consist of mutable lists of `DrawingConnectionJoints`. When a user draws a path that intersects itself, the `DrawingConnectionJoint` list within the `DrawingConnectionJoint` is sliced so that the first intersection point becomes the final joint in the list. When a `DrawingConnectionJoint` is released over a performer `FakeNode`, the node grows slightly, indicating that the connection can be ended. When the composer releases their mouse, the composer generates a `DrawnConnection` locally using the joint list, and which is transmitted to the server. The cooldown period of `MAX_NET_DELAY` begins in the composer's client, during which the newly drawn connection appears "greyed out", cannot be deleted, and does not display traversing hits. If the server rejects the connection, the inactive connection disappears, potentially before the cooldown period is complete. If there is no conflict between the new connection and any existing visual artifacts, the joint list is broadcast to all clients and scheduled to appear at least `MAX_NET_DELAY` after being transmitted by the composer, such that it is displayed by all clients simultaneously.

Other clients generate a `DrawnConnectionDisplay`, which cannot be clicked, but displays traversing hits. Clients also implement classes `AllDrawnConnections` and `AllDrawnConnectionsDisplay`, which maintain all connections present.

5.6.2 Playing and Displaying Instructional Drums

Composers play drums by pressing the keys on the left and right side of their keyboard. Composers have access to two drum sounds. A different drum is played depending on which side of the user's keyboard the pressed key is on, as indicated by `key_positions.js`. Composer drum hit messages sent to the server consist of the beats at which the drums were played, their drum indices, and the composer ID. All clients implement a method `play_drum(drum_index)` for playing back drum sounds corresponding to drum hits. Functionality for displaying traversing hits and for audio playback are decoupled on the client.

The composer drum hit message transmitted to listeners, composers, and non-connected performers consists of the sender ID and beat at which the drum was played. Connected performers also receive the drum index of the performer drum hit, as well as the drum delay between itself and the performer that played the hit, so that the appropriate sound can be played back at the appropriate beat.

Drum hits are drawn based on their playback time, or the time at which they are intended to reach downstream clients. Messages received after the estimated hidden delay but before their corresponding action is scheduled to be executed (passive playback or active response) appear part of the way through their path. If a composer's drum hit is received late, a corresponding traversing hit appears part of the way through a connection. If a performer's drum hit is received late, a corresponding propagating hit appears further outward from their avatar.

5.6.3 Routing and Caching on Server

Clients can calculate drum delays and schedule drums accurately with grid coordinate information contained in the `UserInfoTable`, as well as the lengths of the `DrawnConnection` and `DrawnConnectionDisplay` objects maintained in the grid.

However, we expect drum hit messages to be sent far more often than drum delay or connection change messages. Because the positions of avatars on the grid are consistent across clients (aside from listener positions during cooldown periods), all clients calculate the same drum delays corresponding to drum hits. Thus, to ease the load on clients, the server caches information about drum delays between clients in a field of their associated `ConnectionInfo` object called `DownstreamUsersDistancesMap`, which maps downstream users' IDs (connected performers for composers, all listeners for performers) to their corresponding drum delays.

To prevent position conflicts between grid elements, the server also oversees which grid coordinates are occupied and rejects actions that would cause conflicts between visual artifacts. The server implements a mapping `occupiedCoordsBijections` of positive integers (keys, representing client IDs) to sets of positive integers (values) that encapsulate the x and y grid coordinates of said client's avatars (and additionally, if it is a composer, integers that encapsulate the grid coordinates of any connections between itself and performers). When a new connection is drawn, the server checks to see that no grid coordinates occupied by the new connection (besides the start and end points) are already contained in any of the values in `occupiedCoordsBijections`. If there is no conflict, integers that encapsulate the x and y grid coordinates occupied by the new connection are added to the value the composer's ID is mapped to. When a connection is deleted, the integers representing the connection are deleted from the value. `occupiedCoordsBijections` and all `downstreamUserDistancesMap`s are updated every time a new connection is drawn or deleted, or a listener is moved.

5.7 Performer Functionality and UI

5.7.1 Following and Playing Drums

Performers only interact passively with the grid, their avatars are represented with `FakeNode`s on all clients. Within the composer client, however, performer nodes to which said composer is not already connected grow slightly when hovered over with the mouse, indicating potential for connections.

Performers have access to the same two drum sounds as composers. Performers play along with connected composers' instructional drums. While the movement of traversing hits from composers to performers could be used as a scrolling score, user feedback has indicated that this is impractical. The size of traversing hits, their changes in direction along drawn connections, and the amount of other information being communicated to the user via the grid make it difficult to follow as one would a scrolling score. The performer client implements a secondary "Scrolling View" which can be shown and exited via a button below the mixer in the UI (Figure 24). At the time of the test session, the scrolling view consisted of circles representing instructional drums traveling from left to right, along invisible horizontal lanes representing separate composers, through vertical lines representing single drum delays, as seen in Figure 27. This was meant to encourage users operating performer clients to play drums as they perceived them intersecting with the rightmost vertical line. The number of drum delays displayed in the scrolling view was equal to the shortest drawn connection ending at the performer avatar; it changed as connections were drawn and deleted. Feedback from my thesis committee after the test session indicated that this scrolling score was illegible, so its design was modified. The new version is shown in 28. Rhombuses representing drum hits scroll upward as the two rhombuses at the top of the UI element flash with the metronome, indicating that left or right drum hits should be played when the rhombuses overlap.

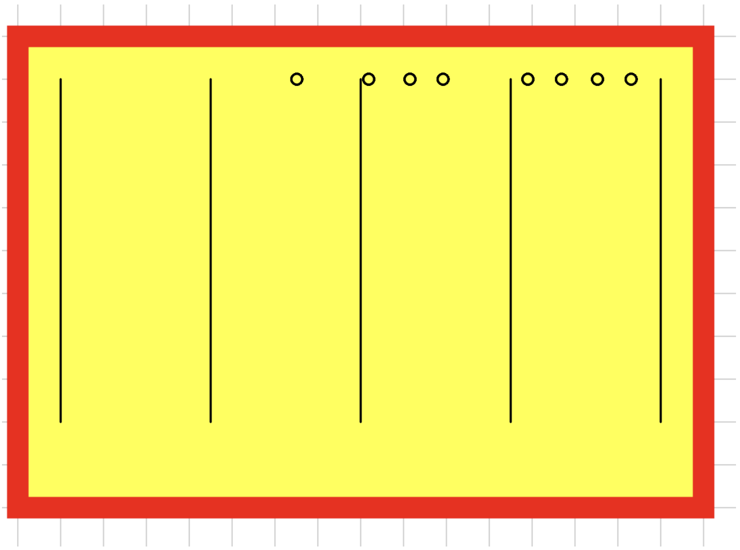


Figure 27: The scrolling view, as it was implemented during the test session.

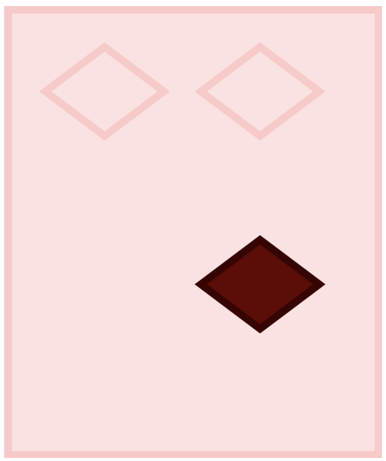


Figure 28: The scrolling view in the current implementation, based on test session feedback.

Performer drum hit messages consist of the performer's ID, the beat at which the drum hit was played, and the drum index. Functionality for displaying propagating hits and for audio playback are decoupled. As with composer drum hits, only the beat at which a drum was played is needed to display a propagating hit, as the drum index does not affect how the hit is displayed. Thus, the performer hit message returned to composer and performer clients only consists of the sender IDs and beat at which it was played. Listeners receive the drum indices of performer drum hits, as well as the drum delays between itself and the performer that played the hit, so that the appropriate sounds can be played back at the appropriate beats.

5.7.2 Displaying Propagating Hits

Clients display performer drums on the grid as rotated squares that propagate outward from the performer's node at a rate of one grid unit per beat (with the exception of performers' own drums, thanks to the distance distortion process described in Section 5.4.2). Clients implement a class `PropagatingHit` containing the beat at which a drum was played by a performer and implements functionality for displaying a rotated square at the appropriate location. A class `AllPropagatingHits` maintains, displays, and culls `PropagatingHit`s as they propagate out of the grid over time.

5.8 Listener Functionality and UI

Listeners do not play drum hits. They are the endpoint of the "life cycle" of a beat and "hear" drums based on their position in relation to performers. Listeners can change their position on the grid throughout a composition by clicking and dragging themselves to a non-occupied grid coordinate. When the player's mouse is released, the new listener location and the beat at which it was moved b_0 is sent to the server, where its validity is checked and cached as described in Section 5.6.3. The server notifies all clients of the listener's new position and the end beat for the cooldown time. All clients update the new listener's position in their `UserInfoTable`, and display the listener node (either a `FakeNode` or `ListenerNode` on the grid at at the new coordinate, colored in gray. To express the cooldown time, at beat b, an arc with length $2\pi * (b - b_0)/(4 * B)$ (where B is the duration of one beat) is drawn over the listener node, giving the impression of a countdown to the end of the cooldown time. Traversing hits that intersect with a listener during its cooldown time are not played back in the listener client or recorded by the server. At beat $b > b_0 + (4 * B)$, the moved listener's node is displayed normally.

5.9 Logging and MIDI File Reconstruction

5.9.1 Computational Artifacts Utilized

Different records of the composition are generated based on when drums are perceived by listeners at different locations on the grid. During performances, the application continuously logs information that can be used to generate accurate listener perspectives of the sessions, including tempo changes and drum hit times / indices / corresponding performer IDs.

We choose to generate records in the form of MIDI files [94]. Most digital audio workstations are compatible with MIDI files and implement functionality for interacting with MIDI files via a "piano roll". MIDI files can contain information about tempo changes and time signatures, as well as the pitch and velocity (0-127) of notes played at different times, and by what instruments (represented by separate "tracks").

MIDI files are delivered after a session is complete and do not have to be generated in real time. Information logging for MIDI file generation is decoupled from the file generation process.

We anticipated difficulty with transmitting binary files to clients from the server, due to variations in browser permissions between clients, or difficulty transmitting files larger than the average O2lite message. The process of converting logs to MIDI files is implemented such that the conversion process could be carried out easily on the client, but the current prototype and testing have involved generating and transmitting MIDI files on the server.

We chose to use Python to convert composition logs to MIDI. Python provides access to a range of libraries for tasks like processing data and generating MIDI files, which can be easily installed via package managers like pip. Python's MIDIUtil library implements functionality for writing notes and tempo changes to specific tracks in MIDI files at specific times, following a time-beat mapping convention similar to the one explained in section 5.2.3. This plus its general ease of use made it a good candidate for a tool to use to generate MIDI files. While Java contains packages for generating MIDI files, installing Python packages in a new virtual environment (or on top of a user's base Python installation) is simpler than compiling a Java application.

5.9.2 Information Logged

The server implements a class `RecordedComposition` that logs pertinent information as it is routed as strings. It reserves space for 10,000 strings, which it exports to a CSV when filled, clears the space, and prepares to write to a new CSV when the application has either stopped or the `RecordedComposition` again runs out of space.

Recall that the beat at which a drum is heard by a listener is the beat it was played by a performer, plus the drum delay between performer and listener. Recall that drum delays between clients and their downstream clients are cached in the former's associated `DownstreamUsersDistancesMap` object. If a performer plays a drum hit at beat b, we can determine the time all listeners hear it using the performer's `DownstreamUsersDistancesMap`. Recall that a performer's `DownstreamUsersDistancesMap` is updated every time a listener is moved. Thus, the server logs performers' `DownstreamUsersDistancesMaps` in its `RecordedComposition` every time they are changed (as shown in Table 2, Row 3). When a drum hit message is received, the server logs it in its `RecordedComposition` as shown in 2, Row 1. This data is utilized to reconstruct individualized logs of when each drum beat is heard by each listener.

Also recall that when a listener is moved, the drum hits it would have heard during its cooldown time are silenced, and all performers' `DownstreamUsersDistancesMaps` are updated. Thus, when a listener is moved, the server logs its cooldown time in its `RecordedComposition` as shown in 2, Row 2. This data is utilized to process the logs of when each drum hit is heard by each listener; drum hits that fall within the given range of beats are muted.

Other data logged by the server in the `RecordedComposition` are tempo changes (in the format indicated in 2, Row 4) and the configuration of clients and their IDs (2, Row 5).

`RecordedComposition` implements functionality for writing logged strings to CSV files in a directory `logs`. The first CSV file is named `log0.csv`, and the numerical identifier is increased every time a new CSV is generated.

Table 1: Information logged and converted to MIDI

Event	**Parameters**
composition start	start beat
composition stop	stop beat, start beat
tempo change	new tempo, beat at which tempo change takes place
drum hit	beat at which drum was hit, which drum was played
listener position change	beat at which position change takes effect, new drum delay between moved listener and each performer

5.9.3 Reconstructing MIDI Files

A Python script `python/generatemidis.py` reads in all log CSV files and concatenates them into a ragged two-dimensional array. The script implements a class `ListenerMIDI` containing functionality for preprocessing information related to its associated listener and converting this information into a MIDI file using the MIDIUtil library.

The first row of the data has the string ID `U`, and is used to generate maps between IDs and client types. A `ListenerMIDI` object is preconfigured for each listener given in this row. The script iterates over the data and constructs individualized logs of when each drum beat is heard by each listener, when tempo changes occur, and when the listener is muted `ListenerMIDI`. When the script finishes iterating, each `ListenerMIDI` contains a record of the listener's sonic perspective of the session, including tempo changes and which drums were played by which performers.

The MIDIUtil library is then used to write this information to a MIDI file. In the ELBS application, only the timing and drum index are significant, so we input default values for MIDI fields other than the beat played whenever it is feasible. Drum indices 0 and 1 are differentiated with pitch C3 and

C#3. All notes written to MIDI have velocity 127. Each performer is assigned a different track in the generated MIDI files, which are labeled with their IDs. The MIDI files are generated in the same directory as the script. The naming convention for MIDI files is `listener<ID>.mid`, where `<ID>` corresponds to the listener's ID.

Table 2: Information logged to reconstruct MIDI files

Event	**String ID**	**Information Logged**
Performer drum hit	H	drum hit beat (`b`), performer ID (`ID`), drum index (`di`)
Listener position changed	L	listener ID (`ID`), cooldown start beat (`s_b`), cooldown end beat (`e_b`)
Distances between performer and each listener (changed or defined)	P	performer ID, [listener IDs], [drum delay in beats between performer and each listener]
Tempo change	T	beat of tempo change, new tempo
Client types	U	[composer IDs], [performer IDs], [listener IDs]

The `RecordedComposition` is continuously updated by the server during its lifetime. When a composer stops the session, or when a client disconnects from the session, all data in the `RecordedComposition` is written to CSV. The server invokes `generatemidis.py` and notifies clients that MIDI files are ready to be downloaded. Clients display MIDI file names next to download buttons, which send HTTP POST requests for the file.

6 Test Session

A test session of the ELBS application was carried out by the two members of my thesis committee and myself. Participants convened in a Zoom meeting, where they were presented with a slideshow introducing the project. The application was set up on a Google Cloud virtual machine, and a link to join the session was distributed, along with a password to join the ensemble. Three rounds of play were carried out with the system.

The first was a "free for all" to gauge the clarity of the instructions that appeared in the chat windows. Each user managed one client's responsibilities with a single open tab. While users were playing drums, no further instructions were given via Zoom. After the session ended, confusions were addressed and clarified. In the second round of play, all users switched roles.

In the third round, two users managed two clients' responsibilities with two open tabs. Roles were allocated based on the ease of use of the application at the time of the application; due to the test users' difficulty interpreting instructional drums as performers, only one performer was used. One participant controlled a performer and a listener, another controlled two listeners, and another controlled a composer. We did not play in a configuration with multiple performers connected to the same composer, which could have resulted in multiple interpretations of the same pattern, which would have been interesting to hear and incorporate into a composition.

Data was logged in all three rounds to be used as the starting point for the composition presented at the final defense.

Suggestions from the test session that were implemented before the final defense included making clients' nodes more identifiable on the grid, making the performer's scrolling view easier to display and interpret, clarifying the instructions given during setup, and allowing users to choose which MIDI files to download, rather than downloading them automatically.

The MIDI and CSV files are available here: https://bit.ly/4bzZFfW

7 Completed Composition

The piece is available here: https://bit.ly/4bzZGR2

After reviewing the MIDI files from the test session, I decided to base my composition off of the third small test. The performer first played regular eighth notes along with the conductor or metronome, then fell nearly silent, then played more chaotically, almost like they were smashing their keys in frustration. This cycle continued throughout the performance. From here I refer to the three listeners involved in the session as listeners 0, 4 and 5, as given by their IDs in `elbs_test3.csv`

I chose to voice the MIDI using a marimba sample library from the New Steinway Kontakt library by 8dio. This mallet sound was percussive but could be pitched around to make the piece more compelling. To make sure the parts of the piece that came from the test session were identifiable, I did not add any more overtly rhythmic elements. I generated atmospheric sounds to accompany the marimba using the plugins Synplant by Sonic Charge and the Soundmagic Spectral Suite, as well as the standalone application Argeiphontes Lyre by Akira Rabelais.

The complete piece elaborates on one cycle of playing eighth notes, then near-silence, then key-smashing. The first section of the piece (0:00 - 0:44) uses MIDI from Listeners 4 and 5. In this section, I edited the pitch of each MIDI note to evoke a call and response taking place. These listeners were 5 and 9 grid units away from each other, so they heard the same thing staggered by four beats. At 0:26 a bass drum accompanies Listener 4, playing the same notes. The second section of the piece (0:44 - 1:03) begins as the mallets fade out and only background sounds are present. The third section (1:03 - 2:08) begins as the mallets fade back in, become more chaotic up to 1:40, then recede as the piece ends. I edited the pitch of each MIDI note to play a clear chord progression, contrasting with the chaotic nature of the drumming.

When a listener was moved and only two other listeners registered MIDI during the cooldown period, I quantized some of the mallets to 16th notes to give the impression of order temporarily rising from chaos. From 1:12 to 1:18, and from 1:47 to 1:52, Listener 5 is muted after being moved, and I quantized Listener 0's MIDI to 16th notes From 1:03 to 1:08, and from 1:26 to 1:31, Listener 0 is also muted after being moved, and I quantized Listeners 0 and 4's MIDI. The MIDI velocity of the mallet instrument was slightly randomized, and its notes were altered.

The only additional edits I made to the MIDI were slightly randomizing the velocity of the mallet instruments and changing their pitch.

8 Related Work

This section goes into more detail about existing systems for simultaneous in-time collaborative music creation that accommodate remotely located participants. We describe one realistic NMP system and several non-realistic NMP systems.

8.1 Eternal Music

Eternal Music, a 2002 non-realistic NMP project by Chris Brown [95], allows up to four players at a time to modify the parameters of eight modulated sine wave oscillators, generating a drone [51]. The application implements a hub-and-spoke network topology. The server utilized is TransJam, a Java server designed to route traffic for Eternal Music and a few other applications [96]. The client application is available to download as an executable JAR file. Latency or bandwidth requirements are not given, but are likely low due to client-side audio synthesis and transmission of control data only.

Upon opening the application, users are asked to enter a username, log in, and enter a "room", or a virtual space in which a subset of users connected to the system can interact. As in ELBS, simple, colored, circular avatars' positions within the space affect the music produced. Each oscillator is depicted as a colored dot in a 2D space, and its position determines the oscillator's parameters.

Players drag and drop dots within this virtual space; when another player drops an oscillator, the dot appears to change position abruptly. The number of players accommodated by the system is not bounded, but the number of avatars present in the space is. The size of the dots compared to the space allows ample exploration without feeling restrictive. However, this means that it's not possible to attribute the movement of any one dot to any one user, which is useful for coordination and enjoyment of the performance experience. Users can join as passive listeners, functioning as an audience without a separate designation [95, 97].

This tool reduces the dimensions of salience to texture to mask latency. The music produced is not rhythmic and precise synchronization is not required to maintain coherence. The tool is theoretically capable of being used to create complex music, there is a strong precedent for music made with similar timbral limitations. Situational visibility is afforded by the clear relationship between player actions, dot positions, and oscillator qualities. Users always see a visual representation of the ensemble's music and can base their decisions on other participants' actions. Users are imparted with a clear understanding of how different dot positions affect oscillator parameters. Situational visibility could be enhanced by alerting users before another user moves a dot, as the change in position is abrupt, or by attributing dot movements to specific users.

8.2 NINJAM

NINJAM is open source software designed to enable remotely located instrumentalists to play together. It can be used as a standalone application or integrated with REAPER, a digital audio workstation, via the ReNINJAM plugin. NINJAM accommodates latency by delaying the playback of performance information such that participants play along with audio delayed by exactly one measure, which is far greater than their estimated transmission delay of one interval. Users play continuously while "intervals" of audio the length of one measure are recorded. When an interval is complete, it is compressed to OGG Vorbis, transmitted to the server, broadcast to all other participants, and played synchronously. Users receive separate streams, enabling individual mixing and remixing after a session has concluded.

NINJAM implements a hub-and-spoke network topology. A guide to assist users in configuring a custom server is available to download on the NINJAM homepage. The server itself is implemented in C and C++. The primary requirement for utilizing the server is the outbound bandwidth of the host machine. NINJAM's developers provide bandwidth suggestions based on the number of connected users. For a session with four users, they suggest 768kbps outbound bandwidth and 240kbps inbound bandwidth. Given that these requirements are met, the minimum expected latency associated with the recording, compression, transmission and reception of one interval of audio is approximately 65 milliseconds. This is much shorter than one measure. REAPER itself is the preferred NINJAM client.

NINJAM supports tempos between 20 and 240 beats per minute, as well as a "beats per interval" (BPI) between 2 and 1024. These values are maintained on the server. The maximum number of users allowed is also maintained on the server. Users can distinguish individual contributions through separate audio streams, a qualitative visual representation of the ensemble is not provided. [98]

While the playing paradigm presented by NINJAM is unconventional, it is debatable whether or not it constitutes realistic or non-realistic NMP. NINJAM is unique in that it aims to facilitate conventional instrumental performance, albeit unconventionally; the application prioritizes successful transmission of audio before playback over real-time streaming, ensuring stability at the cost of immediacy. This effectively slows the rate of joint interaction down such that it resembles turn-taking rather than continuous listening and observing [48]. The convention of an interval / measure is ingrained in the application. They exist not only to coordinate the ensemble, but to establish when and how audio is transmitted after it is produced.

8.3 JackTrip / Virtual Studio

JackTrip is a popular system that supports bidirectional audio streaming with any number of channels. JackTrip utilizes Jack, a sound server API that provides real-time low-latency connections for audio and MIDI data between applications. [99] JackTrip can be utilized on its own by installing JACK, utilizing an associated GUI program to run it, then running JackTrip itself from the command line. [100].

JackTrip has been incorporated into other networked performances [101], but the product that receives the most emphasis on the JackTrip site is JackTrip Virtual Studio, a realistic NMP system that streams audio and video between instrumentalists participating in "sessions". Virtual Studio utilizes a hub-and-spoke topology. Users can rent access to virtual machines that host JackTrip servers, or "cloud audio studios", whose audio quality and bandwidth requirements vary by pay tier. The lowest tier requires 300-500 kbps inbound and outbound bandwidth.

The developers cite the quality of one's Internet connection, audio interface, and gear as factors that can affect latency. Screenshots found in the documentation indicate that Virtual Studio can achieve latency on the order of milliseconds.

While the potential applications of JackTrip itself are not limited to emulating real-time performance, the aim of Virtual Studio is to create "the sense of proximity that sparks real creativity, the experience of being together in the same room" [59], which refers broadly to well-established qualities of liveness and performance, discussed in depth in Section 2.3.

Virtual Studio facilitates musical activity between users with traditional acoustic instruments/ Low-latency video connections serve users equally as performers and audience members; they provide visual feedback that assists musical coordination, and they are an additional means for evaluating each other's skill.

8.4 Global Net Orchestra (GNO)

The Global Net Orchestra (GNO) is a networked performance tool designed to accommodate remotely located participants. GNO implements a "hub-and-spoke" network topology, in which all communication between network endpoints is facilitated by a server, to which all clients are connected. Players locate the server by accessing a custom domain through a web browser. Before a performance, GNO users upload audio samples to this website, where they are integrated with the standalone GNO client application. The application is then made available to download.

GNO employs a "follower-leader" musical dynamic between conductor and performer user classes. The performer UI consists of a scrolling score for keyboard, a chat box, and mix controls. Performers follow the scrolling score as it is generated by conductors and displayed. Their keyboard input is transmitted to the server as control data (MIDI note / velocity pairs) to minimize bandwidth usage. The server broadcasts this data to clients and the performance is reconstructed and played back locally.

Conductors communicate with performers through a chat box and occasional spoken "talk-back" instructions. Conductors require a certain level of situational visibility to make informed decisions, but other users require less information about the state of the ensemble.

Most messages are transmitted using a custom best-effort protocol based on UDP. Data whose loss could significantly interrupt performance, such as message start times, are sent using ZeroMQ, which is based on TCP. Given B bytes of overhead for each message transmitted every U seconds and P connected players, $P * B/U$ incoming bandwidth and $P * (B * P)/U$ outgoing bandwidth are required. Performers only transmit local state information to the server, so upload bandwidth is much smaller.

GNO uses multiple musical strategies to address latency issues. The scrolling score is transmitted best-effort, without latency compensation. Dimensions of salience are reduced by encouraging more gestural, less exact responses to scrolling scores. Expressive shapes are displayed in the performer UI to guide different styles of playing. Thus, dynamics and "texture" can be prioritized over rhythmic or melodic synchronization.

Additionally, GNO delays entire musical cycles to ensure beat alignment despite latency, similar to the approach outlined in [60]. This does not purposefully result in bi-located patterns, the result is more "semi-synchronous" as described by [50].

GNO is motivated by a desire to explore the constraints associated with remote networked musical performance that suggest new musical directions, and to allow the audience to appreciate the overcoming of obstacles [47]. Both motivations could be thought of as extension of Di Scipio's concept of performance as something that emerges from a self-observing network of users with the system / "site". The latter goal reflects Carlson's idea that virtuosity, or some degree of ability to overcome obstacles, is fundamental for performance. Additionally, the digital instrument presented closely resembles a conventional instrument known for its high virtuosity standards (piano / keyboard).

8.5 Endlesss

Endlesss [sic] is a non-realistic NMP system that began operation in 2020 [102]. It is a loop-based music making platform with a DAW-like interface that allows users to create loops given a set of tools agreed upon by a set of performers, send the loops to other users in the same session, and collaboratively build up a composition made up of combinations of loops in real time. Users can create loop-based music that exhibits characteristics of popular electronic music genres, such as synthesizer and drum machine sounds. Users can switch between playing patterns, experimenting with combinations of submitted patterns, and submitting combinations to move the piece forward. The pattern-based method for contributing compositional elements meant that users did not have to play notes in real time to contribute to a collective result.

Endlesss can be used for both composition support and performance. While playing, an evolving piece of music composed of the sequences that have been created is gradually generated. At the end of a session, audio stems can be downloaded, but not MIDI. This makes it difficult for material produced during jam sessions to be repurposed by musicians who are interested in networked performance but whose personal taste may not lean towards the timbres available via the application.

The Endlesss client is available to download as a standalone application or an iPhone app. No server setup is required for users.

Endlesss was discontinued in June 2024, and is no longer available for use.

8.6 Global Drum Circle (GDC)

Since May 2022, I have been working with Professor Roger Dannenberg on a distributed web-based music system, which we have been calling the Global Drum Circle Project (GDC). The goal of this application is to enable drum circle-like musical performances across the Internet in quasi-real time by delaying all users' contributions by an estimated worst-case end-to-end latency. Sections 5.1.1, 5.1.2, 5.2, and 5.5.3 through 5.5.7 refer to implementation details that are unchanged or only slightly adapted from GDC.

As in ELBS, the client application is browser-based. Each user is expected to operate a single client, the application does not prevent them from opening and operating multiple clients in different browser tabs.

The upper bound for end-to-end communication set in GDC is `MAX_NET_DELAY = 3.0` seconds. This estimate was based on the estimated transmission time of $\approx$ 1kB of data. To aid in coordinating users in the ensemble, the application implements the between time and "beats" according to an established tempo, as well as functionality for scheduling events to occur at future beats.

As in GNO, instructions for playing in GDC are delivered by one class of clients to another. "Players" play along to drums played by "Conductors", which have been precisely delayed by eight beats / one measure. Conductors are upstream from players; players are downstream from conductors.

During GDC performances, musical interaction takes place via the following series of steps: A Conductor plays a drum at beat b, and a drum hit message containing b is transmitted to the server. The server calculates the beat at which the action associated with this message (playing a drum sound) takes place, $b + 8$, and broadcasts a drum hit message with this value to all clients. Upon receiving the message, players schedule drum playback to take place at this time. Because the duration of eight beats is greater than or equal to `MAX_NET_DELAY`, it can be safely assumed that all players have received and scheduled the drum hit message before its execution time.

8.6.1 User Interfaces and Play

The GDC UI consists of a chat box, volume sliders, beat lights / metronome, tempo slider, and start / stop indicator. The elements were slightly modified and incorporated into the ELBS application; the other UI elements discussed in this section are unique to GDC.

All users can choose from a selection of drums to play from the "My Drums" section, shown in Figure 29. As in ELBS, users play different drums by pressing keys on the left or right side of their keyboard.

To encourage the performance of more complex and coherent music with the system, musical games called "modes" were incorporated into the system. Modes consist of a looping series of "steps" lasting eight beats. Mode steps consist of the written instructions delivered to conductors and players, and whether or not their output is muted to maintain coherency. For example, a "Follow the Leader" mode coordinates and maintains a call-and-response drumming pattern between conductors and players. Modes are scheduled to start at the first measure occurring `MAX_NET_DELAY` after they are triggered.

Instructions for each step of a mode are displayed onscreen. Mode and tempo changes are scheduled and executed via the same process as drum hits, by which it is ensured that performance information is delivered to all users before it is put into practice. One of the modes currently implemented is a variation of call and response, where a Player plays along with drums played by a conductor as specified in the mode instructions.

During a GDC session, times at which drum hits, users entering and leaving, etc. are logged in a dedicated class. A "Save" button allows conductors to export session data to a CSV file for possible reconstruction.

On the bottom half of the interface, a continuously updated score displays users' drum hits quantized to eighth notes as filled squares on a single row of a grid. A histogram next to the score shows the

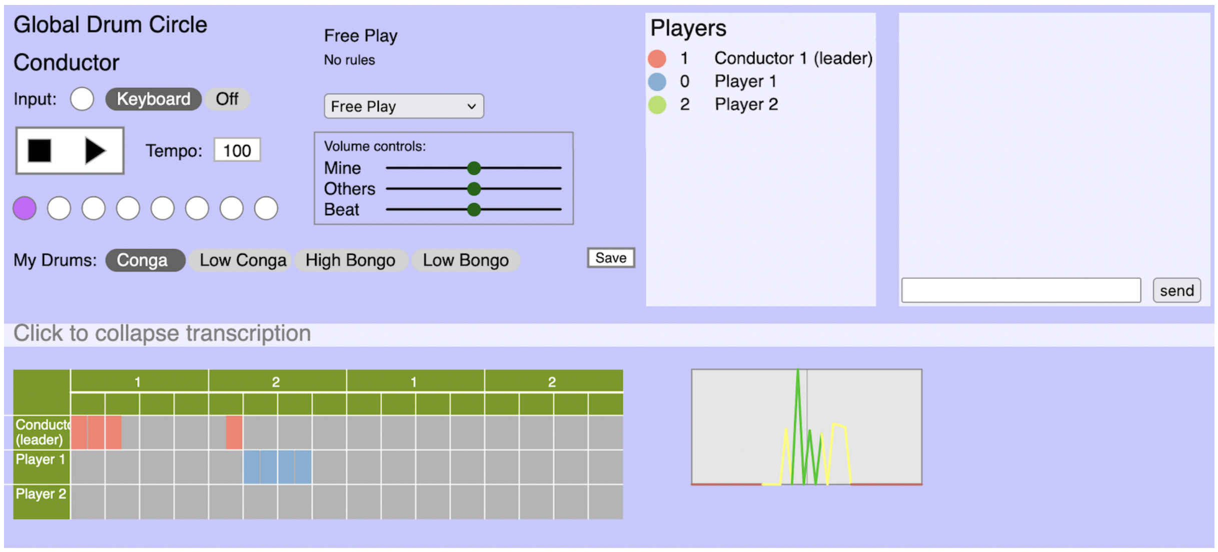


Figure 29: The conductor interface, in “Free Play” mode.

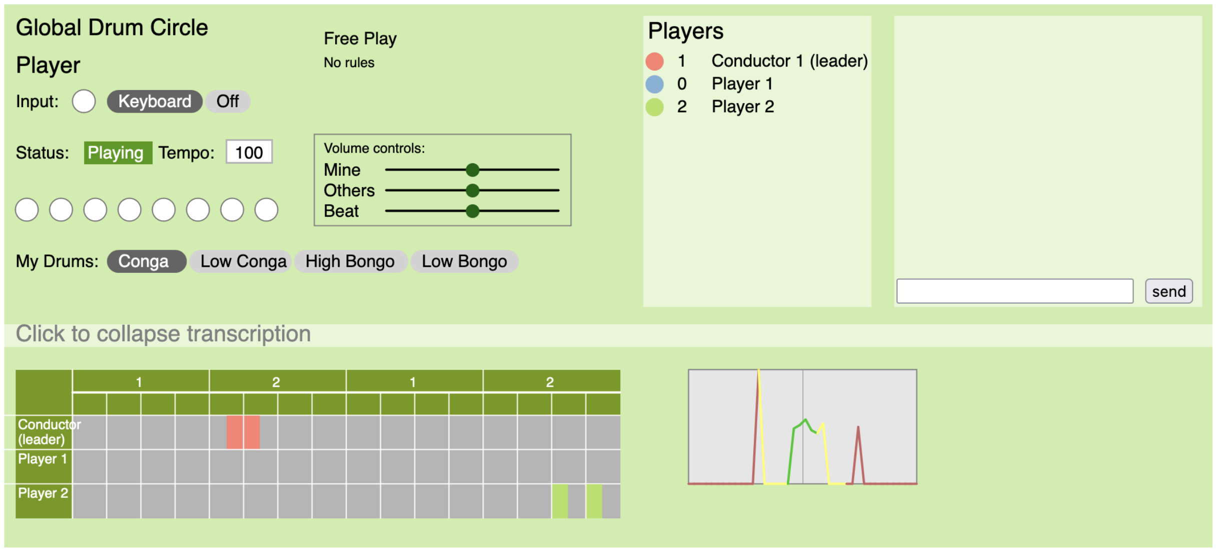


Figure 30: The player interface, in “Free Play” mode.

distribution of differences of drum hits and their nearest beat; this allows users to assess each other’s timing, but the identity of each user whose activity is reflected in the histogram is not specified in the histogram itself.

In GDC, the convention of a “measure” is less ingrained in the application’s implementation than in NINJAM or Endlesss. Measures exist to coordinate the ensemble and are enacted via modes and beat lights, but control information corresponding to individual musical actions is transmitted as it is produced.

9 Conclusions and Future Work

Asynchronous or “out of time” musical collaboration via file sharing is commonplace. However, there are very few compelling systems for quasi-real time collaboration, other than emulations of being present in the same physical space in real time, an experience that is not possible to replicate perfectly due to communication latency. Some degree of latency is inevitable in networked systems, and is usually considered a problem whose effects on emulating real-time communication should be mitigated. Less explored is the idea of embracing latency as a feature of a networked application, and developing composition and performance techniques that incorporate delay. This report documents the design, implementation and testing of a prototype experimental latency-based system (ELBS) for networked musical performance, which creatively incorporates an otherwise disruptive amount of latency as a feature of performance and an opportunity for novel musical experiences.

I reviewed existing networked performance tools, as well as works in other fields that were guided by design principles I found compelling or constraints I thought were worth considering. Research on performance theory also informed the system's design, and helped me narrow down the scope of what I considered related work. I first addressed what was required to facilitate musical performance in general, then adapted the methods by which those requirements were satisfied in traditional performance to a setting with certain technical restrictions. The methods by which limitations were addressed were motivated by their novelty.

To show that the prototype is functional and nominally operable, I held a test session with my thesis committee in which we carried out three short performances using the system. In between this test and my final defense, I used the record of the third session as the basis for a piece of music, created asynchronously. I also made some changes to the application itself, incorporating feedback given at the test session, the thesis proposal, and several other informal test sessions with groups of electronic musicians. The version of the prototype described in this report incorporates this feedback. It is operable and ready for further testing, which would constitute valuable future work.

While latency is a common concern with networked quasi-real time performance systems, there exist relatively few existing quasi-real time performance systems that acknowledge and incorporate it into their design. Similarly, while joint action and situational visibility are often considered in installations and co-located electronic music performances, it has been acknowledged that little research has been carried out that applies these ideas to the design of NMP tools [30]. During my literature review, I was surprised by how few networked tools allowed users to refine material produced during a session, and how few tools embraced artists' desire for some degree of instability in their work [77].

One of the goals of this project was to "sonify and artfully represent latency". In retrospect, the degree to which latency was "incorporated" into the design is debatable. The process described for estimating latency between users in 5.3.3 is crude and not thorough. A future implementation of this prototype might seek to estimate round trip time by averaging more measurements spread out over a longer period of time [1, Chapter 9.14], or using an adaptive algorithm like TCP's [1, Chapter 13.16, 13.17].

Future research could focus on exploring the idea of prioritized integrative attention further. If prioritizing one's own responsibilities (following upstream users) were further emphasized in the implementation over awareness of others' (indirectly related users) actions, it could justify altering distance distortions in 5.4.2 to allow visual representations of non-essential contributions to arrive late. This could make the application more interesting to use or completely inoperable. It would be novel to decrease the added latency to a point at which play would become challenging, but not impossible. A survey of amateur electronic music producers, conducted as research for the development of a machine learning-assisted tool to generate rhythmic patterns, discovered a preference for tools that work within stylistic constraints and "allowed accidents or unexpected results when pushed beyond their declared limits" [76]. Loosening certain constraints such that the inherent instability of a remote connection could creep further into the work produced could prove interesting for consumers.

Future work that is necessary, even given a fundamentally unchanged version of the system, is improving the graphics and general user experience. While the p5js library was ideal for prototyping, a more immersive user experience could be facilitated given a graphics library with deeper functionality, such as WebGL [103], which can render more complex visualizations, such as particle systems, more effectively.

The worst-case transmission time is only rarely required to transmit a data frame. Assuming `MAX_NET_DELAY` cannot be reduced, joint action could take place faster if certain control information were permitted to arrive late. For example, in the ELBS prototype, propagating and traversing hits are displayed several beats before they need to be played back. Users display drum hits based on their play time, such that drum hit messages that arrive late appear part of the way through their path and continue their normal visual progression from that point. Play is not significantly disrupted if drum hit messages occasionally arrive late. There is precedent in prior work ([61], 8.4) for using a UDP-based protocol to transmit data whose late arrival will not significantly interrupt play, and a

TCP-based protocol to transmit vital information. An acceptable-case transmission time can then be established for less vital data. This time is both interaction-specific and application-specific. A general design principle that follows is that as the acceptable-case transmission-action time for an interaction is lowered, the impact of occasional delays on musical coherence and user experience should also decrease.

Another worthwhile change to the system would be to implement all user classes in the same client. The ELBS prototype allows one user to open multiple clients in different tabs, whose clocks are synchronized independently from one another. Temporary synchronization issues in individual clients can cause noticeable timing discrepancies between tabs. If multiple roles could be accessed from the same client, not only would the clock synchronization procedure only need to be carried out once, but temporary issues with the local time would be less noticeable.

One of the goals of this project was to represent all users' musical activity, regardless of its relevance to rudimentary musical responsibility. Musical information from all users is displayed in equal detail. A common complaint during testing was the fact that the grid felt cluttered, even when less than ten clients were connected. This could be addressed in future versions of the prototype by adjusting how activity is visualized. For example, musical information from clients not directly upstream or downstream from any given client could be rendered in less detail, or in a less distracting manner, in the latter. Clients could see detailed visuals relating to other users in their chain of responsibility, while accuracy histograms like the one implemented in GDC (as described in Section 8.6.1) could be displayed above other avatars. All musical activity would still be represented, but to different extents based on its relevance to rudimentary musical responsibility. Alternatively, clients could zoom into specific sections of the grid to better focus on tasks involving certain clients. The full ensemble would remain visible and all musical activity would be displayed, but users could concentrate on selected sections. Both of these strategies could make scaling the system up more viable, but require major changes to the prototype's music and display systems.

In conclusion, this work set out to explore new organizing principles for collaborative, non-realistic and displaced networked musical performance. To offset the well-documented and significant effects of communication delay and visual and auditory isolation on musical ensembles, I implemented, tested and extended several existing but underutilized strategies for improving musical coordination, complexity, and user experience. I shifted clients' time systems to hide transmission delay and create the illusion of immediate interaction with the ensemble. I represented delayed musical input as visual artifacts moving along paths on a grid-quantized display, thus encouraging rhythmic coherence, enforcing the beat alignment of overlapping contributions, and sonifying network latency. I developed a playing paradigm that introduced cascades of rudimentary musical responsibility and created bi-located rhythmic patterns, allowing players to combine basic actions to produce sophisticated music. To make the prototype more marketable and to encourage wide use, I implemented the transmission delay-disguising mechanism to operate in a range of "non-ideal" scenarios. I also implemented functionality for generating and distributing flexible records of the performance after its completion, which could be incorporated into compositions developed out of time. I hope that this work inspires others to consider how musical agency can be distributed in performing ensembles to explore novel social dynamics and create complex music, and how latency can be treated as an enabling constraint rather than a problem to be eliminated.

References

[1] D. E. Comer, *Internetworking with TCP/IP Vol I: Principles, Protocols, and Architecture*, 4th ed. Prentice Hall, 2004, vol. 1.

[2] L. Peterson and B. Davie, *Computer Networks: A Systems Approach*, 6th ed., ser. ISSN. Elsevier Science, 2021. [Online]. Available: https://book.systemsapproach.org/

[3] E. W. Eddy, "Transmission control protocol (tcp)," RFC Editor, RFC 9293, 8 2022. [Online]. Available: https://www.rfc-editor.org/info/rfc9293

[4] V. Paxson, M. Allman, J. Chu, and M. Sargent, "Computing tcp's retransmission timer," RFC Editor, RFC 6298, 6 2011. [Online]. Available: https://www.rfc-editor.org/info/rfc6298

[5] J. Cao, W. S. Cleveland, and D. X. Sun, "Bandwidth estimation for best-effort internet traffic," *Statistical Science*, vol. 19, no. 3, pp. 518–543, 2004. [Online]. Available: http://www.jstor.org/stable/4144400

[6] I. MOS Technology, *KIM-1 Microcomputer Module User Manual*, MOS Technology, Inc. [Online]. Available: https://www.kim-1.com/docs/usrman.htm#61

[7] M. Mathews, *The Technology of Computer Music*, 2nd ed. Massachusetts Institute of Technology, 1974.

[8] C. Brown and J. Bischoff, *Computer Network Music Bands: A History Of The League Of Automatic Music Composers And The Hub.* Cambridge, Massachusetts: MIT Press, 2005, pp. 372–392, in *At a Distance: Precursors to Art and Activism on the Internet*, Annmarie Chandler and Norie Neumark (ed.).

[9] L. Gabrielli and S. Squartini, *Wireless Networked Music Performance*, 1st ed., ser. SpringerBriefs in Electrical and Computer Engineering. Springer Singapore, 2016.

[10] J. Bischoff, R. Gold, and J. Horton, "Music for an interactive network of microcomputers," *Computer Music Journal*, vol. 2, no. 3, pp. 24–29, 1978. [Online]. Available: http://www.jstor.org/stable/3679453

[11] R. Fencott and N. Bryan-Kinns, "Hey man, you're invading my personal space ! privacy and awareness in collaborative music," in *Proceedings of the International Conference on New Interfaces for Musical Expression*, Sydney, Australia, 2010, pp. 198–203. [Online]. Available: http://www.nime.org/proceedings/2010/nime2010_198.pdf

[12] G. Weinberg, "Interconnected musical networks: Toward a theoretical framework," *Computer Music Journal*, vol. 29, no. 2, pp. 23–39, 2005. [Online]. Available: http://www.jstor.org/stable/3681711

[13] M. Iorwerth, *Networked Music Performance: Theory and Applications*, 1st ed. Focal Press, 2023.

[14] J. Lazzaro and J. Wawrzynek, "A case for network musical performance," in *Network and Operating System Support for Digital Audio and Video, 11th International Workshop, NOSSDAV 2001, Port Jefferson, NY, USA, June 25-26, 2001, Proceedings.* ACM, 2001, pp. 157–166. [Online]. Available: http://doi.acm.org/10.1145/378344.378367

[15] M. Fritsch and S. Strötgen, "Relatively live: How to identify live music performances," *Music and the Moving Image*, vol. 5, no. 1, pp. 47–66, 2012. [Online]. Available: http://www.jstor.org/stable/10.5406/musimoviimag.5.1.0047

[16] P. Auslander, *Liveness: Performance in a Mediatized Culture*, 3rd ed. Routledge, 2022.

[17] M. Leman, P.-J. Maes, L. Nijs, and E. V. Dyck, *What is Embodied Music Cognition?*, 1st ed., ser. Springer Handbooks. Springer International Publishing, 2018, pp. 747–760, in *Springer Handbook of Systematic Musicology*, Bader, R. (ed.). [Online]. Available: https://link.springer.com/book/10.1007/978-3-662-55004-5

[18] S. Wurtzler, *"She Sang Live, But The Microphone Was Turned Off:" The Live, the Recorded and the Subject of Representation.* Routledge, 1992, pp. 87–103, in *Sound Theory, Sound Practice*, Rick Altman (ed.).

[19] P. Sanden, *Liveness in Modern Music: Musicians, Technology, and the Perception of Performance.* Routledge, 01 2013.

[20] J. W. Croft, "Theses on liveness," *Organised Sound*, vol. 12, pp. 59 – 66, 2007.

[21] M. Carlson, *Performance: A Critical Introduction (2nd ed.).* Routledge, 09 2004.

[22] T. Blaine and S. S. Fels, "Contexts of collaborative musical experiences," in *Proceedings of the International Conference on New Interfaces for Musical Expression*, Montreal, Canada, 2003, pp. 129–134. [Online]. Available: http://www.nime.org/proceedings/2003/nime2003_129.pdf

[23] A. D. Scipio, *Thinking Liveness in Performance with Live Electronics: The Need for an Eco-systemic Notion of Agency.* Leuven University Press, 2021, pp. 171–194, in *Sound Work: Composition as Critical Technical Practice*, Jonathan Impett (ed.). [Online]. Available: http://www.jstor.org/stable/j.ctv1ccbg96

[24] A. D. Scipio and D. Sanfilippo, "Defining ecosystemic agency in live performance: The machine milieu project as practice-based research," in *2019: array. the journal of the ICMA - Topic "Agency"*, October 2020. [Online]. Available: https://journals.qucosa.de/array/issue/view/216

[25] J. Prince, W. Jon Thompson, and M. Schmuckler, "Pitch and time, tonality and meter: how do musical dimensions combine?" *Journal of experimental psychology. Human perception and performance*, vol. 35, pp. 1598–617, 10 2009.

[26] P. Keller, "Joint action in music performance," *Enacting intersubjectivity: A cognitive and social perspective to the study of interactions, 205-221 (2008)*, vol. 10, 01 2008.

[27] P. Janata, "Neurophysiological mechanisms underlying auditory image formation in music," in *Elements of Musical Imagery*, H. J. R. I. Godøy, Ed. Lisse: Swets & Zeitlinger Publishers, 2001.

[28] F. Schroeder and P. Rebelo, "Sounding the network: The body as disturbant," *Leonardo Electronic Almanac*, vol. Vol 16 Issue 4 – 5, p. 1, May 2009, franziska Schroeder is a saxophonist and theorist who writes on the network, the body and performance.

[29] A. Williamon and J. W. Davidson, "Exploring co-performer communication," *Musicae Scientiae*, vol. 6, no. 1, pp. 53–72, 2002. [Online]. Available: https://doi.org/10.1177/102986490200600103

[30] M. Iorwerth and D. Knox, "Playing together, apart: Musicians' experiences of physical separation in a classical recording session," *Music Perception: An Interdisciplinary Journal*, vol. 36, no. 3, pp. pp. 289–299, 2019. [Online]. Available: https://www.jstor.org/stable/26586173

[31] S. Delle Monache, L. Comanducci, M. Buccoli, M. Zanoni, A. Sarti, E. Pietrocola, F. Berbenni, and G. Cospito, "A presence- and performance-driven framework to investigate interactive networked music learning scenarios," *Wireless Communications and Mobile Computing*, vol. 2019, no. 1, p. 4593853, 2019. [Online]. Available: https://onlinelibrary.wiley.com/doi/abs/10.1155/2019/4593853

[32] N. Sebanz, H. Bekkering, and G. Knoblich, "Joint action: bodies and minds moving together," *Trends in Cognitive Sciences*, vol. 10, pp. 70–76, 2006.

[33] F. Berthaut and L. Dahl, "The Effect of Visualisation Level and Situational Visibility in Co-located Digital Musical Ensembles," in *NIME 2022*, jun 16 2022, https://nime.pubpub.org/pub/n20kw7if.

[34] C. Bartlette, D. Headlam, M. Bocko, and G. Velikic, "Effect of network latency on interactive musical performance," *Music Perception: An Interdisciplinary Journal*, vol. 24, no. 1, pp. 49–62, 2006. [Online]. Available: http://www.jstor.org/stable/10.1525/mp.2006.24.1.49

[35] T. R. Merritt, W. Kow, C. Ng, K. McGee, and L. L. Wyse, “Who makes what sound?: supporting real-time musical improvisations of electroacoustic ensembles,” in *Australasian Computer-Human Interaction Conference*, 2010.

[36] G. Luck and P. Toiviainen, “Ensemble musicians’ synchronization with conductors’ gestures: An automated feature-extraction analysis,” *Music Perception: An Interdisciplinary Journal*, vol. 24, no. 2, pp. 189–200, 2006. [Online]. Available: http://www.jstor.org/stable/10.1525/mp.2006.24.2.189

[37] J. Zorn, “John zorn’s cobra: Live at the knitting factory,” Knitting Factory Works KFW 124, compact disc, New York, 1997.

[38] J. Walshe, “meanwhile, back at the ranch,” Milker Corporation, Dublin, 2005.

[39] J. O. Borchers, “A pattern approach to interaction design,” *AI & SOCIETY*, vol. 15, pp. 359–376, 2001.

[40] G. D’Arcangelo, “Creating contexts of creativity : Musical composition with modular components,” in *Proceedings of the International Conference on New Interfaces for Musical Expression*, Seattle, WA, 2001, pp. 42–45. [Online]. Available: http://www.nime.org/proceedings/2001/nime2001_042.pdf

[41] J. Borchers and M. Muhlhauser, “Design patterns for interactive musical systems,” *IEEE MultiMedia*, vol. 5, no. 3, pp. 36–46, 1998.

[42] D. Wessel and M. Wright, “Problems and prospects for intimate musical control of computers,” *Computer Music Journal*, vol. 26, no. 3, pp. 11–22, 2002. [Online]. Available: http://www.jstor.org/stable/3681975

[43] S. Jordà, “Digital instruments and players: Part i – efficiency and apprenticeship,” in *Proceedings of the International Conference on New Interfaces for Musical Expression*, Hamamatsu, Japan, 2004, pp. 59–63. [Online]. Available: http://www.nime.org/proceedings/2004/nime2004_059.pdf

[44] ——, “Digital instruments and players: Part ii – diversity, freedom and control,” in *Proceedings of the 2004 International Computer Music Conference*, Miami, USA, 2004, pp. 706–709.

[45] D. Robson, “Play! : Sound toys for the non musical,” in *Proceedings of the International Conference on New Interfaces for Musical Expression*, Seattle, WA, 2001, pp. 51–53. [Online]. Available: http://www.nime.org/proceedings/2001/nime2001_051.pdf

[46] C. Chafe, M. Gurevich, G. Leslie, and S. Tyan, “Effect of time delay on ensemble accuracy,” in *Proc. 2004 Intl. Soc. Musical Acoustics*, 2004.

[47] R. Dannenberg and T. Neuendorffer, “Scaling up live internet performance with the global net orchestra,” in *International Conference on Mathematics and Computing*, 2000.

[48] C. Chafe, J. P. Cáceres, and M. Gurevich, “Effect of temporal separation on synchronization in rhythmic performance,” *Perception*, vol. 39, pp. 982 – 992, 2010.

[49] S. Delle Monache, M. Buccoli, L. Comanducci, A. Sarti, G. Cospito, E. Pietrocola, and F. Berbenni, “Time is not on my side: Network latency, presence and performance in remote music interaction,” in *Machine Sounds, Sound Machines - Proc. of the XXII CIM Colloquium on Music Informatics*, 02 2019.

[50] N. Bryan-Kinns, “Annotating distributed scores for mutual engagement in daisyphone and beyond,” *Leonardo Music Journal*, vol. 21, pp. 51–55, 2011. [Online]. Available: http://www.jstor.org/stable/41416823

[51] Álvaro Barbosa, “Displaced soundscapes: A survey of network systems for music and sonic art creation,” *Leonardo Music Journal*, vol. 13, pp. 53–59, 2003.

[52] C. Rottondi, C. Chafe, C. Allocchio, and A. Sarti, “An overview on networked music performance technologies,” *IEEE Access*, vol. 4, pp. 8823–8843, 2016.

[53] R. Mills, "Flight of the sea swallow: A crossreality telematic performance," *Leonardo*, vol. 49, no. 1, pp. 68–69, 2016. [Online]. Available: http://www.jstor.org/stable/43834325

[54] L. Comanducci, *Intelligent Networked Music Performance Experiences.* Cham: Springer International Publishing, 2023, pp. 119–130, in *Special Topics in Information Technology*, Riva, C. G. (ed.). [Online]. Available: https://doi.org/10.1007/978-3-031-15374-7_10

[55] G. Föllmer, "Electronic, aesthetic and social factors in net music," *Organised Sound*, vol. 10, no. 3, p. 185–192, 2005.

[56] C. Alexandraki and D. Akoumanakis, "Exploring new perspectives in network music performance: The diamouses framework," *Computer Music Journal*, vol. 34, no. 2, pp. 66–83, 2010. [Online]. Available: http://www.jstor.org/stable/40731283

[57] C. Drioli and N. Buso, "Networked performances and natural interaction via lola: Low latency high quality a/v streaming system," *Lecture Notes in Computer Science*, vol. 7990, pp. 240–250, January 2013.

[58] P. Holub, L. Matyska, M. Liška, L. Hejtmánek, J. Denemark, T. Rebok, A. Hutanu, R. Paruchuri, J. Radil, and E. Hladká, "High-definition multimedia for multiparty low-latency interactive communication," *Future Generation Computer Systems*, vol. 22, no. 8, pp. 856–861, 2006. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0167739X06000380

[59] "Jacktrip virtual studio - product page," JackTrip Labs, visited 2023-07-01. [Online]. Available: https://www.jacktrip.com/product

[60] M. Goto, R. Neyama, and Y. Muraoka, "Rmcp: Remote music control protocol - design and applications," in *International Conference on Mathematics and Computing*, 1997, p. 446–449.

[61] Y. Obu, T. Kato, and T. Yonekura, "M.a.s.: A protocol for a musical session in a sound field where synchronization between musical notes is not guaranteed," in *International Conference on Mathematics and Computing*, 2003. [Online]. Available: https://quod.lib.umich.edu/cgi/p/pod/dod-idx/m-as-a-protocol-for-a-musical-session-in-a-sound-field-where.pdf?c=icmc;idno=bbp2372.2003.016;format=pdf

[62] S. R. Klemmer, B. Hartmann, and L. Takayama, "How bodies matter: five themes for interaction design," in *Symposium on Designing Interactive Systems*, 2006.

[63] D. Trueman, "Why a laptop orchestra?" *Organised Sound*, vol. 12, no. 2, p. 171–179, 2007.

[64] S. Smallwood, D. Trueman, P. R. Cook, and G. Wang, "Composing for laptop orchestra," *Computer Music Journal*, vol. 32, no. 1, pp. 9–25, 2008. [Online]. Available: http://www.jstor.org/stable/40072661

[65] S. Serafin, C. Erkut, J. Kojs, N. C. Nilsson, and R. Nordahl, "Virtual reality musical instruments: State of the art, design principles, and future directions," *Computer Music Journal*, vol. 40, no. 3, pp. 22–40, 2016. [Online]. Available: https://www.jstor.org/stable/26777007

[66] G. Novembre and P. E. Keller, *Music and Action*, 1st ed., ser. Springer Handbooks. Springer International Publishing, 2018, pp. 523–537, in *Springer Handbook of Systematic Musicology*, Bader, R. (ed.). [Online]. Available: https://link.springer.com/book/10.1007/978-3-662-55004-5

[67] Sponge San Francisco, FoAM, S. X. Wei, C. Salter, L. Farabo, M. Kuzmanovic, N. Gaffney, E. Kusaite, C. Bohner-Vloet, S. Auinger, J. Ryan, O. Cakmakci, K. van Laerhoven, E. Fonteyne, and D. Tonnesen, "T-garden," 2001. [Online]. Available: https://foam.net/publications/t-garden/

[68] N. Schnell and S. Robaszkiewicz, "Soundworks – a playground for artists and developers to create collaborative mobile web performances," in *Proceedings of the Web Audio Conference (WAC'15)*, 2015, https://hal.science/hal-01580797.

[69] G. Weinberg, R. Aimi, and K. Jennings, “The beatbug network - a rhythmic system for interdependent group collaboration,” in *Proceedings of the International Conference on New Interfaces for Musical Expression*, Dublin, Ireland, 2002, pp. 186–191. [Online]. Available: http://www.nime.org/proceedings/2002/nime2002_186.pdf

[70] J. Steurer, “Defining virtual reality: Dimensions determining telepresence,” *Journal of Communication*, vol. 42, pp. 73–93, 1992.

[71] M. Minsky, “Telepresence,” *OMNI Magazine*, June 1980.

[72] B. Loveridge, “Networked music performance in virtual reality: Current perspectives,” *Journal of Network Music and Arts*, vol. 2, 2020, issue 1. [Online]. Available: https://commons.library.stonybrook.edu/jonma/vol2/iss1/2

[73] J. A. Paradiso, K.-y. Hsiao, and A. Benbasat, “Tangible music interfaces using passive magnetic tags,” in *Proceedings of the International Conference on New Interfaces for Musical Expression*, Seattle, WA, 2001, pp. 30–33. [Online]. Available: http://www.nime.org/proceedings/2001/nime2001_030.pdf

[74] L. S. Pardue and J. A. Paradiso, “Musical navigatrics: New musical interactions with passive magnetic tags,” in *Proceedings of the International Conference on New Interfaces for Musical Expression*, Dublin, Ireland, 2002, pp. 145–147. [Online]. Available: http://www.nime.org/proceedings/2002/nime2002_145.pdf

[75] J.-P. Cáceres and A. B. Renaud, “Playing the network: the use of time delays as musical devices,” in *International Computer Music Conference*, August 2008, pp. 244–250.

[76] F. Schroeder and P. Rebelo, “Addressing the network: Performative strategies for playing apart,” in *Medium of Output: Proceedings of the 2007 International Computer Music Conference; International Computer Music Conference (ICMC 2007) ; Conference date: 01-08-2007 Through 01-08-2007*, Aug. 2007, pp. 133–140.

[77] S. Jordà, D. Gómez-Marín, Ángel Faraldo, and P. Herrera, “Drumming with style: From user needs to a working prototype,” in *Proceedings of the International Conference on New Interfaces for Musical Expression*. Brisbane, Australia: Queensland Conservatorium Griffith University, 2016, pp. 365–370. [Online]. Available: http://www.nime.org/proceedings/2016/nime2016_paper0071.pdf

[78] G. Niemeyer, “Ping: Poetic charge and technical implementation,” *Leonardo*, vol. 38, no. 4, pp. 312–300, 2005. [Online]. Available: http://www.jstor.org/stable/20206073

[79] C. Chafe, S. Wilson, and D. Walling, “Physical model synthesis with application to internet acoustics,” in *2002 IEEE International Conference on Acoustics, Speech, and Signal Processing*, vol. 4, 2002, pp. IV–4056–IV–4059.

[80] K. Karplus and A. Strong, “Digital synthesis of plucked-string and drum timbres,” *Computer Music Journal*, vol. 7, no. 2, pp. 43–55, 1983. [Online]. Available: http://www.jstor.org/stable/3680062

[81] V. Bauer and T. Bouchara, “First steps towards augmented reality interactive electronic music production,” *2021 IEEE Conference on Virtual Reality and 3D User Interfaces Abstracts and Workshops (VRW)*, pp. 90–93, 2021.

[82] Y. Wang, M. Xi, M. Adcock, and C. P. Martin, “Mobility, space and sound activate expressive musical experience in augmented reality,” in *Proceedings of the International Conference on New Interfaces for Musical Expression*, M. Ortiz and A. Marquez-Borbon, Eds., Mexico City, Mexico, May 2023, pp. 128–133. [Online]. Available: http://nime.org/proceedings/2023/nime2023_17.pdf

[83] MozDevNet, “Websocket - web apis: Mdn.” [Online]. Available: https://developer.mozilla.org/en-US/docs/Web/API/WebSocket

[84] J. Walnes, “Webbit: A java event based websocket and http server,” 2022. [Online]. Available: https://github.com/webbit/webbit

[85] Processing Foundation, “p5.js.” [Online]. Available: https://github.com/processing/p5.js

[86] Facebook, “React documentation.” [Online]. Available: https://react.dev/learn

[87] Google, “Google cloud - network intelligence center - guides,” https://cloud.google.com/network-intelligence-center/docs/performance-dashboard/how-to/view-google-cloud-latency, visited multiple times between 2024-07-05 and 2024-08-29.

[88] R. Dannenberg, “O2 2.0.” [Online]. Available: https://rbdannenberg.github.io/o2/html/index.html

[89] Google, “Google cloud - compute engine documentation - network bandwidth,” https://cloud.google.com/compute/docs/network-bandwidth, visited multiple times between 2024-07-05 and 2024-08-29.

[90] C. Chafe and R. Leistikow, “Levels of temporal resolution in sonification of network performance,” in *Proceedings of the 2001 International Conference on Auditory Display*, 2001. [Online]. Available: https://ccrma.stanford.edu/~cc/shtml/tempResNetPerf.shtml

[91] Linux Manual, *ping(8) - Linux man page*. [Online]. Available: https://linux.die.net/man/8/ping

[92] T. Cortina, “Introduction to computing for creative practice, fall 2020, lecture 22: Mutual interaction,” https://www.cs.cmu.edu/ tcortina/15104-f20/lectures/, 2020.

[93] J. London, “Cognitive constraints on metric systems: Some observations and hypotheses,” *Music Perception: An Interdisciplinary Journal*, vol. 19, no. 4, pp. 529–550, 2002. [Online]. Available: http://www.jstor.org/stable/10.1525/mp.2002.19.4.529

[94] MIDI Association, “Official midi specifications,” June 2023. [Online]. Available: https://www.midi.org/specifications

[95] C. Brown, “Eternal music client,” https://www.transjam.com/downloads/eternal.html.

[96] Phil Burk / SoftSynth, “Transjam - jammin’ on the web!” 1997. [Online]. Available: https://www.transjam.com/

[97] C. Brown, “Eternal music client,” https://www.transjam.com/eternal/eternal_client.html.

[98] Cockos Incorporated, “Ninjam,” https://www.cockos.com/ninjam/.

[99] “Jack audio git repository,” JackTrip Labs. [Online]. Available: https://github.com/jackaudio

[100] “Jacktrip software homepage,” CCRMA, Stanford University. [Online]. Available: https://ccrma.stanford.edu/software/jacktrip/osx/

[101] S. Wilson, N. Lorway, R. Coull, K. Vasilakos, and T. Moyers, “Free as in beer: Some explorations into structured improvisation using networked live-coding systems,” *Computer Music Journal*, vol. 38, no. 1, pp. 54–64, 2014.

[102] “Endlesss homepage.” [Online]. Available: https://endlesss.fm/

[103] MozDevNet, “Webgl - web apis: Mdn.” [Online]. Available: https://developer.mozilla.org/en-US/docs/Web/API/WebGL_API