PIANO MASTER CLASSES VIA THE INTERNET

John P. Young and Ichiro Fujinaga

Peabody Conservatory of Music
Johns Hopkins University
1 East Mt. Vernon Place
Baltimore, MD 21202 USA
[jpyoung | ich]@peabody.jhu.edu

This paper describes the long-distance reproduction of professional-level piano performance by transmission of redundant indexed UDP packets containing MIDI data over the Internet. Our project was developed with the express intent of enabling remote instruction of piano Master Classes where highly accurate and subtle musical gestures must be reproduced. With MIDI grand pianos on both ends of a network connection, the sound quality of our system exceeds other means of performance reproduction. Regardless of the fidelity of live audio streaming, no arrangement of microphones and speakers can reproduce the acoustics of a grand piano as accurately as hammers striking the strings of a real, physical instrument.

1. Introduction

Everyone wants to do music over the Internet. Schools of thought in transmitting musical information converge towards two options: using relatively high bandwidth to send digitized audio samples, or using relatively low bandwidth to send musical representations. Audio samples generally have the advantage of fidelity to the source material, while representations generally have the disadvantage of timbral imprecision. Considerations for transmitting live performance over the Internet are well known, with high latency and potential data loss among the most prominent issues. Given these parameters, we reasoned that it should be possible to have the best of all possible worlds—low bandwidth and high fidelity over almost any Internet connection—for the special case of MIDI piano.

2. Background

Next to standard notation, MIDI is arguably the most universal representation of musical gesture. For many applications, MIDI is understood to be an imperfect compromise. However, for capturing the full range of possibilities in a piano performance, MIDI can be quite faithful and comprehensive. We realized, in fact, that sending MIDI to a piano equipped to interpret and play back the messages would result in exceptionally realistic reproduction of remote performance.

There were, however, a host of small issues to confront. First came the consideration of protocols. Clearly the only choice of network protocol was IP (Internet Protocol), but there remained the decision between TCP (Transport Control Protocol) and UDP (User Datagram Protocol) for transport. TCP is designed for reliability, to minimize data transmission errors by maintaining a constant dialogue between sender and receiver, acknowledging receipt of packets and adjusting for variable network conditions. As a result, TCP functions poorly in time-sensitive applications over long distances with many routing hops. There are basically three problems with TCP from a musical viewpoint: (1) To guarantee delivery of all data, TCP retransmits lost packets, causing music to stop while dropped notes are re-sent. (2) TCP ensures that packets arrive in the same order they were sent, causing notes to back up waiting for the arrival of differently routed notes. (3) TCP includes slight extra bandwidth and processing overhead to perform these error corrections (Comer 1995).

To avoid such problems, we chose UDP for packet transmission. UDP is an “unreliable” protocol, with nominal overhead, but no inherent compensation for dropped and out-of-order packets. UDP is also “connectionless”, meaning there is no dialogue between sender and receiver, so UDP has no provisions for “flow control” or response to changing network conditions. The sender transmits packets as fast as it can, and the receiver must be equipped to process them as they arrive (Comer 1995). Thus, UDP forces the application to deal with these issues when they occur. First, we address packet loss by sending multiple copies of each MIDI bundle, statistically ensuring that at least one copy of each message arrives at the recipient. MIDI data is so compact, even hundreds of copies use minimal bandwidth. Second, by adding index numbers to each bundle, we guarantee that they play back in the proper sequence, regardless of their order of arrival. These indexes also easily identify which duplicate packets can be discarded. Third, to allow for inevitable network slow-downs, we incorporate a mandatory buffer of a few seconds. This prohibits

two-way “jamming”, but the inherent delay of Internet communication already makes simultaneity nearly impossible (Eliens 1997). The number of duplicates and buffer length can be manually adjusted or allowed to periodically self-calibrate to optimize for network conditions. It should be noted that UDP, like TCP, incorporates a packet-level checksum verification to ensure data integrity, thus sparing the application from this necessity (Comer 1995).

3. Implementation

Our initial plan for this project intended to exploit Java’s networking and user interface strengths to develop a cross-platform solution. However, even with current releases of the Java Media Framework and JavaSound APIs, there are no standard Java classes for processing MIDI device input. Third-party implementations of MIDI I/O exist, but there seems to be general agreement that the current Java VM (Virtual Machine) is not suitable for “on-the-fly” event processing because of latency and timing instability. Our testing confirmed these constraints. We also observed general degradation in OS stability, presumably introduced by the need to constantly exchange data between Java code and Native Interface drivers compiled in C. So, abandoning cross-platform compatibility for the time being, we chose instead to use Opcode’s Max environment on MacOS, which has excellent support for MIDI processing, but conspicuously lacks any built-in networking objects. Fortunately, in support of their Open Sound Control initiative, CNMAT at Berkeley has made available exactly what we needed—OTUDP (Open Transport UDP), a Max object that transmits data over IP using UDP (Wright 1998).

Once a platform and environment were chosen, implementing the logic of our system was relatively straightforward. Incoming MIDI events are parsed and attached to a delta-time measured as the number of milliseconds elapsed since the previous MIDI event. These bundles are subsequently attached to a numeric index that counts every discrete parsed MIDI event (note on, note off, controller message, etc.):

Index	ms	MIDI msg		
324	44	176	64	127
325	15	144	60	98
326	10	144	67	70
327	59	176	64	0

Each bundle is next fed through a duplication loop based on the current level of redundancy derived from measurement of network packet loss rates. The replicated bundles are sent across the network to the receiving station, where all but one of each indexed duplicate is discarded, and that remaining bundle is placed into a buffer. The buffer size is derived from periodic measurement of network latency. After the buffer duration has elapsed, each bundle is sent out in numerical order, delayed by the specified time value, and sent to MIDI output.

In addition to the stream of MIDI data traveling from sender to receiver, we bounce a small amount of return traffic back to the sender, which allows for the monitoring of network latency and packet loss, and consequent adjustment of buffer size and level of redundancy. Sender and receiver agree that every nth bundle will be monitored upon sending and echoed upon receipt, for calculation of round-trip time. The percentage of echoed bundles that do not return successfully is assumed to approximate levels of packet loss.

Of course, a master class involves more than just musical information; it also requires interaction between teacher and student(s). Our expectation is that current and future videoconferencing technology will be sufficient for this purpose. Inexpensive cameras and free software are already used for many types of distance education, and the quality and affordability of these solutions should only continue to improve.

4. Technical Discussion

We continue to test, refine, and optimize the system, as there are many parameters that can be independently modified. For example, OTUDP itself allows adjustment of the size and number of internal UDP buffers for both outgoing and incoming data. Aligning the UDP buffer size with the underlying network MTU (Maximum Transfer Unit) size should increase efficiency, although in practice this is difficult over a heterogeneous network where MTUs may vary (Comer 1995).

We tested Internet UDP performance within the U.S. over a range of times and endpoints, and observed the following behavior:

Route	Latency: Min/Max/Avg	Peak Loss
In-State	~ 10 / 220 / 25 ms	~ 1%
National	~ 270 / 2600 / 350 ms	~ 4%

Currently, our resolution of delta-time between MIDI events is roughly 5 ms, which has provided acceptable performance. Rather than choose an arbitrary maximum-load scenario such as has been done with MIDI over an Ethernet LAN, we have instead used transmission of an actual piece of music for our “worst-case” evaluation (Foss 1996).

Using our measurements as a guideline, we can calculate the appropriate parameters for transmitting, for example, Liszt's Piano Sonata in B Minor, a 25-minute piece with approximately

40,000 MIDI events. Achieving a 1/8000 probability of a dropped event, or one every five minutes, implies 80 duplicates for the in-state connection, and 320 duplicates for the national connection. A buffer equal to or slightly greater than the maximum latency should produce an uninterrupted performance, even during periods of peak musical density. Upon execution of these parameters, we have so far been pleased with the fidelity of the Liszt received.

However, optimal system parameters are difficult to determine, for a variety of reasons. Conditions on the Internet can change dramatically from moment to moment, so effective on-the-fly adjustment can become complex and challenging, as demonstrated by the array of tactics used to make TCP reliable. A large sudden latency increase can leave the receiver stranded with no data before the sender is even aware of the problem. And because packet losses occur in clusters, mainly due to overloaded routers that simply drop everything until congestion declines, statistical certainties based on average drop rates don't guarantee success (Bolot 1993).

We will soon begin (or may be underway by the time you read this) setting up demonstrations with conservatory musicians to tune usability and performance of the system, and will incorporate their feedback into the design where appropriate. The Max code will be freely available, and any suggestions or improvements will be welcomed.

Currently, OTUDP requires a destination IP address or hostname and port number to be hard-coded into the patch. This restriction has been inconvenient for distributing the receiving client for use with MaxPlay, as the patch needs to be modified for each destination. Hopefully we can work with CNMAT to change this requirement in an upcoming version of OTUDP.

To enhance our error-correction schemes, we are experimenting with algorithms to gracefully release stuck notes if the expected note off message does not arrive.

Once the theoretical basis of our system has proven robust, we will once again pursue a cross-platform strategy to encourage maximum dissemination and usefulness to the music and education community.

Updates will be periodically posted to the Web at http://www.peabody.jhu.edu/~jpyoung.

We would like to thank Matt Wright and The University of California at Berkeley CNMAT for providing the public with their Max objects OpenSoundControl and OTUDP, and for providing the authors with feedback on their use.

References

Bolot, J.-C. 1993. End-to-end packet delay and loss behavior in the Internet. *Computer Communication Review* 23 (4): 289-98.

Comer, D. 1995. *Internetworking with TCP/IP*. 3d ed. Vol. 1, *Principles, Protocols, and Architecture*. New Jersey: Prentice-Hall.

Eliens, A., M. van Welie, J. van Ossenbruggen, and B. Schonhage. 1997. Jamming (on) the Web. *Computer Networks and ISDN Systems* 29 (8-13): 897-903.

Foss, R., and Thabo Mosala. 1996. Routing MIDI messages over Ethernet. *Journal of the Audio Engineering Society* 44 (5):406-15.

Wright, M. 1998. Implementation and performance issues with OpenSound Control. *Proceedings of the International Computer Music Conference*. 224-7. Ann Arbor, MI: ICMA.

Resources

Third-party implementations of MIDI I/O in Java, free for non-commercial use:

JavaMIDI (SoftSynth) – http://www.softsynth.com/javamidi/

MIDIShare/Java (Grame) – http://www.grame.fr/MidiShare/Develop/Java.html

NoSuch MIDI (Tim Thompson) – http://209.233.20.72/nosuchmidi/

Java MIDI Kit (Michael McNabb) – http://www.mcnabb.com/software/fantasia/index.html

MIDIChat (Niels Gorisse) – http://www.bonneville.nl/software/MidiChat/