# MINI: Making MIDI Fit for Real-time Musical Interaction over the Internet

Michael Welzl
*University of Innsbruck*
*michael.welzl@uibk.ac.at*

Max Mühlhäuser
*Darmstadt University of Technology*
*max@tk.informatik.tu-darmstadt.de*

Jan Borchers
*RWTH Aachen University*
*borchers@cs.rwth-aachen.de*

## Abstract

*While MIDI is still the most viable music protocol, it is not appropriate for use in live networked music performances. We propose a network data format called MINI. It eliminates the disturbing arpeggio effect caused by network packet delay jitter, and yields smaller packets that can be further reduced by omitting marginal MIDI features. MIDI–MINI transcoding is transparent, and works with standard MIDI instruments. We present the MINI design rationale, protocol, and empirical performance results.*

## 1. Introduction

Musicians have frequently attempted to improvise together over long-distance network connections. Due to the real-time player interaction that such "jam sessions" require, network delay and jitter quickly lead to a total musical communication breakdown between the players. The internet, however, typically only provides a "best-effort" service with no guaranteed limits on latency or jitter. These effects are mainly caused by growing queues at congested bottleneck routers receiving more traffic than they can forward.

Thus, to minimize the chance of delays, applications should reduce the likelihood of causing congestions to occur, by sending as few and as small packets as possible. Smaller packets also reduce the time the sender has to wait until all the data to fill a packet is available. VoIP applications such as Skype use both strategies, sending extremely small packets at a very low rate.

MIDI is not as small as it could be, wasting bandwidth and increasing the chance of congestion. And even with today's gigabit wired connectivity, WiFi connections or multi-player scenarios can quickly lead to congestion. A MIDI-based real-time protocol should therefore aim to reduce the amount of data considerably, even at the cost of some nonessential MIDI features.

We present the *MINI (Musical Instrument Network Interface)* format that addresses the problems with MIDI over long-distance network connections in three ways: It (1) is *smaller* than MIDI; (2) *encodes chords* as such instead of as a series of individual notes like MIDI, avoiding the arpeggio effect (see section 2.1); and (3) provides flexible *feature–speed tradeoffs*.

Adaptive internet multimedia applications frequently sacrifice features to reduce delay, e.g., by reducing video frame quality in congested networks [16]. A reduced quality also lowers the sending rate and hence packet loss, often leading to a more agreeable result at the receiver. Although congestion is a dynamic effect, reacting continuously to the current network state can lead to undesirable quality fluctuations—users prefer a continuously poor quality over frequent changes [12, 20]. A successful way to handle this discrepancy between the dynamic network and its not-so-dynamic user is to let the user switch between quality levels [2]. MINI also employs this strategy.

We describe the MINI encoding scheme in the next section, then present a MINI implementation in a GUI application in section 3, our evaluation in section 4, and an overview of related work in section 5.

## 2. MINI

MIDI encodes the onset of each musical note with a *Note-On*, and its termination with a *Note-Off* message. A chord onset is encoded as a sequence of *Note-On* messages, one for each of its notes, and its termination in the same way. The MIDI standard [1] assumes a local connection that will not yield audible delays (limiting the number of MIDI devices that can be daisy-chained,

for example), and that has a fixed bandwidth. Sending MIDI data across the internet violates that specification. Network delays between Note-On message packets belonging to a chord can turn it into an arpeggio—musically, a highly undesirable effect we call *intra-chord jitter*. Delay fluctuations *between* chords or single notes (*inter-chord jitter*) are equally disturbing, and to be avoided. MINI addresses both issues.

MINI's scope is restricted to transporting musical MIDI data. Distributed performances usually require additional initial agreements on the software level, using an application specific protocol and format we call SETUP. A SETUP phase may precede the exchange of MINI messages, and additional SETUP messages may occur during the performance. As shown later, MINI offers several trade-off options between expressiveness and message size. Some can be changed from one message to the next, and are part of the MINI format. Others are expected to change rarely or not at all during a performance, and must be negotiated between the distributed applications using SETUP messages. MINI does not define these application-specific messages.

### 2.1. Chord encoding

To remove intra-chord jitter and save space at the same time, MINI encodes chords as a single code as opposed to individual notes. While a musician can start and stop playing some notes of a chord at different times, most of them are usually played or muted within an interval that yields the impression of concurrence for the listener. This interval can be defined as a fixed delay threshold. Of course, this makes it impossible to preserve individual notes being played or terminated within that interval; this is one of the expressiveness–speed tradeoffs MINI introduces.

The MINI chord encoding scheme considers the number $m$ of all possible chords with $k$ notes within a given range of $n$ notes as a $k^{\text{th}}$-order combination of $n$ elements without repetition or ordering:

$$m = \binom{n}{k} = \frac{n!}{k!(n-k)!} \qquad (1)$$

For example, there are $m = 455$ different possible triads ($k = 3$) in a range of $n = 15$ notes. By unambiguously mapping each chord to a number 1...455 in a table, these chords could be encoded using $\lceil \log_2(455) \rceil = 9$ bits.

Encoding and decoding MINI chords via tables would be straightforward, but require significant memory. Fortunately, there is no need for tables, as the mapping is a simple combinatorial problem—see the implicit unambiguous mapping from higher-dimensional

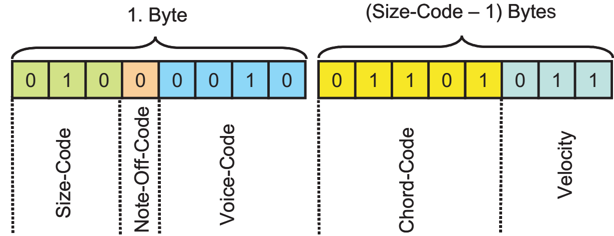


**Figure 1. Note-On with velocity in MINI.**

to one-dimensional arrays in the *C* programming language. For MINI, we use an algorithm [15] that maps an array representing a unique subset of size $K$ from a set of size $N$, bijectively to an integer number, the "rank" (or order) of the subset. (We used the C encoding and decoding functions KSUB_RANK and KSUB_UNRANK from http://people.scs.fsu.edu/~burkardt/.) Note that, just as the C array mapping works without knowing the size of the array it is working on, this algorithm does not need $N$ as input.

$n$ and $k$ can be chosen to adapt to the instruments and network capacity: to connect instruments with a small tonal range, $n$ can be small, and such instruments usually also limit $k$: e.g., $k > 10$ makes little sense on a keyboard. The smaller $n$ and $k$, the smaller the data format; this can be used to trade comfort against sending rate when bandwidth becomes scarce.

### 2.2. Note-On / Note-Off message layout

Fig. 1 shows a sample MINI word. Since each word can vary in size, we encode its length in bytes in the 3-bit Size-Code field. Note-Off-Code is 0 for Note-On or 1 for Note-Off messages.

The Voice-Code field indicates the number $k$ of notes in the encoded chord. Its 4 bits allow for up to of 16 voices. Common instruments such as keyboards will rarely require more voices. If they do, the chord can simply be encoded in multiple MINI words. The Chord-Code is the rank of the chord as defined above.

Velocity is the volume (e.g., keyboard attack or release speed) of MIDI Note-On and Note-Off messages. For most instruments, it is difficult to play notes in a single chord at different velocity; therefore, MINI uses a single velocity value for the entire chord. We believe most musicians will be happy to sacrifice these small nuances in their performance in exchange for smaller data sets and thus reduced delay.

MIDI encodes velocity in 7 bits, but some instruments cannot produce that many velocity values, and keyboard experiments showed us that the difference to using 3 bits is barely audible. In MINI, velocity resolu-

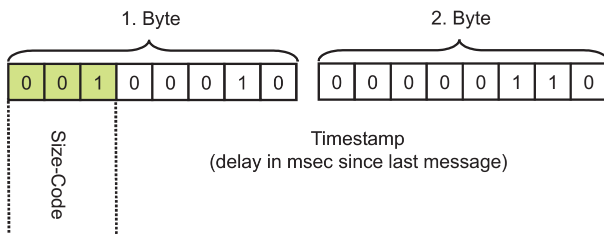


**Figure 2. MINI Timestamp message.**

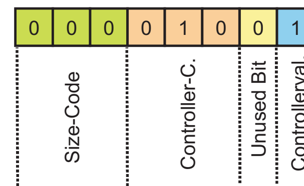


**Figure 3. MINI pedal Controller Message. 8 possible controllers were setup initially.**

tion is negotiable from 1–7 bits using SETUP messages.

### 2.3. Timestamp messages

Queueing delay *between* MINI words causes *inter-chord jitter*. To remove it, the receiver needs to restore the correct timing. Since this requires precise knowledge about when notes were played, the MINI Size-Code value "001" encodes a Timestamp message (see Fig. 2). MINI senders can insert it in front of a MINI Note-On or Note-Off message to specify how much time has passed since the last MIDI event. It encodes milliseconds in 13 bits, giving a max. delay of about 8 seconds. Longer delays can be encoded via multiple Timestamp messages, to be summed up by the receiver.

The MINI receiver can then remove jitter and restore correct timing by using a playout buffer. Arriving MINI words are placed in this buffer and played from it with the right timing. The length of this buffer enables another trade-off: a longer buffer removes more severe inter-chord jitter, but adds more constant delay before incoming messages are played. If the goal is to make a performance as interactive as possible, and inter-chord jitter is more acceptable than a fixed additional delay, this feature should not be used.

### 2.4. Controller messages

In MIDI, Note-On and Note-Off are "Channel Voice Messages", bound to one of 16 logical Channels. To save space, Channels are not included in MINI: one MINI stream represents one Channel. Multiple channels can be represented by using the underlying protocol's multiplexing, e.g., as multiple MINI streams on different UDP ports.

MIDI also defines "System Common Messages" including "Song Select" (only relevant for sequencers), "Tune Request" (irrelevant when musicians do not truly hear each other), device-dependent "System Exclusive Messages", and "System Realtime Messages" to ping a device (mostly sequencers) to see if it's still there. None of these are crucial to typical real-time performances, so MINI does not incorporate them, saving space.

MIDI "Controller Messages", however, encode sound changes during a note, generated through mechanical knobs, sliders, or pickups. They *are* relevant for MINI because they affect real-time musical interaction. MINI distinguishes between Note-On/Note-Off and controller messages similarly to MIDI. If all three initial bits (the Size-Code) are zero, the remaining MINI word is interpreted as follows:

- The next $n$ Controller-Code bits identify the controller. $n$ is initially negotiated between 1 and 3.
- The $m$ rightmost bits encode the conroller value; $m$ is controller-specific.
- Since MINI words are a multiple of 8 bits, we insert padding bits between the two values as needed.

Fig. 3 shows a sample message with $n = 3$ and $m = 1$.

MIDI allows for up to 128 different controller types. Many of these are still undefined, and some controllers are quite exotic and not available on most devices or soundcards. MINI includes the following subset:

**Program Change:** 7 bits select an instrument sound ("patch"). They are encoded as in MIDI.

**Pitch Bend, Modulation, Volume, Reverb, Chorus:** In MIDI, these controllers have a resolution of 7 or 14 bits. MINI maps them all to 7 bits.

**Sustain Pedal, Sostenuto Pedal:** MIDI wastes 7 bits to encode these binary pedal values (0–63=off; 64–127=on); MINI uses one bit.

While MIDI actually defines Program Change and Pitch Bend as "Channel Voice Messages", MINI groups them together with controller messages for simplicity.

## 3. Implementation

Based on MINI, we implemented *Netmusic* (http://www.welzl.at/research/projects/netmusic). It was designed as a fun tool to jam together over the internet,

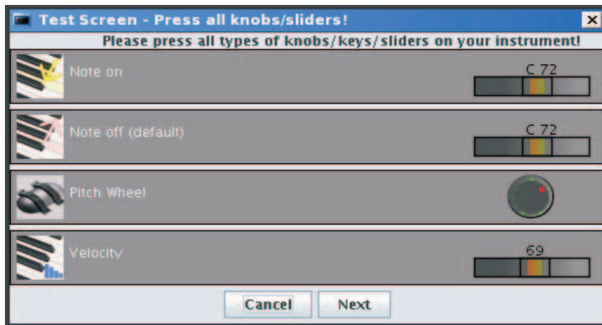


**Figure 4. Controller detection window**

but offering all the features to fully exploit the capabilities of MINI. Netmusic's core components were written in C for performance reasons, its user interface in Java. It was developed under Linux and tested with a Fedora Core 4 system (kernel v.2.6.17.1-2142). Here is a rough overview of its functionality:

- It connects to another host and maintains a TCP connection to exchange parameters, start and end the session. MINI data is exchanged via UDP.
- It captures MIDI via the ALSA(http://www.alsa-project.org) library, regarding notes within a user-defined interval as chords. It then converts them to MINI and sends them to the other host.
- It converts incoming MINI messages to MIDI immediately, and plays them via ALSA (i.e., without buffering or Timestamp messages). ALSA lets the user patch MIDI ports to hardware or software instruments, so the newly generated MIDI messages can be played on a connected MIDI device or a software synthesizer, invisible to *Netmusic*.
- The user can adjust MINI's feature–sending-rate trade-offs via a GUI. It first asks him to activate all controllers (Fig. 4) to determine the number of encoding bits needed. Controllers become visible as they are operated. Then they can be selected and configured (Fig. 5): e.g., the right-hand slider controls resolution. This also shows a new space-saving *NetMusic* feature: leaving out Note-Off messages for instruments such as a xylophone whose sound ends automatically. This application-specific mechanism is negotiated via SETUP messages over the TCP connection.

The GUI provides feedback to the user in 4 panels: a Property Window shows network details (time connected, IP addresses, port numbers). An on-screen

**Figure 5. Options window**

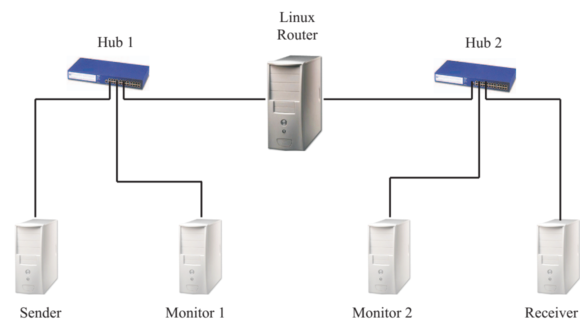


**Figure 6. The testbed**

keyboard shows notes being played. A Controller Window shows selected controllers, and a Log Window lists setup data, parameter changes and error messages.

## 4. Test setup and results

We tested MINI against MIDI using our Netmusic application and a local testbed to simulate network congestion that has been used successfully before ([14],[4]). We used a unidirectional flow scenario (as if one musician plays and the other one just listens). This scenario is valid because a flow in the other direction is completely independent of the flow considered.

We used 5 PCs connected via 100Mbps Ethernet (Fig. 6). Monitor 1&2 created and received background traffic. Monitor 1 also logged traffic as sent, i.e., be-

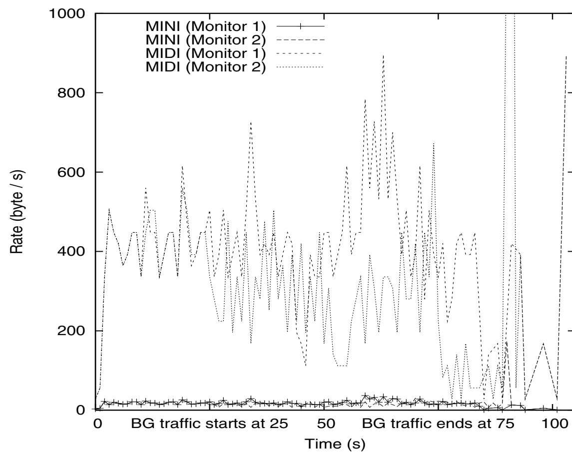


**Figure 7. MINI&MIDI rates logged by Monitors 1&2, background traffic 80 packets/s.**

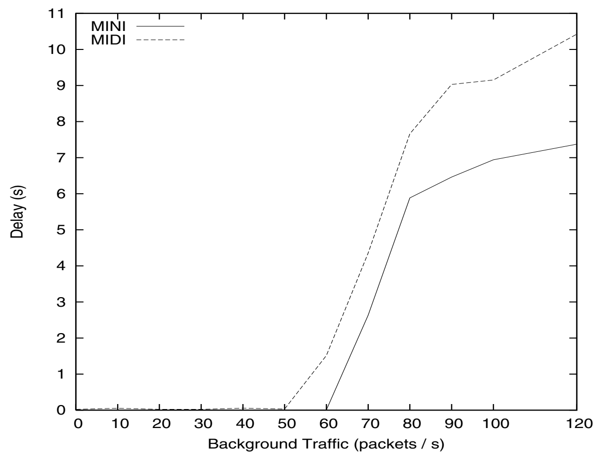


**Figure 8. Average delay of MIDI and MINI**

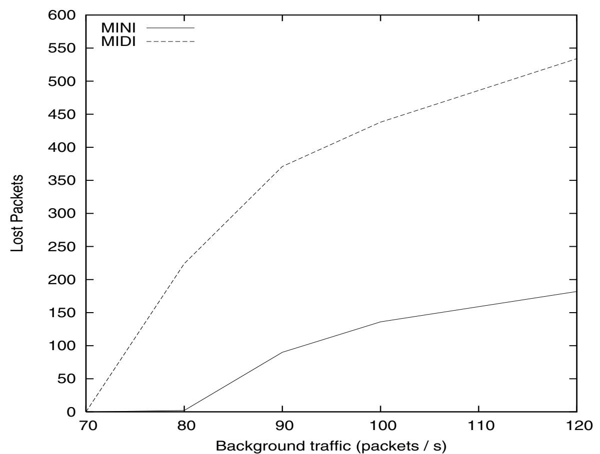


**Figure 9. Average loss of MIDI and MINI**

fore congestion; Monitor 2 logged it as received. Packet drops become easy to see as gaps between the two lines showing the rates perceived by the two Monitors (Fig. 7). Peaks on the two lines, however, can differ due to the delay in the router's queue when congested.

To cause congestion, the maximum traffic rate was limited using the tc (traffic control) Linux command and Class-Based Queuing with only one class for the PC on the receiver-side link of the router. We did not use Token Buckets as they influence traffic characteristics [5]. The monitors measured traffic over hubs 1 and 2 with tcpdump (http://www.tcpdump.org). Loss was calculated as bytes sent (logged by Monitor 1) minus throughput (logged by Monitor 2).

mgen (http://mgen.pf.itd.nrl.navy.mil/mgen.html) generated a constant-bitrate UDP data flow of 100-byte packets as background traffic. It was sent from the router to Monitor 2, thus avoiding collisions but congesting the router's outgoing queue. We generated 11 classes of background traffic for 50s each, starting after 25s and consisting of 10–100 (in steps of 10) and 120 packets/s, respectively. These rates were chosen for our Netmusic application and network setup to allow investigating their impact on the application behavior. Initial mgen packets were used to synchronize the test machines. For reproducability, we always transmitted the 1:46 piece “Préludes Nr. 4, Largo, Op. 28” by Frédéric Chopin which has frequent chords. With background traffic of 60–80 packets/s, the arpeggio effect and thus degradation of MIDI becomes quite clear to the listener.

Fig. 7 shows the rates perceived by Monitors 1 and 2 with a background traffic of 80 packets/s. Clearly, for MIDI, the incoming is much higher than the outgoing rate, so packets were dropped. The outlier of the Monitor 2 rate at the end of background traffic after 75s results from the router's queue emptying. The diagram also shows that the rate for MINI is much lower, and that its lines from Monitors 1 and 2 are close to each other, so there is little queuing delay or packet loss.

Fig. 8 and 9 show cumulative results (average delay and total loss) of all our studies; tests with traffic of 0–60 packets/s are not included in Fig. 9 as no packets were dropped. They clearly show that MINI not only fulfils its primary goal of reducing delays in real-time internet jams, but also that packet loss is reduced—an important outcome as lost packets mean lost notes or controller messages. These results were confirmed by the significantly enhanced quality perceived when listening to MINI and MIDI in our tests.

## 5. Conclusion and related work

The idea of playing music together over a network is far from new. We documented the state of the art in this field in 1998 [19]; back then, existing projects already ranged from Frank Sinatra singing with U2's Bono via a dedicated fiber link, to the "Res Rocket" (later "Rocket Power") system which was even linked to the popular "Cubase VST" sequencer for a while. In Res Rocket, musicians collaborated by editing sequencer tracks in real-time, but interactivity was limited, as musical updates were only disseminated when a button was clicked. Res Rocket is not available anymore (see http://www.jamwith.us/about_us/rocket_history.shtml), and current music-sharing portals are even less interactive [13], [18], [7], [3]. Their existence and success ([13] reports 20000+ members) shows the ongoing demand of distributed musicians to collaborate.

Of particular interest is the "Networked Musical Performance (NMP)" [8] system that transmits MIDI over IP using RTP [17] by means of a new RTP packetization format [9]. It is specified in [11, 10]. Delayed or lost packets are compensated for by adding timestamps at the sender and using them to properly handle problems at the receiver. 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 takes an interesting approach: it does not proactively reduce latency or packet loss like MINI; its additional RTP header even slightly increases the chance of packet loss during congestion. But MINI does not include the retroactive compensation of NMP. By avoiding to prescribe such mechanisms, we ensured that MINI is flexible: we intend to implement a MINI based application that includes these NMP features.

## 6. Acknowledgments

We thank Klaus Hörmann, Georg Regensburger and Elmar Weiskopf for their contributions to this work.

## References

[1] MIDI Manufacturer's 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", *Proc. of the MULTIMEDIA '99 Seventh ACM International Conference on Multimedia (Part 1)*, ACM Press, NY, 1999, pp. 37–46.

[3] C. Chafe, "Distributed internet reverberation for audio collaboration", *24th AES International Conference*, Audio Engineering Society, Banff, 2003.

[4] S. Hessler and M. Welzl, "An empirical study of the congestion response of Real Player, Windows Media Player and QuickTime", *Proc. of the 10th IEEE International Symposium on Computers and Communications (ISCC 2005)*, La Manga del Mar Menor, Cartagena, Spain, IEEE Computer Society Press, June 27–30, 2005.

[5] G. Huston, "Next steps for the IP QoS architecture", *RFC 2990*, IETF, Nov. 2000.

[6] ISO, *14496 Standard (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", *Proc. of the NOSSDAV '01 11th International Workshop on Network and Operating Systems Support for Digital Audio and Video*, ACM Press, 2001, pp. 157–166.

[9] J. Lazzaro and J. Wawrzynek, "An RTP payload format for MIDI", *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)*, IETF, Nov. 2006.

[11] J. Lazzaro and J. Wawrzynek. "RTP payload format for MIDI", *RFC 4695 (Proposed Standard)*, IETF, 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", *Proc. of IHM-HCI 2001*, Lille, France, AFIHM/BCS, Sep. 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", *Proc. of the 14th ACM NOSSDAV International Workshop on Network and Operating Systems Support for Digital Audio and Video*, ACM Press, June 2004.

[15] A. Nijenhuis and H. Wilf, *Combinatorial Algorithms*, 2nd ed., Academic Press, Reading, MA, 1978.

[16] R. Rejaie, M. Handley, and D. Estrin, "Quality adaptation for congestion controlled video playback over the internet", *Proc. of the SIGCOMM '99 Conference on Applications, Technologies, Architectures and Protocols for Computer Communication*, ACM Press, 1999, pp. 189–200.

[17] H. Schulzrinne, S. Casner, R. Frederick, and V. Jacobson, "RTP: a transport protocol for real-time applications", *RFC 3550 (Standard)*, IETF, July 2003.

[18] V-Band, 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, 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.