MINI — Making MIDI fit for Real-time Musical Interaction over the Internet

Michael Welzl
Institute for Computer Science
University of Innsbruck
Innsbruck, Austria
michael.welzl@uibk.ac.at

Max Mühlhäuser
Department of Computer Science, Telecooperation
Darmstadt University of Technology
Darmstadt, Germany
max@tk.informatik.tu-darmstadt.de

Jan Borchers
Media Computing Group
RWTH Aachen University
Aachen, Germany
jan@rwth-aachen.de

Abstract

Distributed live music performances over the Internet have become rather popular. For true real time exchange of electronic instrument output, the classical MIDI format is still the only viable choice — but MIDI is inappropriate for the Internet. We propose a MIDI compatible network data format called MINI. MINI eliminates the very disturbing arpeggio effect caused by packet delay jitter in the Internet. Moreover, MINI yields smaller packets and the data size can be further reduced by trading in musical features of marginal interest. Standard MIDI instruments can be used since MIDI-MINI transcoding can be transparently introduced. The paper introduces the design rationale and details of MINI and presents performance results from realistic experiments.

1. Introduction

Musicians have played together over the Internet and other long-distance networks on several occasions. If such an interaction should feel like a "jamsession", i.e. with the spontaneous real-time interaction that is the most essential ingredient of jazz music for example, it is probably safe to say that there are no side effects except for total communication breakdown that are as unpleasant and undesirable as delay and jitter.

On the Internet, however, users typically obtain a so-called "best effort" service — that is, the network does its best to forward data packets from the source to the destination as quickly and efficiently as possible, but provides no guarantees whatsoever. This includes the potential for delay and jitter, both of which is known to be mainly attributed to the queue at the bottleneck router, which grows in the presence of congestion (i.e. when more traffic is arriving than can immediately be delivered).

Bearing this in mind, the best way for applications to reduce delay is therefore to reduce the likelihood of of congestion to occur and hence send as little as possible. While it is always possible that other applications send enough data for the bottleneck queue to grow, it is an obvious strategy for a delay-sensitive application to itself avoid being the cause of congestion. One way to do this is to send a small number of packets. Another method is to keep packets small — since a sender has to wait until all the data that should fit in a packet are available before it can send the packet, the use of large packets causes delay on the sender side. Common VoIP applications like Skype, which need to minimize the impact of delay and jitter, address both of these problems at the same time: they send extremely small packets at a very low rate.

While MIDI is quite small by nature, it is not as small as it could be, and any unnecessary waste of bandwidth increases the chance of congestion; it should therefore be the foremost goal of any MIDI based real-time communication system to reduce the amount of data even further, even if it may be at the cost of some of the features that the format offers. Since most public networks are typically overprovisioned, a user would usually not experience congestion when sending MIDI data via a standard desktop PC that is connected to the Internet, but this can be entirely different in WiFi environments, for instance, and in situations with a large number of interacting musicians.

We present a new format, *MINI (Musical Instrument Net-*

work Interface), which addresses the problems with MIDI across long-distance networks in three ways:

1. It is smaller than MIDI
2. It encodes chords as what they are, instead of simply encoding them as a series of individual notes as MIDI does; this way, it avoids the arpeggio effect that occurs with MIDI (we will elaborate on this in section 2.1)
3. It provides the necessary flexibility for making a trade-off between feature richness and having a low sending rate

The idea of sacrificing features for the sake of preserving little delay is in line with the wealth of work on *adaptive multimedia applications* for the Internet, where it was, for example, suggested to reduce the quality of video frames in the presence of congestion in the network [16]. This would be done because the quality reduction also reduces the sending rate, and hence reduces packet loss, which may lead to a generally more agreeable result at the receiver. Congestion being a dynamic effect, one might be tempted to believe that such quality adaptations should permanently react to the current state of the network, but such behavior can lead to quality fluctuations which are quite undesirable — in fact, users have been found to prefer a continuously poor quality over frequent changes [12, 20]. One way to handle the discrepancy between the dynamic network and its not-so-dynamic user is to give the user the choice, i.e. let the user switch between quality levels; this was successfully done by the authors of [2], and it is the strategy that we have foreseen for MINI.

The MINI encoding scheme is described in the next section. We elaborate on an implementation of MINI in a GUI supported application in section 3, explain our test setup and present results in section 4. Section 5 concludes with an overview of related work.

2 MINI

MIDI encodes any musical note that is played with a *Note-On* message, and it encodes the termination of a note with a *Note-Off* message. A chord that is played or muted is encoded via multiple *Note-On* and *Note-Off* messages, respectively — one for each note that the chord consists of. As the MIDI standard [1] assumes an interconnection technology that will not yield audible delays, and has a fixed bandwidth, sending MIDI data across the Internet actually violates the specification. The unexpected delay that such usage can interject between packets containing notes that belong to a chord can effectively turn a chord into an arpeggio. This is of course highly undesirable as it is entirely different from what the musician who originally played the chord wanted it to sound like. We term this effect *intra-chord jitter*; clearly, delay fluctuations can also occur *between* chords or single notes (*inter-chord jitter*), and these should also be avoided. MINI provides mechanisms that address both of these issues.

The MINI format is restricted in scope to the transport of MIDI musical data. Distributed musical performances will usually require additional agreements between the participating distributed software components (MINI-MIDI transcoders, user interface, etc.). These additional agreements must be exchanged in an application specific protocol and format which we refer to as SETUP. A SETUP phase is supposed to precede the exchange of MINI messages, additional SETUP messages may be intertwined with MINI messages during the performance. As we will show later in the paper, MINI offers a number of choices in the trade-off between expressiveness and message size. Some of them can be changed from one message to another one; these choices are encoded as part of the MINI format. Other choices are supposed to change rarely or to be fixed for an entire musical performance; these choices must be negotiated between the distributed application components using SETUP messages. As argued above, SETUP is out of scope of MINI since it is supposed to be largely application specific.

2.1 Chord encoding

As a solution to the intra-chord jitter problem, MINI encodes chords via a single code as opposed to encoding them as individual musical notes. While a musician can start and stop to play the notes that make up a chord at any time, it is often the case that several of them are simultaneously played or muted (as perceived by a listener, i.e. the events happen within a period of time that is short enough to yield the impression of concurrence). Thus, in addition to solving the arpeggio problem mentioned above, space can be saved by encoding the beginning or end of multiple notes in a single message; in an implementation, the decision whether notes are to be considered as being played at the same time or not can simply be based on a fixed delay threshold.

The chord encoding scheme in MINI is based on regarding the set M of all possible chords with k notes within a given range of n notes as a k^{th} order combination of n elements without repetition and without ordering:

$$M = \binom{n}{k} = \frac{n!}{k!(n-k)!} \tag{1}$$

For example, by setting $n = 15$ and $k = 3$, it can be calculated that there are 455 different possible triads in a range of 15 notes. Thus, by unambiguously mapping each of the numbers from 1 to 455 to a chord in a table, it would theoretically be possible to encode any such chord combination with only 9 bits.

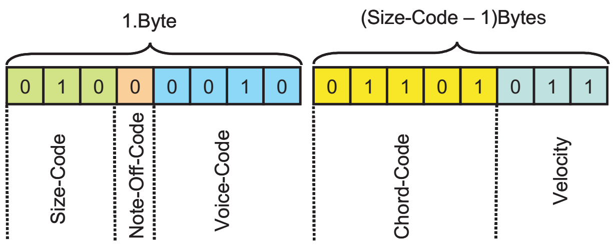


Figure 1. MINI Note-On message with Velocity

Encoding and decoding MINI chords with tables would be straightforward, but it would require significant memory space. Fortunately, there is no need to maintain tables, as the desired mapping is merely a combinatorial problem — consider, for example, the unambiguous mapping between arrays of higher dimensions and unidimensional arrays that the C programming language implicitly provides. For MINI, we use an algorithm from [15] which provides a bijective mapping between an array representing a unique subset of size K from a set of size N, and an integer number, which is called the "rank" (or order) of the subset.[1] Notably, this algorithm does not need N as input; again, consider the mapping in the C programming language, which also works without knowing the size of the array that it operates on.

One degree of freedom for adapting MINI to the network capacity is given by the arbitrary choice of n and k — for instance, if MINI is used to connect instruments that have a small tonal range, n can be small, and the instrument usually also imposes an upper limit on k (e.g., $k > 10$ would normally not make much sense when the instrument is a keyboard). The smaller these two variables are, the smaller the data format becomes; this is therefore one of the factors that can be used to trade comfort against sending rate when bandwidth becomes scarce.

2.2 Layout of Note-On and Note-Off messages

Figure 1 shows an example MINI word. Since each word can vary in size, it is necessary to encode the length of the word itself; this is the purpose of the three-bit "Size Code" field, which encodes the size of the word in bytes. The next field, "Note-Off-Code", is set to 1 when the word encodes a MIDI "Note-Off" message (i.e. the encoded note or chord ends), and 0 when it encodes a "Note-On" message (i.e. the note or chord begins).

[1]We used the C implementation of the encoding and decoding functions which are available from `http://people.scs.fsu.edu/~burkardt/` by the names of "KSUB_RANK" and "KSUB_UNRANK", respectively.

The "Voice-Code" field indicates the number of musical notes that the encoded chord contains (k in equation 1). The four bits that this field consists of allow for a total number of 16 voices. This choice was made with common instruments such as electronic keyboards in mind, where it would be unlikely that more bits would be required for real-time musical interaction. Notably, this does not impose a true limit on the voices in a chord, it only limits the number of chords that can be encoded within a single MINI word. The Chord-Code is the rank of the cord as explained above.

"Velocity" is the MINI representation of Velocity, which is embedded in MIDI Note-On and Note-Off messages, representing the speed at which (in case of a keyboard) a key is hit or released, respectively. Since the encoding scheme in MINI concerns chords as well as individual notes, it seemed obvious to let a single Velocity value affect a whole chord in our format. For most instruments, it is technically quite challenging and therefore somewhat unusual to have different velocity values for individual notes in a chord — on a keyboard, this corresponds with hitting several keys simultaneously, albeit with different speeds. While this choice can hide some musical nuances from the listener, we believe that this is a sacrifice that most musicians would be willing to make in exchange for potentially reduced delay (because the data set becomes smaller).

In MIDI, Velocity has a resolution of 7 bits, which may be too much for most practical situations where musicians jam over a network (some instruments may not even be able to generate Velocity values with such a fine granularity; in our experiments with electronic keyboards, the difference between 3 and 7 bits was barely audible). Therefore, the Velocity resolution is configurable with a range from 1 to 7 bits in MINI. The resolution choice must be negotiated between the distribution application components by way of SETUP messages.

2.3 Timestamp messages

Inter-chord jitter is caused by queuing delay *between* MINI words, and can therefore only be counteracted by restoring the correct timing after their reception. Since this requires precise knowledge about the time at which notes were played, the otherwise unused MINI Size-Code value of "001" encodes a "Timestamp" message, which a MINI sender can insert in front of a MINI Note-On or Note-Off message in order to tell the receiver about the time which has passed since the last MIDI event. This message is shown in Figure 2. As the Timestamp field consists of 13 bits, and the time is given in milliseconds, a maximum delay of approximately 8 seconds can be encoded with one such message; in order to encode longer durations, a Timestamp message can be followed by another Timestamp message, the value of which must be added to the value of the pre-

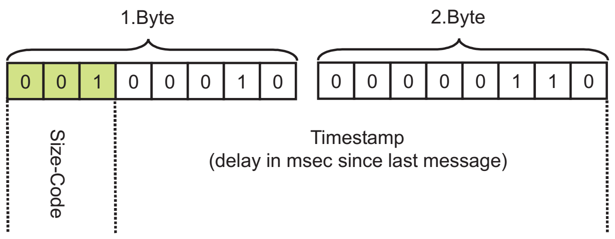


Figure 2. MINI Timestamp message

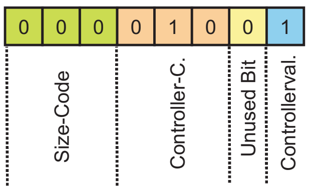


Figure 3. MINI pedal-message when 8 possible controllers were initially selected

ceding one by the receiver.

A MINI receiver can restore the correct timing by means of a playout buffer. Arriving MINI words are immediately placed in this buffer, and they are played from the buffer at the right time. The length of this buffer is another tradeoff for users of MINI: a long buffer can compensate for severe timing fluctuations, but it adds significant delay before incoming messages can be played, whereas a short buffer makes the system feel more interactive but may not always be able to compensate for inter-chord jitter. If the goal is to make the system feel as interactive as possible, and inter-chord jitter is more acceptable than some fixed additional delay, this feature should not be used.

2.4 Controller messages

In MIDI, Note-On and Note-Off belong to the class of "Channel Voice Messages". These messages are bound to a logical Channel, of which there are up to 16. The concept of Channels was not included in MINI, as it takes up space in the format, which we considered unnecessary because multiplexing is already provided by underlying protocols (e.g. via ports in UDP, onto which Channel numbers could be mapped). In other words, it is assumed that one MINI stream represents one logical channel. The MIDI standard also foresees the class of "System Common Messages" — these include "Song Select" (only relevant for sequencers), "Tune Request" (irrelevant when musicians do not truly hear each other) and system exclusive messages which are device dependent — and "System Realtime Messages", which are mainly designed for sequencers. These messages provide functionality similar to the "ping" command in order to check whether a device is still reachable. The goal being a data format which is as slim as possible, we decided that none of these messages need to be incorporated in MINI.

Unlike System Common and Realtime Messages, controller messages are relevant for MINI because they concern real-time musical interaction. They are used to encode changes to the sound which are typically generated via a mechanical device such as a knob, slider or pedal. Other than Velocity, these messages are not embedded in Note-On or Note-Off messages in MIDI. For MINI, we followed the same method — that is, a MINI word is either a Note-(On or Off) message or a controller message, the latter of which is detected by checking whether all initial three bits (the Size Code) are zero, which is a previously undefined value of this field. If that test succeeds, the rest of the word is interpreted as follows:

- The Size-Code is followed by n "Controller-Code" bits, where n is initially negotiated in the range from 1 to 3. These bits are used to identify the controller.
- The m rightmost bits of the MINI word encode the value of the controller; m depends on the controller.
- Since the size of MINI words are a multiple of 8 bits, unused ("padding") bits are inserted between the Controller-Code and the value of the controller as needed in order to fill up the space.

As an example of this layout, Figure 3 depicts a MINI controller message where $n = 3$ and $m = 1$.

MIDI allows for up to 128 different controller types. Several of these numbers are still undefined, and some controllers are quite exotic, and hence not available on most devices or soundcards. We decided to include only the following subset of controllers:

Program Change: 7 bits are used to select an instrument (a so-called "patch"), and they are encoded in MINI as they are in MIDI.

Pitch Bend, Modulation, Volume, Reverb, Chorus: some of these controllers can have a 14- or 7-bit resolution, and pitch bend only has a 14-bit resolution. We decided not to support the 14-bit resolution in MINI and always map any such value onto a 7-bit value.

Sustain Pedal, Sostenuto Pedal: here, in MIDI, 7 bits are used to encode a binary value (a value in the range 0-63 means "off", while a value in the range 64-127 means "on"). We use a single bit to encode the state of the pedal.

Strictly speaking, Program Change and Pitch Bend are not controller messages because they are defined as "Channel Voice Messages" in MIDI; we decided to group them together with controller messages in MINI for the sake of simplicity.

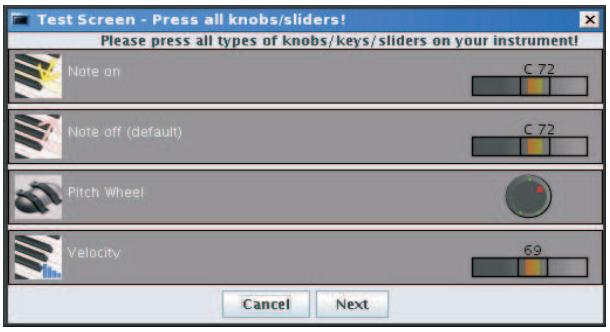
3 Implementation

Equipped with a format for efficiently transmitting MIDI data across long-distance networks, we were able to build a comprehensive application for real-time music playing over the Internet, which we called "Netmusic".[2] The design goals of this software were clear: it had to be a fun tool which enables musicians to jam together over the Internet, and it had to have all the features that would allow its users to fully exploit the capabilities of MINI. Netmusic consists of code which is written in C (the core components, for efficiency reasons) and Java (the user interface), was designed for Linux and tested with a Fedora Core 4 system with kernel version 2.6.17.1-2142. Here is a rough overview of its functionality:

- It connects to another host and maintains a TCP connection for exchanging parameters as well as starting and ending the session. MINI data are exchanged via UDP.
- It captures MIDI via the ALSA[3] library; if the delay between notes is less than a defined threshold (under control of the user), they are regarded as chords. Then, it converts them to MINI and sends them to the other host.
- Whenever a MINI message arrives, it is immediately converted to MIDI and played via ALSA (i.e. no buffering is used, and there are no Timestamp messages). With the standard interface that ALSA provides, a user can map readable MIDI ports onto writeable ones, including software-based input/output systems — that is, the newly generated MIDI messages can either be played on a connected MIDI device or on a software synthesizer, and this choice is not visible to our Netmusic application.
- The user can use the GUI to gage the inherent feature richness vs. sending rate trade-off of MINI; the number of controllers to be used is determined by first showing a window which asks the user to activate all of them (shown in Figure 4). In this window, the number of visible controllers grows with every one that was operated. Then, the existing controllers and some additional features can be selected and tuned in a separate window, which is shown in Figure 5. Here, the slider on the right controls the resolution. Figure 5 shows another feature of the Netmusic application: its ability to save even more space by sending Note-On messages without ensuing Note-Off messages. This mechanism makes sense for a few sounds ("patches") such as a xylophone or vibes, where the duration of the sound is fixed; here, no harm will come from simply omitting Note-Off messages. This application-specific mechanism is negotiated between the hosts via SETUP messages, using the TCP connection.

[2]This application is available from
http://www.welzl.at/research/projects/netmusic/

[3]http://www.alsa-project.org


Test Screen - Press all knobs/sliders!

Please press all types of knobs/keys/sliders on your instrument!

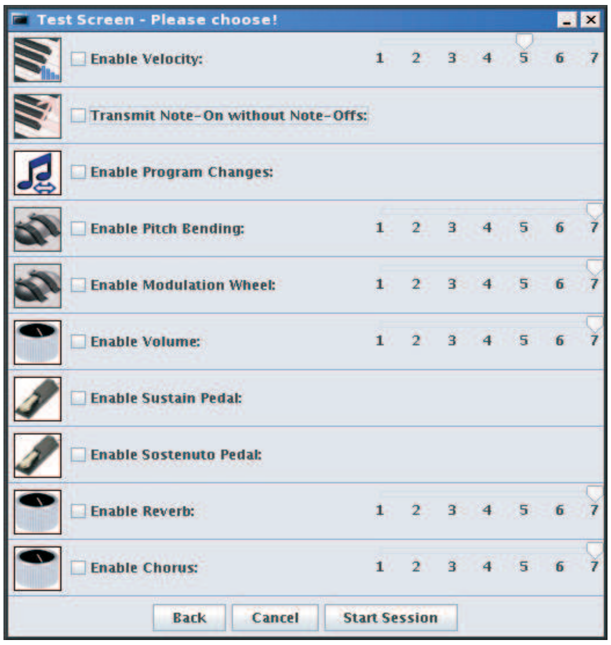
Note on — C 72

Note off (default) — C 72

Pitch Wheel

Velocity — 69

Figure 4. Controller detection window


Test Screen - Please choose!

Enable Velocity: 1 2 3 4 5 6 7

Transmit Note-On without Note-Offs:

Enable Program Changes:

Enable Pitch Bending: 1 2 3 4 5 6 7

Enable Modulation Wheel: 1 2 3 4 5 6 7

Enable Volume: 1 2 3 4 5 6 7

Enable Sustain Pedal:

Enable Sostenuto Pedal:

Enable Reverb: 1 2 3 4 5 6 7

Enable Chorus: 1 2 3 4 5 6 7

Figure 5. Options window

The GUI contains four different panels which provide visual feedback to the user: the "property window" shows all the relevant network details, such as the duration of the connection, IP-addresses and port numbers. There is a virtual keyboard, which always shows all notes that are played.

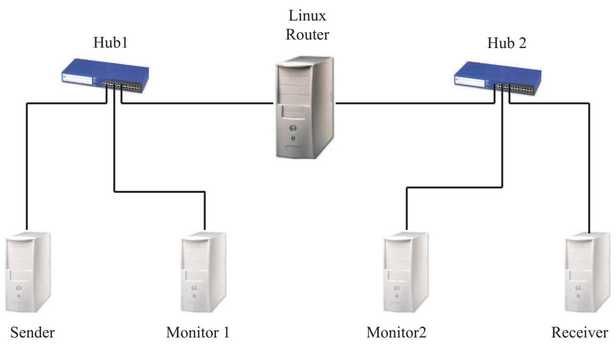


Figure 6. The testbed

All of the selected controllers are shown in the "controller window", and there is a "log window" which provides status information regarding connection setup and informs the user about parameter changes as well as error messages.

4 Test setup and results

In order to judge the benefits of MINI under assessable conditions, we tested it using our real implementation within the Netmusic application. Since Internet based tests do not always exhibit the full spectrum of adverse effects that one might encounter, we followed a typical approach for network tests where the impact of congestion is of concern: we constructed a local testbed. Our setup was successfully used on several occasions before, e.g. in [14] and [4]. For the sake of simplicity, we only consider a unidirectional flow — that is, our scenario is the same as if only one musician plays and the other one listens. This does not impair the validity of our results, as a flow in the other direction is completely independent of the flow under consideration.

Our testbed, shown in Figure 6, consists of five machines which are interconnected using 100 Mbps Ethernet links. Since we wanted to be absolutely sure that this activity does not interfere with the timing of our Netmusic process, we used two separate machines to generate and receive background traffic (Monitor 1 and Monitor 2 in Figure 6). The same machines were also used to log traffic; Monitor 1 would see traffic that is sent by the sender, i.e. before it experiences congestion, whereas Monitor 2 would see the same traffic pattern as the receiver. This way, it is easy to notice packet drops, as they are represented by the gap between the two lines if one plots the rates perceived by the two Monitors; it is however worth pointing out that peaks on the two lines can sometimes differ due to the delay that is caused by the router's queue in the presence of congestion.

In order to cause any congestion at all, the maximum traffic rate had to be limited by using the `tc` (Traffic Control) Linux command and Class Based Queuing with only one class for the receiver-side link of the router, which is a PC running Linux (Fedora Core 4, kernel 2.6.17.1-2142). We did not use Token Buckets due to their influence on traffic characteristics (cf. [5]). The monitors used tcpdump[4] to measure the traffic that traversed Hub 1 and Hub 2, respectively. Loss was calculated as the difference between the bytes sent (logged by Monitor 1) and the throughput (logged by Monitor 2).

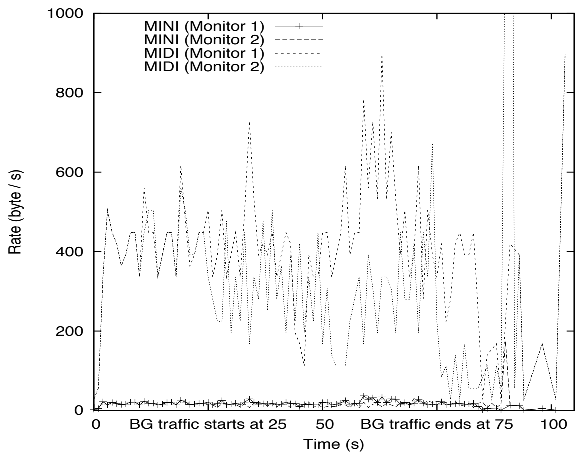


Figure 7. Rates of MINI and MIDI perceived by Monitors 1 and 2 (background traffic 80 packets per second)

Background traffic was generated as a Constant Bit Rate (UDP) data flow of 100 byte packets using the `mgen` traffic generator.[5] It was sent from the router to Monitor 2, which means that it could not cause collisions but only lead to congestion in the queue of the router's outgoing network interface. We generated 11 classes of background traffic, which always lasted for 50 seconds, starting from the 25th second, and consisted of 10 to 100 (in steps of 10) and 120 packets per second, respectively. These rates were customized according to the Netmusic application and the network setup so that the impact of increasing background traffic on the application's behavior could be investigated. Additionally, initial `mgen` packets were used to synchronize the test machines. In order to obtain reproducible results, we transmitted the piece "Préludes Nr. 4, Largo, Opus 28" by Frédéric Chopin; chords occur frequently in this 1:46 minute piece. With background traffic of 60, 70 and 80 packets per second, the continuous qualitative degradation of MIDI is quite clear to the listener, as the arpeggio effect becomes more and more pronounced.

Figure 7 shows the rates perceived by Monitors 1 and 2 with a background traffic of 80 packets per second. Clearly, for MIDI, the incoming rate is much higher than the outgo-

[4]`http://www.tcpdump.org`
[5]`http://mgen.pf.itd.nrl.navy.mil/mgen.html`

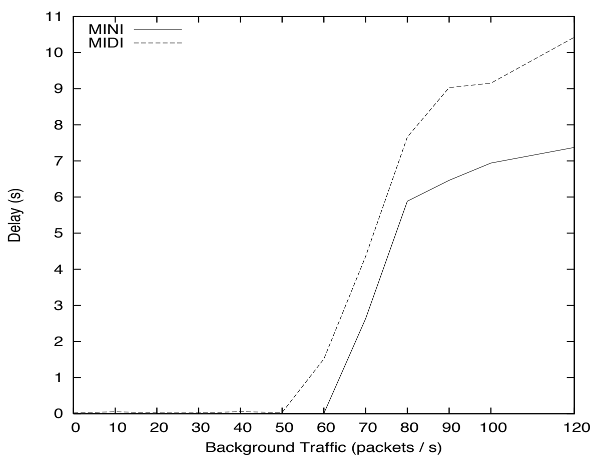


Figure 8. Average delay of MIDI and MINI

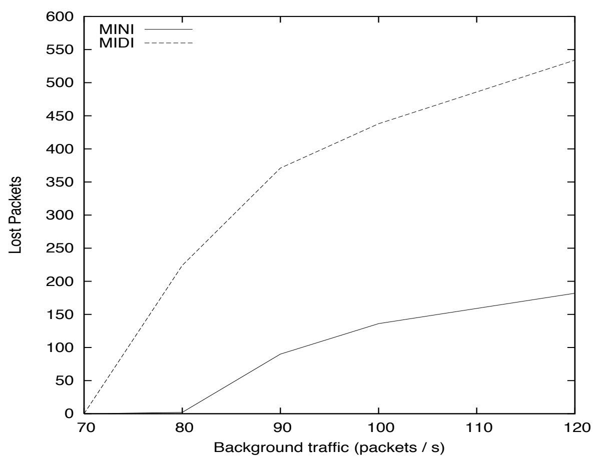


Figure 9. Average loss of MIDI and MINI

ing rate, and hence, packets were dropped. The outlier of the rate perceived by Monitor 2 that coincides with the end of background traffic after 75 seconds is the result of the router's queue emptying. The diagram also shows that the rate of MINI is generally much lower, and that the MINI lines corresponding with Monitors 1 and 2 are generally close to each other, i.e. there is little or no queuing delay and packet loss.

Figures 8 and 9 show cumulative results (average delay and total loss) of all the measurement studies; the tests with background traffic of 0 to 60 packets per second are not included in Figure 9 as no packets were dropped in these scenarios. The diagrams clearly show that we did not only reach our primary goal of reducing the delay that a musician can experience during a real-time jam over the Internet, but also that packet loss could be reduced. Since a lost packet means that either a note or a controller message was lost, this is a notable improvement of the overall outcome. The result is confirmed by the significantly enhanced quality that is evident when listening to MINI and MIDI in our tests.

5 Conclusion

The idea of playing music together via some network, or even the Internet, is far from new. We documented the state of the art in this field in 1998 [19]; back then, a large number of experiments were already conducted, ranging from Frank Sinatra singing a duet with Bono of U2 via a dedicated fiber connection to a system called "Res Rocket" (later called "Rocket Power"). In the latter system, which was even linked to the popular music tool "Cubase VST" for a while, musicians actively worked together by editing sequencer tracks in real-time, but the interactivity was limited, as musical updates were only disseminated when a button was clicked.

Res Rocket is not available anymore[6], and its place is now taken by even less interactive portals for sharing music such as [13], [18] and [7]. The existence of these portals, the first of which claims to have more than 20000 members at the time of writing, clearly indicates the continuing demand for ways to cooperate between musicians who may not be in the same place. There are, of course, also more recent examples of related work, e.g. [3].

One particularly noteworthy system for true real-time musical interaction via IP-based networks is "Networked Musical Performance (NMP)" [8]; here, MIDI is transmitted over IP using the "Real-Time Transport Protocol" (RTP) [17] by means of a new RTP packetization format [9], which is specified in [11, 10], and delayed or lost packets are compensated for by adding timestamps on the sender side and using them to properly handle problems on the receiver side. For example, if a note is delayed, it may sometimes be better not to play it at all, thereby making the outcome sound closer to mistakes produced by imperfect musicians than to the unpleasant effects that are produced by data-starved audio codecs. MPEG-4 Structured Audio (SA) [6] is used for music synthesis.

NMP tackles an interesting part of the design space: on the one hand, it does not proactively reduce latency or packet loss like MINI does; as a matter of fact, its additional RTP header would slightly increase the chance of packet loss in the presence of congestion. On the other hand, MINI does not include retroactive compensation methods like NMP does. By avoiding to prescribe such mechanisms, we ensured that MINI stays flexible: one could, for instance, implement a MINI based application that includes these features of NMP. Indeed, this is the path that our next steps will follow.

[6]Its somewhat sad history is documented at `http://www.jamwith.us/about_us/rocket_history.shtml`

6 Acknowledgments

We would like to express our gratitude towards the following colleagues and students, all of whom contributed to MINI in one way or another (in alphabetical order): Klaus Hörmann, Georg Regensburger, Elmar Weiskopf.

References

[1] M. M. Association. The complete midi 1.0 detailed specification, 1996.

[2] S. Boll, W. Klas, and J. Wandel. A cross-media adaptation strategy for multimedia presentations. In *MULTIMEDIA '99: Proceedings of the seventh ACM international conference on Multimedia (Part 1)*, pages 37–46, New York, NY, USA, 1999. ACM Press.

[3] C. Chafe. Distributed internet reverberation for audio collaboration. In *AES 24th International Conference*, 2003.

[4] S. Hessler and M. Welzl. An empirical study of the congestion response of realplayer, windows mediaplayer and quicktime. In *Proceedings of 10th IEEE International Symposium on Computers and Communications (ISCC 2005)*, La Manga del Mar Menor, Cartagena, Spain, June 27-30 2005. IEEE Computer Society Press.

[5] G. Huston. Next steps for the ip qos architecture, November 2000. RFC 2990.

[6] ISO. Iso 14496 (mpeg-4), part 3 (audio), subpart 5 (structured audio), 1999.

[7] jamwith.us, http://www.jamwith.us.

[8] J. Lazzaro and J. Wawrzynek. 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*, pages 157–166, New York, NY, USA, 2001. ACM Press.

[9] J. Lazzaro and J. Wawrzynek. An RTP payload format for midi. In *The 117th Convention of the Audio Engineering Society*, Oct. 2004.

[10] J. Lazzaro and J. Wawrzynek. An Implementation Guide for RTP MIDI. RFC 4696 (Informational), Nov. 2006.

[11] J. Lazzaro and J. Wawrzynek. RTP Payload Format for MIDI. RFC 4695 (Proposed Standard), Nov. 2006.

[12] J. Mullin, L. Smallwood, A. Watson, and G. M. Wilson. New techniques for assessing audio and video quality in real-time interactive communication. In *IHM-HCI*, Lille, France, September 2001.

[13] Netmusicmakers, http://www.netmusicmakers.com.

[14] J. Nichols, M. Claypool, R. Kinicki, and M. Li. Measurements of the congestion responsiveness of windows streaming media. In *Proceedings of the 14th ACM International Workshop on Network and Operating Systems Support for Digital Audio and Video (NOSSDAV)*, June 2004.

[15] A. Nijenhuis and H. Wilf. *Combinatorial Algorithms*. Academic Press, Reading, Massachusetts, second edition, 1978.

[16] R. Rejaie, M. Handley, and D. Estrin. Quality adaptation for congestion controlled video playback over the internet. In *SIGCOMM '99: Proceedings of the conference on Applications, technologies, architectures, and protocols for computer communication*, pages 189–200. ACM Press, 1999.

[17] H. Schulzrinne, S. Casner, R. Frederick, and V. Jacobson. RTP: A Transport Protocol for Real-Time Applications. RFC 3550 (Standard), July 2003.

[18] v-band.de, http://www.v-band.de.

[19] M. Welzl. Netmusic: Echtzeitfähige konzepte und systeme für den telekooperativen austausch musikalischer information". Master's thesis, University of Linz, Linz, Austria, 1998.

[20] M. Welzl. User-centric evaluation of tcp-friendly congestion control for real-time video transmission. *Elektrotechnik und Informationstechnik*, 2005(6), June 2005.