netMUSE: Networked Multi-user Sonic Environment

Bogdan Vera[i]
Centre for Digital Music, Queen Mary University of London
School of Electronic Engineering and Computer Science
bogdan.vera@eecs.qmul.ac.uk

In: David Holland, Louise Rossiter (Eds.) Proceedings of Sound, Sight, Space and Play 2013
Postgraduate Symposium for the Creative Sonic Arts
De Montfort University Leicester, United Kingdom, 5-7 June 2013.

ABSTRACT

netMUSE is a piece of software designed to enable networked computer music composition and performance involving many users, without sending live audio signals over the network, inspired by recent developments in the sandbox genre of massively multiplayer computer games. It was developed as a Master of Arts final project at the University of York, in the MA Music Technology programme.
netMUSE was developed in Java, with the use of the Processing (Stanford University, 2008) libraries, Beads (Renaud et al, 2007) and Open Sound Control (Renaud, 2010). It is based around a custom developed sever-client system designed to host a persistent session for long periods of time. This allows users to build sound generating structures from basic interactive building blocks in one very large 2D space, and interact with them together in real time.
This paper focuses on the aesthetic and compositional principles driving the project, as well as touching on the technical implementation and describing the current state of the software. Finally, the possibilities created by this type of networked music application are explored.

1. Literature Review

Networked music performance technology was in part pioneered by research done at CCRMA's SoundWIRE group, at Stanford University, led by director Chris Chafe. Research at SoundWIRE resulted in software such as JackTrip, described as 'A System for High-Quality Audio Network Performance over The Internet' (Stanford University, 2008). Publications from this department cover a wide range of topics, including studies into network propagation latency, internet acoustics, network performance systems and software design.

A few approaches to networked music performance can be identified, but so far there appear to be no universally established classifications. Considering the research of academics such as Alain Renaud and Alexander Carôt, the approaches can be thought of as latency critical and latency accepting subtypes. In the first type, the latency of the network is critical to the performance, and high latencies are not acceptable. Generally a latency higher than around 25 ms will make it difficult for musicians to play at fast tempos, causing them to slow down to keep up with each other. Renaud et al. Define 25ms as the EPT, ensemble performance threshold, which they describe as 'the maximum delay that allows for musicians to play in synchronisation' (Renaud et al, 2007). The latency accepting approach considers latency as non-critical, and the musicians are not expected to interact in real time, or timing is not critical to the music being performed.

There have been various attempts to tackle latency in networked music. Renaud et al describe an example timing system used in Ninjam (Renaud et al, 2007) (Novel Intervallic Jamming Architecture), from Cockos (creators of the Reaper DAW). In Ninjam, intentional delays are added based on a central metronome, to the point where performers receive each other's audio with latency equal to the duration of a set number of bars, so the musicians play asynchronously but technically stay in time. Additionally, Renaud (Renaud, 2010) presents a cueing system dubbed 'Master Cue Generator', which he describes as being 'used to trigger performers (humans or computers) over an IP-based network'. Renaud identifies three types of cues, temporal, behavioural and notational. The MCG exploits these ideas and, as Renaud explains, it 'broadcasts important musical information by providing a basic structure to the nodes playing over the network, such as which section of the piece the nodes are in, as well as warning messages that the piece is about to switch to another section.' The apparent success of these techniques and continuing research seems to indicate that in the future users will possibly embrace interacting in a latent performance environment

1.1 Multiplayer Games and the Sandbox Paradigm

In recent years, the computer game industry has been revolutionized by the emergence and subsequent popularity of MMO (massively-multiplayer online) games. This type of game employs a server-client based architecture (Caltagirone, 2002), allowing a very large number of users (sometimes ranging in the tens of thousands) to interact in one shared game world, over a very long period of time. In the MMORPG variety, which is arguably the most popular, long-term player participation is driven primarily by the endeavour to 'level up' or improve a character by gaining character experience. Character leveling, in turn, allows players access to more areas of the game world, populated by more powerful enemies with increasingly high experience and loot returns. Another variety of MMO game has emerged quite recently, and is called a Sandbox MMO. This type of game often does not provide a set of rules for 'winning' or character progression, but rather allows users to do anything they like. Often a 'building' mechanic is involved, which allows players to work together (or alone) to create things from a palette of building blocks. Perhaps the most well known game of this type is Markus Persson's 'Minecraft' (Mojang, n.d.), which has garnered an immense amount of popularity in recent years. Minecraft's world is made up almost entirely of equally sized cube-shaped blocks, with various properties and appearance. Users can mine blocks from the environment, collect them, and use them again to build structures, such as buildings or devices. Some blocks can be crafted into blocks with special properties, such as an extensive set of electronic circuitry-inspired blocks called 'redstone', which allow users to create interactive structures and logical circuits. Consequently, some Minecraft servers contain immense and intricate player content, built over months or years. Many servers start new players in areas with immense player-constructed castles, designed to showcase the architectural skills of users on that server. In this regard, every Minecraft server is completely different, and is a world and a community onto itself. On a note perhaps most related to this project, there is a type of block in Minecraft called a 'note block', which plays a pitched sound when activated. The blocks can be activated using redstone circuits, allowing players to construct very complex programmable sequencers, although the quality and quantity of the sounds is very limited.

1. Introduction

Using the massively multi-user sandbox paradigm as an inspiration, netMUSE is a software system that allows a large number of users to interact musically in a novel way, encouraging collective creative input in a large, distributed performance space. Just as user creations in a sandbox game such as Minecraft can remain persistently inside a server's game world for other users to explore. NetMUSE explores the idea of storing musical arrangements for long periods of time, during which they are available for other users to manipulate or simply experience.

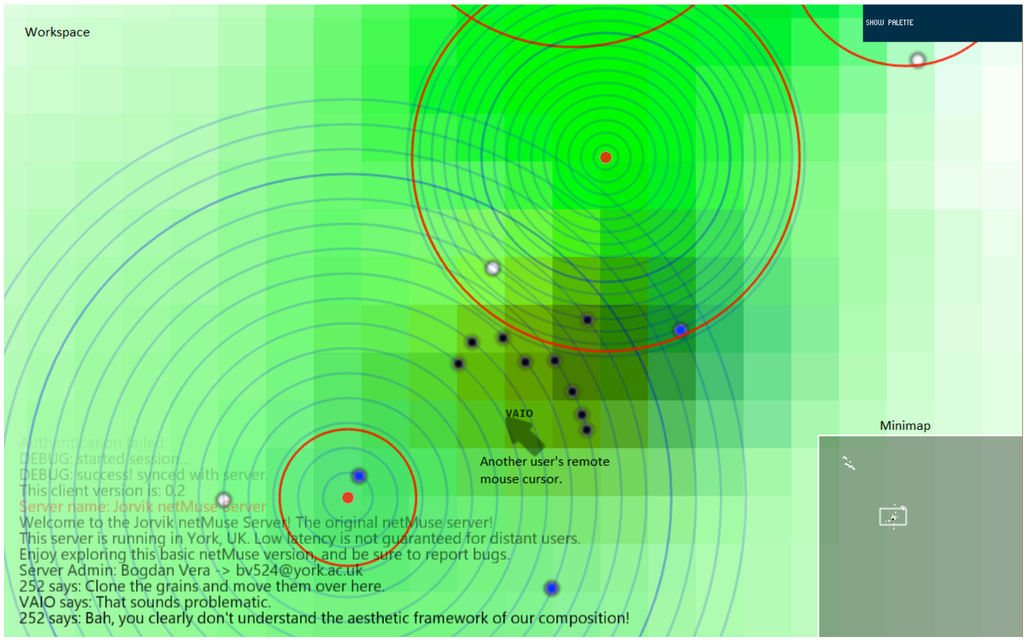
The system provides users with a visual interactive interface to sound generating entities that are shared seamlessly between users connected to the same netMUSE server. From a set of familiar building blocks such as samplers, oscillators, granulators and sequencers, users build sound generating structures from a top down perspective, and all user actions including adding or manipulating entities are replicated in real time to all other connected users.

Another way in which netMUSE mimics the sandbox game paradigm is by making the available workspace very large. A user can navigate along this large 'map' using the keyboard, but can only hear sounds that are generated directly around their location, using realistic sound attenuation. This adds an element of exploration to the user experience, and also encourages creative users to exploit this element in their compositions, for instance by creating fields of sound generating entities over wide areas which users can never listen to as a whole, but rather must explore the piece in a non-linear fashion by physically navigating through it.

2. Framework

netMUSE's interface (in its current form) shows a top-down 2D view of a large space called a 'map'. Form a drop down menu, users can select from a number of interactive entities, such as oscillators, sample players, circular sequencers (called 'wavefronts') and granulators. These entities can be added onto the map using the mouse, after which they are editable through context menus revealed by right clicking. Object context menus include options such as removal and cloning, oscillator frequencies, sample selection, etc. Overall the interface is familiar to users that have used software such as Max/MSP, however, rather than relying on connections or patch cords, netMUSE's workflow relies on proximity between entities as well as entity placement. Due to the fact that output volume attenuates with distance, each user's position on the map also directs their perception of the performance, and could be thought of as an element that directly affects it.

Communication between users connected to a session is facilitated by a built in terminal-like interface which enables chatting, while also allowing various types of commands (such as commands for 'kicking' or banning users and testing network latency).



```
Authentication failed.
DEBUG: started session...
DEBUG: success! synced with server.
This client version is: 0.2
Server name: Jorvik netMuse Server
Welcome to the Jorvik netMuse Server! The original netMuse server!
This server is running in York, UK. Low latency is not guaranteed for distant users.
Enjoy exploring this basic netMuse version, and be sure to report bugs.
Server Admin: Bogdan Vera -> bv524@york.ac.uk
252 says: Clone the grains and move them over here.
VAIO says: That sounds problematic.
252 says: Bah, you clearly don't understand the aesthetic framework of our composition!
```

Figure 1 Client-side interface.

3.1 Objects

In its proof of concept form, the system provides a basic set of usable entities, consisting of a sine wave oscillator with envelope controls (labeled a 'Bloop'), a variable rate sample player, a granulator entity (labeled 'Grain' with a variety of controls) and a type of circular sequencer, labeled 'Wavefront'. The various sound generating entities respond differently to the influence one or more Wavefront entities.

3.1.1 Wavefront

The Wavefront entity implements a polyphonic 16-step sequencer displayed as a series of concentric circles, representing metrical steps, extending around one central point which emits periodic waves. Objects in the Wavefront's area of influence are triggered by the wave according each entity's activation rules (for example, samples are simply triggered while Grains are faded in at various speeds by the wave). Objects can either be snapped to a particular Wavefront's 'grid' or be placed in a free-form manner. Objects that are snapped to a parent Wavefront respond to their parent's change in parameters such as size or position. This means that a Wavefront's area can be extended to fill the entire screen, entities can be snapped easily to the Wavefront, after which the parent Wavefront can be collapsed into a very small area, with the entities preserving their position relative to their parent. Several Wavefronts can have overlapping areas, triggering or affecting common entities together. Additionally, a Wavefront can be set to trigger another Wavefront when its activation wave finishes one cycle, so an endless number of Wavefronts can be linked together to form more complex patterns.

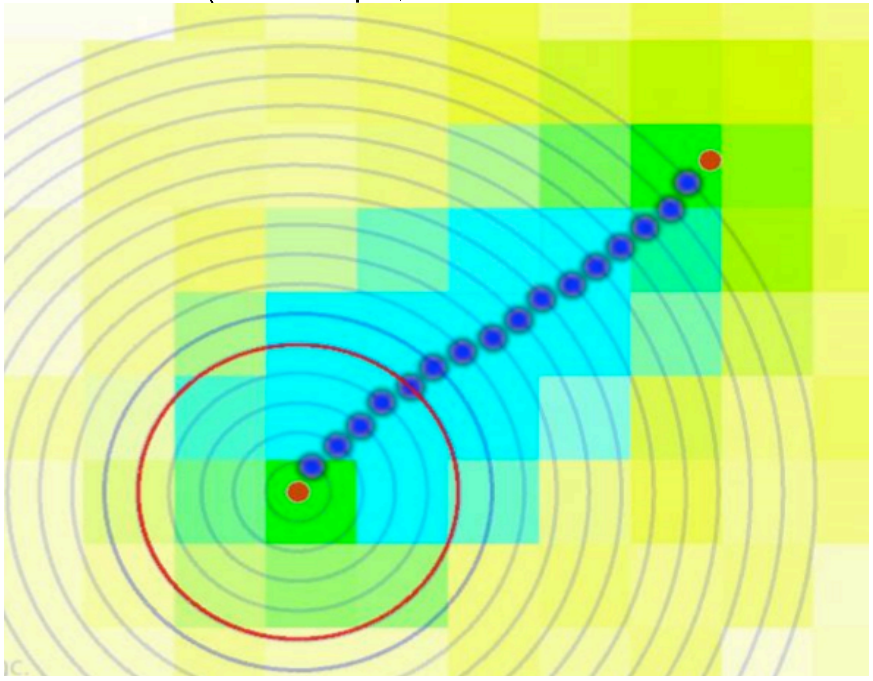


Figure 2 Sample entities sequenced around one Wavefront.

3.1.2 Bloop

The Bloop entity is a simple sinusoidal oscillator with an attack and release envelope as well as pitch controls. Wavefronts trigger Bloop entities by starting their envelopes. By placing Bloops around one or several Wavefronts, complex sounds can be generated by additive synthesis.

3.1.3 Sample

The Sample entity implements a variable speed sample player which can be used for tasks such as creating drum beats, sound collage/musique concrete, etc. Currently the Sample object can only load audio files that are locally available to a user, however ideally the server could store a collection of sound files chosen by users, enabling many more possibilities for composition.

3.1.4 Grain

The Grain entity implements a granular synthesizer with a variety of controls such as play head position, grain size, grain speed, grain separation and pitch. Similarly to the Sample entity, Grain can load a file from disk for granulation. The entity will essentially play the same grain of sound from the same position in the audio file continuously, however it will by inaudible by default. When the activation wave of a Wavefront passes close to a Grain entity, the position of the closest point on the wave relative to the Grain will being to fade it in, and then out, as it passes over it. Multiple waves passing over the same grain at the same time have an additive influence (although the gain factor cannot exceed 1.0).

The Grain entity creates compositional possibilities due to the fact that several Wavefronts playing at different speeds and having areas of influence of various sizes can activate a large field of diverse grain entities. Distance gain attenuation then allows users to explore potentially very large granular sound 'clouds'.

3.2 Compositional Methods

While the sandbox paradigm often implies that players have free reign to create or do anything they desire, this is not often the case. In fact, it can be argued that most sandbox computer games impose certain limits on creative freedom, either for reasons of technical implementation difficulty or, more often, due to a clear desire to steer users into a particular creative direction. For example, Minecraft’s visual style revolves around an 8-bit aesthetic, with pixelated textures and ‘voxelated’ geometry made up of small cubes. It would be impossible for users to create curved surfaces, for example, and hence the overall look of user content can be described as blocky. Nevertheless, this gives Minecraft its characteristic style, and this retro aesthetic has been praised. Within the limits of this aesthetic, however, players have great creative freedom.

In netMUSE, the aim was the steer users towards constructing soundscapes while guiding them away from simply using the software as a DAW and just sequencing sounds or creating drum beats (although this is still possible, while difficult). The choice of sounds objects made available to the users makes the creation of granular or additive synthesis-based compositions very easy, as the visual interface makes it intuitive to arrange sounds.

4. Technical Implementation

netMUSE was created using the Processing framework, along with a set of libraries including OscP5, which enables communication over computer networks using the Open Sound Control format. The overall system includes a server application, which must run continuously on a publicly reachable machine, and a client application that draws graphics and generates sound, serving as an interface for user input. Sound synthesis was implemented using Ollie Brown's Beads library, which provides a set of Java classes for sound synthesis similar to SuperCollider's set of UGENs. The interactive interface was developed using OpenGL, for graphics rendering, and the ControlP5 library for creating a custom user interface.

4.1 The Server-Client Model

netMUSE is based around networked communication between *clients* and one *server.* This is somewhat similar to the server-client structure of massively multiplayer games, but the server application can also be compared to the concept of a *dedicated server*, as it is designed to be freely hosted by end users on home machines rather than the entire system being based around a small number of servers. In the case of netMUSE, a user runs an instance of the main netMUSE application, which allows users to interact with the shared workspace and allows them to see and hear each other's actions. The server application runs on a public machine, handles connections between clients connected to it and persistently stores a representation of the entities and users existing on its session. *Figure 3* shows a diagram of some of the likely networking scenarios, and mentions some of the important concepts such as Network Address Translation (or more importantly, how to bypass it) and the server's location at the centre of the system as a means of allowing clients from various connections to communicate with each other and create the impression of a seamless virtual space. The efficacy of this model is documented by various publications (Caltagirone, 2002) (Assiotis and Tzanov 2005), and has proven its use in commercial applications, although most recently there has been a trend towards decentralizing this model and using peer-to-peer connections. The paper 'Architecture for a Massively Multiplayer Online Role Playing Game Engine' (Caltagirone, 2002) explains:

> Two architectures are possible with multiplayer games (with some variations), centralized server and peer-to-peer (distributed). The architecture being presented is a centralized server, and was designed as such for five reasons, world consistency, security, avoiding the game clock problem, simpler implementation, and a viable business model.

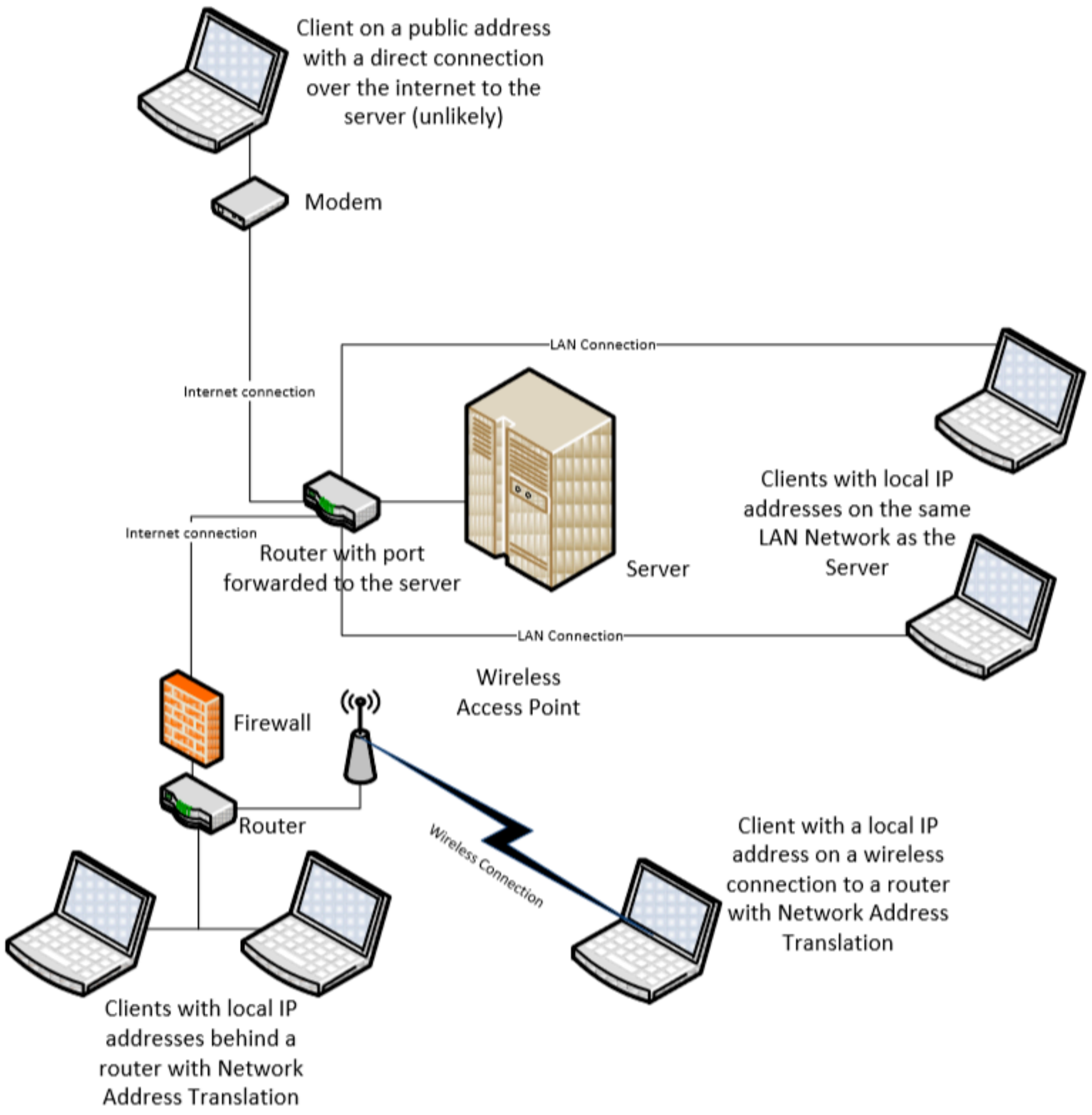

Client on a public address with a direct connection over the internet to the server (unlikely)
Modem
Internet connection
LAN Connection
Router with port forwarded to the server
Server
Clients with local IP addresses on the same LAN Network as the Server
Internet connection
LAN Connection
Wireless Access Point
Firewall
Router
Wireless Connection
Client with a local IP address on a wireless connection to a router with Network Address Translation
Clients with local IP addresses behind a router with Network Address Translation

Figure 3 Some common networking scenarios.

Communication between the clients and the server is done through standard Open Sound Control messages, using a custom formatting scheme. A message generally consists of a pattern matching address, used in netMUSE to specify the type of message, followed by a series of arguments. The choice of using the pattern matching address simply to specify the type of message somewhat goes against the intended purpose of this feature, which is normally used in OSC applications to route messages to specially addressed controls. Because netMUSE has a completely freeform architecture, it would not make very much sense to use the address this way, and instead various arguments specify entities, users, etc. The following example message informs a server (or client) that a user has added an entity.

"/update_existingEntity, "user1", "Wavefront", "2349512", 100, 200

The pattern matching address starts with a "/" character, conforming to the pattern matching address standard. The following arguments are the name of the user that added the entity, the type of entity, the unique identifier of the entity and its x and y coordinates on the map. In comparison, a classically formatted OSC message might simply have a pattern matching address of the type "/controlPanel1/knobs/cutoff", followed by a controller value. In this case, the message might change the position of the cutoff knob on the first control panel of some hypothetical synthesizer.

A client would send the message above when the user adds an entity, and would then forward it to the server. The server listens for messages and handles them based on their pattern matching address. In this case, the server would add the entity to its list of entities, and then forward identically formatted messages to all of the clients connected to the server, except the one that originally sent the message. The server and client both continuously listen for incoming messages on a dedicated thread. An example message flow is illustrated in Figure 4.

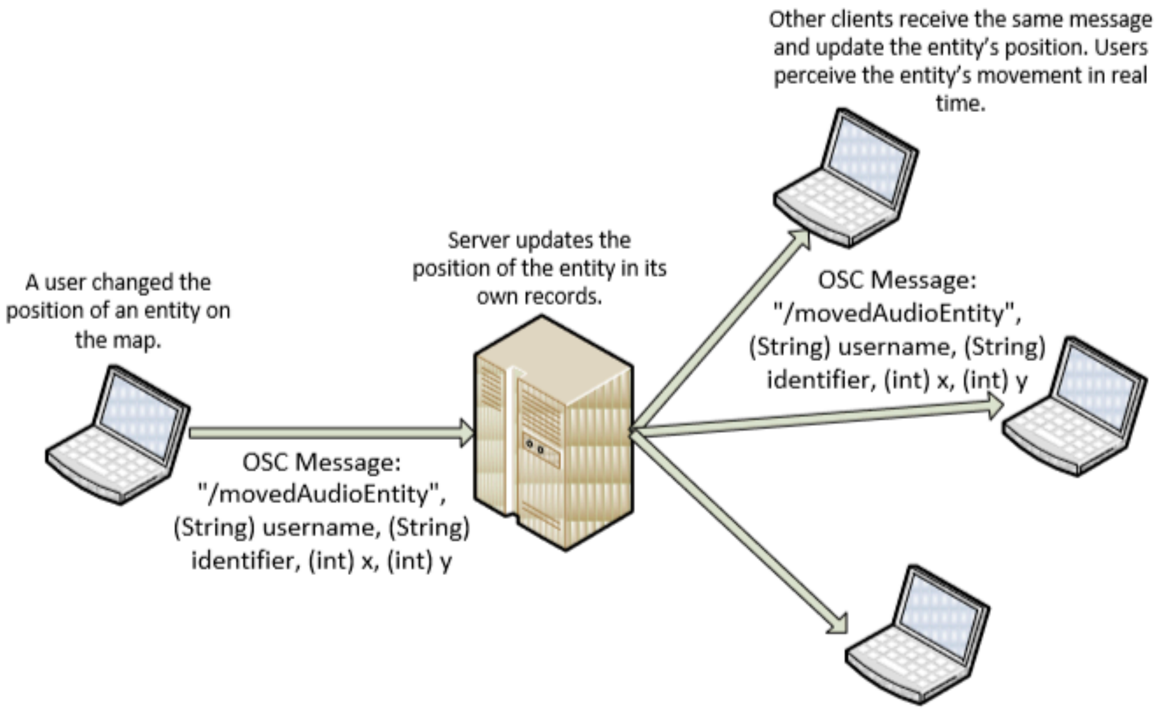

Other clients receive the same message and update the entity's position. Users perceive the entity's movement in real time.
Server updates the position of the entity in its own records.
A user changed the position of an entity on the map.
OSC Message: "/movedAudioEntity", (String) username, (String) identifier, (int) x, (int) y
OSC Message: "/movedAudioEntity", (String) username, (String) identifier, (int) x, (int) y

Figure 4 Message flow for when a user moves an entity.

4.2 Bypassing Network Address Translation

The architecture described above suffers from a significant flaw if applied naïvely to an online context. When the server is running on a Local Area Network located behind a router with a NAT firewall, it is possible for users on the same network to connect to the server simply by knowing its IP. The server then records the users' IPs and can route the messages to these addresses. Attempting to host the server for users outside of the local network presents a problem in that the router will have an external address, which is accessible to machines connected to the internet, but the server itself will be running on a local machine with an inaccessible and possibly random address, making it impossible to reach. This issue is resolved by forwarding a port in the router to the server machine. External clients can then reach the server by sending messages to the router's address on the given port, and the messages will then be automatically forwarded to the server. This issue occurs for the clients as well, and the server will not be able to send messages back to clients behind a NAT. Rather than making users forward their own ports, it is possible to solve this problem with a technique called *NAT hole punching* (Ford et al, 2013).

NAT hole punching is generally used to establish connections between clients called *peers*, which are all behind NAT firewalls, an issue that commonly occurs in applications such as torrent clients and online games. Classic NAT hole punching involves a *hand shaking server,* which is completely accessible, to which the clients all initially connect. The server than records the original port and address of the clients, and instructs them to message each other on the same ports that the original connections were established through. Sending network packets back through a router's output port will normally cause that port to remain open indefinitely, although the port will generally be closed automatically after a certain amount of time has passed without any packet throughput. *Keep alive* packets are then sent periodically to force the port to remain open. The individual clients can then communicate to each other through the same ports they originally used to contact the server.

netMUSE was a prime candidate for this technique, as it inherently uses a server to route messages, although in this case the server acts as more than just a handshaking server. The server application is designed to listen and send messages on port 8000, whereas the client will simply find a random port to both listen and receive on. When a client connects to the server, the server retrieves the port of the output (and in this case the input) socket of the NAT in the client's router, and stores it. Keep alive messages are then periodically sent to keep this connection open, and the server communicates to the client on that port. The exchange of keep alive messages also serves as a way to detect when a client has disconnected, so that it can be removed from the active users list. Thus, clients in netMUSE do not communicate with each other, but rather through the server, which can keep the connection open between itself and the clients.

This techniques allows users from most networking conditions, and most importantly people with no technical knowledge, to join the server and begin experimenting with the software.

4.3 Network Latency

netMUSE takes a predominantly latency accepting approach, and there is very little functionality implemented to alleviate the effects of latency, although there are certain cases in the software where synchronization is important, as covered in the later section about Wavefront entities. Much like in the multiplayer games it takes inspiration from, latency and *lag* remains an unavoidable nuisance, especially in long-distance interaction scenarios, and there remains a large spectrum of further research to be done in this area.

netMUSE attempts to treat the problem of latency by intentionally turning the focus away from live networked performance and towards preparation of sequences and sounds. Even so, it is possible to interact with content on the server in a way that resembles live, synchronized performance. For example, by placing and manipulating various entities such as Grain or Bloop entities around a Wavefront entity, and varying the various parameters available, something reminiscent of an ensemble performance can be achieved. However, even though users participating in such a performance may not immediately notice, latency introduces slight variations in timing between users, making the experience of the performance ultimately inconsistent. Generally speaking, the finer the interaction between users, the more latency becomes noticeable, as users' responses to each other become delayed more and more with respect to the tempo. In experiments with live networked music, latency has been described as primarily having an effect on the tempo of a piece, appearing to *force* networked users to lower their tempo by making them constantly wait to respond to each other until the performance slows down to a level where the latency falls beneath a perceptual threshold, as described by Nathan Schuett (Schuett, 2002) and demonstrated by Chris Chafe in a series of experiments involving subjects trying to keep in time by clapping whilst separated in different rooms and hearing each other with a time delay (Chafe et al, 2004). In netMUSE's case, where the performance will often revolve around looping sequences, latency will often result in the perceived sequences being at least slightly out of phase with each other, unless a stricter and more precise synchronization is used to ensure timing precision, which may involve intentional limitations that can affect the performance. netMUSE treats this desynchronisation as ultimately inevitable, and it can thought of as a reasonable sacrifice in the endeavor to allow a large number of users with varying connections and from various locations to connect and interact. In this case it is overall more important to ensure that the relative order and timing between events in each user's perception is consistent then to attempt to ensure that the clients all perceive events at exactly the same time.

5. Critical Analysis and Conclusions

The software presented exemplifies some of the key issues within the subject of networked music performance and composition, while attempting to address some of them with mixed but promising results. One original goal of netMUSE was not to dictate any aesthetic or musical parameters or limitations to the users involved, allowing them to simply take a set of very basic building blocks and construct completely original sounds. Surprisingly, this is not the case, and the users involved in testing of the prototype were found to construct somewhat similar sounds, although this can be attributed to the scarce palette of basic tools and sounds in the prototype. As an example, the Grain entity allows users to build potentially unlimited types of sounds, but only as long as the source sound material is sufficient. For this reason, perhaps a feature allowing users to upload and share their own sounds for granulation would greatly enhance the type of music and sound that can be generated. A similar conclusion can be drawn for the Bloop object, which essentially allows users to perform additive synthesis. While additive synthesis can yield incredibly complex sounds when automated in various ways, the sounds resulting from the test users' interactions were often quite similar. Users would often attempt to create sequences of related harmonics, or to simply construct melodies consisting of clearly defined individual notes.

As a general conclusion, the end result of longer sessions has primarily been the creation of large, continuous fields of sounds, which can be explored via simple movement of the viewpoint across the map. This is consistent with the author's original view of what longer term multi-user interaction may lead to: the inevitable conglomeration of users' creations into one mass. It is difficult to characterize the genre of music produced in this situation, although its often cacophonous nature can be likened to that of the aptly named genre of "noise music". In truth, however, the presence of many users' contributions, spread across a large map, creates what may be thought of as a stylistic continuum. The performance element can be attributed both to the users that have prepared the content existing in the session and equally to the observer, who may essentially remix the sound generated through simple movement across the map, allowing the built in distance attenuation to mix the sounds. This is reminiscent of the concept of a performance ecosystem, described by Tom Davis (Davis, 2011), paraphrasing Bowers (Bowers, 2003), as an assemblage of 'artifacts and practices that enable him to participate in collective music making'. In this sense, a live netMUSE session is perhaps a very literal instance of such an ecosystem, and much like a biological ecosystem, it can grow and evolve over time based on complex interactions between users with varying motives and aesthetic preferences. As described in section 5, implementing some form of game-inspired conflict or competition mechanic would take these ideas even further and explore the concept of challenge in music. Martin Parker (Parker, 2004) has presented some related ideas. He talks about making musical interfaces purposefully challenging, writing that the limitations of various instruments and the challenge involved in playing them gives them their specific sound and expression. He then presents a set of ideas for a piece of musical software that actively works against the user. The author of this dissertation believes that this concept could be taken further, and that the challenge may arise from the actual competition between users, leading to an interesting dynamic that would be interesting to observe in a massively multi-user context.

In conclusion, netMUSE offers a reasonable test bed for exploring long term networked music performance. It offers a good possibility for developing a series of analytical tools that can shed light on the way users might interact in such an environment, and the presence of a central server means that, in a research scenario, data can be gathered and analyzed in a central point, over a long period of time. After some further development, an effort will be made by the author to host a permanent netMUSE server, for anyone to access at any time, although the resources required to scale up the user base are yet to be determined, and it may be discovered that true *massively* multi-user interaction can only be achieved with the considerably larger computing resources of a large company. At some point in the future, the software may be released as open source, making it easy for users to host their own servers. Splitting up the potential overall user base into several local servers would mitigate the problem of latency, while making netMUSE even more similar to a sandbox MMO like Minecraft, and more dissimilar to a MMORPG like World of Warcraft.

All of these possibilities remain to be explored in the near future, with equal emphasis on the artistic and research aspect of the project, even if netMUSE itself may be rewritten and reconsidered with the added benefit of hindsight.

References

Stanford University (2008) Jacktrip, [Online]. Available from: https://ccrma.stanford.edu/groups/soundwire/software/jacktrip/

Renaud et al (2007), Networked Music Performance: State of the Art, in *30th International Conference on Intelligent Audio Environments*.

Renaud, A. (2010) Dynamic Cues for Network Music Interactions, in *Sound and Music Computing Conference*.

Caltagirone, S. et al (2002), Architecture for a Massively Multiplayer Online Role Playing Game Engine, *Journal of Computing Sciences in Colleges,* pp. 105-116.

Mojang, "Minecraft," [Online]. Available from: https://minecraft.net/.

Assiotis, M. and Tzanov, V. (2005) A distributed architecture for massive multiplayer online role-playing games," http://pdos.csail.mit.edu/6.824-2005/reports/assiotis.pdf

Ford, B. et al (2013), Peer-to-Peer Communication Across Network Address Translators, [Online]. Available from: http://www.brynosaurus.com/pub/net/p2pnat/.

Schuett, N. (2002), "The Effects of Latency on Ensemble Performance," Stanford University.

Chafe, C. et al (2004), "Effect of Time Delay on Ensemble Accuracy," in *International Symposium on Musical Acoustics*, Nara.

Davis, T. (2011), "Towards a Relational Understandsing of the Performance Ecosystem," *Organized Sound,* pp. 120-124.

Bowers, J., (2003), Improvising Machines: Ethnographically Informed Design for Improvised Electro-Acoustic Music, in *University of East Anglia*.

Parker, M., (2004), Resisting the Seamless, *International Journal of Architectural Computing,* pp. 430-442.

Notes

[1] Many thanks to Oli Larkin (Music Research Centre, University of York) for supervision of this work, which was part of my Master of Arts final project.