Challenges and Performance of High-Fidelity Audio Streaming for Interactive Performances

Nicolas Bouillot
Centre for Interdisciplinary Research
in Music Media and Technology
McGill university
Montreal, Qc, Canada
nicolas@cim.mcgill.ca

Jeremy R. Cooperstock
Centre for Interdisciplinary Research
in Music Media and Technology
McGill university
Montreal, Qc, Canada
jer@cim.mcgill.ca

Abstract

Low-latency streaming of high-quality audio has the potential to dramatically transform the world of interactive musical applications. We provide methods for accurately measuring the end-to-end latency and audio quality of a delivered audio stream and apply these methods to an empirical evaluation of several streaming engines. In anticipation of future demands for emerging applications involving audio interaction, we also review key features of streaming engines and discuss potential challenges that remain to be overcome.

Keywords: Networked Musical Performance, high-fidelity audio streaming, glitch detection, latency measurement

1. Introduction

Current high-fidelity audio streaming activity covers a wide range of interactive applications between remote performers. Sample applications include ensemble recording [1], group interaction with minimal latency [2, 3], group interaction with controlled latency [4, 5], and outdoor mobile interaction [6]. Recent advances in distributed interactive performance have shown the need for coupling video, gesture and other sensor-based interactions with audio transmission in order to support simultaneous music, dance, theater and other interactions in contexts including rehearsal, teaching and performance. These requirements motivate high-fidelity audio streaming as the foundation for many aspects of music-making and remote performing. High-fidelity audio streaming engines will be a key part of large-scale distributed applications, such as the i-Maestro [1] and the World Opera [2] projects, where many distributed applications are run simultaneously in a highly heterogeneous environment. In this context, high quality streaming engines face several challenges, including simplicity of configuration, latency control, and adjusting to the competition for network resources resulting from additional independent streams.

In prior literature, latency has typically been the first challenge to be addressed. Although the International Telecommunication Union defines a latency of 150 ms as the threshold for acceptable quality telephony, musicians perceive the effects at much lower values, often less than 25 ms, depending on the style of music they are playing. Experiments conducted at CCRMA and USC [7][8] have investigated these limits. Chew found that latency tolerance in piano duet performance varied depending on the tempo and types of onset synchronization required; for a particularly fast movement, this figure was as low as 10 ms. However, for certain genres of pattern-based music, significantly higher latency (several seconds) was found to be acceptable, and in fact, preferable to low latency, provided that the delay value corresponded to a whole-number multiple of the pattern duration [4].

These studies helped characterize the impact of latency on musicians interacting exclusively via audio but did not consider the quality of audio delivery itself. Here, quality implies playout of the correct audio sample at the designated instant of time. Latency can be reduced at the expense of quality, but deployment of audio streaming in most practical contexts requires both accurate control of user-to-user latency *and* sufficient quality of audio delivery, regardless of the hardware and software used. We are thus motivated to gain a better understanding of the capabilities and performance provided by current high-fidelity streaming engines, as relevant to supporting distributed audio interaction. In order to do so, we must first define the methods and metrics for evaluating user-to-user latency and the quality of delivered audio, across systems. With such tools, we can then conduct a performance evaluation of uncompressed audio streaming engines, taking into account these issues in addition to the choice of parameter configuration. The experiments take a consistent black-box approach, in which only the audio input and output are analyzed. Our subsequent analysis provides the basis for an extended discussion of the challenges

[1] http://www.i-maestro.org/
[2] http://www.theworldopera.org/

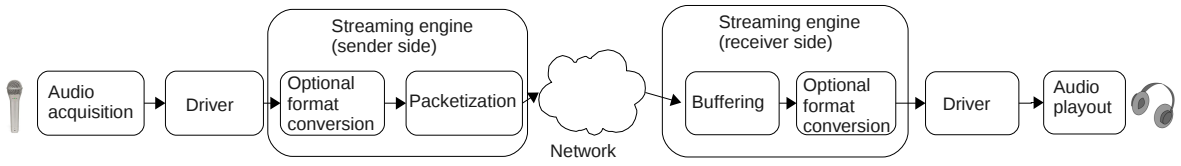

Figure 1: Components involved during audio streaming

that must be addressed by such systems. To our knowledge, such methods and comparisons have not previously been addressed formally in this context.

The remainder of this paper is organized as follows. First, we present a general overview of high-fidelity audio streaming engines in Section 2. We then introduce our methodology for measuring audio latency and delivered audio quality in Section 3. Next, we provide experimental results in Section 4, and finally, we review current streaming engine features and discuss challenges that audio streaming engines will need to overcome in the future in Section 5.

2. Streaming engines

In general, streaming engines are intended to transmit media from a source location to one or more destinations. Whereas the ideal for networked musical performance is to enable performers to interact through the audio link as if physically co-present, traditional audio streaming systems fail to support such interactivity due to their use of high-latency buffering and/or heavy compression. In contrast, *live performance* streaming uses small buffers and no compression or computationally lightweight compression algorithms to minimize latency. A significant challenge such engines face is maintaining the periodicity, at delivery, of the audio data. However, in IP networks such as the Internet, available bandwidth and scheduling are shared equally among clients, which often leads to variable performance, including packet delay variation or *jitter*.[3] Live audio streaming engines employ various strategies to reduce these network effects.

Figure 1 shows the components involved during live audio streaming. At both ends, sound cards accessed by the streaming engine through a driver provide audio acquisition and playout. Signal quality and introduced latency can be optimized using professional audio equipment; however, remote sound cards are subject to clock skew [9], which can cause data underflow or overflow. Streaming engines optionally provide a format conversion component to enable audio transmission among heterogeneous hardware, but this introduces additional latency. Audio compression can be considered a format conversion, but this may also introduce significant latency and quality degradation.[4]

[3] Note that this is a general assumption concerning end-to-end communication. Even if some sub-parts of the Internet provide guaranteed performance, this cannot be assumed for every possible IP communication.

[4] Note that ultra-low-delay codecs, such as ULD [10] and CELT (`www.celt-codec.org`), which introduce less than 10 ms of latency, allow for high-fidelity audio transmission over low bandwidth networks.

The packetization component, present on the server side, is responsible for filling a network packet with audio data. Since data are produced periodically, packetization determines the tradeoff between introduced latency and network scheduling: using a small *packet size* means waiting less time before sending the current packet, but this increases the number of transmitted packets, as well as the packet rate, which can have negative effects on performance. On the receiver side, the buffering component is responsible for ordering received packets, concealing the effects of network losses, and dealing with network jitter. This is achieved by placing the data in a queue, from which samples are passed to the sound card at regular intervals. The queue size must balance between variability in packet arrival times and latency overhead: a small queue size minimizes latency, but may result in a disrupted audio stream if data is received late due to higher than expected jitter. A buffer manager decides when audio data must be unqueued and forwarded either directly to the sound card driver, or to an optional format conversion component.

The description above provides an overview of some of the challenges faced by streaming engines designed for interactive installations and performances. However, the engine itself is not the sole determinant of audio quality. Rather, this is additionally dependent on external parameters, including the varying state of the network and operating system scheduling. Accurate measurement of end-to-end latency and resulting audio quality has thus remained an open problem, despite the fact that it is critical to the successful deployment of interactive remote applications.

3. Methodology

Streaming engine settings engine are typically tweaked manually to optimize perceived quality. In most cases, an undersized queue, failing to avoid network jitter, and network losses lead to missing audio data that produces audible glitches during playout. While subjective evaluation by the actual performers exacerbates the complexity of deployment, it is nevertheless important to allow the performers to evaluate the effects of latency while actually interacting. In this section, we provide methodologies for latency and signal defect measurement. To our knowledge, such methodologies have not previously been described formally in this context.

Although end-to-end latency measurements are difficult to accomplish over long distances, they can be completed easily in a laboratory. Figure 2 shows a setup where a trans-

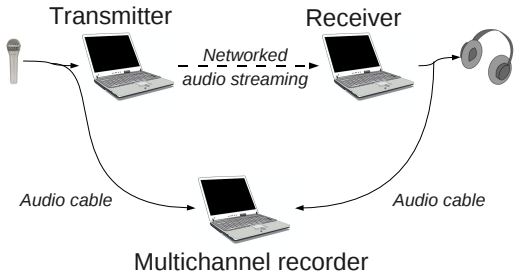

Figure 2: Setup for end-to-end latency measurement of a single audio stream

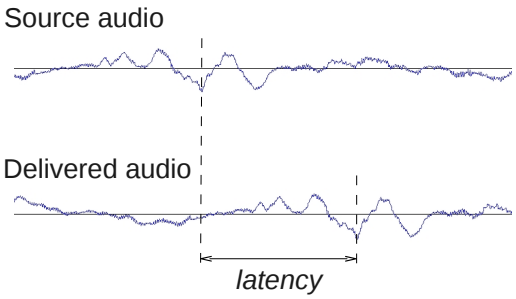

Figure 3: End-to-end latency measurement using a multichannel editor. Here a 30 ms window is displayed.

mitter streams audio to a physically nearby receiver. In this way, the sound source, as well as the resulting audio, can be duplicated and recorded using a multichannel recorder. Since we assume that every channel is recorded simultaneously, the resulting audio file allows for an off-line comparison of both audio quality and temporal differences. [5]

3.1. Accurate end-to-end latency measurement

End-to-end latency measurements, e.g., from sound source to the listener, rely on many aspects of hardware, software and the underlying network. As such, accurate measurement cannot be obtained from within the operating system or streaming engine. Fortunately, the previously described configuration allows for simultaneous recording of the source and delivered audio, outside of the operating system and sound engine. Although it may be useful to compute latency and audio quality automatically and continuously from the sound file by using cross analysis, such a tool does not, unfortunately, appear to exist within the sound processing community. The MATCH software [11] includes an algorithm for temporal alignment of audio, but this cannot provide useful statistics for our purposes since it is unable to cross analyze sound files containing glitches. A cross-analysis algorithm that would allow such an evaluation is under development by the Audacity software team, [6] but at the time of this writing, is not yet available for use.

Instead, we employ a simple manual method for latency evaluation, illustrated in Figure 3. Using a multichannel audio editor, sound files are displayed and temporally compared. We can zoom into the waveform and identify two samples that represent the exact same sound, using their offset to measure the temporal difference. This method, although manual, provides measurements accurate to a single sample. In our case, we employ a 48 kHz sampling rate, which provides an accuracy of 0.02 ms. Note that the audio content must be non-periodic in order to be amenable to visual pattern recognition; we must avoid purely sinusoidal or other repeating signals.

3.2. Measuring audio quality

As noted in Section 2, the typical primary source of quality degradation is missing audio data at one of the components. When such a problem occurs, a glitch can be heard in the playout. A secondary source of distortion is a format conversion component that alters the audio fidelity. For the purposes of our present study, since we use high-fidelity transmission without compression, we only consider glitch distortions due to missing audio data.

During our experiment, we observed that delivered audio was subject to two kinds of defect. The first, seen in Figure 4, is the result of significant amplitude variation over a small time interval, while the second, called micro silence, is the result of silence insertion inside an audio stream. These two distortions are heard as clicks.

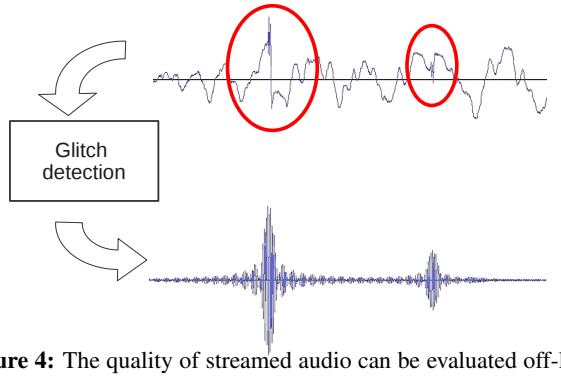

Figure 4: The quality of streamed audio can be evaluated off-line by counting glitches. A band pass filter set to 8-9.5kHz, followed by an amplification of 25 dB, allows for isolation of glitches in the resulting audio file, and allows us to count glitches over time by using an additional algorithm.

In order to characterize the quality of the delivered audio after transmission, we developed an algorithm that detects and counts such audio glitches. This detection, illustrated by

[5] This use of a multichannel audio recorder, connected directly to both the transmitter and receiver constrains the two units to be close together. To perform similar measurements over a long distance, i.e., with the audio input and output recorded remotely from each other, over a network, globally synchronized clocks are required. In this case, each audio sample could be time-stamped and re-aligned offline. However, further investigation is required to demonstrate the reliability of such a method.

[6] `http://audacityteam.org/wiki/index.php?title=Audio_Diff_Notes`

Figure 4 consists of applying a band-pass filter that isolates frequencies between 8 and 9.5 kHz, followed by an amplification of +25 dB. The resulting audio file, when displayed, exhibits spikes that correspond to the previously described audio distortions. A simple algorithm is then employed to count these spikes, based on amplitude analysis. As seen on Figure 4, this approach provides accurate glitch detection, but is subject to false positive error when the original audio source contains significant frequency components inside our window of analysis. To address this problem, we also apply our algorithm to the recorded source to locate (false) glitches in the original signal, and exclude these from the corresponding analysis of the received audio signal.

4. Experimental results

Our experiments used a one hour radio broadcast containing both speech and music. We measured latency every 10 minutes in the recording, then computed averages for the entire sound file. We compared the following four low-latency, high-fidelity streaming engines:

- JackTrip version 1.0.2 alpha, [7] developed by Juan-Pablo Caceres at Stanford University is a client for the Jack audio connection kit [8]
- jack-tools version 0.0.2-4, [9] developed by Rohan Drape, includes a streaming engine that acts as a Jack client
- Soundjack, [10] developed by Alexander Carôt at the University of Lübeck, is a stand-alone program that can also act as a client for Jack
- nStream version 1.0, [11] developed by Nicolas Bouillot at McGill University, is an external object for PureData [12] and PureData anyWhere [12] [13]

The configuration of these streaming engines consists primarily of manually setting the *packet* and *queue* sizes. Although some engines allow dynamic parameter changes, we did not vary the parameters during our one hour sessions. This ensured that we could analyze the stability of end-to-end latency throughout an extended session.

Our experiments consisted of three conditions. In the first, we used large parameter values to evaluate the quality of the resulting audio when the engine components executed essentially without temporal constraints. Second, we evaluated the latency and audio quality of transmissions with the engine configured using the minimum values allowed. Third, we manually optimized the parameters for each engine to find a balance between reducing latency and minimizing the number of resulting glitches.

[7] http://ccrma.stanford.edu/groups/soundwire/software/jacktrip/

[8] http://jackaudio.org/

[9] http://slavepianos.org/

[10] http://www.livemusicportal.eu

[11] http://www.audioscape.org

[12] a fixed point version of PureData for small form factor hardware

Our setup, illustrated in Figure 2, used computers running Ubuntu Linux 8.04 with a real-time kernel and RME Hammerfall Multiface II sound cards. Each streaming engine was configured to access the sound card through the Jack audio connection kit, configured with 64 samples per frame and two frames per buffer. We used a sampling rate of 44.1 kHz, except for the Soundjack engine, which only supported a 48 kHz sampling rate. [13]

For the first set of experiments, a common theoretical latency was specified by setting the *packet* and *queue* size parameters to the values required to produce an overhead of 10240 samples, corresponding to an anticipated latency of 232 ms at 44.1 kHz sampling, or 213 ms at 48 kHz. However, the actual (much smaller) observed latencies, seen in Table 1, for jack-tools and JackTrip, suggest that these two engines use the parameters to determine buffer capacity rather than buffer latency. Interestingly, perhaps due to sound card clock skew, Soundjack latency decreased from 218 to 165 ms, indicating that its buffer management did not try to maintain a constant amount of queued data within the buffer. While this is not necessarily critical, it could lead to undesirable underflow or overflow in longer sessions. The quality results indicate a range of 7 to 16 glitches through the transmission of the one hour recording. Since these values are all non-zero, this justifies the need to establish the range of parameters we seek in the third experiment.

Table 1: Best audio quality observed, expressed as the number of glitches measured during a one hour streaming session (ordered by decreasing quality)

Engine	# of glitches	Anticipated Latency (ms)	Actual latency (ms)
Soundjack	7	213	165–218
nStream	10	232	243
jack-tools	13	232	22
JackTrip	16	232	11

In our second set of experiments, we set the *packet* and *queue* sizes to their minimum allowed values. In theory, this should result in the smallest possible end-to-end latencies. For all the engines, the minimum packet and queue sizes were both 64 samples, apart from jack-tools, for which the minimum buffer size was 256 samples.

Not surprisingly, as observed in Table 2, there was a clear inverse relationship between latency and number of glitches. Both JackTrip and Soundjack exhibited more than one glitch per second, on average, over the one hour recording. While this is clearly unacceptable from the perspective of listening experience, it is nonetheless encouraging to note the feasibility of networked audio transmission under a 10 ms thresh-

[13] Note that using a higher sample rate should result in better end-to-end latency results, but at the expense of increasing both the bandwidth used and the packet rate.

Table 2: Latency and quality observed using minimum theoretical latency (ordered by increasing latency)

Engine	latency (ms)	# of glitches
JackTrip	5.12	20229
Soundjack	8.75	5387
nStream	11.87	130
jack-tools	12.47	78

old using high-performance networks.

The last set of experiments involved setting *packet* and *queue* sizes manually to obtain an approximate "best" tradeoff between latency and quality. As it was not feasible to do so exhaustively for every possible combination of parameter values, configurations for optimized transmission were determined manually starting from the minimum packet and queue sizes and increasing the latter until this resulted in a subjectively acceptable low number of glitches. In the case of Soundjack, however, further increase to the packet size was necessary before acceptable quality was achieved. [14] Algorithmic measurement was then used to confirm that a good tradeoff was achieved. The resulting configurations, along with measured latency and number of glitches, can be seen in Table 3. As a proof of concept, this demonstrates that low latency can be achieved using existing high-fidelity audio streaming engines, provided that the parameters (at least, buffer size) is adjusted manually to achieve the best tradeoff between latency and audio quality. However, it is important to note that this experiment was conducted using an isolated 100 Mbps Local Area Network, where performance was significantly better than one can expect from the wide-area Internet. Moreover, a given configuration that works well at one time may fail later because of network performance variations. Such fluctuations dramatically complicate the deployment of distributed applications that require audio streaming.

Table 3: Results observed while minimizing latency and maximizing quality (ordered by increasing latency)

Engine	Results		Settings	
	latency (ms)	# of glitches	packet (samples)	buffer (samples)
JackTrip	7.04	18	64	128
jack-tools	12.92	13	64	320
nStream	14.54	18	64	128
Soundjack	21.44	52	512	512

[14] This may be due to limitations of the operating system scheduler, which is unable to support the Soundjack requirements efficiently when packet size is below 512 samples.

5. Current features and future challenges

Today, projects involving distributed musical systems touch on many aspects of interactivity, and integrate a variety of software technologies and communication modes. At the communication level, audio, as well as video, real-time data, file sharing, monitoring data and other streams are transmitted simultaneously, each requiring its own level of performance. Far from being fully integrated, a high degree of flexibility is needed to ensure adequate usability in meeting each scenario. In this context, we enumerate multiple challenges for future research and for implementing high-fidelity streaming engines.

Auto-configuration Our experiments have illustrated the difficulties that arise when deploying a system that seeks to achieve optimal audio quality. This complexity is a significant obstacle to creating a system covering multiple sites. Appropriately configuring *packet* and *queue* sizes, the two parameters used by the streaming engines, is crucial for overcoming the problems presented by a variable network.

Fortunately, network state can be monitored in real-time by the receiver side of the engine, which can continuously track both network losses and jitter for adapting packet and buffer sizes. Format conversion, including sample depth variation and ultra-low delay compression, is also an interesting option that can provide the necessary foundation for designing adaptive algorithms that will be able to dynamically compute appropriate data rates. By ensuring efficient use of network resources while optimizing latency vs. quality, such algorithms may significantly enhance the capabilities of streaming engines.

User control While we believe that auto-configuration is critical for the future use of audio streaming, engines should allow users to increase latency optionally. This can be critical in various performance scenarios, where the perceptible delay is a key part of the desired interactivity, or when audio delivery must be synchronized with other real-time data.

Integration Facing the challenges involved with future large scale distributed interactive applications, where interaction is not limited only to audio, we anticipate that combining the audio information with other data, plus engine state monitoring, will both be required. In this environment, it is critical that engines provide state monitoring and control using standard mechanisms, such as the Open Sound Control (OSC)[14] protocol.

Mobility As interactive mobile applications become more dominant, we must contend with limitations of mobile processors, such as low-power and fixed-point architectures. Support for heterogeneity within larger distributed interactive systems, taking into account both fixed- and floating-point systems, along with any associated format conversions, as appropriate, will be needed.

Table 4 summarizes the features provided by each of the streaming engines considered in this paper. Most of these are basic requirements for addressing the high-level chal-

Table 4: Streaming engine features

Engine	Multichannel	Compression	User controlled delay	Multicast	Fixed-point version	Dynamical configuration
nStream	√		√	√	√	√
Soundjack	√	√	√			√
JackTrip	√					
jack-tools	√					

lenges described above. In addition, we consider *dynamic configuration*, i.e., the ability to modify configuration parameters during streaming, since this is necessary both to support *user control*, as previously discussed, and as a first step toward auto-configurable engines.

Another option to consider is IP multicasting, which enables a single stream to be received by multiple hosts, taking advantage of a specific packet routing performed directly by the network hardware. While this allows a more effective use of available bandwidth and increases system scalability, it also increases the challenge of sender auto-configuration when transmitting to multiple heterogeneous receivers.

6. Conclusion

Many distributed audio performances employ high-fidelity audio streaming engines to support audio interactivity. These performances require both low latency as well as high audio quality. In this paper, we investigated the trade-off between these two factors in four high-fidelity streaming engines. After describing our methods for accurately measuring latency between source and destination and evaluating quality based on the resulting number of audio glitches, we presented the results of a preliminary experiment involving these metrics. The engines we tested provided high fidelity transmission with end-to-end latency ranging between 7 and 21 ms, well withing the thresholds suggested by the literature (10 - 25 ms) for networked musical performance.

In addition, taking into account recent trends in distributed interactive audio applications, we discuss several challenges, including *auto-configuration*, *user control*, *integration* and *mobility*, which should be considered as the next generation of streaming engines is designed for use in increasingly demanding applications. Similar investigations should also be performed over long distance transmission, and might also be extended to include other computer music related data, including video and sensor data. We expect the results of such studies will continue to provide a better understanding of distributed musical interactions.

7. Acknowledgments

The authors would like to thank Jeff Blum for his valuable comments and suggestions for revising earlier drafts of this paper and Alexander Carôt for his generous assistance with the Soundjack software.

References

[1] J. Cooperstock and S. Spackman, "The recording studio that spanned a continent," in *WEDELMUSIC '01: Proc. First International Conference on WEB Delivering of Music*, (Washington, DC, USA), IEEE Computer Society, 2001.

[2] U. Kraemer, J. Hirschfeld, G. Schuller, S. Wabnik, A. Carot, and C. Werner, "Network music performance with ultra-low-delay audio coding under unreliable network conditions," in *Proceedings of the 123rd Audio Engineering Society Convention*, (New York, USA), 2007.

[3] Z. Kurtisi, X. Gu, and L. Wolf, "Enabling network-centric music performance in wide-area networks," *Commun. ACM*, vol. 49, no. 11, 2006.

[4] N. Bouillot, "nJam user experiments: Enabling remote musical interaction from milliseconds to seconds," in *Proc. of the International Conference on New Interfaces for Musical Expression (NIME)*, (New York, NY, USA), ACM, 2007.

[5] J.-P. Caceres, R. Hamilton, D. Iyer, C. Chafe, and G. Wang, "To the edge with china: Explorations in network performance," in *ARTECH 2008: Proceedings of the 4th International Conference on Digital Arts*, (Porto, Portugal), pp. pp. 61–66, 2008.

[6] M. Wozniewski, N. Bouillot, Z. Settel, and J. R. Cooperstock, "Large-scale mobile audio environments for collaborative musical interaction," in *Intl. Conference on New Interfaces for Musical Expression*, (Genova, Italy), 2008.

[7] C. Chafe, M. Gurevich, G. Leslie, and S. Tyan, "Effect of time delay on ensemble accuracy," in *Proceedings of the International Symposium on Musical Acoustics*, 2004.

[8] E. Chew, A. A. Sawchuk, R. Zimmerman, V. Stoyanova, I. Tosheff, C. Kyriakakis, C. Papadopoulos, A. R. J. François, and A. Volk, "Distributed immersive performance," in *Proceedings of the Annual National Association of the Schools of Music (NASM)*, (San Diego, CA), 2004.

[9] E. Brandt and R. Dannenberg, "Time in distributed real-time systems," in *Proc. of the 1999 International Computer Music Conference*, (San Francisco), pp. pp. 523–526, 1999.

[10] S. Wabnik, G. Schuller, J. Hirschfeld, and U. Krämer, "Reduced bit rate ultra low delay audio coding," in *Proceedings of the 120th AES Convention*, May 2006.

[11] S. Dixon and G. Widmer, "Match: A music alignment tool chest," in *in Proc. ISMIR*, 2005.

[12] M. Puckette, "Pure Data," in *Proceedings of the International Computer Music Conference*, (San Francisco), pp. 269–272, 1996.

[13] G. Geiger, "PDa: Real time signal processing and sound generation on handheld devices," in *International Computer Music Conference (ICMC)*, 2003.

[14] M. Wright, "Open sound control 1.0 specification." Published by the Center For New Music and Audio Technology (CNMAT), UC Berkeley, 2002.