RTP MIDI : Recovery Journal Evaluation and Alternative Proposal

Nicolas Falquet, Dominique Fober

GRAME
Laboratoire de recherche en informatique musicale
Technical report TR-050622

RTP MIDI : Recovery Journal Evaluation and Alternative Proposal

Nicolas Falquet, Dominique Fober
nfalquet@insa-lyon.fr, fober@grame.fr

June 2005

Grame
Centre National de Création Musicale
9, rue du Garet BP 1185
FR - 69202 Lyon Cedex 01

Abstract

An RTP payload for MIDI commands [11] [10] is under development. As a part of this draft, a default resiliency mechanism for the transport over lossy networks defines a journalling method called *recovery journal*. But the theoretical size of this recovery journal can be very large and its format is complex. This report will present an empirical evaluation of the recovery journal size based on a few MidiFiles. We will also propose an alternative solution for the resiliency of RTP MIDI streams based on the combined use of redundancy and retransmissions. Our solution is simpler and might be interesting for some scenarios, typically : short grouping times, complex streams or unconventional semantics.

1 Introduction

RTP (Real-Time Transport Protocol) [18] is a generic protocol designed for real-time applications. MIDI (Musical Instrument Digital Interface) [1] is the current *de facto* standard for communication between digital instruments. MIDI defines both the physical layers and the logical command language of the protocol. RTP MIDI [11] [10] proposes an RTP payload for MIDI commands.

MIDI is a fragile code. For example, losing a NoteOff commands could cause the note to sound indefinitely. An RTP MIDI transmission needs both real-time delivery and strong reliability. Real-time applications over the Internet are usually build on UDP and have to cope with packet loss. Providing adequate reliability mechanisms for MIDI commands is quite specific and demanding.

This is an innovative project since no other protocol for the transport of musical events (such as SASL of MPEG-4 Structured Audio [7] or OpenSound Control [20]) propose resiliency tools for the use over lossy networks. Transporting MIDI data is also really different from transporting traditional digitized audio since the sending rate may vary over time according to the quantity of musical events, since a single loss can have a persistent impact on the stream, since commands rely on a complex semantics and finally since they are also harder to compress.

The current RTP MIDI drafts [11] and [10] proposes a resiliency mechanism : each payload includes, in addition to a basic list of MIDI commands, a special section (called *journal section*) that encodes historical informations about the stream and can be used by the receiver to repair the stream when noticing a packet loss. The draft also defines a default format called *recovery journal* for this journal section.

We will start this report with an empirical evaluation of the recovery journal size for a few typical MIDI transmissions. The results will be given in terms of average and maximum journal size as well as resulting bandwidth for the stream.

We will then present the design of any feasible reliability mechanism for MIDI commands as a overall compromise between several criterias.

This study will lead us to propose an alternative reliability mechanism suitable for RTP MIDI transmissions. Our solution is based on the complementary use of redundancy and retransmission. As a part

of this solution, we define a journal section format called *redundant blocks*. It is using several existing IETF specifications or draft proposals. This solution is simple, independent from the command semantics and allows the receiver to reconstruct the stream exactly as it was sent. We compare this proposal with the existing solution in terms of size and in the context of the previously defined compromise.

2 Recovery Journal Size Evaluation

As we already said, the size of the recovery journal described in [11] is not fixed and it could, theoretically, be very large (17394 octets). It depends on the content of the stream in terms of *features*, mainly : the number of channels in use, the types of commands used (the number of controllers, the use of program changes, etc.), the polyphony of each channel, etc. The recovery journal should be regarded as encoding the "state" of the stream. This implies that its size is not affected by the tempo of the piece. It's related to the number of features used by the stream rather than the quantity of events.

A first evaluation of the recovery journal size was made by Lazzaro in [9]. It was based on an analytical study of the semantic content of the stream. In addition to this semantics-based approach, we studied empirically the size of the recovery journal for several MidiFiles.

2.1 Evaluation Context

We created RTP MIDI payloads from a MidiFile just as they would be created by a streaming application. This was done using a library we built at GRAME to automate the creation and the analysis of payloads to the RTP MIDI format. This library was first presented in [5], it's now integrated as a MidiShare [13] library and available on the MidiShare repository [12]

In RTP MIDI it's possible to group several commands in a same payload. The strategy for grouping the commands is a key aspect of the transmission. The two basic options that can be thought of are : either to send each command immediately (as soon as it is produced), either to choose a duration for which all commands of the stream will be grouped in a single packet. We will call this duration the *grouping time*. Sending the commands one by one might be preferable for an interactive application but this will lead to higher bandwidth. Grouping the commands will increase the overall latency but the stream will be lighter.

In [10] and [9], Lazzaro introduce the notion of *guard packets* : empty packets that can be send on purpose to "guard" the stream during silent sections. For simplicity reasons, this study does not use any guard packets.

Another important characteristic of the transmission is the interval between checkpoint updates. Typically, those updates are made through RTCP Receiver Report packets. The default and minimum value defined by RTP is of 5 seconds but it can increase up to several minutes in case of large multicast sessions.

We present here the results obtained with three MidiFiles :

- Arietta, from the Piano Sonata number 32 in C minor by Beethoven.
- Take the "A" Train, a jazz standard by Billy Strayhorn.
- Proximité, a contemporary piece composed in MIDI by Yann Orlarey.

The recovery journal was including informations about the following command types : NoteOn, NoteOff, CtrlChange, ProgChange and PitchWheel. SysEx were transmitted but not included in the recovery journal. The following chapters are used in the recovery journal : P, C, W and N.

2.2 Journal Size

We will first present the evolution of the recovery journal size for those three MidiFiles. In the first experiment, we didn't use any checkpoint update ; then we tried several values for the checkpoint update interval (5, 20 and 60 seconds).

2.2.1 Arietta

This MidiFile represents 17658 MIDI events in 14'55 minutes. This is a piano piece on two channels : one for each hand. Apart from basic NoteOn / NoteOff commands, from 10'12 onwards numerous controllers

64 (Sustain Pedal) are used. The overall tempo is quite slow but the number of notes is still important : on average 9.57 notes per second.

Table 1 shows some statistical data about the recovery journal size for such a piano piece. The overall journal size is quite small (smaller than the IP / UDP / RTP headers of 40 bytes). The use of a classical checkpoint update every 5 seconds can reduce the average size of the journal by 38 %. Longer intervals between checkpoints are still effective.

The average journal size for each channel (see Table 2) is quite typical for simple musical channels.

checkpoint update	average	max	s.d
no update	37	51	7
5 seconds	24	38	6
20 seconds	28	41	5
60 seconds	31	50	5

Table 1: Arietta – Recovery journal size (in bytes) versus checkpoint update interval

channel	instrument	average
0	piano left hand	11.5
1	piano right hand	10.2

Table 2: Arietta – Average journal size (in bytes) per channel for 5 seconds update

2.2.2 Take the "A" Train

This MidiFile represents 8512 MIDI events in 2'29 minutes. This arrangement uses 6 channels (a grand piano, two basses, two trumpets and a drum set). A few controllers are used : Sustain Pedal (CtrlChange 64) on channel 0 (grand piano), Pitch Wheel on channels 1 and 4 (basses). The overall tempo is fast.

With this example (see Table 3 and Figure 1), the differences caused in the journal size whether we are using the checkpoint update or not is crucial. Without checkpoint update the average journal size is of 242 bytes, which still fits in the MTU of usual networks but this will dramatically impact the stream bandwidth. This is due to the fact that, in many MidiFiles, some initialization is done at the beginning of the stream with commands that won't be used afterwards : ProgChange, CtrlChange (e.g Ctrl 7 (Channel Volume), 0 and 32 (Bank Select), 10 (Pan), etc.), Channel Mode Messages, etc. If no checkpoint update is made, those commands will appear in all journals of the stream. In this MidiFile, a controller 121 (Reset All Controllers) is sent to all 16 channels and a ProgChange as well as 7 controllers (Bank Select MSB, Bank Select LSB, Channel Volume, Pan, Expression Controller, Effects 1 Depth and Effects 2 Depth) are set for each of the 8 channels that will be used. This creates for the journal size an offset of about 200 bytes that corresponds to the main difference that can be observed between the curves (with or without checkpoint updates). Still, for applications that are not using RTCP, it is possible to use the SDP parameter *smf_info* set to *sdp_start* as described in [11] Appendix C.6.4.1 and to include a short SMF that will be played at initialization time.

Apart from that feature, a 5 seconds checkpoint update still leads to reasonable sizes. The average sizes for each channel is comparable to the one observed in the previous example (see Table 3 and Table 4). Here, on average, the journal for channel 0 (grand piano) is larger than the journal for channels 3 or 4 (trumpet) because the piano is using a sustain pedal (this adds chapter C and a log for controller 64) and also because the instrument is more polyphonic (each NoteOn log takes 2 bytes).

checkpoint update	average	max	s.d
no update	243	268	13
5 seconds	56	207	25
20 seconds	81	253	56
60 seconds	132	255	85

Table 3: Take the "A" Train – Recovery journal size (in bytes) versus checkpoint update interval

channel	instrument	average
0	grand piano	13.4
1	bass 1	11.8
2	trumpet 1	10.2
3	trumpet 2	8.7
4	bass 2	11.8
9	drums	9.2

Table 4: Take the “A” Train – Average journal size (in bytes) per channel for 5 seconds update

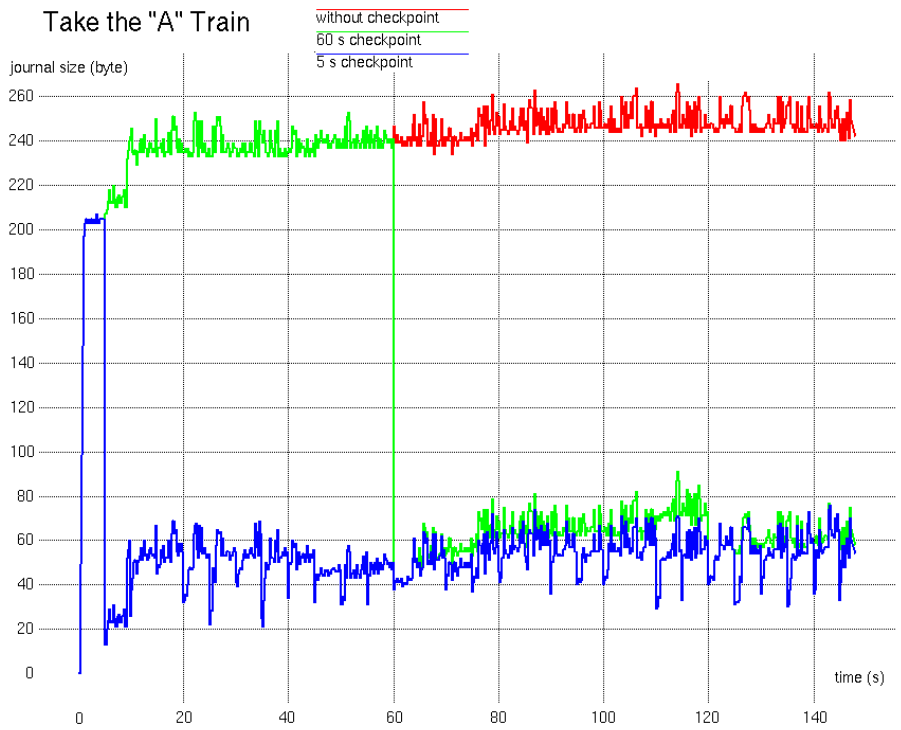

Figure 1: Take the “A” Train – Evolution of the recovery journal size

2.2.3 Proximité

This MidiFile represents 37212 MIDI events in 8’05 minutes. The piece is composed of several movements, using different numbers of channels (at most 8) with different characteristics in terms of polyphony, quantity of notes, etc.

In this example, the overall journal sizes (see Table 5) are still comparable to the one observed in the previous example. The major difference between the curves with or without checkpoint updates still occurs right from the initialization of the stream. It’s due here to Omni Mode On (Ctrl 125) and then Omni Mode Off (Ctrl 124) controllers on the 16 channels plus 3 ProgChange commands.

The first part of the piece (up to 164 s) is using 2 channels with little polyphony but very occasionally up to 7 notes are played at a time. From 164 to 340 seconds, several musical events occurs, involving 8 to 2 channels and using numerous Channel Volume controllers and ProgChange commands. From 340 onwards, 8 channels are used monophonically with many Channel Volume controllers.

2.3 Stream Bandwidth

We will now study more in details the bandwidths obtained with the recovery journal. Tables 6, 7 and 8 shows the bandwidths obtained for various grouping time values. We also created packets with no grouping time. In that case, only the commands that were published simultaneously were still grouped in a same packet. We also indicate some statistical data about the MIDI command sections since it’s the

checkpoint update	average	max	s.d
no update	237	253	20
5 seconds	53	136	24
20 seconds	66	140	33
60 seconds	79	164	36

Table 5: Proximité – Recovery journal size (in bytes) versus checkpoint update interval

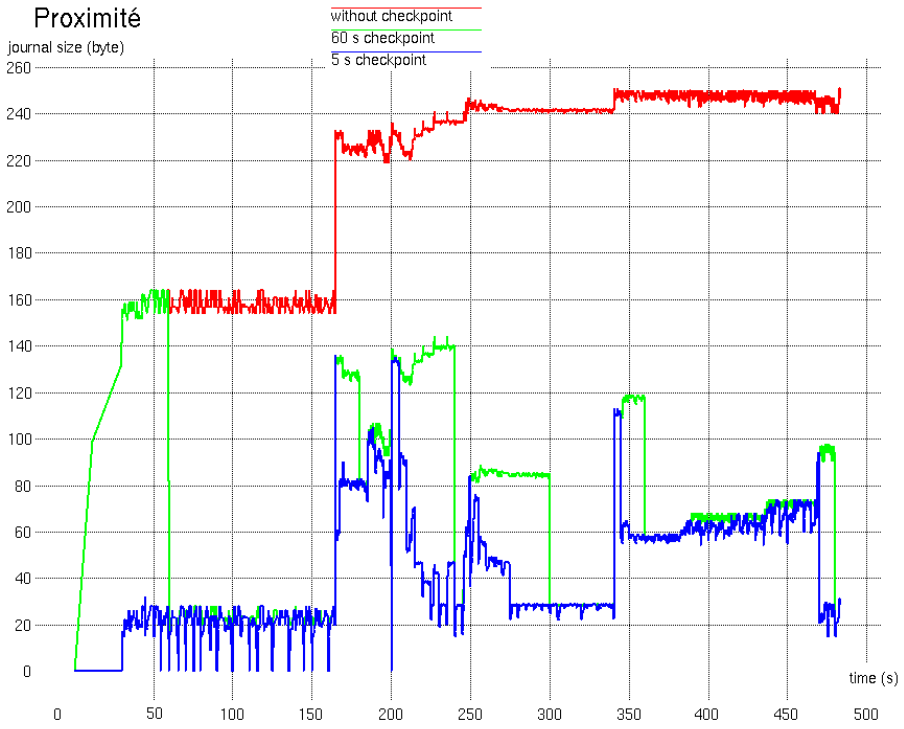

Figure 2: Proximité – Evolution of the recovery journal size.

only part of the payload that varies significantly in size with the grouping time.

The fact that the recovery journal size is independent of the grouping time can make it become an important part of the total bandwidth for short grouping times (see Table 7 for no grouping or 25 ms grouping time). It can be quite unfortunate to impose such overheads for short grouping times since, in the recovery mechanism, the grouping time is related to the minimum possible delay before recovery (the loss detection and the recoveries occur with the reception of the packet immediately following the loss events). Streams with short grouping times will provide faster recoveries and it might be a better option to favor short grouping times.

grouping	bandwidth	commands	commands size	commands %	journal %	header %
no	5.2	2.2	9.8	13.4	31.9	54.7
25 ms	4.7	2.5	11.7	15.5	31.5	53.0
50 ms	4.1	2.9	13.6	17.5	30.7	51.8
100 ms	3.4	3.6	16.4	20.6	29.4	50.0
200 ms	2.6	5.2	22.8	26.4	27.3	46.3

Table 6: Arietta – Stream bandwidth (in kbps), average number of commands, MIDI command section size (in bytes) and section ratios versus grouping time

grouping	bandwidth	commands	commands size	commands %	journal %	header %
no	34.5	1.3	6.4	6.3	54.4	39.3
25 ms	20.2	2.4	11.4	10.7	51.7	37.6
50 ms	14.6	3.5	15.8	14.2	49.9	35.9
100 ms	9.2	6.0	26.0	21.3	45.9	32.8
200 ms	5.7	11.6	48.1	33.3	38.9	27.8

Table 7: Take the "A" Train – Stream bandwidth (in kbps), average number of commands, MIDI command section size (in bytes) and section ratios versus grouping time

grouping	bandwidth	commands	commands size	commands %	journal %	header %
no	50.0	1.3	6.1	5.7	56.1	38.2
25 ms	20.7	3.2	14.6	13.4	49.9	36.7
50 ms	12.3	5.9	25.1	21.3	44.9	33.8
100 ms	7.6	11.0	45.4	33.1	37.7	29.2
200 ms	5.2	20.3	81.7	47.7	28.9	23.4

Table 8: Proximité – Stream bandwidth (in kbps), MIDI command section size and section ratios (in bytes) versus grouping time

3 RTP MIDI Compromise

The characteristics of the transmission that RTP MIDI tries to provide are quite specific and demanding. We need to achieve both timeliness and reliability. On best-effort networks, such as IP networks, those two requirements are traditionally hard to combine. TCP provides reliability but at the cost of late deliveries (by adapting its sending rate and using retransmissions). UDP provides raw best-effort services and the transport remains unreliable. Still, real-time applications using RTP are usually build on UDP to favor timeliness over reliability.

There's another reason to favor UDP : an important point of the RTP philosophy is to consider that only the application has sufficient knowledge of its data to make an informed decision about how that data should be transported. This is the concept of application-level framing, as identified in [3]. The application will provide the data in meaningful units (application data units) and the transport layer should respect those units and expose the details of the delivery to the application. The application can then choose and adapt, according to the type of data and the content of the stream, the solutions it wants to apply when facing network problems like packet loss, packet reordering or network congestion.

3.1 Ideal Solution

In this study, we will still consider that the main stream is transported on UDP and can suffer from packet loss. Reliability solutions must be used to repair from losses and fix the stream.

An ideal solution that would preserve timely delivery and allow in the same time to repair the stream from packet loss as soon as possible would be to include in each packet a complete history of all the commands that appear previously on the stream. This would allows the receiver to repair the stream from any number of loss events as soon as he receives the first packet following the losses. But such a solution would not be feasible since the size of the complete history of the stream would continuously increase over time and and would rapidly be too large. The actual solutions that can be proposed will have to make a compromise in order to reduce this size.

In the case of RTP MIDI, the proposed solution (the recovery journal system) follows the main ideas of this solution but bases the format of its history on the semantics of the commands. In fact, it tries to code the state of the stream rather than a log of all the commands. However, the evolution of the solution has shown that it was not fully satisfying, in particular in terms of complexity and dependence on the semantics of the commands.

We will build a set of criterias that will help us to go on with evaluating the current solution, to propose thereafter some alternatives and to compare the various solutions in terms of a compromise between those criterias.

3.2 Command Semantics

The various solutions may or may not depend on or use the semantics of the MIDI commands. This is not really a criteria that can be used to compare solutions qualitatively but the use of the semantics for the design of the draft's recovery journal in [11] occurred to have a heavy influence on several of its final characteristics (its complexity among others).

The actual usages of MIDI shows, in its diversity and its complexity, that it's very difficult to define a common use or to take into account all possibilities. The recovery journal is based on a detailed study of the semantics of MIDI commands in order to adapt its structure and its encodings. But this proved to be a hard task.

3.3 Size

As we already said in Section 2, the load induced by the recovery solution can be evaluated according to :

- the size of the solution for a single packet (and this can be limited by the MTU of the network)
- the resulting stream bandwidth as well as the ratio of the stream bandwidth occupied by the solution and this for efficiency reasons.

Exceeding the MTU of the network will lead to the IP fragmentation of the datagram. This could cause an efficency loss and a loss multiplier effect since the entire packet will be discarded if any of its fragments is lost.

Recovery Journal

The evaluation of the recovery journal size we did in Section 2 for a few common MidiFiles showed that its maxima were bearable and that its average size was reasonable. The overall stream bandwidths were still small in comparison with traditional audio encodings.

However, an important characteristic of this journal is that its maximum size (17394 octets) can be theoretically larger than the MTU of common networks. For the rare streams that would need a recovery journal larger than the MTU it is still possible to split such dense streams into several smaller streams.

3.4 Accuracy of the Recoveries

The recoveries can vary in terms of quality / accuracy. For example, the velocity of a NoteOn command is coded with 7 bits corresponding to the starting volume of the note, we could imagine a less accurate recoveries on 2 or 3 bits, the same could apply to 7 bits, 14 bits controllers or other values. A solution proposing recoveries less accurate that the original commands will surely depend on the command semantics.

Recovery Journal

The recoveries are generally as accurate as the original commands except for some very special cases (e.g without chapter E, it won't be possible for the receiver to reconstruct the velocity value of a lost NoteOff command).

3.5 Completeness of the History

The solution can provide the application with three types of guarantees considering the completeness of the delivered commands after the possible recovery mechanisms :

- the delivered commands correspond strictly to the emitted commands,
- the commands are delivered on a best-effort basis and the recovery mechanisms guarantee the stream integrity,
- the commands are simply delivered on a best-effort basis.

In most cases, the second type of guarantee will be sufficient and adapted (let us note that between different solutions, the characteristics of this *best-effort* delivery can vary). Solutions proposing this second type of guarantee will surely depend on the semantics of the transported commands. The first type of guarantee might still be necessary in case of applications that use unusual MIDI semantics or applications that want to archive the received stream. The third solution could lead to a violation of the recovery journal mandate described in [11] Section 4.

Recovery Journal

It is not guarantee that all lost commands will lead to a recovery command. The recovery journal is coding a single information for all commands of a same type (same note, same controller, etc.) and if we lose, for example, several NoteOn and NoteOff commands for a same pitch successively, the recovery journal will propose a single recovery command corresponding to the last command for this note.

3.6 Time Ordering

This criteria relates to the ability of the solution to indicate, in addition to the recoveries, the order (and perhaps the temporal arrangement) in which the corresponding commands were emitted on the stream.

Recovery Journal

The recovery journal is reducing the size needed to code the history of the stream by grouping and encoding the commands according to their types and not to their order of appearance. As a consequence, it's not, generally speaking, possible to reorder the lost commands. This is however possible between commands coded by a same chapter (e.g two different notes or two controllers on the same channel). But it is not possible, for example, to reorder interlaced notes and controllers. For example, in the case of a NoteOff and a command *ON* for the sustain pedal (controller 64, value 127) lost in a same packet, it won't be possible to determine if the note has to be sustained or not.

3.7 Complexity

The complexity of the solution has also to be taken into account. It shouldn't be considered as a flaw by itself but it will influence the ease of implementation, the debugging and the use of the protocol as well as the transparency of its mechanisms to the client application. It can also informs us on the overall elegance of the solution.

Recovery Journal

It occurs to be really complex. The size of the memo dedicated to the recovery journal can testify to that : it's more than 50 pages long whereas [14], [15] and [17] are respectively 52, 11 and 26 pages long. The recovery journal defines 13 different chapters and a specific type of encoding for each chapter.

3.8 Delay before the Recovery

The consequences of the packet loss can be repaired more or less quickly. If the solution, like the ideal solution, does not imply multiple transmissions, the recovery can take place at earliest with the packet arriving next after the packet loss. This emphasis the importance for short grouping times since large grouping times will increase the minimum delay before recovery.

We could imagine that content streaming applications with quite important buffers on the receiving side could accommodate with later recoveries.

Recovery Journal

The stream can be completely fixed as soon as the first packet following the packet losses is received.

3.9 Adaptability to the Network Conditions

In the current networking environments where UDP streams are sharing resources with other TCP connections, our UDP streams have to compete fairly with TCP and, notably, they have to adapt their sending rates to the network congestion. In the case of RTP MIDI it's hardly conceivable to adapt the bandwidth used by the MIDI commands included in the MIDI command section. It's more appropriate

to adapt the load of the reliability mechanisms or the grouping time of the commands. But it can be argued that, anyhow, RTP MIDI streams that correspond to usual MIDI transmissions won't create very large bandwidths as it has been shown in Section 2.3.

Recovery Journal

With the recovery journal, the main way to modulate the bandwidth that do not impose restrictions on the incoming commands is to increase the grouping time of the commands. This will reduce the number of packets sent and will, as a consequence, reduce the bandwidth used for the recovery journal as well as the IP / UDP / RTP headers. This method is effective for this solution as it was already shown in Tables 6, 7 and 8.

With this method, the bandwidth is reduced at the price of a increased latency for all the commands of the stream. We could imagine that this could rather be made at the expense of other criterias of the compromise (like the speed of the recoveries) which would not affect the latency of the normal commands.

Two other option can be thought of while using the recovery journal.

- To reduce the delay between checkpoint updates. But they are made through RTCP Receiver Reports and RTP imposes that they cannot be made more frequently than every 5 seconds.
- To change the inclusion rules for the recovery journal. But it has been stated in [11] that, as a general rule, session parameter changes are not supported very well.

3.10 Scalability

The recovery mechanisms that can be used may vary whether the stream is unicast, multicast for a small group or large scale multicast. A reliability solution that would need the sender to perform actions according to the feedback of each receiver could still work for small multicast groups but won't scale very well. A reliability mechanism using TCP won't be adapted to multicast transmissions, usually based on UDP.

Recovery Journal

The recovery journal is not implying, in case of packet loss, any action to be taken by the sender. All recovery decisions can be made on the receiving side. This leads to a high scalability.

4 Alternative Proposal

We will now propose an alternative solution. The previous studies showed the pros and cons of the recovery journal. The next step for us was not to find out a better solution in an absolute way, but to find a real alternative, proposing a different compromise that could be useful for some scenarios.

The first two aspects we wanted to favor were : simplicity and independence of the command semantics (the two append to be closely related in the recovery journal). We also wanted a solution that would allow the receiver to reconstruct the stream as it was sent by the sender.

The fact that we didn't want to be dependent on the MIDI semantics imposes a solution that could not be based, like the recovery journal, on encoding an interpreted "state" of the stream. This also relates to other criterias of the compromise, namely the completeness of the history, the time ordering and the accuracy of the recoveries, since using them to reduce the size of the solution leads to interpret the semantics of the commands. That's why our solution uses logs of the commands.

We also thought that the recovery journal was probably adding to much redundancy on the stream : the informations of a command can figure in each packet for several seconds. Such a strong guarantees will be rarely needed.

4.1 Solution Overview

Our solution is based on the usual characteristics of packet loss patterns on IP networks as summarized by Perkins in [15] :

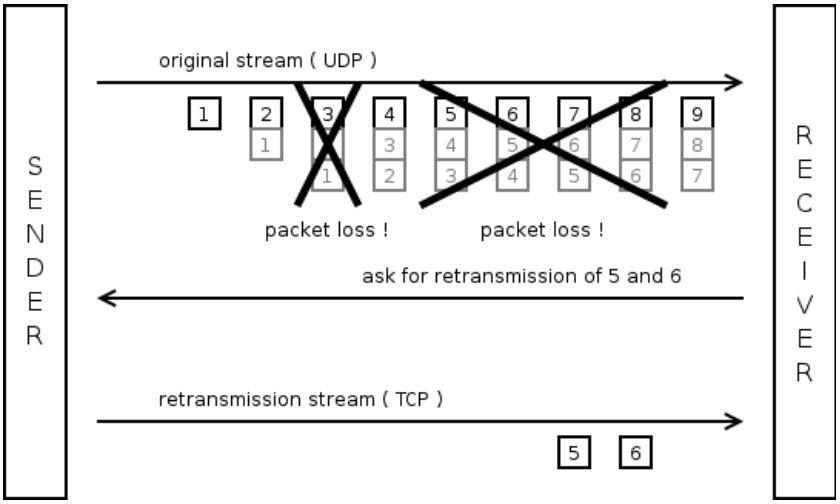

Figure 3: Redundancy and retransmission

1. the vast majority of packet losses are of single packets (about 90 %),
2. however, loss events are not independent and often occurs in burst which are typically short,
3. on rare occasions, some longer bursts and even short outages in the transmission can occur.

Our solution relies on two complementary mechanisms to deal with those loss events.

- We use redundancy to cope with isolated loss or small bursts of loss events. For that, each packet will include, in addition to its own list of commands, a small number r of redundant data blocks, corresponding to the command lists of the previous packets.
- We use retransmissions to cope with longer bursts and outages. A receiver noticing a burst of losses that cannot be repaired with the redundancy blocks will ask the receiver to retransmit the command lists of the missing packets. Those retransmissions will be made on a separate TCP stream.

A example transmission is given in Figure 3. The sender is transmitting packets on an original UDP stream (here packets 1 to 9). Each packet includes, as redundant data, the command lists from r previous packets in addition to its primary data (here $r = 2$). In case of a single loss (e.g loss of packet 3), the receiver will use the logs included in the first packet following the loss to repair the stream (here log of packet 3 included in packet 4). In case of a long burst of loss events (e.g loss of packets 5, 6, 7 and 8), the receiver won't be able to reconstruct all the lost commands by using only the first packet following the losses (here with packet 9 he can only reconstruct packets 7 and 8). So, he will ask for retransmission of the packets he was not able to reconstruct (here packet 5 and 6). The sender will then retransmit the command lists of the missing packets on a separate TCP retransmission stream.

4.2 Redundancy

To support redundancy we will define a new journalling method for RTP MIDI payload called *redundant blocks*.

Redundancy could have been provided in accordance with RFC 2198 [16] which describes an RTP payload for redundant audio data. We won't use this specification since our purpose is not to reuse existing audio encodings to provide redundancy but to define a new journalling method for RTP MIDI, defining its own encodings. This also makes sense since this kind of encodings won't probably be used apart from RTP MIDI. Still, our solution will be inspired by RFC 2198.

4.2.1 Redundant Blocks Section

The *journal section* defined in [11] will be replaced by a *redundant blocks section* consisting of r blocks. The redundant block section of packet i will include redundant blocks corresponding to the MIDI command section of packets $i-1$ to $i-r$. The structure of the redundant blocks section is given in Figure 4 and the generic structure for a redundant block is given in Figure 5.

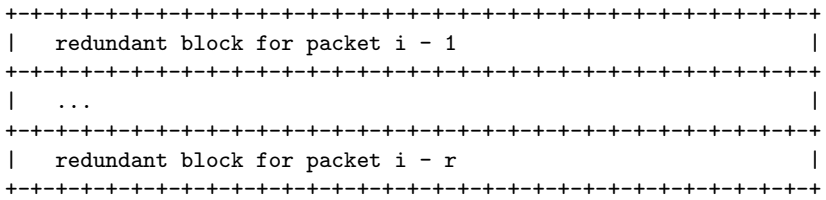

Figure 4: Structure of the redundant blocks section

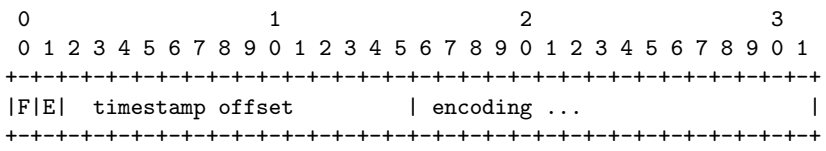

Figure 5: General structure of a redundant block

- "F" bit indicates whether another block follows. If 1 further block follow, if 0 this is the last block.
- "E" bit indicates the type of encoding for the "encoding" part.
- "timestamp offset" is a 14 bits unsigned offset of timestamp for this block relative to the timestamp given in RTP header.

4.2.2 Possible Encodings

An important difference with the mechanisms of RFC 2198 as they can be used for digitized audio, in [2] for example, is that with MIDI commands it's harder, but still possible to some extent, to propose multiple encodings of various size and quality for the same command or set of commands (at least it's not as easy as using a lower bit rate encoding).

For that reason, MIDI data are more similar to text data. A payload type for text conversation has been standardized in [6] and is also providing reliability based on redundancy without compression.

We will now define two possible encodings for the redundant blocks.

Default Encoding : MDEF

The default encoding is based on the encoding defined for MIDI command sections of the RTP MIDI draft, Section 3. In this encoding, each command is associated with a delta time value that is used to calculate the absolute timestamp of the command. When using this encoding, the "encoding" part of the redundant block for packet n will be a simple copy of the MIDI command section of that packet. For this encoding, the "E" bit of Figure 5 is set to 0.

Raw MIDI Encoding : MRAW

A first option to reduce the size of the log is to remove the delta time values associated with the commands. This should reduce the size of at most 25 % (if we consider 3 bytes commands and 1 byte delta time values). Though, the first command of a packet might be associated with an implicit 0 delta time value, not encoded in the command list. When using this encoding, the "encoding" part of the redundant block for packet n will be a copy of the MIDI command section of that packet without any delta time information. For this encoding, the "E" bit of Figure 5 is set to 1.

MIDI Summary

Another option to reduce the size of the logs is to simplify some commands. An application could decide to simplify the logs from commands that are not relevant for recovery, for example : NoteOn commands, associated NoteOn and NoteOff commands, multiple values of continuous controllers, etc.

Those decisions imply to use the semantics of the commands but this can be done at the application level and has no consequence on the format or the mechanism used for transmissions. Such a simplification

could lead to important reductions in size : in case of a simple melodic stream, removing all NoteOn commands from logs will already reduce the number of commands by 50 %.

This technique does not define another type of encoding since it can be applied to both previously defined encodings (MDEF and MRAW). Note that it could be important to signal to the receiver if this technique will be used in the stream. This can be done through session description.

4.3 Retransmission

Redundancy can be used to cope with isolated loss or small bursts of loss events and this has to cover most loss cases. But still MIDI streams needs a higher level of reliability to ensure that the receiver can recover from all indefinite artifacts.

We propose to provide this next level of reliability through retransmissions.

Retransmission have often been dismissed as an option for error control in real time transmissions. But it has be shown [4] that, even if it's not feasible in all cases, it can be a reasonable option in number of scenarios.

It can also be argued that those rare cases will occur after a burst of loss events where the stream was anyways already disturbed. For example, in the case of a sender adding two redundant blocks in each payload, the retransmission will be used in case of three successive losses. This will already cause a delay corresponding to three times the grouping time of the commands before any possible recovery.

Here comes another difference between transmissions of MIDI and digitized media. In real time transmissions it's often agreed that a late data is useless. In traditional audio or video streaming, a data arriving after it's playout time is purely discarded. In MIDI however, as a single command can have a persistent influence on the resulting output, even a late command can still be crucial for the integrity of the stream. The notion of delay bounds has thus a different meaning and it allows us to choose a solution which can deliver late commands but which is still relevant as a part of the overall compromise.

4.3.1 Existing Framework for Retransmission

No mechanisms are designed in the current RTP specifications [18] [19] to support retransmissions. However an extension to the common AVP profile is under development [14]. It provides the receiver with mechanisms for, statistically, more immediate RTCP feedback. In this mechanism, a receiver can send its RTCP feedback earlier than normal, at the expense of delaying the following packet.

In addition to this profile extension, an RTP payload for retransmission is also being worked on in [17]. In this proposal, a separate stream is used for the retransmissions. In our case where we have to favor the reliability of this stream, it should be transported over TCP. Sending RTP packets over a connection-oriented transport such as TCP needs a framing method to recompose the original packets. Such a method is being standardized by [8].

4.3.2 Asking for Retransmissions

If a receiver is receiving a packet i that follows a burst of l loss events with l larger than r, he will need to ask for retransmission of packets $i - l$ to $i - r - 1$.

Using the extended RTP profile for RTCP-based feedback described in [14], a receiver detecting a burst of packet loss will try to send an immediate feedback to the sender as a part of a compound RTCP packet. He will be allowed to do so if the resulting bandwidth of RTCP packets respects the bandwidth limitation of RTCP. After sending an early RTCP packet he will have to delay the following packet.

The retransmissions will be asked through NACK messages to be included in compound RTCP packets. Figure 6 shows the format for a single NACK message as defined in [14] Section 6.2.1. Note that in our case, NACK messages are not used to signal lost packets but to ask for retransmissions, since a lost packet might already be reconstructed with redundant blocks.

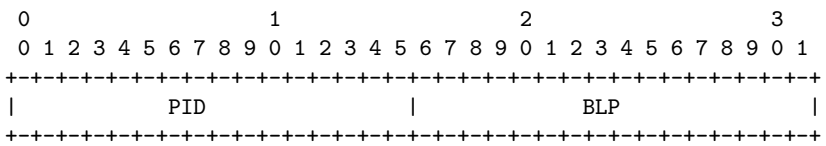

Figure 6: NACK message

- "PID" is a 16 bits field corresponding to the RTP sequence number of a packet that needs to be retransmitted.
- "BLP" is a 16 bits bitmask field that allows for asking retransmission of any of the 16 RTP packets immediately following the RTP packet indicated by PID. The i^{th} bit set to 1 asks for the retransmission of packet $PID + i$.

The time and bandwidth constraints imposed by RTP for sending RTCP packets cannot ensure that a receiver will be allowed to ask in all cases immediately when a burst of loss occurs. See [14] for more details. This is one of the reasons for which our mechanisms needs to adapt to the network conditions. See Section 5.1.

4.3.3 Retransmitting

Once the sender receives a feedback asking for retransmissions, he will retransmit the missing packets through a separate retransmission stream as proposed in [17]. Though retransmissions could take place on another UDP stream for digitized audio, it has to be transported on TCP in the case of MIDI transmissions since we need to achieve full reliability to correct the stream in any case. For the transport over TCP, a framing method as described in [8] must be adopted. This leads to the packet structure described in Figure 7.

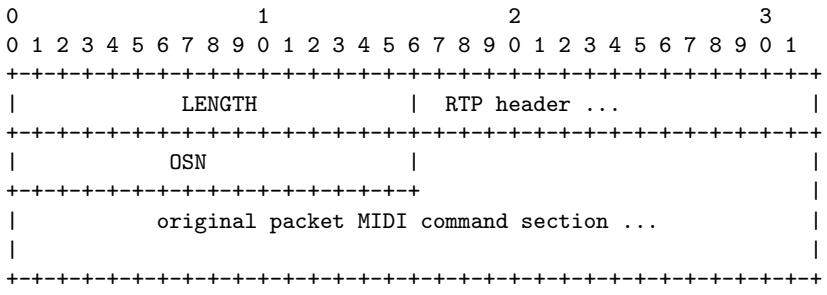

Figure 7: Retransmission packet structure

- "LENGTH" is a 16 bits field indicating the length of the following RTP packet (RTP header, OSN field and MIDI command section).
- "RTP header" is the RTP header of the packet for this separate stream. This means for example that the sequence number of the packet figuring in the RTP header is not equal to the sequence number of the original packet. However, the timestamp value of this header must be set to the timestamp value of the original packet, as described in [17].
- "OSN" is a 16 bits field indicating the sequence number of the original packet in the original UDP stream.
- Then follows the MIDI command section of the original packet. Note that we are not retransmitting the redundant blocks section included in the original packet.

Since the retransmission takes place on a separate channel, the receiver has two options after asking for retransmissions. First, the simpler solution can be to stop and wait for the retransmitted packets. While waiting for the retransmissions, he will queue the packets coming on the UDP stream and play all the commands in order once the retransmitted packets arrived. But this option won't be acceptable in much cases. The other solution is to keep on playing the packets coming through the original UDP stream and to interpret, when the retransmissions arrive, which recoveries are still needed. For that, the application will need to interpret the semantics of the commands since it won't be appropriate to just play the missing commands as they arrive (for exemple, it won't be appropriate to play a missed NoteOff if the corresponding note has already been started by another NoteOn). This should be possible for an application if it maintains the history of all the commands played since the loss occurs.

4.4 Size Evaluation

An interesting point of this solution is that its size is directly dependent on the number of commands that are sent. It becomes more predictable.

Given the average command section size c for a stream, we can predict the average packet size and bandwidth of the stream using such a solution with r redundancy blocks. In this estimation we do not consider the load induced by retransmission stream. This underestimation needs to be signaled but it can be made since retransmissions have to be rare and the resulting bandwidth will be negligible. A packet will be composed of :

- IP / UDP / RTP headers : 40 bytes
- command section size : c bytes
- redundant data blocks : $r * (c + 2)$ bytes

Tables 9, 10 and 11 compares the sizes and bandwidths obtained with the recovery journal solution and the sizes that can be expected with the redundant blocks solution. First, we can see that the size of our solution increases with grouping time on contrary to the recovery journal which is constant. But this relates to the statement we made in Section 2.3 and Section 3.8 : the grouping time directly impacts the delay before recoveries and thus, short grouping times should be favored. For short grouping times and stream using basic features (in Arietta simple NoteOn and NoteOff commands are used on 2 channels) the sizes of the two solutions are comparable. However for stream using more complex features our solution is lighter (e.g for Take the "A" Train, with 8 channels, some controllers and important initialization : 27 % gain in bandwidth for 25 ms grouping).

		recovery journal		redundant blocks	
grouping	commands	section size	stream bandwidth	section size	stream bandwidth
no grouping	9.8	23.3	5.2	23.6	5.2
25 ms	11.7	23.8	4.7	27.4	4.9
50 ms	13.6	23.8	4.1	31.2	4.5
100 ms	16.4	23.5	3.4	36.8	4.0
200 ms	22.8	23.6	2.6	49.6	3.4

Table 9: Arietta – Sizes (in bytes) and bandwidths (in kbps) for the two solutions

		recovery journal		redundant blocks	
grouping	commands	section size	stream bandwidth	section size	stream bandwidth
no grouping	6.4	55.4	34.5	16.8	21.4
25 ms	11.4	55.0	20.2	26.8	14.8
50 ms	15.8	55.7	14.6	35.6	11.9
100 ms	26.0	55.1	9.2	56.0	9.2
200 ms	48.1	55.2	5.7	100.2	7.4

Table 10: Take the "A" Train – Sizes (in bytes) and bandwidths (in kbps) for the two solutions

		recovery journal		redundant blocks	
grouping	commands	section size	stream bandwidth	section size	stream bandwidth
no grouping	6.1	59.0	50.0	16.2	29.7
25 ms	14.6	54.5	20.7	33.2	16.7
50 ms	25.1	53.0	12.3	54.2	12.5
100 ms	45.4	51.7	7.6	94.8	10.0
200 ms	81.7	49.5	5.2	167.4	8.8

Table 11: Proximité – Sizes (in bytes) and bandwidths (in kbps) for the two solutions

5 Condition on the Command Section Size

In our solution, the size of the redundant blocks section for packet i can be directly deduced from the sizes of the command sections of the previous packets.

Let c_i be the size of the command section of packet i and r_i the size of the redundant blocks section of packet i. Equation 1 gives two expressions for r_i if using the default encoding. The total space u that can be used in each payload for c_i and r_i is given, according to the network MTU, by Equation 2. Let d_i, defined in Equation 3, denote the difference between u and $c_i + r_i$: the space which is still available in the MTU for packet i.

$$r_i = \sum_{n=i-r}^{i-1} c_n \Leftrightarrow r_{i+1} = r_i + c_i - c_{i-r} \tag{1}$$

$$u = MTU - 40 - 2r \tag{2}$$

$$d_i = u - c_i - r_i \tag{3}$$

Given a limited MTU for the network packets we have to find the conditions for which Equation 5 will be verified. If Equation 4 is verified, using Equations 1 and 3 can lead to the general condition on the command section size for a new packet given by Equation 6.

$$r_i + c_i \leq u \tag{4}$$

$$r_{i+1} + c_{i+1} \leq u \tag{5}$$

$$c_{i+1} \leq c_{i-r} + d_i \tag{6}$$

5.1 Adapting to Network Conditions

They are three main reasons for which our mechanisms needs to adapt to the network conditions.

- Every RTP application should manage some form of congestion control to compete fairly with TCP traffic. This implies that the stream should be able to adapt its sending rate.
- To be efficient, our retransmission mechanism must be used rarely, on emergency occasions, since it imposes larger delays and since a receiver won't be allowed to ask immediately for all retransmissions when the RTCP bandwidth and timing rules do not permit it. This implies that the stream should be able to adapt to the loss rate in order to handle almost all loss events through redundancy.
- The level of redundancy must be adapted to the loss rate also to avoid sending redundant blocks when the network is loss free.

Several options can be used to adapt to network conditions in order to reduce the sending rate or to adapt to loss rate.

- Adapting the grouping time will slightly reduce the stream bandwidth and it should also reduce the size of the bursts of loss. But increasing the grouping time will increase the overall latency of the transmission and it can be problematic for the receiver to adapt its playout buffer dynamically.
- Adapting the redundancy level r will increase the resiliency of the stream to packet losses but it will increase, in the same time, the bandwidth of the stream and this might lead as a reaction of the network to more packet losses.
- Adapting the encodings used for the redundant blocks could decrease the bandwidth of the stream.

Those three options should be combined to propose a smart adaption to the network conditions.

6 Conclusion

The experiments we made in Section 2 show, as a conclusion, that the recovery journal leads to good results for usual MIDI transmissions. The average journal sizes were reasonable and its maximum were bearable. The resulting bandwidth were still very small in comparison with traditional audio codecs for music.

We proposed in Section 4 an alternative solution having different characteristics from the draft's solution. We will now summarize the comparison between the two solutions according to the criterias of the compromise described in Section 3.

- **Complexity** and **Command Semantics** – Our solution is simpler and allows to transport MIDI commands independently from their semantics. However, it is still possible for the application to use the semantics of the commands to improve the performances of the solution.
- **Size** – The recovery journal is lighter for streams with few features or large grouping times. Our solution is more predictable in size and lighter for complex streams and short grouping times. Our solution is also bounded in size.
- **Delay before the Recovery** – The recovery journal is more robust since a single packet can repair the stream completely after any number of packet loss. Our solution is occasionally using retransmission and this can leads to larger delays. Being lighter for short grouping times, we can consider that our solution is encouraging short grouping times that lead, on a regular basis, to shorter delays before recovery.
- **Completeness of the History** and **Time Ordering** – Our solution allows the receiver to reconstruct all lost commands as well as their time arrangement.
- **Scalability** – Due to occasional retransmissions on a TCP stream, our solution is not as scalable as the recovery journal.

References

[1] MIDI Manufacturers Association. *The Complete MIDI 1.0 Detailed Specification*. 1996.

[2] J.-C. Bolot, S. Fosse-Parisis, and D. Towsley. Adaptive FEC-Based Error Control for Internet Telephony. In *Proceedings of the 18th Joint Conference of the IEEE Computer and Communications Societies (INFOCOM'99)*, March 1999.

[3] D. D. Clark and D. L. Tennenhouse. Architectural Considerations for a New Generation of Protocols. In *Proceedings of the Symposium on Communications Architectures and Protocols (SIGCOMM'90)*, September 1990.

[4] B. J. Dempsey. *Retransmission-Based Error Control for Continuous Media Traffic in Packet-Switched Networks*. PhD thesis, University of Virginia, May 1994.

[5] N. Falquet and D. Fober. Implémentation d'un flot de données Midi sur RTP. In *Actes des Journées d'Informatique Musicale (JIM'05)*, June 2005.

[6] G. Hellstrom. RTP Payload for Text Conversation. RFC 4103 (Proposed Standard), June 2005.

[7] International Standard Organisation. *ISO/IEC 14496 MPEG-4*, chapter Part 3 (Audio), Subpart 5 (Structured Audio). 2001.

[8] J. Lazzaro. Framing RTP and RTCP Packets over Connection-Oriented Transport. IETF Internet Draft (Work in Progress), January 2005.

[9] J. Lazzaro and J. Wawrzynek. A Case for Network Musical Performance. In *Proceedings of the 11th International Workshop on Network and Operating Systems Support for Digital Audio and Video (NOSSDAV'01)*, June 2001.

[10] J. Lazzaro and J. Wawrzynek. An Implementation Guide for RTP MIDI. IETF Internet Draft (Work in Progress), April 2005.

[11] J. Lazzaro and J. Wawrzynek. RTP Payload Format for MIDI. Work in Progress, April 2005.

[12] Y. Orlarey, D. Fober, and S. Letz. MidiShare. http://midishare.sourceforge.net.

[13] Y. Orlarey and H. Lequay. MidiShare : A Real Time Multi-Tasks Software Module for Midi Applications. In *Proceedings of the International Computer Music Conference (ICMC'89)*, 1989.

[14] J. Ott, S. Wenger, N. Sato, C. Burmeister, and J. Rey. Extended RTP Profile for RTCP-based Feedback (RTP/AVPF). IETF Internet Draft (Work in Progress), August 2004.

[15] C. Perkins. *RTP : Audio and Video for the Internet*. Addison-Wesley, 2003.

[16] C. Perkins, I. Kouvelas, O. Hodson, V. Hardman, M. Handley, J.-C. Bolot, A. Vega-Garcia, and S. Fosse-Parisis. RTP Payload for Redundant Audio Data. IETF RFC 2198 (Proposed Standard), September 1997.

[17] J. Rey, D. Leon, A. Miyazaki, V. Varsa, and R. Hakenberg. RTP Retransmission Payload Format. IETF Internet Draft (Work in Progress), March 2005.

[18] H. Schulzrinne. RTP : A Transport Protocol for Real-Time Applications. IETF RFC 3550 (Proposed Standard), July 2003.

[19] H. Schulzrinne. RTP Profile for Audio and Video Conference with Minimal Control. RFC 3551 (Proposed Standard), July 2003.

[20] M. Wright and A. Freed. OpenSound Control : A New Protocol for Communicating with Sound Synthesizers. In *Proceedings of the International Computer Music Conference (ICMC'97)*, September 1997.