LANdini: a networking utility for wireless LAN-based laptop ensembles

Jascha Narveson
Princeton University
narveson@princeton.edu

Dr. Dan Trueman
Princeton University
dtrueman@princeton.edu

ABSTRACT

Problems with OSC communication over wireless routers are summarized and the idea of a separate networking utility named LANdini is introduced. LANdini's models and current structure are explained, and data from tests is presented. Future improvements are listed.

1. BACKGROUND MOTIVATION

1.1 Wireless routers are good

The two laptop ensembles of which we are a part - the Princeton Laptop Ensemble (PLOrk) and Sideband - both use wireless routers for group networking. This is a decision based on convenience and logistics: not having to worry about cables dramatically reduces set-up and take-down time for performances and rehearsals, and makes it easy to scale the ensemble up or down in size. In the case of PLOrk, which has at times exceeded 30 simultaneous performers, these are extremely valuable features. Avoiding networking cables also allows for a freedom of location, both on a performer basis (up in balconies, spread out around the audience, etc...), and on an ensemble basis (such as playing out-doors or in non-traditional venues). Even confined to on-stage setups, the presence of cables can impede fluid re-arrangement of performer stations from piece to piece, which is a common element in both PLOrk and Sideband concerts. In order to stay as flexible as possible, both PLOrk and Sideband intend to use wireless routers for the foreseeable future.

1.2 Wireless routers are bad

As a trade-off for all of the logistical convenience afforded by wireless routers, PLOrk and Sideband have had to deal with the less reliable performance of UDP protocols over wireless systems for OSC [1] communication, as well as the propensity of routers to occasionally drop users from the network. Latency and dropped packets have been a constant source of trouble for any piece in our repertoire that uses networking and, while some pieces can suffer a dropped packet and/or timing inconsistencies, others may be structured such that enough networking errors result in the piece failing. This is a well-known fact amongst laptop-ensemble musicians, and has been the subject of other academic inquiries [2]. Because of this, ensembles such as LSU's LOL and Virginia Tech's L2Ork choose to make the tradeoff in the other direction and perform with a wired router and a mass of ethernet cables [3, 4].

1.3 Previous solutions have left us unsatisfied

Composers who have worked with PLOrk and Sideband have approached the problems of wireless unreliability in different ways from piece to piece:

- To deal with dropped packets, some pieces take a shotgun approach by sending redundant OSC messages in short bursts in hopes that one of them will make it to its destination. This method is not guaranteed to work, and has the potential to create a lot of overhead, depending on the number and density of messages that need to be transmitted in this fashion.
- For network sync, some pieces have tried a simple server-side broadcast which either gets picked up or not on the client computers, with predictable results in terms of reliability and timing. Other pieces have employed versions of Cristian's algorithm [1] to coordinate execution time across different computers, and these solutions are often paired with the shotgun approach to set up timed messages in a network, again with imperfect results.
- Pieces requiring a specific spatial ordering of the players often require the players to select their user number manually before launching the patch - in one memorable case, a guest composer had written a piece that required everyone to change the network sharing name of their laptop to an integer and re-log-in in order to run the piece!

All of the methods above (with the exception of the username change) have worked well enough for composers to keep using them, but the current situation has three main drawbacks:

- Composers are wasting time solving problems of user-list maintenance, reliable message delivery, and useable timing anew with each piece. PLOrk and Sideband have both experienced lost rehearsal time and significant individual coding-time because of these problems.
- In addition to regular idiosyncratic behavior within the piece itself, each composer's approach to solving

the above mentioned networking issues potentially introduces another layer of erratic behavior.

- Composers sometimes avoid certain networking strategies altogether for fear of failure in a live situation. While this is pragmatic, this is obviously bad for the state of the art as a whole.

2. A PROPOSED SOLUTION

A solution to the above situation is to relegate all networking duties to a separate application which would run in the background for the duration of a rehearsal or concert. This modular approach directly addresses the three problems listed at the end of the previous section:

- Composers would no longer need to worry about how to deal with networking problems.
- Faults with basic networking issues would be easily traceable back to one common application. Assuming a stable enough application, this would make networking problems both rarer and easier to address as they come up.
- Composers could feel free to explore uses of networking that had hitherto felt too risky or complicated.

Our current attempt at addressing these problems is a software utility we've named LANdini. It is still in an early stage of development, but has to date been used in performance three times with encouraging results. It addresses issues with delivery and timing, as well as implementing some extra features that we thought would be useful, such as the "stage map" (see section 3.8).

2.1 Pre-existing solutions

There are other laptop ensembles who have also worked on solutions. One prominent example is OSCthulhu [5], a similarly motivated application developed by the group Glitch Lich (in particular Curtis McKinney). Its absence of clear documentation was an initial hurdle in its being adopted, but it also focussed on a state-based model of information flow which we didn't feel drawn to, and lacked some of the features which we envisioned, such as the "stage map" feature (see section 3.8).

Neil Cosgrove's LNX Studio [6] is quite a different type of application, being a collaborative music making environment that can work over LANs and internet connections. It's extremely well implemented, but it isn't designed to be a networking utility. What it does have is an impressively resilient network sync and message delivery system and excellent network time synchronization. It also boasts admirably open source code, and many of the features of LANdini were the result of studying and adapting solutions that were used in LNX Studio.

Ross Bencina's OSCGroups [7] is a core component of LNX Studio, but in that context is used for internet connections, which it was primarily designed for. Neil Cosgrove's code for LAN connections uses classes of his own making which feature the same API. Since the authors of this paper are interested in LAN-based music, we elected to follow Neil Cosgrove's example and implement our own solutions, leaving OSCGroups to those who are working over internet connections.

3. LANDINI'S IMPLEMENTATION

The following is a list of features that we felt would be reasonable demands to make of any networking utility that was going to be truly useful, along with explanation of how they are currently implemented in LANdini. It should be noted that, while LANdini was developed primarily for wireless networks, some of the features described could be useful for laptop ensembles on wired networks as well.

3.1 Self-contained

LANdini is a simple double-clickable application that doesn't require any extra installs. At the moment, LANdini is implemented in SuperCollider [8] to run on Mac OS 10.6+. SuperCollider was used because it was the language Narveson knew best, it has the potential of being cross-platform (though this hasn't been implemented and tested, due to a lack of Linux and PC machines), and it is easy to create stand-alone applications. SuperCollider doesn't currently support sending OSC via TCP, so UDP was used, and TCP-style behavior implemented directly. Narveson suspects this is better than the built-in latency that comes with TCP, but a parallel version in a TCP-enabled language would need to be tested to be sure.

3.2 No client/server differentiation

For simplicity and flexibility, LANdini is the same on each computer in the ensemble. In this way, members of the group can simply start LANdini at the beginning of a session and then let it run without worrying about having an extra server laptop on hand.

3.3 Dynamic user list

Each running copy of LANdini maintains a dynamically updated list. The details of how this is done are summarized in Figure 1, but involves each user broadcasting their name, ip, and port number once per second, and using incoming messages from other users to assemble a list of active LAN participants. Each user replies with a copy of their entire list, so that the sender of the original broadcast message can compare and add any users they haven't detected yet.

Once connected, each user creates local user profiles of everyone on the network and sends regular status pings to everyone on their user list. These pings contain positional information as well as info pertinent to the "guaranteed delivery" and "ordered guaranteed delivery" protocols (see sections 3.5 and 3.6 for details on the message protocols and status pings, respectively). The most recent ping time is also stored and used for removing that user from the list if too much time elapses without an update. This is currently set to 2 seconds, but can be adjusted.

Local applications can ask LANdini for a copy of the current user list either as a simple set of names or as a "stage map," as described below in section 3.8.

3.4 Minimal change to pre-existing OSC messages

In order to facilitate the updating of old pieces and the happy adoption of LANdini in to new pieces, we wanted to change as little as possible about the way OSC messages were sent and received by composers' patches.

Incoming OSC messages are, from the perspective of the receiving patch, completely unchanged, requiring no updating in the composer's code.

Outgoing OSC messages are sent through LANdini and are simply prefaced with two extra strings: a protocol and a destination. The protocol tells LANdini how to send the message, and the destination tells LANdini where to send the message. Other outgoing OSC messages are sent to LANdini to request specific information, like the current network time or user list.

3.5 Message protocols for different tasks

LANdini uses three message protocols that are based off of a subset of the many to be found in Neil Cosgrove's LNX Studio, mentioned above (see 2.1). In adopting the code, changes were made to accommodate the different internal organization of LANdini's data structures, but the general strategies are similar. Each protocol is referred to by a name that is used as a prefix in outgoing OSC messages from locally running applications:

- `/send` - this is just normal OSC
- `/send/GD` - this is the "guaranteed delivery" method, which indexes outgoing messages and stores local copies in a look-up dictionary in order to re-send upon request. Messages known to have been safely received are deleted on the sending computer, to save memory. A more detailed summary can be seen in Figure 2.
- `/send/OGD` - this is the "ordered guaranteed delivery" method, which works in a manner similar to `/send/GD`, with the addition that incoming messages are stored in an intermediate queue and passed on to the local application strictly in order. As above, the sending computer deletes messages that are known to have arrived safely, and the receiving computer deletes messages as they're performed and leave the queue.

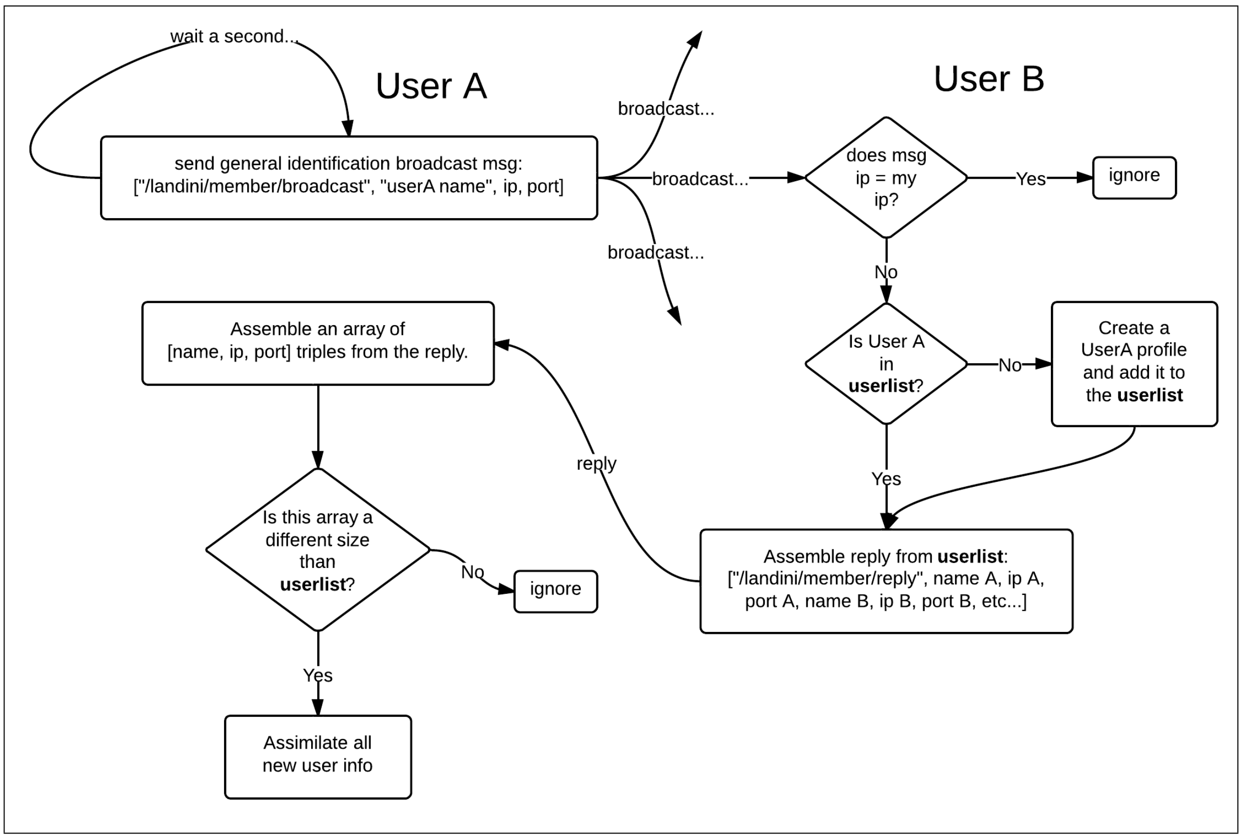
3.6 Status Pings

The network's current state is maintained through frequent status pings that are sent between all active users who are running LANdini. At the time of writing, these pings default to 3 pings per second. A breakdown of the contents of the ping and how they're used is provided below:

- name: The sender's name is used so that the receiver can put the rest of the info in to the appropriate user profile. **All variables mentioned below are stored in the receiver's user profile for that particular sender** and appear in **bold** - analogous ones exist for every separate user profile on the receiver's machine. The underscores used in the variable names have been omitted for formatting purposes.
- update position: The sender sends an updated current x/y location on the "stage map" (see section 3.8)
- check `/send/GD` IDs:
  - The sender includes the ID of the last outgoing `/send/GD` ID they sent the receiver. This is compared against the receiver's **last incoming GD ID** variable: if their last outgoing ID is bigger, **last incoming GD ID** is updated to equal it.
  - The ID of the `/send/GD` message beneath which all other messages have been safely received is included so that the receiver can delete locally stored copies of those messages it sent the sender. This keeps memory usage at a minimum.
  - The receiver looks at all the IDs in the range from (**min GD** + 1) to **last incoming GD ID**, and ask for re-sends of the ones whose IDs don't appear in the **performed GD IDs** list.
- check `/send/OGD` IDs:
  - The sender includes the ID of the last `/send/OGD` from the receiver that they performed, allowing the receiver to delete locally stored copies of those messages.
  - The ID of the last `/send/OGD` the sender sent the receiver is compared to the receiver's **last performed OGD ID**. As above, all IDs between the receiver's **last performed OGD ID** and the sender's last sent `/OGD` ID are collected, and ones that don't appear in the receiver's **msg queue for OGD** are requested to be re-sent.
- check network time server: Every status ping message includes the name of the current network time server. In the event that the network time server leaves the group, the next user in alphabetical order is automatically chosen to take up the role. There is no deep reason for choosing alphabetical order as the organizing principle for this role: better methods will be explored and implemented in future versions of LANdini.

3.7 Synched network time

LANdini automatically establishes a shared network time on boot. At the moment, this is a simple implementation of Cristian's algorithm (see Figure 3), with the commonly added refinement of using the shortest recorded round-trip time. Once more than one player is on the network, the



wait a second...
User A
User B
send general identification broadcast msg: ["/landini/member/broadcast", "userA name", ip, port]
broadcast...
broadcast...
broadcast...
does msg ip = my ip?
Yes
ignore
No
Is User A in **userlist**?
No
Create a UserA profile and add it to the **userlist**
Yes
Assemble reply from **userlist**: ["/landini/member/reply", name A, ip A, port A, name B, ip B, port B, etc...]
reply
Assemble an array of [name, ip, port] triples from the reply.
Is this array a different size than **userlist**?
No
ignore
Yes
Assimilate all new user info

Figure 1. how LANdini assembles a user list

user with the first name in alphabetical order becomes the network time server. If this server goes offline, the next highest in alphabetical order takes over with minimal interruption. Network time can be polled and used as a reference when sending messages that need to be executed at a certain point in the future.

3.8 Stage Map

This feature is, to our knowledge, unique to LANdini. Up until this point, composers have required players to choose a player number at the startup of their patch; this is clearly vulnerable to human error, and can result in lost rehearsal time or incorrect performances with missing and/or redundant parts. Moreover, requiring players to choose a player number doesn't necessarily scale well - some patches are hard wired to expect N number of players, sometimes with no better reason than the server needing to know ahead of time how many players there are.

LANdini's stage map window (when opened) represents each user on the LAN with a simple named square. Imagining the window to represent the stage or area they're in, users arrange themselves in the window in relation to the other LAN members. This information is updated through the regular ping messages, and is therefore always current. Local applications can request a copy of the stage map as an OSC message containing ordered triples of name/x/y for each user. Once this data is received, it can be sorted and used to send commands to the ensemble in whatever order the local application specifies: left-to-right, front-to-back, or other, more subtle constructions, such as those found in Gil Weinberg's paper on network topologies [9].

Asking users to arrange themselves on the stage-map is also vulnerable to human error, of course; the thinking here is that having the ensemble do this at the start of a performance is safer and more convenient than choosing player numbers for every piece that requires a specific order.

3.9 Traffic monitoring windows

LANdini has windows for monitoring incoming OSC traffic on both the local and LAN ports. Text fields on the top of the windows allow users to filter the displayed messages by typing the text of the relevant messages, allowing users to search for specific messages. For instance, if one was interested only in seeing messages sent from a performance patch that used the OSC path texttt/drums, typing drums in the filter would cause all other messages to stop being printed. This is useful functionality for debugging.

4. PERFORMANCE

At the moment, LANdini has been used in several rehearsals and three concerts, with encouraging results. Background network traffic for the upkeep of the user lists and user status peaked at about 17 KB/sec for a group of 8 laptops, which is acceptably light background usage. As of this writing there has been no chance to test LANdini with a large group of 20+ laptops, although this is a necessary and planned future step.

Narveson did run a smaller-scale test, the results of which are summarized in Figure 4. The test consisted of four

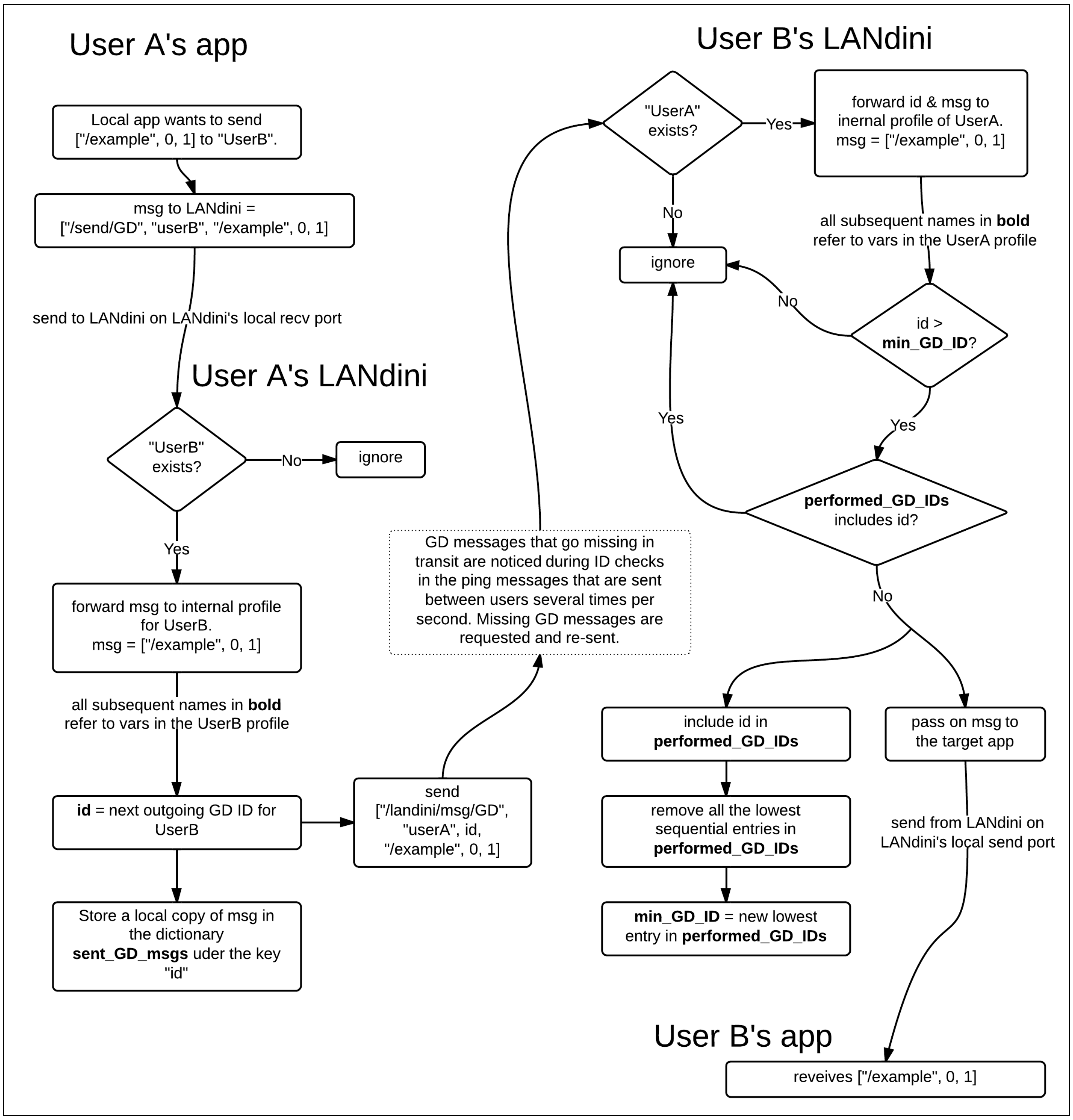

User A's app
Local app wants to send ["/example", 0, 1] to "UserB".
msg to LANdini = ["/send/GD", "userB", "/example", 0, 1]
send to LANdini on LANdini's local recv port
User A's LANdini
"UserB" exists?
No
ignore
Yes
forward msg to internal profile for UserB. msg = ["/example", 0, 1]
all subsequent names in bold refer to vars in the UserB profile
id = next outgoing GD ID for UserB
Store a local copy of msg in the dictionary sent_GD_msgs uder the key "id"
send ["/landini/msg/GD", "userA", id, "/example", 0, 1]
GD messages that go missing in transit are noticed during ID checks in the ping messages that are sent between users several times per second. Missing GD messages are requested and re-sent.
User B's LANdini
"UserA" exists?
Yes
forward id & msg to inernal profile of UserA. msg = ["/example", 0, 1]
No
ignore
all subsequent names in bold refer to vars in the UserA profile
id > min_GD_ID?
No
Yes
performed_GD_IDs includes id?
Yes
No
include id in performed_GD_IDs
remove all the lowest sequential entries in performed_GD_IDs
min_GD_ID = new lowest entry in performed_GD_IDs
pass on msg to the target app
send from LANdini on LANdini's local send port
User B's app
reveives ["/example", 0, 1]

Figure 2. LANdini's `/send/GD` protocol

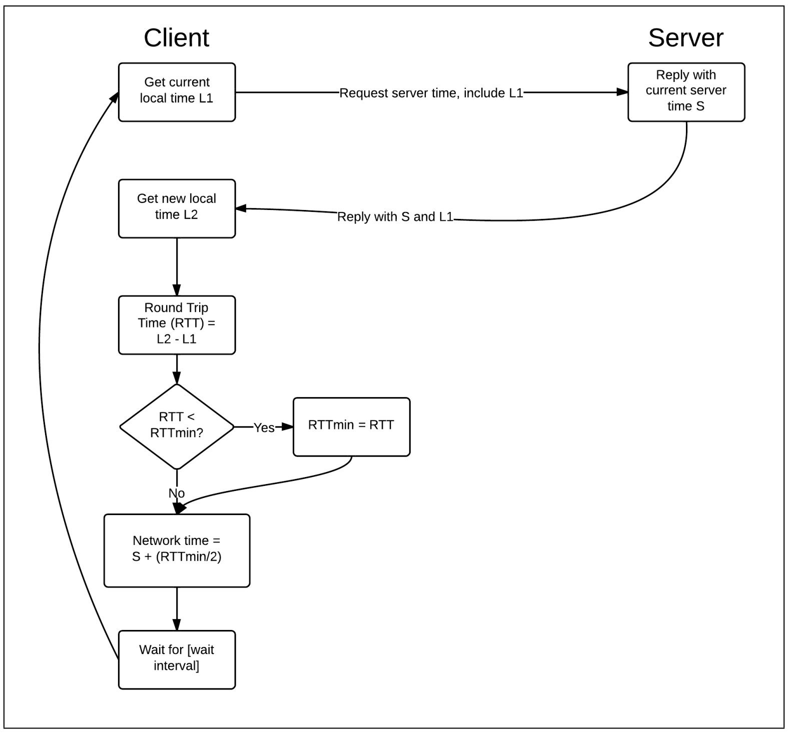

Client
Server
Get current local time L1
Request server time, include L1
Reply with current server time S
Get new local time L2
Reply with S and L1
Round Trip Time (RTT) = L2 - L1
RTT < RTTmin?
Yes
RTTmin = RTT
No
Network time = S + (RTTmin/2)
Wait for [wait interval]

Figure 3. LANdini's simplified version of Cristian's Algorithm

rounds, one for regular OSC without LANdini, and one each for LANdini's `/send`, `/send/GD`, and `/send/OGD` protocols. The tests were performed with a D-Link Dir655 router (802.11n, 2.4Ghz), a 15" MacBookPro from 2011 running OS 10.8.2 as the server, and two 13" white plastic MacBooks from 2009 running OS 10.6.8 as the clients. Each round consisted of 50 tests of 1000 messages from a server laptop to two client laptops. The tests were simple patches written in SuperCollider - each round of 1000 messages was separated by 3 seconds, and the messages themselves used a 5ms spacing. The test patches recorded message ID, protocol, and arrival time.

The results show a clear performance advantage to using the `/send/GD` and `/send/OGD` protocols, which managed a 100% arrival rate without any appreciable cost in terms of average message spacing or total time between the arrival of messages 1 and 1000 in a given test. As expected, regular non-LANdini OSC and LANdini's `/send` protocol were matched in performance, with an arrival rate in the mid-to-high 90% range, depending on the machine.

The chart in the bottom left of Figure 4 shows outliers for the between-message deltas in these tests, reaching 1.4 seconds for the `/send/OGD` protocol on machine B in the worst case. This points to further work that needs to be done to make timing issues more reliable.

A positive side-effect of the LANdini GUI is that rehearsals and concerts are sped up by making it easy to confirm that everybody is on the network simply by looking at the user list. Previously, connection problems only showed up once we started playing and noticed unresponsive behavior.

5. EXAMPLE IMPLEMENTATIONS

As of this writing, three pieces in our repertoire have been converted to make use of LANdini - below are short summaries of how this has worked so far.

5.1 Beepsh

Narveson's piece Beepsh involves the group passing around pitch and rhythm sequences. The pre-LANdini implementation used its own method for establishing the player list and for establishing group pulse synchronization. The new version uses LANdini to get the list of available participants, and uses the simple `/send` protocol and network time for sending out the rapid pulse metronome messages. Pulses are scheduled for half a pulse of latency using LANdini's network time, and this provides good synchronization. The `/send/GD` protocol is used for players to update the server about the beginning and end of their sequences, since these are more important pieces of information. The Stage Map functionality makes easy to arrange the player messages in order so that sequences can be heard traveling around the group from one side of the stage to the other.

5.2 In Line

Narveson's other piece, In Line, also involves group synch. In this case, a regular metronomic pulse at 1 pulse/second is sent to the entire group using the `/send` protocol with a scheduling latency of 100ms, which performs well. Earlier versions of this piece relied on setting up an internal metronome which would be constantly corrected by `/pulse` messages from the Server as they came in, which resulted in occasional jitter. Other, crucial messages about state-change in certain players are handled using the `/send/GD`

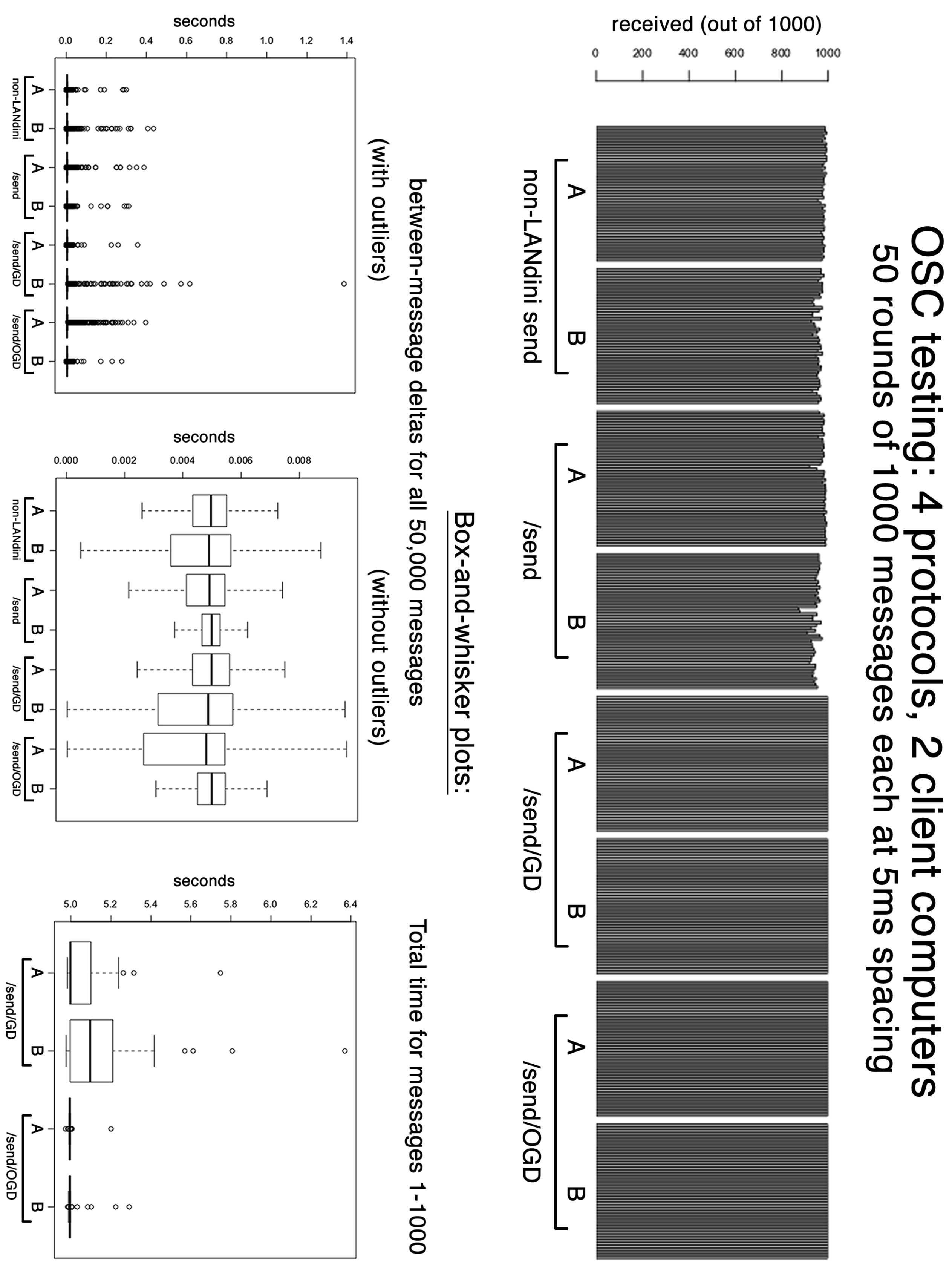


Figure 4. Test results for 50 bursts of 1000 OSC messages at 5ms intervals using four protocols across two machines

protocol, again with very encouraging results; earlier versions of the piece relied on a "shotgun" approach to ensuring the arrival of these messages, with occasional glitches. This piece has a history of crashing the router, which the current LANdini-enabled version has not done to date.

5.3 CMMV

Trueman's piece CMMV also uses a mixture of `/send` and `/send/GD` protocols, the former for time-sensitive beat information and the latter for less time-sensitive but musically important pitch information. Again, the performance of both has been very encouraging, with noticeable savings in rehearsal time due to the absence of networking problems.

5.4 Linked List test

Trueman has long tried to implement a simple "linked list" style OSC test in ChucK [10] in which two (or more) computers on a LAN pass a "play" message around a list: computers wait for the message and then, upon receiving it, play a simple sound and send a "play" message to the next user on the list. This deceptively simple test has so far failed to work, likely due to dropped packets. Using LANdini's `/send/GD` protocol solves this particular issue and enables the test to run smoothly.

6. FUTURE WORK

LANdini is currently being developed in SuperCollider, though porting it to other languages and environments is something that could be done if the need arose. One imminent future application is a port to iOS for Daniel Iglesia's new mobile music platform MobMuPlat [11].

There is currently a SuperCollider class - LANdiniWatcher.sc - which takes care of regularly polling LANdini for network time and user list info. This has the advantage of saving composers working in SuperCollider the need to setting up their own OSC loops to and from LANdinin to access network information, thus cutting down on time and potential problems with implementation from piece to piece. Skeletal classes for Max and ChucK exist which currently handle polling for network time, but need to be fleshed out to handle the "stage map" information.

While LANdini has been used in performance with 8-9 players, it has yet to carefully tested with groups that size or larger. Testing large group sizes is important because LANdini's background traffic currently grows in quadratic time: a group of size N sees (N * (N-1)) ping messages being sent every ping interval, which is set to 3 pings/second by default. For a group of 8 people this results in 8*7*3 = 168 pings per second - for 10 people that number goes up to 270 pings/sec, and, for 30 people, it would be 2610 pings/sec (!). Alternatives that get LANdini closer to "N log N" performance are clearly required, either in the form of automatic or manual throttles to the status ping frequency, a dynamic centralized server model (as is currently the case with network time, as explained in section 3.7), or something else entirely. In general, more tests related to timing and latency need to be done, as well.

At the time of writing, `/send/GD` looks for missing messages at each status ping, whereas `/send/OGD` looks for missing messages at the receipt of each new message. This should be changed since `/send/GD` currently shows marginally slower total-completion times than `/send/OGD` (see Figure 4).

Improvements to the implementation of synced network time include possible tweaks to the algorithm and the addition of visual feedback on the GUI about which machine is currently the time server. A manual over-ride of this function should be implemented in the event that the current time server machine is faulty in some way. As already stated, using alphabetical order as an organizing principle for determining the time server is an ad-hoc solution, and better methods need to be explored and implemented.

7. REFERENCES

[1] M. Wright, A. Freed, and A. Momeni, "Open sound control: State of the art 2003," in *International Conference on New Interfaces for Musical Expression*, Montreal, 2003, pp. 153–159.

[2] M. M. Cerqueira, "Synchronization over networks for live laptop music performance," 2010, senior thesis for Princeton University. [Online]. Available: http://lorknet.cs.princeton.edu/cerqueira_thesis.pdf

[3] S. D. Beck, August 2011, private communication.

[4] I. I. Bukvic, September 2011, private communication.

[5] C. McKinney and C. McKinney, "Oscthulhu: Applying video game state-based synchronization to network computer music," in *Proceedings of the 2012 International Computer Music Conference*, Ann Arbor, MI: MPublishing, University of Michigan Library, 2012.

[6] N. Cosgrove, "LNX Studio, version 1.4," 2012, open source software. [Online]. Available: http://lnxstudio.sourceforge.net

[7] R. Bencina, "Oscgroups," 2005, software utility. [Online]. Available: http://www.rossbencina.com/code/oscgroups

[8] J. McCartney, "Rethinking the computer music language: Supercollider," *Computer Music Journal*, vol. 26, pp. 61–68, 2002.

[9] G. Weinberg, "Interconnected musical networks: Toward a theoretical framework," *Computer Music Journal*, vol. 29, pp. 23–39, 2005.

[10] G. Wang and P. Cook, "Chuck: A concurrent, on-the-fly, audio programming language," in *Proceedings of the 2003 International Computer Music Conference*, San Francisco, California: International Computer Music Association, 2003, pp. 219–226.

[11] D. Iglesia, "Mobmuplat," 2013, mobile music platform for iOS. [Online]. Available: http://mobmuplat.com