Networking Infrastructure for Collaborative Laptop Improvisation

Greg Surges, University of Wisconsin-Milwaukee (gssurges@uwm.edu)
Christopher Burns, University of Wisconsin-Milwaukee (cburns@uwm.edu)

Abstract: We describe a network interface which enables the exchange of control data and communications among laptop performers participating in ensemble improvisation. Design objectives include simplicity, flexibility, and stylistic neutrality, supporting our vision of the network as a resource for the transformation of musical data. Integrated functions include data request/exchange, autonomous status reporting, and instant messaging. Our first software implementation runs in Pd, using the Open Sound Control protocol, with additional implementations planned for other platforms. Implementations are designed for ease of integration into existing software – critical for our target ensemble, MiLO (the Milwaukee Laptop Orchestra), where each musician performs with their own custom software. The complete protocol specification and the open-source implementation code are available online.

1 Introduction: the laptop ensemble and musical networking

Laptop performance is now a ubiquitous practice in contemporary music, and large ensembles of laptop musicians are increasingly prevalent. All-laptop ensembles are active at universities including Carnegie Mellon, Cincinnati, the Moscow State Conservatory, Princeton, and Stanford, among other places [Dannenberg 07, Kirn 07, Trueman 06, Trueman 07]. The Evan Parker Electro-Acoustic Ensemble offers an alternative model of organization, combining instrumentalists and multiple laptop performers in a large improvising ensemble.

Our group, the Milwaukee Laptop Orchestra (MiLO), defines itself as a mixed ensemble of instrumentalists and laptop performers, with some musicians serving in both roles. MiLO emphasizes interdisciplinarity, with filmmakers, artists, and poets participating alongside musicians, and with live visuals a regular feature of performance. Most importantly, free improvisation is the basic practice of the group, with each laptop performer using the software and musical approach of their own preference.

NRCI (Networked Resources for Collaborative Improvisation) is a suite of software tools designed for use by the members of MiLO. NRCI serves an introductory and pedagogical function for novice laptop performers, as a rapid application development environment for experienced users, and as a platform for research in live coding and musical networking. This paper describes the networking component of NRCI, which facilitates the exchange of musical control data and communications over local-area wireless networks. Musical networking can facilitate new forms of improvisation, composition, and computation; our protocol and implementations are intended to enable creative experimentation in all of these areas.

2 Precedents

The network music created by the League of Automatic Music Composers (1978-83) and the Hub (1986-) represent an important context for this work [Bischoff 85, Brown 02, Gresham-Lancaster 98]. The League and the Hub both exemplify the sustained and creative use of network technologies to make music which would otherwise be inconceivable. Chris Brown describes the Hub's concerns: "the idea of having musicians play with each other from distant locations was... of considerable interest to promoters, publicists, and audience.... But the band itself was always for more interested in the aspects of performer interactivity, algorithmic complexity, and the web of mutual influence that the network provided" [Brown 02]. Brian Kane goes a step farther and argues that network music is obligated to pursue the unique possibilities facilitated by the network: "Any aesthetics of Net music would, correspondingly, imply a set of musical practices that exploit... specific affordances of networks" [Kane 06].

Previous work with ensemble performance in this area has mostly involved one-off protocols and specific solutions. The Hub implemented its second-generation compositions as series of "hacks" on the MIDI protocol, using a standard communications protocol in non-standard ways [Brown 02]. The Carnegie Mellon Laptop Orchestra opted to establish a unique networking protocol specific to their December 2006 performance [Dannenberg 07]. Dan Trueman describes reusable networking code as part of the "PLOrk Utilities" library of Max/MSP abstractions; most of the applications mentioned refer to rhythmic synchronization or other forms of coordination between a "conductor" laptop and non-conductor "players" [Trueman 06]. The TUIO protocol, like the PLOrk Utilities, uses Open Sound Control, and has served as a useful model for our work [Kaltenbrunner 05].

3.1 Design goals: the protocol

The NRCI network protocol is designed to facilitate experimentation with network music in the context of a laptop improvisation ensemble. The protocol emphasizes the use of local-area networks for distributed computing, as opposed to the wide-area transmission of performance data for telepresence. While wide-area connections are possible with the NRCI protocol, our interests align with those of Matt Wright, who argues that "only when each computer is doing something interesting does a network of computers behave like a network of computers instead of unreliable microphone cables with built-in delay lines" [Wright 05].

Design objectives for the protocol include simplicity, flexibility, universality, and robustness. Simplicity implies ease of use, ease of implementation, and common-sense design concepts which can be grasped and used in the heat of live performance. Flexibility suggests that the protocol should make as few assumptions as possible about the source and generation of control data by the sender, or the transformation and use of the data by the receiver; ideally, the protocol should be stylistically neutral. Universality implies openness: the protocol should work on an many software platforms as possible, with easy integration of straightforward implementations. Furthermore, the network protocol should invite use by the ensemble, but not require that every laptop performer provide network data (at least in the context of free improvisation; network compositions may be a different matter). Robustness suggests the graceful handling of errors and network dropouts, as well as the economical use of available bandwidth.

Note that the protocol does not address the networked exchange of audio. A number of members of the ensemble have been working with the live transformation of audio created elsewhere around the group. However, microphones, mixer aux send routings, and submixes have proved sufficient for this purpose to date, while sidestepping issues of latency, technical complexity, and network bandwidth.

3.2 Design goals: the implementation

Our first implementation of the network protocol was developed in the Pd environment [Puckette 96]. Pd was chosen for its familiarity, its open-source ethos, its support of OSC and networking, and because of MiLO's general preference for Pd (a majority of the laptop performers in the group use Pd at least some of the time in performance).

The primary goal for the Pd implementation is to provide a working environment for the development, testing, and validation of the protocol. Another design goal concerns simplicity and hackability - because each member of the ensemble uses their own software, and because we want to encourage (rather than mandate) widespread adoption of networking within the group, the network interface needs to be as easy to integrate into existing and new software instrument designs as possible. A third objective concerns the larger NRCI project; the networking code should be seamlessly integrated into the larger NRCI toolset, so that a performer choosing to use NRCI in performance gets networking "for free." Finally, we have tried to minimize the number of external libraries required by the networking code, in order to maximize the longevity and maintainability of the implementation.

4 The Protocol

NRCI uses a star network topology, in which all data is broadcast across the entire network via a central 802.11g wireless router. There is no central server, and all computers on the network are treated as equal peers. If any one performer experiences a crash, the rest of the performers can still take full advantage of the network. While the router still represents a single point of failure, to date we've had no issues with the reliability of installed routers at various testing, rehearsal and performance locations.

NRCI networking is built on top of the OSC network protocol. OSC provides much more flexibility than MIDI, including natural-language messaging, is widely available in computer music and art software systems, including Pd, Processing, Isadora, SuperCollider, and Max, and runs natively over modern networks. Each user directs outgoing OSC messages to 255.255.255.255 - a standard IP address for broadcasting to an entire local-area network. All messages are sent and received on port 9999, and OSC messages are transported across the network using UDP as the underlying protocol.

OSC messages follow a simple hierarchical addressing format. In our protocol, messages are generally formatted:
/receiver/sender/message-type arguments
and can also be broadcast:
/all/sender/message-type arguments
Standardized usernames are required to describe network peers (senders and receivers). MiLO currently uses first names in all lowercase characters, to encourage informality and a non-hierarchical relationship between performers.

The protocol allows for transmission of control-rate data streams from one performer to one or more other performers. Two types of communication are supported: requests (control data exchange driven by the receiver) and commands (exchange driven by the sender). The request protocol targets on-the-fly use in unscripted improvisation, while the command protocol targets more compositionally structured or otherwise preplanned situations.

4.1 Request-based control data

With the request method, one performer sends a message to another indicating a request for a data stream. Requesters indicate when a stream is wanted, and when it is no longer desired. Upon receipt of a request, the requestee begins to broadcast the desired data. An OSC message requesting a data stream is formatted:
/receiver/sender/data-type-request on/off-tag
and a message broadcasting data is formatted:
/all/sender/data-type value

In the event that multiple users request the same data stream from a given performer, they simply "tune in" to the existing broadcast stream. (Non-requesting receivers of the broadcast stream discard the stream data). Senders track the number of active requests for a given data-type; when no active requests remain, broadcasting ceases. This method minimizes redundant network traffic within the broadcast approach.

The available stream data types are pitch, amplitude, duration, and rhythmic onset. Pitch is represented by floating-point MIDI values (facilitating both equal-tempered and microtonal representations of pitch), amplitude as dB values, durations as floating-point time values, and rhythmic onsets as a stream of trigger values. The corresponding data type names are *pitch-report*, *amp-report*, *duration-report*, and *onset-report*. Each time the sender's software instrument determines that a new value is appropriate, it is immediately broadcast to the network, maintaining timing as accurately as possible within the limits of computation and network latency.

A specific request for pitch data is formatted as:
/chris/greg/pitch-request 1
(where the "1" represents "active request") and an element in the corresponding data stream broadcast would be formatted as:
/all/chris/pitch-report 78.7

While the flexibility of OSC would allow for many more than four data types, the limitation of the protocol has several virtues. First, arbitrary software instruments are relatively likely to implement processes that correspond to these types. While pitch may not necessarily be a meaningful component of a software instrument, it is much closer to universality than "feedback coefficient" or "modulation index." Second, these types are perceptually "strong": they are more likely to be clearly audible phenomena than lower-level instrument parameter changes, and this perceptual correlation suggests the use of the network as a kind of unusually precise and detail-oriented listening to one's improvising colleagues. Finally, in the setting of improvised performance, choice (and its associated interface complexity) can be overwhelming. A simple and limited number of options seems easier to manage than an open-ended situation.

4.2 Command-based control data

The second method of data transmission, commands, provides some of the flexibility eschewed by the request system. While planning a realization of Scot Gresham-Lancaster's composition *Stuck Note*, originally designed for the Hub, we realized that it was necessary for senders to have the ability to specify a specific target for a data stream [Brown 02]. The sender specifies the receiver and a descriptive name for the data stream, such as “brightness” or “filter cutoff”. This allows for any type of numerical data to be streamed from one performer to another, though unlike the request protocol, it requires coordination between sender and receiver. Command messages are formatted:
/receiver/sender/command stream-name value

For instance, *Stuck Note* requires transmission of two data types: amplitude and “x-factor” - a timbral control. Rather than modifying their locally-generated sound, musicians shape the piece through amplitude and "x-factor" control of other performer’s sounds. In our realization, command messages were formatted as:
/chris/greg/command amplitude 35.3
/greg/chris/command x-factor 75.2

4.3 Instant messaging

The NRCI protocol also includes a chat system for textual communication and coordination during performance. Performers can evaluate and discuss a performance in progress, and group decisions about musical processes and direction can be negotiated on-the-fly. This system could also be used for various kinds of extra-musical interaction between poets, artists, and musicians or even between audience and performers. Textual chat also provides the opportunity for visual projection of intra-ensemble communication, as used in Hub performances of *Vague Notions of Lost Textures* [Brown 02]. Currently, all chat is broadcast as ASCII text:
/all/sender/chat text
For example:
/all/greg/chat lets bring it to a close here...

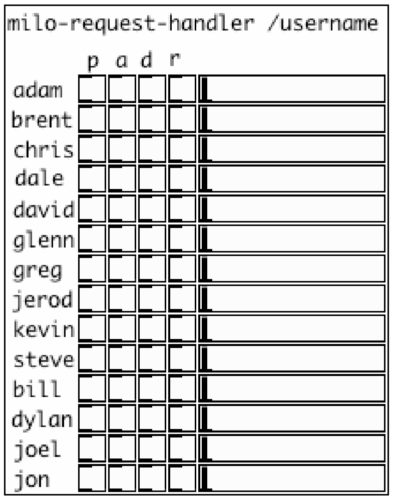
4.4 Status reporting/VU metering

Finally, the network protocol provides instantaneous amplitude reports for networked performers. The data is formatted as a floating-point value in dB, and each performer broadcasts an amplitude value every 100 ms. Amplitude reports respond to one challenge of laptop ensemble performance - the difficulty of associating a particular performer with a particular sound, gesture, or texture. Peak metering provides an easily grasped way for ensemble members to assess the activity of specific musicians in the group. These messages are formatted:
/all/sender/amp-status value
for example:
/all/chris/amp-status 64.3
We have also experimented with the broadcast of time-averaged counts of discrete interface actions by each performer; while an intriguing concept, this information seems more difficult to use in the heat of improvised performance. Perhaps this idea can be revisited (via the command protocol) as a compositional device.

5 Pd Implementation

The first implementation of the NRCI network protocol is as a set of abstractions written in Pd, using the OSCx and iemlib external libraries found in the Pd-extended distribution. Each performer initializes the networking code by entering his or her username, after which messages are dynamically created by inserting the username into a generic template.

A GUI abstraction named *request-handler* (figure 1) allows users to select and monitor their data-stream requests. The interface is comprised of a grid of mouse-selectable toggle boxes. Usernames are arranged in a vertical row, and the four request data-types are arranged in an adjacent horizontal row. To the right is a horizontal VU meter for each user. The VU meter receives the instantaneous amplitude reports broadcast by the corresponding user, and provides a visual representation of the data.

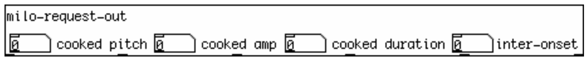

milo-request-handler /username

p a d r

adam
brent
chris
dale
david
glenn
greg
jerod
kevin
steve
bill
dylan
joel
jon

figure 1: *request-handler* screenshot. From left to right, the requestable data types are pitch, amplitude, duration, and onset, with a VU meter to the right for each user.

A second Pd object, *request-out,* provides six outputs for requested data (figure 2). From left to right: the first two are unformatted "raw" data and data parsed by username, while the last four are pitch, amplitude, duration, and rhythmic onset values from all currently requested streams. The quick access to these parameters facilitates adoption of the protocol and enables live-coding. This data can be used in any way, and cross-mapping of data-types is encouraged. For example, incoming pitch data could be used to control local amplitude, or incoming amplitude could be used as an index into an array of pitch values. The *request-out* object keeps track of a user's requested streams, and parses out all other data.


milo-request-out

0 cooked pitch 0 cooked amp 0 cooked duration 0 inter-onset

figure 2: *request-out* screenshot. The four number boxes display the data streams corresponding to the four rightmost outlets.

Broadcast of requested data is accomplished by *transmit-handler*. This object monitors incoming OSC messages, and parses out requests for each of the four possible data-types. Counters monitor the number of active requests for each data-type, and turn broadcasting for that type off when no requests exist. Local audio processes (such as the NRCI audio tools) communicate relevant control data to *transmit-handler* through standard Pd global send and receive objects; when broadcast is active, *transmit-handler* converts the control data values into OSC messages.

The command protocol is implemented using a pair of Pd abstractions, *command-send* and *command-receive*. *Command-send* takes arguments specifying the recipient and data-type. Each user can have an arbitrary number of *command-send* objects active at a time, streaming different data-types or sending to different receivers. The receiver uses matching *command-receive* objects, which take an argument specifying the data-type to be received. These objects can be created and removed during performance, but users must agree on naming conventions for data – in advance of performance, through chat, or via other means.

The chat module, *ui-chat*, receives on/off messages from another object called *key-manager*. *Key-manager* allows a user-specified keystroke to toggle between various uses of keyboard input. When chat is active, *ui-chat* monitors ASCII keyboard input and creates text strings which are broadcast with a press of the enter key. Backspace is used to clear out one entire line of text, before transmission. Received chat messages are displayed in the main Pd window, and overall the interface is similar to a standard "chat-room" environment. Due to the limited string processing capabilities in Pd, some punctuation characters are currently unavailable. Future plans for improvement include character-by-character backspace.

6 Evaluation and Future Work

We tested the NRCI networking protocol in a live public performance on December 6, 2007, at the University of Wisconsin-Milwaukee. The command protocol was used to support a realization of *Stuck Note* (as described above), the request protocol was used in several free improvisations, and the chat and VU metering features were used throughout the concert. Networking functioned without interruption or failure; in general, we have found the system to be robust.

In performance, the request protocol facilitates new kinds of interactive relationships among the ensemble members, and the very precise kind of "listening" it affords augments (rather than replacing) traditional modes of listening. Similarly, the chat and VU systems extend, rather than usurp, traditional types of inter-ensemble communication and attention. (This is especially true since the instrumentalists in MiLO don't participate in the chat system – musical modes of improvisational negotiation are still necessary in performance).

The command protocol is perhaps the least explored to date, but we are confident that it will facilitate novel kinds of musical thinking, including new types of distributed responsibility for ensemble behavior. We are very excited about the use of networking in the context of NRCI and MiLO, and look forward to more work and experimentation with these tools.

Future work for this project centers around the design of performances and compositions using the NRCI network protocol. The work of the Hub is both a model for this activity, and a target: we intend to realize and perform several more of the Hub's pieces using the NRCI tools. Additionally, we intend to implement the network protocol in other software platforms; a Processing implementation is in progress, and a SuperCollider implementation is planned.

Source code for NRCI (including the Pd implementation of the network protocol) is freely downloadable from: http://ccrma.stanford.edu/~cburns/NRCI

Acknowledgements

Special thanks to the to the members of MiLO for their enthusiastic support of this project.

References

[Bischoff 85] Bischoff, J., Gold, R., and Horton, J.. "Music for an Interactive Network of Microcomputers", in *Foundations of Computer Music*, Roads, C., and Strawn, J., eds. (Cambridge, Mass.: MIT Press), 1985.

[Brown 02] Brown, C., and Bischoff, J. "Indigenous to the Net: Early Network Music Bands in the San Francisco Bay Area", http://crossfade.walkerart.org/brownbischoff/ (2002).

[Dannenberg 07] Dannenberg, R. "The Carnegie Mellon Laptop Orchestra", *Proceedings of the International Computer Music Conference 2007.*

[Gresham-Lancaster 98] Gresham-Lancaster, S. "The Aesthetics and History of the Hub: the Effects of Changing Technology on Network Music", *Leonardo Music Journal* vol. 8 (1998).

[Kane 07] Kane, B. "Aesthetic Problems of Net Music", *Proceedings of the SPARK Festival 2007.*

[Kaltenbrunner 05] Kaltenbrunner, M., et. al. "TUIO: A Protocol for Table-Top Tangible User Interfaces", *Proceedings of the 6th International Workshop on Gesture in Human-Computer Interaction and Simulation*, 2005.

[Kirn 07] Kirn, P. "Electronica Unplugged", *Keyboard Magazine*, http://www.keyboardmag.com/article/electronica-unplugged/Jul-07/29770, 2007.

[Puckette 96] Puckette, M. "Pure Data", *Proceedings of the International Computer Music Conference 1996.*

[Trueman 06] Trueman, D., et. al. "PLOrk: the Princeton Laptop Orchestra, Year 1", *Proceedings of the International Computer Music Conference 2006.*

[Trueman 07] Trueman, D. "Why a Laptop Orchestra?", *Organised Sound* 12/2 (2007).

[Weinberg 05] Weinberg, G. "Interconnected Musical Networks: Toward a Theoretical Framework", *Computer Music Journal* 29/2 (2005).

[Wright 03] Wright, M., Freed, A., and Momeni, A. "OpenSound Control: State of the Art 2003", *Proceedings of the Conference on Instruments for New Musical Expression 2003.*

[Wright 05] Wright, M. "Open Sound Control: an Enabling Technology for Musical Networking", *Organised Sound* 10/3 (2005).