A Comparison of Protocols for Communication of Digital Music Gestural Data

Sam Whelan

Masters(Taught) Dissertation
MSc. in Computing, Communications Software
Department of Computing, Maths and Physics
School of Science
Waterford Institute of Technology

Supervisor : Mr. Robert O'Connor
Head of Department : Dr. Mícheál ÓhÉigeartaigh

August 21, 2009

Table of Contents

A Comparison of Protocols for Communication of Digital Music Gestural Data

Sam Whelan

Waterford Institute of Technology
Department of Computing, Maths and Physics
Cork Rd., Waterford City, Ireland

Abstract. Established protocols, Musical Instrument Digital Interface (MIDI) and Open Sound Control (OSC), provide a means of digitally storing and manipulating the gestures made by musicians while performing on musical instruments. OSC also provides support for sending this gestural data across networks. This research compares two approaches to sending MIDI data across networks.

1 Introduction

The purpose of the proposed research topic is to compare two approaches to the transport of Musical Instrument Digital Interface (MIDI) [1] data across a network. This data is to be transported over established protocols namely Real Time Protocol (RTP) [2] and Open Sound Control (OSC) [3].

Musicians express notes and rhythm through various physical gestures, strumming a guitar, hitting keys on a piano, blowing into a trumpet and so on. Both MIDI and OSC provide a means of storing and manipulating such gestures in a text based digital format. The MIDI protocol was originally developed to allow a musician to trigger several music synthesisers from one piano keyboard. It has stood the test of time not least on account of its simplicity. MIDI is not without its shortcomings being both too undefined for seamless device interaction while also being too narrowly defined for customisation [4]. OSC, a promising alternative, is built with modern networking in mind, facilitating the transport of media over local area networks, wireless networks and the internet [5].

There are many applications where musicians wish to collaborate over a network as if they were in the same room. This can be achieved by sending real time audio data such as WAV or MP3 files or gestural data using protocols such as MIDI or OSC. Gestural data has the advantage of using less bandwidth than real time data thus reducing the impact of network congestion [6]. For example "The Entertainer" by Scot Joplin is 20kb when represented in MIDI versus 9.72MB as a WAV file at a sample rate 1411kbps. Gestural data also facilitates applications that would not be possible with real time audio. Gestural data for example facilitates different digital music instruments to express the same gestural information at different end points. This enables applications such as laptop orchestras [7], [8] and remote control of distant instruments for remote musical ensembles [9].

There have been general comparisons of MIDI and OSC [10], [11]. This research compares MIDI combined with RTP and MIDI combined with OSC based implementations for communicating gestural information over a network.

2 Research Question and Hypothesis

2.1 Research Question

How does the transmission of MIDI over RTP and MIDI over OSC compare with respect to data loss for communicating gesture-driven information describing musical passages over a network?

2.2 Research Hypothesis

It is proposed that two mechanisms be used to transport gestural data, describing the same musical passages, across the a network. The approaches being considered are MIDI over RTP and MIDI over OSC. For a selection of MIDI files, each file will be played and recorded locally. The same files will then be played over a network using each of the two mechanisms and recorded at a remote client. These recordings will be exported as MIDI files. It is expected that the differences in these files will demonstrate the quality of each approach.

3 Literature Review

3.1 Digital Representation of Sound

In [12] Kirn describes how the tiny fluctuations in air pressure we call sound are represented digitally. Sine waves are presented as a basic waveform that can be used as a building block to describe more complex waveforms. Sine waves oscillate repeatedly at a single frequency and have no sound energy at any other frequency. Real world audio can be analysed mathematically as a combination of sine waves. A technique called Fourier transform can be used to analyse a complex waveform as being made of a number of simple components, each of which are sine waves.

After presenting an overview how sound works and how it can be measured and modelled mathematically Kirn moves on to discuss sound in digital form. Once transformed to electrical form by means of a transducer, such as a microphone, sound signals can be converted into numerical data that can be transmitted digitally to computers and other digital devices.

3.2 MIDI

Kirn also introduces MIDI, a particular digital representation of sound that captures the physical gestures of the musician as they perform a piece of music on a particular instrument. This digital method of recording gestures is analogous

to transcribing musical performances as written scores. The three parts of the MIDI specification are outlined as, a file format, a protocol specification and the physical interface and cabling between units. MIDI data is said to be more compact and easier to edit than real time audio data. However MIDI requires a synthesiser to produce sound.

A more detailed description of MIDI is provided by Kirn in [4]. Here Kirn describes MIDI as a simple way of storing sequences of musical events, likening it to the player piano for the modern age. The traditional use case for MIDI is described as having three elements, a controller, a receiver and a sequencer. The controller is most commonly a musical keyboard, the receiver a sound module or virtual instrument and the sequencer a means to edit the MIDI performance. MIDI data is described as a series of number values, from 0 to 255 that allow control events to be universally understood by different hardware and software. These 256 bytes are broken into status bytes and data bytes. Each MIDI message consists of a status byte followed by one or two data bytes. The status byte describes the kind of message e.g. notes or control changes. The second and third bytes describe the note number, e.g. 60 for middle C and other relative information, e.g. velocity with which the note was struck, respectively. The MIDI Manufacturers Association MMA, MIDI file format[1] allows MIDI messages to be collected and stored in a computer file system. MIDI files are used in the experimental procedure of this research.

3.3 OSC

Wright and Freed [5] present Open Sound Control (OSC), a transport independent protocol for communication among computers, sound synthesisers and other multimedia devices that is optimised for modern networking technology. OSC data is delivered in packets rather than streams leading to a protocol that is as stateless as possible. They describe the OSC data representations messages and bundles, which contain multiple messages. A message addressing scheme is also outlined as a hierarchical set of dynamic objects allowing each system that can be controlled by OSC to define its own address hierarchy, thus avoiding the addressing limitations inherent in MIDI. The authors state that "OSC addressed our need for a network protocol usable for interactive computer music that could run over existing high-speed network technologies such as Ethernet". It is this aspect of OSC that is particularly relevant to this research.

3.4 MIDI vs OSC

Wessel and Wright [13] describe their efforts in developing a live performance computer based musical instrument. They consider among other aspects low latency and the relationship between the musicians gestures and the musical result. Particularly relative to this research, they describe MIDI and OSC as discrete event protocols. They point out that MIDI has no mechanism for atomic updates. This means that a chord is always represented as an arpeggio. An arpeggio is a broken chord in which the individual tones are sounded one after another instead

of simultaneously [14]. They point to [15], [16] and [17] as providing examples of what is termed the disfunction of MIDI. They also suggest that OSC addresses these disfunctions. For example in OSC notes of a chord represented as a group of messages can be given the same time tags so that they are processed simultaneously.

In [10] the MIDI Manufacturers Association (MMA) set out to correct what they term as public misinformation about MIDI and how it compares to OSC. They discuss how recent implementations have brought OSC a lot of attention and that this has lead to articles being published which contain inaccurate comparisons between MIDI and OSC. They also point to [11] as providing inaccurate comparisons.

The first correction made is that MIDI is hardware independent. They point out that although there is a transport specification for MIDI it is not required and MIDI has been commonly carried on transports such as USB, FireWire and Ethernet. The provision of MIDI over Ethernet will be part of the focus of this research. They also consider comparisons of throughput. Here they point out that the data rate across Ethernet is a factor of the transport and not the protocol. They also state that MIDI messages tend to be smaller than OSC messages describing equivalent gestures and as such MIDI could be said to have better throughput than OSC.

3.5 Network Music Applications

While the earliest known example of computer music dates to 1951 [18], the idea of using computer networks for musical performances may have originated with the Hub in the mid seventies. In his discussion of the aesthetics and history of the Hub, Gresham [19] states that "Music is, at its core, a means of communication". He describes computer network music as interactive data environments that can be electronically transcribed into music. The advent of the microprocessor and the MIDI controlled synthesiser are propounded as having made a new type of network performance possible.

The shortcomings integral to some such systems are embraced by computer network musicians as providing welcome surprises as a part of this art form. For example one of the members of the Hub was observed to have set up a piece of equipment incorrectly, this however was a deliberate mistake as this configuration had been found to generate "a type of feedback that he found particularly stunning".

The first Hub concerts took place in 1985 from two locations in New York over phone lines via a modem, several years before the proliferation of the internet. Around 1990 the Hub experiments moved to using MIDI to directly control the set up of another member of the group. The third generation of the Hub is cited as wrapping MIDI messages in OSC to enable communication of MIDI messages to computers over the internet using IP, which is one of the approaches considered for comparison by this research. This dissertation continues research in the same vein as that researched by the Hub for over thirty years.

A discussion on the affect of network and community infrastructures on the creative musical process is the subject of Tanaka et al in [20]. They present two examples of work demonstrating musical creation at the "intersection of electronic music, interaction and social computing" as opposed to a commodity or dead medium. They suggest that network latency can be viewed as the "acoustic of the network", likening it to the acoustics of a concert hall. The example is given of medieval composers using the long reverberation times of Cathedrals to write long slow lines versus jazz musicians playing extremely fast solos in jazz clubs.

Lazzaro et al [6] discuss how musicians have a desire to interact over networks as if they are in the same room. This creates a case for a practical networked music application (NMP). They describe their approach, including mechanisms to "ameliorate the effects of late and lost packets".

They begin by discussing the network performance requirements of an NMP application in particular with regard to latency and handling of late and lost packets. They point out that a group of musicians playing a composition can be seen as a distributed sensory motor feedback loop. As musicians hear each others performances they adjust their playing to create a coordinated performance. Latency is always present in any performance due to the speed of sound. The distance that musicians would have to be from each other in the same room, i.e. the acoustic latency, is compared to the latency created by musicians performing together over a network in their experiments. It is noted that there are also computational delays at each host. If the overall delays are low enough then a usable NMP system should result.

A description is given of how an NMP might be achieved using audio streaming. Steps necessary to reduce latency are outlined including, use of RTP, uncompressed sample based packets, head phones, close microphone placement and modern sound cards. It is suggested that if such design principles were followed an NMP could be achieved. However in networks with a lack of Quality of Service (QoS) and bandwidth limitations this would not be practical and would lead to garbled audio which would make a successful performance very unlikely.

It is suggested that packet loss could be concealed if the musical performance data was sent at a higher level of abstraction, such as sending gestural data that describes the musicians performance. The impact of late and lost packets in this case would be akin to mistakes made by a musician for example missing a note or playing a note too late. This would be more conducive to a successful NMP than the clicks and gurgles that would result from problems with the streaming audio approach. They suggest that "with clever design, the qualitative effect of an occasional lost packet can be made quite subtle". This dissertation is based on this idea. In particular with how different implementations of this higher level abstraction compare in terms of the quality yielded from the transport of each over a network.

They proceed to give an overview of the client/server architecture of the proposed NMP system. The client is a "real-time sound synthesis engine". It is used to synthesise both information from the local musician and that arriving

across a network from a remote musician. They detail the client design, built using Structured Audio (SA) and SAOL which supports MIDI input. The sfront SA engine was used and was extended to accept UDP packets. The client uses RTP under the Audio Video Profile (AVP) profile to exchange MIDI between end points and Session Initiation Protocol (SIP) [21] is used to set up sessions between musicians at the different end points.

A new payload format that extends RTP to transport MIDI is described in [22]. This format encodes all possible MIDI commands and includes the definition of tools for graceful recovery from packet loss. RTP is presented as an extensible transport. Additional payload formats can be created to add support for new audio or video codecs. The authors presented a new payload format for MIDI at the 52nd IETF meeting based on earlier work on network musical performance [6]. The format partitions MIDI data into packets in a way that minimises the impact of lost and late packets and includes a resiliency scheme that is feed forward in nature and does not use packet retransmission. A recovery journal system is described which ensures that audio rendered from an RTP MIDI stream does not contain indefinite artifacts even if the stream is sent over a network that loses packets. The dynamic qualities of RTP MIDI are considered in particular the algorithms used by senders and receivers. Finally there is a description of using SIP and the Session Description Protocol (SDP) [23] for customising the characteristics of a stream at the start of a session. The RTP payload for MIDI was published in RFC 4695 [24].

Biaz et al [9] introduce a number of RTP and TCP based MIDI over IP protocols and perform quantitative experiments on the selected protocols. They attempt to determine which protocols would be best for the performance of a musical duet and the transmission of a MIDI sequence over a network. They point out that applications include remote control of distant instruments and remote musical performances such as distributed duets. As MIDI cannot tolerate missing or out of sequence messages only TCP was considered as a transport mechanism by these authors.

They explain how they used the Java Media Frameworks RTP Manager to change the underlying transport protocol for RTP to TCP. RTP has two channels so using both UDP and TCP this allows for four different combinations, TCP in the control channel with UDP in the data channel, UDP in the control channel with TCP in the data channel, TCP in both channels and UDP in both channels. The nagle algorithm [25] was used to collect data into larger packets to prevent floods of packets that contain little data And this was enabled and disabled for different experiments.

The TCP only approach was also taken by Dannenberg and Lageweg [26] in the development of Aura a “distributed real time object system” which is capable of transporting audio, MIDI, or any other data between objects. Although they point out that previous studies have found UDP to be reliable over local area networks, they observed dropped packets. This made UDP unusable without adding a mechanism to resend dropped packets. As a result they used TCP only.

The preceding papers describe the state of the art with regard to MIDI transport using RTP which is one of the mechanisms being compared in this research. The other mechanism for comparison is transport using OSC. In [27] Wright discusses how people prefer to work face to face rather than over a network. Following on from this the prospect of developing tools for musical networking that people would prefer to use in place of working in physical proximity are considered. An example is given of a music teacher teaching multiple students in remote locations simultaneously from the comfort of their home. The author points to the communication as being the key element in the success of such tools. He points out that communication should be reliable, accurately timed, sufficiently general in data content and semantics, as simple as possible, as fast as possible and ideally an accepted standard. The Open Sound Control (OSC) protocol is presented as having been designed to meet these goals. He discusses the motivation behind making OSC transport independent and having generality in the meaning of its messages, achieved by the use of symbolic parameter names. He provides a list of some of the projects that have used OSC including SuperCollider [28]. A discussion of physical distance and network latency with regard to musical applications is presented describing how jitter is particularly harmful.

Other implementations of network music applications include Auracle [29] and the Public Sound Objects (PSOs) [30]. Auracle is a "group instrument controlled by voice for real-time interactive distributed music making over the Internet". Vocal sounds are analysed and transmitted across the internet to control a synthesiser. PSOs is an "experimental framework to implement and test new concepts in networked music". Notably specific software features were implemented to counteract the affect of network latency on real time collaboration. In [31] Barbosa provides details of how PSOs incorporates latency as a software function by dynamically adapting the instruments tempo to the communication delay caused by the network latency.

3.6 Quantifying Network Music Applications

This section describes the experiments performed in [6] and [9]. These experiments informed the design of the experiments that were executed as part of this research.

The experiments in [6] were designed to analyse the effectiveness of the proposed network music performance system. The system was qualitatively tested by having the musician reply on the network stream returning from the remote client for audible feedback. In this case network congestion would directly disturb the sensory-motor feedback loop.

This is followed by a discussion on the quantitative experiments. Payload bandwidth was measured by recording the sending time, payload size and MIDI payload command type for all RTP packets in a five minute performance. This gave a measurement of the distribution of MIDI commands. This data was analysed and the number of packets per second, number of bits per packet and

number of bits per second were calculated. These results were summarised using several common statistics.

To quantify the effect of late and lost packets the musician performed for five minutes. The client modeled the RTP timestamp the packet is expected to have, tm. tm-tp was recorded for each received packet, where tp is the time the packet actually arrived at. This provides quantitative information on late packets. Information about each lost packet detected was also recorded. In this experiment the musician heard only local feedback so as not to affect the performance.

The last experiment described in [6] estimates the latency in the system. This was achieved by having a virtual instrument generate a 5KHz 100ms tone for a MIDI NoteOn Command. A second client was setup to detect this tone by means of a microphone. By using mirrors as different locations the latency at each location could be measured. The round trip times were also measured from RTCP reports received during the test.

In [9] the experiments were designed to compare different protocols to determine which protocols would be suitable for the network transmission of a MIDI sequence. The experiments used in involved sending a MIDI sequence from one computer to another measuring the inter-departure and inter-arrival times and the total time at the destination to play the sequence. This was repeated multiple times for each of the protocols being considered. The inter-departure time is the time between the departure of packets at the source. The inter-arrival time is the time between the arrival of packets at the destination.

4 Methodology

This section considers the methodology to address the proposed research question. It begins with an overview of development of the experimental procedure before describing the methodology in detail. It is worth noting that, while the experiments described in 3.6 informed the design of these experiments, a different methodology had to be designed for this research. One of the reasons for this which is elucidated below is that this methodology involved both proprietary software on one hand and a custom built software application on the other. For this reason the comparison could only take place based on the output of both approaches.

4.1 Evolution of Methodology

The methodology to address the research question evolved over a process of minor iterations as the research enlightened the development of an experimental procedure. Initially it was intended that the experiment would be carried out across different networks over the internet. To this end early experiments were run from two different external networks. Firstly, using a mobile internet connection as an external network connecting to a fixed line broadband connection. Secondly, using one fixed line broadband connection to another. It was found that both external networks introduced uncontrollable variables and so

it was decided to limit the experiments to a LAN. Early experiments on the LAN utilised ipfw [32] to limit the bandwidth. It was discovered that unlimited bandwidth on the LAN yielded sufficient discrepancies between the control files and the files sent across the LAN so ipfw was discarded. Using a controlled LAN meant that all experiments could be run in a controlled environment with only the computers involved in the experiment using the network. As a consequence this experiment does not demonstrate how each protocol would perform on a congested network. Figure 1 shows the network topology.

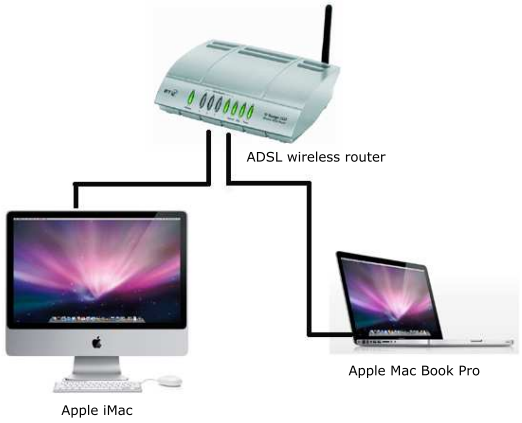


Fig. 1. Network topology

Development of applications to facilitate the experiments began with some simple MIDI applications in Java using the Java Sound API [33]. These programs were written based on [34] and played back midi files using the Java sequencers and synthesizers. Further simple applications were developed to send audio files over RTP using the Java Media Framework (JMF). These applications were then to be modified, based on [35], into an application to send MIDI over RTP. It was also discovered that the Apple OS X operating system included an implementation of RTP MIDI in its MIDI Network Driver. Two Apple computers with OS X were obtained and further investigation led to the decision to park the development of the Java MIDI over RTP program in place of this proprietary solution.

For the MIDI over OSC implementation several languages were considered. The Java implementation of OSC, JOSC [1] combined with SuperCollider's server was the first approach considered. Secondly a pure SuperCollider approach was considered. Ultimately, it was decided to proceed with an implementation using Chuck [36], a strongly-timed, concurrent, and on-the-fly audio programming

[1] http://www.illposed.com/software/javaosc.html

language, as this offered a complete light weight solution. Two Chuck programs were developed. These programs are listed in appendix B.

Various applications to support the experiments were investigated for both the Windows and Apple platforms. In the initial experiments on the Apple computers recording of MIDI was performed using Garageband[2], but as this program did not support exporting of MIDI files it was abandoned. Next Mulab[3] was considered for this purpose but as this gave some unexpected results and was relatively awkward to use it also was abandoned. Ultimately Logic Studio 8[4] was settled on for this task as it provided all the features required, ease of use and gave the most consistent results in the initial experiments. Mighty MIDI[5] was chosen for playback of the MIDI files on the local client.

When using the Apple Network MIDI Driver as the output for Mighty MIDI the entire file was sent to the remote client. Individual events were preferred as the experiments were attempting to be as similar as possible to a live performance while also providing an identical performance each time. To prevent the entire file being sent at once Mighty MIDI's MIDI out was routed through MIDI Keys[6]. MIDI Keys in turn had its MIDI thru configured to route to the relevant MIDI port for the protocol being considered by that experiment.

Analysis of the output from the experiments was performed on a Windows laptop. For this purpose several applications were considered, the most suitable of which was found to be MIDI-OX[7]. MIDI-OX was chosen as it logged MIDI events to a text file in a human readable format which was suitable for file comparison. This meant that both the control files and files exported from the remote recordings could be played back in MIDI-OX and the log files from each used for comparison. ExamDiff Pro[8] was used for the comparison of these log files.

4.2 Description of Methodology

The research question proposed the comparison of two methods of transporting MIDI data over a network. The two approaches, RTP and OSC are illustrated in the partial protocol stacks in tables 1 and 2.

In order to carry out a comparison the experiments needed to collect a data set of MIDI files transported using each protocol. To generate this data set four MIDI files were selected. The selected files are well known classical piano solos with relative complexity, mixed tempos, passages of single notes and chord progressions. A list of the selected pieces is included in appendix A. For each of these files a MIDI track was recorded by playing the file in Mighty MIDI

[2] http://www.apple.com/ilife/garageband/
[3] http://www.mutools.com/index.html
[4] http://www.apple.com/logicstudio/
[5] http://www.versiontracker.com/dyn/moreinfo/macosx/18967
[6] http://midikeys.en.softonic.com/mac
[7] http://www.midiox.com/
[8] http://www.prestosoft.com

Table 1. Partial protocol stack for RTP approach

Application	MIDI-RTP
Transport	UDP
Internet	IP
Link	Ethernet

Table 2. Partial protocol stack for OSC approach

Application	MIDI-OSC
Transport	UDP
Internet	IP
Link	Ethernet

through MIDI keys to a local MIDI port which was routed to Logic Studio. The recordings were exported as MIDI files and saved for use in the analysis.

To generate the MIDI files for each protocol the same method was used up to the output from MIDI Keys. For the Apple Network MIDI Driver experiments the following set up was employed. A Network MIDI port was configured on both the local and remote clients and a connection was made between these MIDI ports. Once this connection was made the network MIDI port appeared to any MIDI application as an additional MIDI port. The MIDI out of the MIDI Keys program was routed to the configured Network MIDI port. Each MIDI file was played back ten times and recorded and exported each time at the remote client.

For the MIDI OSC approach a Chuck program was run on each of the remote and local clients. On the local client a Chuck program was run that would accept input from a MIDI port. The MIDI messages received on this MIDI port were wrapped in OSC messages and sent to a parameterised IP address (the address of the remote client) and port (the network port the remote Chuck program was listening on). On the remote client a Chuck program was run that would listen on a network port for incoming OSC messages. This program would strip the MIDI messages from these OSC messages and send them to a MIDI port. The MIDI messages received on this port were recorded as a track in Logic Studio. For each track recorded a MIDI file was exported.

For each protocol each MIDI file was played back ten times and recorded and exported at the remote client, this yielded the data set in table 3

Table 3. Data collected for each protocol

Data	**Quantity**
Songs	4
RTP recordings	40
OSC recordings	40

Analysis was carried out on a Windows machine using MIDI-OX. Each file was played back using MIDI Bar with its MIDI OUT port set to the MIDI IN port of MIDI-OX. As MIDI-OX plays back midi it logs each message to a text file in a human readable format. Each of the control MIDI files were processed first. This process yielded four MIDI files, one for each of the original four MIDI files selected. Further log files were then created for each of the MIDI files exported from the network session for both RTP and OSC. The network MIDI log files were each compared against the relevant control file for each piece using ExamDiff Pro. ExamDiff Pro provided statistics on the differences in the files and these were used as the basis of the comparison between the two approaches.

In this experiment design the independent variable is the gestural data contained in the MIDI file for each selected piece of music. The controlled variable is the MIDI file recorded on the local client, this is the same in each case for a selected piece of music. The dependent variable is the MIDI file on the remote client, this file is different for each independent variable considered.

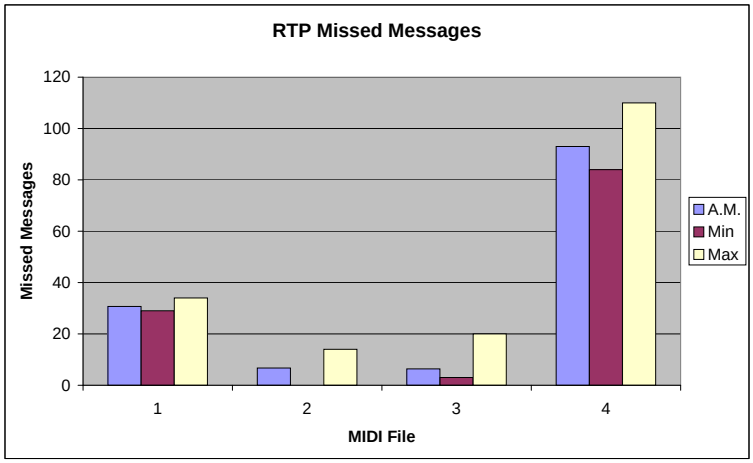
5 Experimental Evaluation and Results

This section analyses the data collected from the experiments described in section 4. The full results collected from the comparison of each of the network MIDI files against its control file are listed in appendix C.

5.1 Data Collected

The file comparison showed a relatively high number of messages had been added and removed. Closer inspection revealed that most of this could be explained by notes of a chord appearing in a different sequence in the different files. This is acceptable as the net result to the listener is the same.

This analysis is concerned with the quantity of messages that were lost during communication over the network. To get this figure the deleted lines total was taken from the added lines. This yielded ten results for each protocol for each original MIDI file. The arithmetic mean of the lost messages was taken for the ten runs for each protocol. This data is presented in tables 4 and 5. The minimum and maximum are also included to give an indication of the spread of data. This information is also depicted in figure 2.



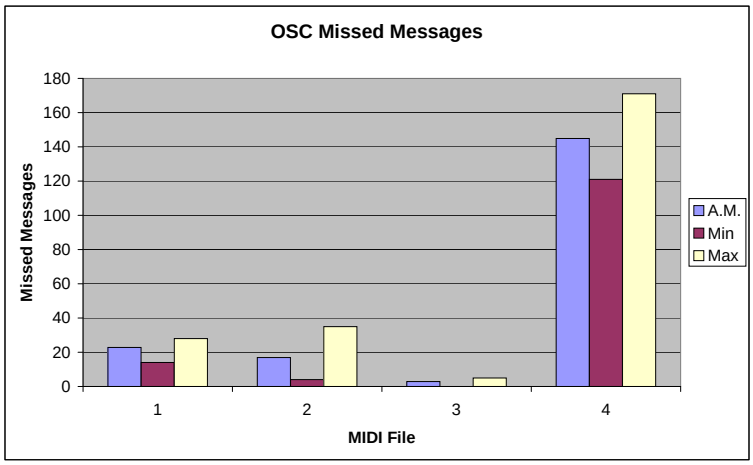


Fig. 2. Missed Messages for each protocol

Table 4. Network MIDI (RTP) Missed Messages

MIDI File	Arithmetic Mean	Minimum	Maximum
1	30.7	29	34
2	6.7	0	14
3	6.4	3	20
4	93	84	110

Table 5. Chuck Programs (OSC) Missed Messages

MIDI File	Arithmetic Mean	Minimum	Maximum
1	22.8	14	28
2	16.9	4	35
3	2.9	0	5
4	144.9	121	171

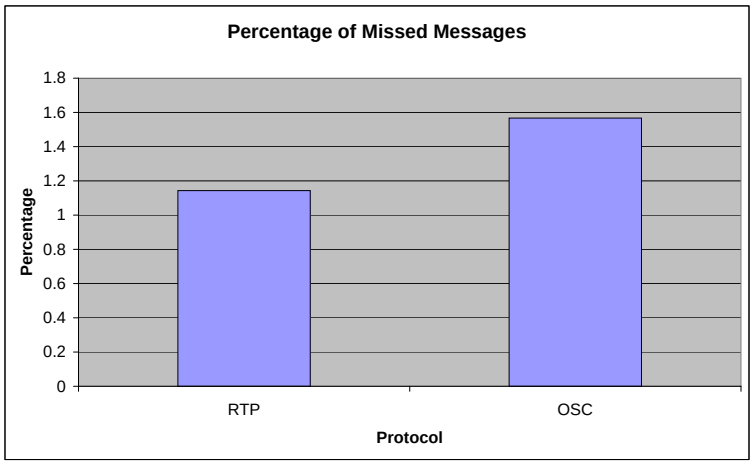


Fig. 3. Messages missed as a percentage of total messages sent per protocol

5.2 Analysis

The results show that for a MIDI file with greater complexity there are higher numbers of missed events. MIDI file 4 was the most complex and as a result has the most missed messages in comparison to its control file. MIDI file 1 was the next most complex and the other two MIDI files were relatively less complex again.

Overall the data shows that the RTP based approach performs better. To clarify this the total messages across all four files was calculated. The total missed messages was also calculated across all files for each protocol. Using the results of these calculations an overall percentage of missed messages per protocol was calculated. This data is represented in figure 3. It can be seen from this figure that both protocols performed well and that the difference in the quality of each approach is marginal. The missed messages for both protocols average less than 1.6%.

The missed messages fall into three categories, note on messages, note off messages and control messages. Control messages have the least impact on the listener as the effect of for example of a sustain pedal is quite subtle. Note on messages do mean that elements of the performance are missed but at such a low percentage this would be negligible. Missed note off events could have a more noticeable impact on the performance. In this case notes which should have ended may clash with subsequent chords. For each MIDI recording taken the researcher listened to the audio received at the remote client as it was recorded. There was no noticeable discrepancies in the performances. This correlates with the low percentage of missed messages. The virtual instrument used by the synthesiser in the experiments was a piano which would reduce the impact of missed note off messages as a characteristic of this instrument is relatively low sustain. Had the experiments used an instrument which sustained notes the impact of any missed note off messages may have been greater.

The data set used here is too restricted to determine if one approach should be favoured over the other. Further data and analysis would be required to conclude which approach would be best suited for example in the implementation of the networking component in a distributed music application like those outlined in section 3.5. It can been seen from the analysis that, for the development of a network music application, which would transport MIDI data across a network, either approach examined in this research would produce similar results. Based on the results presented here the choice of which approach to use in the development of network music system would be based on other factors such as target platform. For example if Apples OS X was the target platform then the Network MIDI Driver could be utilised in which case no development effort would be required for the network component of the system. If Windows was the target platform, the network component would have to be developed. In this case the Chuck MIDI over OSC approach can be seen to be more suitable.

6 Conclusion and Future Work

6.1 Conclusion

In previous research a case has been made for network music applications using gestural data. It had been shown that viable implementations were possible using MIDI over RTP and by wrapping MIDI in OSC. This research took both of these approaches and performed quantitative experiments to allow a direct comparison of the performance of each approach. It has been seen from the analysis of the results of these experiments that the RTP based approach gave marginally better results. As such other factors may need to be considered in selecting an approach to such an implementation as both approaches recorded adequate results for use in a network music application.

6.2 Future Work

This research was limited in the number of musical pieces considered and the number of runs recorded for each piece. Further pieces with more diverse styles could be run many more times to yield a greater data set for analysis. The analysis itself could also be extended to consider the data at a more granular level. This would allow consideration of misplaced events as well as missed events as analysed in this research.

The experiments ran as part of this research were conducted on a LAN using MIDI files to ensure consistent performances in a controlled environment. It might also be useful to collect data across a WAN to see what affects this would have on the data set. The analysis of performances by live musicians across both protocols would also be of interest. Extending from that multiple live performers on remote clients sending gestural data to a central server to record an ensemble MIDI performance using both approaches for comparison.

References

1. MMA: The complete midi 1.0 detailed specification (second edition). http://www.midi.org/techspecs/midispec.php (Nov 2001)
2. Perkins, C.: RTP: Audio and Video for the Internet. Addison Wesley, Reading, Massachusetts, USA (2003)
3. Wright, M.: Open sound control 1.0 specification. (2002)
4. Kirn, P.: Real World Digital Audio: Industrial-Strength Production Techniques. Peachpit Press, Berkeley, CA, USA (2006)
5. Wright, M., Freed, A.: Open sound control: A new protocol for communicating with sound synthesizers. In: International Computer Music Conference, Thessaloniki, Hellas, International Computer Music Association (1997) 101–104
6. Lazzaro, J., Wawrzynek, J.: A case for network musical performance. In: NOSSDAV '01: Proceedings of the 11th international workshop on Network and operating systems support for digital audio and video, New York, NY, USA, ACM (2001) 157–166
7. Roger B. Dannenberg, Sofia Cavaco, E.A.e.a.: The carnegie mellon laptop orchestra. In: In Proceedings of the 2007 International Computer Music Conference, Volume II, San Francisco, USA (2007) 340–343
8. Daniel Trueman, Perry Cook, S.S.G.W.: Plork: Princeton laptop orchestra, year 1. In: In Proceedings of International Computer Music Conference, New Orleans, USA (2006)
9. Biaz, S., Chapman, R.O., Williams, J.P.: Rtp and tcp based midi over ip protocols. In: ACM-SE 43: Proceedings of the 43rd annual Southeast regional conference, New York, NY, USA, ACM (2005) 112–117
10. MMA: White paper: Comparison of midi and osc. http://www.midi.org/aboutmidi/midi-osc.php (Nov 2008)
11. Freed, A., Schmeder, A., Zbyszynski, M.: Osc showcase for maker faire 2007. In: Maker Faire 2007, San Mateo, CA, USA (20/10/2007 2007)
12. Kirn, P.: Real World Digital Audio: Industrial-Strength Production Techniques. Peachpit Press, Berkeley, CA, USA (2006)
13. Wessel, D., Wright, M.: Problems and prospects for intimate musical control of computers. Comput. Music J. **26**(3) (2002) 11–22
14. Taylor, E.: The AB Guide to Music Theory Vol 1. Associated Board of the Royal School of Music, London W1B 1LU, United Kingdom (1989)
15. Moore, F.R.: The dysfunctions of midi. Comput. Music J. **12**(1) (1988) 19–28
16. McMillen, K.: Zipi: Origins and motivations. Computer Music Journal **18** (1994) 47–51 Winter 1994 vol.18, no.4.
17. Wright, M.: A comparison of midi and zipi. Computer Music Journal **18** (1994) 86–91 Winter 1994 vol.18, no.4.
18. Fildes, J.: 'oldest' computer music unveiled. http://news.bbc.co.uk/2/hi/technology/7458479.stm (June 2008)
19. Gresham-Lancaster, S.: The aesthetics and history of the hub: The effects of changing technology on network computer music. Leonardo Music Journal **8** (1998) 39–44
20. Tanaka, A., Tokui, N., Momeni, A.: Facilitating collective musical creativity. In: MULTIMEDIA '05: Proceedings of the 13th annual ACM international conference on Multimedia, New York, NY, USA, ACM (2005) 191–198
21. Rosenberg, J., Schulzrinne, H., Camarillo, G., Johnston, A., Peterson, J., Sparks, R., Handley, M., Schooler, E.: SIP: Session Initiation Protocol. RFC 3261 (Proposed Standard) (June 2002) Updated by RFCs 3265, 3853, 4320, 4916, 5393.

22. Lazzaro, J., Wawrzynek, J.: An rtp payload format for midi. In: AES: Proceedings of the 117th convention, San Francisco, California (2004)
23. Handley, M., Jacobson, V., Perkins, C.: SDP: Session Description Protocol. RFC 4566 (Proposed Standard) (July 2006)
24. Lazzaro, J., Wawrzynek, J.: RTP Payload Format for MIDI. RFC 4695 (Proposed Standard) (November 2006)
25. Nagle, J.: Congestion control in IP/TCP internetworks. RFC 896 (January 1984)
26. R.B. Dannenberg, P.v.d.L.: A system supporting flexible distributed real-time music processing. In: Proceedings of the 2001 International Computer Music Conference, ICMA. (2001) 267–270
27. Wright, M.: Open sound control: an enabling technology for musical networking. Org. Sound **10**(3) (2005) 193–200
28. McCartney, J.: Rethinking the computer music language: Supercollider. Comput. Music J. **26**(4) (2002) 61–68
29. Ramakrishnan, C., Freeman, J., Varnik, K.: The architecture of auracle: a real-time, distributed, collaborative instrument. In: NIME '04: Proceedings of the 2004 conference on New interfaces for musical expression, Singapore, Singapore, National University of Singapore (2004) 100–103
30. Barbosa, A.: Ten-hand piano: A networked music installation. In: Proceedings of the International Conference on New Interfaces for Musical Expression (NIME 2008), Genova, Italy (2008)
31. Barbosa, A., Cardoso, J., Geiger, G.: Network latency adaptive tempo in the public sound objects system. In: NIME '05: Proceedings of the 2005 conference on New interfaces for musical expression, Singapore, Singapore, National University of Singapore (2004) 184–187
32. freebsd.org: Ipfw. http://www.freebsd.org/doc/en/books/handbook/firewalls-ipfw.html (Aug 2009)
33. Microsystems, S.: Java sound api. http://java.sun.com/products/java-media/sound/ (August 2009)
34. Microsystems, S.: Java sound programmers guide, chapter 8 overview of the midi package. http://java.sun.com/products/java-media/sound/ (August 2009)
35. Lazzaro, J., Wawrzynek, J.: An Implementation Guide for RTP MIDI. RFC 4696 (Informational) (November 2006)
36. Ge Wang, P.C.: Chuck : Strongly-timed, concurrent, and on-the-fly audio programming language. http://chuck.cs.princeton.edu/ (August 2009)

Appendix A: Songs Used

This appendix contains details of the songs used in the experiments. Each song is well known solo piano piece chosen for their relative complexity.

1. BWV 846 from The Well-Tempered Clavier (Prelude and Fugue No. 1 in C major), composer Johann Sebastian Bach
2. BWV 847 from The Well-Tempered Clavier (Prelude and Fugue No. 2 in C minor), composer Johann Sebastian Bach
3. Für Elise (A Bagatelle in A minor), composer Ludwig van Beethoven
4. Clair de Lune, (in D-flat major, third movement from the suite bergamasque), composer Achille-Claude Debussy

Appendix B: Chuck Programs

The first program is miditoosc.ck listens on a MIDI port. Any MIDI messages received on that MIDI port are wrapped in OSC messages and sent to an IP address and network port number. The IP address, network port number and MIDI port number are provided as parameters passed when calling the program.

```
if (me.args() != 3) {
  <<<"usage: chuck miditoosc.ck:HOST:PORT:MIDIDEVICE">>>;
  me.exit();
 }

me.arg(0) => string host;
me.arg(1) => Std.atoi => int port;
me.arg(2) => Std.atoi => int dev;

MidiIn min;
if (!min.open(dev)) {
  <<<"Could not open MIDI device " + dev>>>;
  me.exit();
 }
else
{
  <<<"Opened MIDI in device " + min.num(), " -> ",
  min.name()  >>>;
}
MidiMsg msg;

OscSend send;
send.setHost(host, port);
while(min => now) {
  while(min.recv(msg)) {
    send.startMsg("/midi/raw", "iii");
        <<<"sending midi msg"+msg.data1+"    "+msg.data2+"
"+msg.data3>>>;
    msg.data1 => send.addInt;
    msg.data2 => send.addInt;
    msg.data3 => send.addInt;
  }
 }
```

The second program is osctomidi.ck listens on a network port. The messages received in this port are MIDI messages wrapped in OSC. The MIDI messages are stripped from the OSC messages and sent to a MIDI out port. The network port and MIDI out port are provided as parameters when calling the program.

```
if (me.args() != 2) {
  <<<"usage: chuck osctomidi.ck:PORT:MIDIDEVICE">>>;
  me.exit();
 }

me.arg(0) => Std.atoi => int port;
me.arg(1) => Std.atoi => int dev;

OscRecv recv;
port => recv.port;
recv.event("/midi/raw, iii") @=> OscEvent oe;
recv.listen();

MidiOut mout;
if (!mout.open(dev)) {
  <<<"Could not open MIDI device " + dev>>>;
  me.exit();
 }
 else
 {
  <<<"Opened MIDI out device "+ mout.num(), " -> ",
        mout.name()  >>>;
 }
MidiMsg msg;
while (oe => now) {
  while(oe.nextMsg() != 0) {
    oe.getInt() => msg.data1;
    oe.getInt() => msg.data2;
    oe.getInt() => msg.data3;
    mout.send(msg);
        <<<"Received: " + msg.data1 + "    "
                + msg.data2 + "    " + msg.data3>>>;
  }
 }
```

Appendix C: Full Results

This appendix contains the full results for the experiments

RTP

bach 846	Added	Removed	Missed	%Missed
1	371	401	30	1.085383502
2	369	399	30	1.085383502
3	369	399	30	1.085383502
4	367	401	34	1.230101302
5	367	399	32	1.157742402
6	366	396	30	1.085383502
7	370	400	30	1.085383502
8	369	401	32	1.157742402
9	366	396	30	1.085383502
10	367	396	29	1.049204052
Min	366	396	29	
Max	371	401	34	
Arith. Mean	368.1	398.8	30.7	

bach 847	Added	Removed	Missed	%Missed
1	545	552	7	0.186666667
2	554	559	5	0.133333333
3	549	557	8	0.213333333
4	549	557	8	0.213333333
5	545	545	0	0
6	551	565	14	0.373333333
7	541	545	4	0.106666667
8	544	550	6	0.16
9	546	551	5	0.133333333
10	558	568	10	0.266666667
Min	541	545	0	
Max	558	568	14	
Arith. Mean	548.2	554.9	6.7	

fur elise	Added	Removed	Missed	%Missed
1	275	280	5	0.223413762
2	277	280	3	0.134048257
3	275	280	5	0.223413762
4	274	279	5	0.223413762
5	275	280	5	0.223413762
6	275	280	5	0.223413762
7	276	281	5	0.223413762
8	278	283	5	0.223413762
9	273	279	6	0.268096515
10	274	294	20	0.893655049
Min	273	279	3	
Max	278	294	20	
Arith. Mean	275.2	281.6	6.4	

deb clair	Added	Removed	Missed	%Missed
1	645	738	93	2.893590541
2	638	728	90	2.800248911
3	643	738	95	2.955818295
4	640	750	110	3.422526447
5	652	740	88	2.738021157
6	645	742	97	3.018046049
7	645	731	86	2.675793404
8	645	738	93	2.893590541
9	644	738	94	2.924704418
10	641	725	84	2.61356565
Min	638	725	84	
Max	652	750	110	
Arith. Mean	643.8	736.8	93	

OSC

bach 846	Added	Removed	Missed	%Missed
1	343	371	28	1.013024602
2	353	376	23	0.832127352
3	347	373	26	0.940665702
4	352	374	22	0.795947902
5	351	372	21	0.759768452
6	348	371	23	0.832127352
7	344	365	21	0.759768452
8	354	368	14	0.506512301
9	341	369	28	1.013024602
10	360	382	22	0.795947902
Min	341	365	14	
Max	360	382	28	
Arith. Mean	349.3	372.1	22.8	

bach 847	Added	Removed	Missed	%Missed
1	554	576	22	0.586666667
2	549	553	4	0.106666667
3	569	596	27	0.72
4	558	593	35	0.933333333
5	544	562	18	0.48
6	552	560	8	0.213333333
7	556	568	12	0.32
8	553	559	6	0.16
9	570	583	13	0.346666667
10	537	561	24	0.64
Min	537	553	4	
Max	570	596	35	
Arith. Mean	554.2	571.1	16.9	

fur elise	Added	Removed	Missed	%Missed
1	247	252	5	0.223413762
2	251	255	4	0.17873101
3	256	260	4	0.17873101
4	238	238	0	0
5	247	252	5	0.223413762
6	238	241	3	0.134048257
7	242	246	4	0.17873101
8	251	251	0	0
9	246	249	3	0.134048257
10	246	247	1	0.044682752
Min	238	238	0	
Max	256	260	5	
Arith. Mean	246.2	249.1	2.9	

deb clair	Added	Removed	Missed	%Missed
1	578	734	156	4.853764779
2	556	727	171	5.320472931
3	589	742	153	4.760423149
4	602	729	127	3.951462352
5	586	710	124	3.858120722
6	592	713	121	3.764779091
7	590	750	160	4.978220286
8	574	722	148	4.604853765
9	576	714	138	4.293714997
10	583	734	151	4.698195395
Min	556	710	121	
Max	602	750	171	
Arith. Mean	582.6	727.5	144.9	