Synchronization over Networks for Live Laptop Music Performance

A senior thesis submitted to the
Department of Computer Science of Princeton University

Mark M. Cerqueira

Advisor: Daniel Trueman

Second Reader: Perry Cook

16 April 2010

This paper represents my own work
in accordance with University regulations.

Mark M. Cerqueira

Many thanks to my awesome advisor, Dan Trueman,
whose guidance and good humor made this possible;

to Perry Cook and Jennifer Rexford,
for their time and advice;

to my family,
for their encouragement;

to Alyce Tzue, T.J. Fazio, Theo Beers, Lucas Giron, and Mia Tsui,
for their unfaltering friendship and undying support;

to members of PLOrk 2010,
for their patience and aid in running tests;

and to FFMCN,
for keeping me in beast mode.

Table of Contents

1. Introduction

This project aimed to explore, diagnose and solve issues present in a wireless networking environment affecting the performance of the Princeton Laptop Orchestra – an ensemble that uses laptops to produce live music. Motivated to measure the presence of inherent network properties that would be disruptive to live computer music performance, a toolkit was developed and used to verify the current networking issues PLOrk was experiencing. After discovering that these were attributed to the hardware being used to route network traffic, other devices were tested with the toolkit and found to perform better. This document describes the use of networking for live music performance and the problems it brings, related work on music performance over networks, the design of the toolkit, and the results of extensive testing using the toolkit.

1.1 Introduction to PLOrk

The Princeton Laptop Orchestra (PLOrk) was co-founded in 2005 by Princeton professors Perry Cook and Dan Trueman. As a musical group, PLOrk aims to create music from the range of players' creativity, which has the potential to "both guide the development of new instruments and technologies" while also invigorating the concept of the orchestra [Trueman 2007]. Members of PLOrk use their own laptops and additional human-computer interface devices to play compositions that are coded predominantly in ChucK. During its flagship year, PLOrk compromised only fifteen members. Today, PLOrk has over thirty members and plays with well-known groups and performers such as Matmos, So Percussion, and Anders Åstrand. Members of PLOrk use Macbook laptops running Mac OS X and connect wirelessly to an Airport Extreme Base Station that is used solely by PLOrk and does not connect to the Internet. Each member of PLOrk also has their own six-channel hemispherical speaker and subwoofer,

effectively giving each member their own instrument which allows them to contribute individually to the collective sound of the entire group.

There is no typical PLOrk composition – each composition defines itself by how it uses technology and how different people interact with that technology in a collaborative environment to produce sound. The use of technology also varies from composition to composition – some rely heavily on the wireless network in place to tightly synchronize performers whereas others are led by a person that is conducting that cues performers in and signals change in musical sections.

1.2 Introduction to ChucK

Most compositions performed by PLOrk are written in ChucK, a programming language designed for real-time music synthesis, composition, and performance that was developed by Ge Wang and Perry Cook at Princeton University in 2003. The language was designed to be flexible, allow control over the passage of time in programs, and allow synchronization among programs running at the same time on the same machine or over a network, all while maintaining a strong correspondence between code structure and timing. ChucK resembles Java, but the language relies very heavily on a unique overloaded operator called the ChucK operator: =>. Among its many uses, the ChucK operator is used to ChucK (i.e. throw one entity onto another or assign) values to variables (e.g. `3 => int foo`), and to connect unit generators in a sequential manner (e.g. `sine wave oscillator s => gain g => speaker`).

While programming languages allow a programmer to specify *what* a program will do, ChucK also allows a programmer to specify *when* a program will do certain things, making ChucK a *strongly-timed* language. ChucK has two unique primitive types, time and duration, that allow fine control over time. Time in ChucK is computable – the keyword *now* specifies the time (in samples) since a program began running. A duration is a period of time that is specified

by value::units (e.g. 1::second is a duration of 1 second). Computations are assumed to happen infinitely fast with respect to the program running, so time only advances in the program when it is explicitly advanced, which is done by ChucKing a duration to the keyword *now* (duration => now). By advancing time, the program pauses for the amount of time specified in duration and control is given to the ChucK virtual machine and synthesis engine, which produces sound if anything is connected to the digital-to-analog converter. Time can also be advanced by waiting on an event, which can be a signal sent out by another ChucK program running on the same machine, data from a human interface device, or a message sent over the network from another program [Wang 2008]. This project explores the use and issues of synchronizing over a network in the ChucK environment.

1.3 Networking in ChucK – Open Sound Control (OSC)

For communication over networks, ChucK currently implements Open Sound Control (OSC) - a protocol designed for communication among devices over networks to produce interactive music. There are many implementations of OSC in all types of programming languages, interactive sound synthesizers, and sensor/gesture capture hardware. All of them enable devices to communicate over existing network technologies – communication is most important while reliability, accurate timing, and accepting various data formats to transmit are desired features [Wright 2005]. The ChucK implementation of OSC contains two classes: OscSend for constructing and sending OSC packets, and OscRecv for receiving OSC packets and parsing the contained data. The types of data that can be transmitted are integers, floats, and strings. OSC packets are transported over the User Datagram Protocol (UDP) [Wang 2006].

UDP is commonly used for time-sensitive applications such as VoIP and online gaming. In UDP, packets are simply sent to their destination – there is no acknowledgement from the receiver that the packet has been received and packets are not resent if they are dropped. For

time-sensitive applications, it is not worth sending a packet again if it does not reach its recipient because if the packet is resent, by the time it reaches the recipient, the information contained in the packet would be old and obsolete. Consider a call over voice over IP (VoIP) – packets containing voice data are sent from caller to caller. If a packet is delayed or dropped it is not worth sending the packet again since that voice data is old. Much like a call placed over VoIP, music over networks does not require a large amount of bandwidth, but rather a consistent and reliable connection that delivers a large percentage of packets in a timely manner.

1.4 Networking for PLOrk – Problem and Motivation

Like the conductor of an orchestra, a network connecting computers is a powerful tool that can be used to create a particular type of sound from the orchestra as a whole by linking and coordinating individual performers. But unlike a traditional conductor that typically keeps players on beat, controls dynamics, and cues sections in with visual cues, a network also allows any type of information to be transmitted to all players, such as filter parameters, text messages for players to read, or control messages to manage the volume levels of players. Depending on the design of a particular composition, the network may not be used at all and a person conducts the orchestra in the traditional sense; a program can be set to conduct the orchestra automatically over the network; or a person can control a program that automatically keeps tempo but allows the conductor to adjust other settings [Smallwood, et al. 2008]. Most of the current PLOrk repertoire relies on networking for either tempo synchronization or transmission of other performance-related information.

During its earlier years, when there were fewer members, the ability to synchronize PLOrk orchestra over a wireless network was described by early composers as "remarkable, though not flawless" [Smallwood, et al. 2008]. With a single computer acting as a conductor sending out pulses to which players can synchronize to, pulses could be sent out as frequently as

every 40 milliseconds without a problem. Still, there were times when the network presented difficulties, and those involved in PLOrk knew exploring more robust and reliable means for communicating was important for compositions to function as designed [Smallwood, et al. 2008].

With the growing membership of PLOrk and increased load on the network, things have become less remarkable and more flawed, and compositions that used to function properly over the network no longer do. For example, for compositions in which a server sends a pulse over the network at a constant rate for all players to synchronize to, computers are unable to synchronize with each other. In other cases, computers are able to synchronize but do not synchronize to the steady rate of pulses the server is sending out.

The issues that have been propping up seem odd, especially considering how simple the PLOrk network topology is – all computers are always and exactly one hop away from each other on a network that is used solely for traffic that is performance-related. The amount of traffic is also not unreasonable or worrisome considering the performance of a piece like CliX, which has every performing member multicasting their presence every second for the entire performance and other performance-related information being transmitted over the network, will pass about six thousand packets over a ten-minute period – an amount that a modern router can easily handle.

Even odder are the drastic performance differences between sending packets to all hosts via the multicast protocol versus opening separate and unique direct connections to each client via unicast. Multicast appears to be the best implementation for broadcasting synchronization packets as it is both simple and efficient – the server sends a single packet to a host group address (e.g. 224.0.0.1) and the router handles distributing it to all hosts connected to the router.

Comparatively, unicast is both complicated and inefficient – the server sending out synchronization pulses must first learn the hostnames of all the devices it needs to send packets to, and then it sends out the same data to all the clients separately, increasing the load on the router based on the number of hosts the server sends pulses to.

On wireless networks, the media access control (MAC) protocol is a set of rules that regulates how the communication channels can be used to transfer information among nodes that are competing to send information. In multiple access environments, like wireless networks, only one device can successfully transmit information at any given time. When more than one device tries to transmit information, interference and collisions occur at the router [Golmie 13]. Hence, reducing the number of packets being transmitted over the wireless network should reduce the chance for collisions to occur. While multicast appears to be superior to unicast for broadcasting traffic over a network because of this increased chance to reduce collision, composers of PLOrk pieces have found that multicast just does not work. Even when there are only two devices communicating, multicasting synchronization packets is unable to synchronize the two devices, whereas unicasting to each client achieves noticeably better results.

There are workarounds to using wireless networks for synchronization, such as having players manually synchronize their computers during a piece and then perform the piece as usual. Still, having a reliable network to automate synchronization allows players to focus more on performing and less on making sure their computer is synchronized. Manual synchronization requires a person to play the traditional role of an orchestral conductor, which is not atypical for PLOrk, but should not be a requirement for all compositions. Wired networks using Ethernet to connect all of PLOrk may also be an option to explore, although the directors of PLOrk strongly prefer the current wireless setup.

1.5 Related Work

The concept of performing music over networks has become a widely researched area as the increasing power of networks has removed the traditional barriers performers had to share when performing together. But is performance over networks that are only best-effort and do not provide any quality of service guarantees even possible? This section explores past work and research done by groups that attempt to shed light on this question.

The SoundWire group, a research group based at Stanford University, has been researching how to transfer audio and perform interactive music over networks. In 2000, the SoundWire group put on several teleconcerts by streaming high-quality uncompressed audio between two musical events at separate locations on Stanford's campus. Later in the same year, the group was successful in putting on another teleconcert between Stanford and North Carolina over Internet2 [Chafe, et al. 2000]. The SoundWire group also turned the network between locations into an instrument by measuring network statistics between those two locations and then transforming that data into sound, which provided a quick way to assess the quality of the connection [Chafe, et al. 2001]. Tests were also conducted by the SoundWire group to assess the effect of delay on two performers' rhythmic accuracy when they were in separate rooms. When separated by longer delays, tempo deceleration resulted, while shorter delays (< 11.5 ms) produced tempo acceleration. A delay of about 20 milliseconds between locations was found to be optimal for ensemble performance [Chafe, et al. 2004]. The early work done by the SoundWire group illustrated that performance over networks was certainly possible.

In the Gigapop Ritual performed in 2005, researchers from Princeton University in New Jersey, USA and McGill University in Montreal, Canada set out to perform jointly while being at the two distant locations to test if interactive performance over networks was possible. Although the Gigapop Ritual and the resulting framework that was designed, GIGAPOPR, could transmit

audio, video, and MIDI data, the design principles behind GIGAPOPR attempted to work around many of the networking issues that occur when PLOrk laptops are communicating data over a local wireless network in the same room.

GIGAPOPR was designed to be straightforward while providing optimizations for operating over the low-latency, high-bandwidth Internet2 and CA2Net networks. All data transmitted in GIGAPOPR used the User Datagram Protocol (UDP). While UDP does not provide flow control or congestion control, the designers of GIGAPOPR realized that waiting for the retransmission of dropped or delayed packets that TCP implements would not be useful in a live, real-time performance. Packets sent via the GIGAPOR framework included a sequence number in the header that enforced ordering of incoming packets, allowed detection of packet loss, and made redundant transmission of packets possible. With sequence numbers it was possible to send copies of each packet to increase the chance that at least one of the packets would reach its destination [Cook, et al. 2005].

With any framework, there always exists some latency between sending a packet and receiving packet because there is a limit on how fast packets can move. During this particular performance between Princeton and Montreal, the average measured round-trip latency was between 120 and 160 milliseconds. Since there was no way to have a true real-time feed between the two locations, when performing a piece, one location served as the leading side and the other location served as the following side separated by the round-trip latency of 120 ms. In this particular performance, performers in Princeton served as the leaders and once the data arrived at McGill, the performers there played along to what they were seeing. The researchers who developed the GIGAPOR framework concluded that performances relying on a best-effort network was certainly possible and was a concept worth pursuing especially considering how

fast networks have become [Cook, et al. 2005].

In April of 2008, the SoundWire group set out to put on a joint real-time performance of Terry Riley's *In C* between Stanford University in California and Peking University in Beijing, China. With such a vast distance separating the two locations, many of the same issues present in the Gigapop Ritual had to be dealt with for this performance, which was designed to transmit both audio and video feeds. Like in the Gigapop Ritual, researchers dynamically measured the round-trip time between Stanford and Peking University, and then used that measurement to set the tempo of the eighth notes that comprise *In C*. When a note was played at Stanford, it was transmitted over the network and then played at Peking exactly one eighth note later (relative to the performance at Stanford) and vice versa for notes played in Beijing. This technique of using the round-trip time between the two locations to set a tight rhythmic alignment between vastly separated locations is known as feedback locking. The end result is a tightly synchronized piece at each of the performance locations, which allowed there to be a live, interactive transcontinental performance [Cáceres, et al. 2008].

While the GIGAPOPR framework and SoundWire performance of *In C* were synchronizing over rather large distances, some of the problems the researchers worked around, such as dropped packets, need to be addressed in the PLOrk environment. These performances and frameworks were also designed to transfer audio and video feeds, which are not currently used in any PLOrk pieces. In a typical PLOrk piece, control messages and synchronization pulses are the most common type of traffic transmitted over the network. Researchers at Waseda University developed a protocol, called the Remote Music Control Protocol (RMCP) that integrated MIDI and computer networks – traffic that more closely relates to the traffic being transmitted in the PLOrk environment. RMCP is a connectionless server-client model where

various clients broadcast messages that various servers receive and process. By relying on broadcasting, information is shared among all servers without having to specifically retransmit packets to each server. Each server fulfills a different, specified role – one may be visually displaying what is happening, and another may be producing the sound [Goto, et al. 1997]. The most interesting feature of RMCP lies in the time scheduling feature built into it.

RMCP packets have the option of including a timestamp in the packet header. If a server receives a packet before its timestamp, the packets are queued and only processed when the current time matches the timestamp. Packets received after the time in the timestamp are discarded and packets without timestamps are processed immediately. Timestamps were included to compensate for variation in network latency; however, this implementation requires that the clocks among all computers be synchronized. In RMCP, synchronization of clocks occurs via the RMCP Time Synchronization Server (RMCPtss). Essentially, all machines keep a table of the offset of their clocks to every other machine. Periodically, a machine broadcasts what time it is and all other machines then calculate their offset to that machine and update their table of offsets. With RMCPtss, time among all machines becomes absolute and timestamps can be used [Goto, et al. 1997]. The scheduling functionality present in RMCP may potentially be a useful model to emulate in a PLOrk if jitter in network latency emerges as a problematic issue that cannot be curtailed.

Roger Dannenberg of Carnegie Mellon University developed another system for real-time distribution of data over networks called Aura. Dannenberg designed Aura to be flexible in the data it can transmit while providing low-latency, real-time communication between clients. The most interesting aspect of Aura is that, unlike previously described frameworks, Aura runs over TCP instead of UDP. While UDP seems to be the best protocol for real-time transport, it does

not guarantee delivery of messages. Dannenberg found UDP to be reliable across local area networks in controlled situations, but found that multiple machines transmitting messages via UDP resulted in dropped packets. Because of the reliability issues with UDP, Dannenberg opted to try using the reliable protocol for packet transport, TCP [Dannenberg 2001].

TCP is not the preferred protocol for real-time systems: TCP buffers information to minimize the number of packets that need to be sent which creates a delay that is separate from the delay packets experience over the network; TCP also retransmits lost or heavily delayed packets, which is not useful since the data that packet contains will most likely be obsolete. Still, with a few tweaks to TCP, Dannenberg was able to make TCP's timing behavior very similar to UDP's. By turning off the option to wait for more data to come in to merge so fewer packets are sent, more packets are sent, but there is no delay between sending data and having it sent. To make retransmission of lost packets not obsolete in his real-time system, Dannenberg sent more audio samples than what was typically computed in the period between when packets are sent. If a packet dropped, there would still be data to process while the next packet was retransmitted [Dannenberg 2001]. The success of Aura as a real-time music processing system and its use of TCP helps show that one need not create a real-time system at the cost of reduced reliability – you can have both real-time and reliable communication over a network.

2. Building a Laptop Orchestra Network Toolkit (LOrkNeT)

2.1 Network Properties and Laptop Orchestra Performance

To diagnose the networking issues present in PLOrk, a toolkit was designed to collect data to assess and present a complete picture of what is going on in the network used by PLOrk. The toolkit evaluates metrics that relate directly to inherent properties of best-effort networks – there are no guarantees as to if and when traffic will arrive when it is sent. Four main issues can occur at the network-level to impair the ability of PLOrk compositions to function properly:

1. Packets can be dropped, never reaching their destination, which can result in a player missing a beat completely if a server is sending out pulses to beats of a measure.
2. There is always some latency when sending data over a network – packets take some time to reach their destination. Latency is not a problem in an environment where music is being created if it remains constant – a server sending out pulses that all players consistently receive at the same time in the future is not a problem because all the players will be in sync. For example, if all players receive the pulses a server sends out 10 milliseconds after the server sends them out, that is not problematic as players will still be in sync to a pulse that shares the same delay in reaching them. Problems occur when the latency varies from player to player, resulting in players receiving pulses at different times (e.g. player A receives a pulse 10 milliseconds after the server sends it out, and player B gets the same pulse 15 milliseconds after the server sends it out). This results in machines being out of sync as packets that are sent at the same time arrive at staggered times on different machines.
3. Issues can occur when there is a jitter in latency, in which the time it takes the packet to travel from server to player varies from packet to packet, effectively modulating the tempo

on a per packet basis. This issue can arise on a per-machine basis, where a single machine will receive packets at uneven intervals (e.g. packets are being sent every 200 ms, but packets are being received at rates varying between 150-250 ms). This issue could also arise on the orchestral level, where all machines are receiving packets at the same time, but at uneven intervals (e.g. packets are being sent every 200 ms, and everyone receives the packets at the same time, but the packets arrive at intervals ranging from 150-250 ms).

4. When a server sending packets is also producing sound with clients receiving those packets, a high latency, even without any jitter, can be problematic. Consider an example in which machines pass on messages in a sequence, signaling the next machine in the sequence to begin playing. If the latency between two machines is large, the machine sending the packet would stop playing and there would be an audible pause before the next machine begins playing. Another example – if the server plays a pulse when it sends a packet and a client plays a pulse when it receives a packet, if the latency is too high, then the machines would not sound in unison.

2.2 Designing LOrkNeT

The Laptop Orchestra Network Toolkit (LOrkNeT) was designed to measure and quantify the presence of each of the aforementioned issues present in networks that impact laptop orchestra performance. The toolkit is built around two programs, a server and a client, written in ChucK using the OSC classes that provide networking functionality. ChucK was used because it is the language used most in PLOrk, as well as the fact that it is a strongly-timed language that allowed us to track packets to a high degree of temporal accuracy.

The server program sends out packets and the client receives these packets. The server program can send out any number of packets at any specified rate, and supports both multicasting and unicasting to clients – this last feature was included to verify and diagnose the

issue that PLOrk experiences where the more efficient implementation of multicasting synchronization pulses to clients performs noticeably worse when compared to the more inefficient unicasting implementation. Packets sent out by the server include the server's network name, the rate servers are sending packets, the protocol being used to send messages (multicast or unicast), the time the server sends the packet, and a sequence number that ranges from zero to the number of packets the server is programmed to send. The client program listens for packets sent from server programs running. When a packet is received, the client logs the network name of the server, the sequence number of the packet, the time the server sent the packet, the time the packet was received, and the time interval from when the last packet was received from the particular server.

Data collected by the client programs running is then processed to analyze the issues highlighted in the previous section. Dropped packets are picked up by analyzing the sequence numbers sent from servers – a gap in the sequence numbers that increment sequentially indicates a dropped packet. Calculating the difference between the time the server sends the packet and the time the client receives the packet allows us to calculate the latency a packet experiences while travelling over the network – this will ideally remain low and constant. The intervals between when two packets arrive can be calculated by measuring the time a client receives those packets – this interval will ideally remain fairly consistent and close to the rate packets are being sent at.

One issue with this approach is that for calculating latency over the network, times reported by two different machines (the server and the client) are used to calculate the latency. Time in ChucK starts when the ChucK virtual machine begins running – unless all machines start running ChucK at the same exact time, the time ChucK reports from machine to machine will vary. While the value calculated can still be used to see if the latency remains constant, it

does not give us the actual time the packet takes to travel over the network, because unless the server and client are started at the same exact time, there will always be an offset in their times that is equal to the time difference between when the programs began running. Being able to measure packet travel time by creating an absolute and shared notion of time amongst machines would not only help calculate how long it takes a packet to travel over the network, but also allow us to confirm that packets are arriving at different machines at the same time – helping complete the analysis of the network.

Synchronizing system clocks over a network has already been studied extensively and implemented in the Network Time Protocol (NTP). In NTP, the frequency of a system clock is adjusted so that it will synchronize itself to a reference clock that provides the current time. The protocol consists of multiple exchanges of request and replies from client to server to find the offset of the system time of the client to that of the server. The client first sends a request to the server with its own time (originate timestamp) in a packet. When the server receives this packet, it attaches its own time (receive timestamp) and then attaches the time when the packet is sent back (transmit timestamp). When the client receives this packet, the client logs when the packet was received so it can calculate the delay in receiving the packet to provide more accurate synchronization. It normally takes about five exchanges of this type to fully adjust the clock frequency to achieve synchronization. When synchronization takes place over the Internet, the typical accuracy comes within 5 to 100 milliseconds of the actual time on the reference clock. Over a local area network with fewer hops between clients and the NTP server, higher accuracy can be achieved [Windl, et al. 2006].

The laptops used in PLOrk already have an implementation of NTP running on them, as do most computers since computers use NTP to synchronize their system clocks to a reference

clock. By reconfiguring the NTP daemon running on these computers to synchronize their clocks to a computer on the local area network, it is theoretically possible to achieve tight synchronization of clocks across all machines. Then, passing the system time into a ChucK program, it is possible to access an absolute sense of time in each ChucK VM by adding the system time passed into the ChucK program with the *now* of the program running.

With a notion of an absolute sense of time that is shared among ChucK programs, it is possible to measure the time it takes a packet to travel over a network because times logged by the server sending a packet and the client receiving that packet will be the same across machines. If the system clocks across the machines are tightly synchronized by the NTP daemon, then the difference between the two times will give us an accurate measure of how long it takes a packet to travel over the network. This value will ideally remain constant over time and consistent across machines.

LOrkNeT also supports running multiple servers concurrently to test higher traffic setups. While a common networking setup involves a single server sending out synchronization pulses, it is not uncommon to have a composition where there are multiple machines communicating with all other machines. When running a server, a rank must be specified – a server can either be a parent or a child. When running tests with one server, the rank of the server must always be parent because child servers will not run without receiving a start signal from a parent server. When running multiple servers, all but one instance of the server that is running as a parent should be running as children. Parent and children servers function identically in sending packets, except a child server will always wait for a start signal from a parent server, and a parent server will always send start signals to waiting children. This parent/child setup was chosen to

synchronize the traffic sent by servers so that load on an access point would be maximized by having all servers synchronize when they will send their packets.

Running an instance of the aforementioned server/client setup gives us an idea of what is going on with a certain number of laptops and servers, sending a particular number of packets using multicast or unicast at a particular rate. To make testing more thorough, scripts were written to automatically have servers send packets at commonly-used rates in laptop orchestras – packets sent every 50, 100, 200, 400, 800, and 1600 milliseconds – using both multicast and unicast implementations. Scripts were also written to have the client program run multiple times, sending data to separate files for later analysis and comparison.

To stop a client program collecting data for a particular setup during automation of multiple tests, at the end of a particular run, the parent server sends multiple packets with a magic sequence number that causes clients to terminate, thus letting the next client run for the corresponding setup. In case all the terminate packets are dropped, clients begin execution knowing the rate they should expect packets at. When a client receives a packet from a server, it verifies, using the information in the packet, that the server is sending at a rate the client is expecting. If the rate information in a packet sent from the server does not match the rate the client is expecting packets, the client does not log the information and quits, starting the next client program that should correspond to the particular server setup sending the packets. These measures ensure data collected will correspond to the actual test being run, even in an environment where packets may be dropping.

These scripts allow for comprehensive testing at a range of rates with both multicast and unicast implementations, but testing can be done on laptop orchestras of varying size. By running the client program script on a varying number of machines, network conditions can be assessed

when the number of nodes connected to the access point varies from a single computer to multiple computers. Within each of these orchestras of varying size, the number of servers can also be varied – beginning with only one server running as parent, and then adding children server.

2.3 Psychoacoustics and the Notion of Synchronicity

The client program measuring the latency a packet experiences travelling over the network and the intervals between receiving packets can provide accuracy to a single sample, but what values for these measurements should be considered acceptable? For example – regarding acceptable levels of latency – if a server plays a pulse when it sends a packet and clients play pulses when they receive those packets, how fast must the packet travel over the network for all machines to sound in sync? And regarding measuring the intervals between packets – if pulses are sent by a server every 200 ms, how far from the interval of 200 ms could the packets deviate without there being a perceptual change in what should be a consistent tempo? These questions are at the center of the field of psychoacoustics – the study of human perception of sounds.

The question regarding acceptable levels of latency deals directly with the issue of entrainment. Entrainment is a process where two separate rhythmic processes "interact with each other in such a way that they adjust towards and eventually 'lock in' to a common phase and/or periodicity" [Clayton, et al. 2004]. In the laptop orchestra environment, the aim is to achieve entrainment among all machines playing pulses. In a study, researchers looked at how two performers playing clap sticks in a Djambidj song entrained with each other as patterns and tempo changed in the song. Researchers found a synchronization bandwidth of approximately 30 milliseconds – if they played a note within 30 milliseconds of each other, there was no adjustment by either player. Researchers attribute this window to psychophysics – when the notes come within 30 milliseconds of each other, there is not enough time to evaluate the

temporal-spatial order of the events so that, while players may be able to tell that they are not in sync, they cannot tell who is playing earlier or later, so no action can be taken to improve synchrony. Only when notes were separated by a span larger than the grouping threshold of 30 milliseconds would explicit action by the performers be taken to correct for the difference [Clayton, et al. 2004].

The question regarding allowable deviations in intervals between packet deliveries has been studied by speech and music acoustics researchers Anders Friberg and Johan Sundberg. They found that during a musical performance, the interval between tones typically varies, and that this variance is used in an expressive nature. Still, the variance typically occurs at levels at which listeners do not perceive any changes in tempo. These researchers set out to assess and measure the degree of freedom, which they called the just noticeable difference (JND) in isochronous sequences. The JND is the maximum amount one can deviate from a particular tempo before a change is perceived. For tones played at various rates ranging from every 100 to 1000 ms, test subjects listened to several notes at a specified rate with the middle note out of sync. Subjects were asked to move the note using a scrollbar and as soon as the note was moved, all notes would be played. The subjects would continue to shift the middle note around until they perceived the note being in sync with the rest of the notes. Friberg and Sundberg found that for the intervals ranging from 240 to 1000 ms, the JND is 2.5% of the interval – for a rate of 600 ms, the JND is 15.0 ms, allowing performers to space their beats anywhere from 585.0 to 615.0 ms without there being a noticeable change in tempo. For intervals under 240 ms, the JND was constant, averaging an astonishing 6 ms [Friberg, Sundberg 1995].

Revisiting the data collected by LOrkNeT, we aim to calculate the following characteristics: the number of dropped packets based off of sequential sequence numbers,

whether packets arriving are within the grouping threshold of 30 ms by finding the difference between server send time and client receive time, and whether arriving packets at a client are arriving within JND and producing a consistent tempo. If packets are dropped, this is taken into consideration so that a dropped packet does not necessarily mean the following packet will be considered out of JND – for example, if packets are being sent every 50 ms, and a packet drops, the earliest the following packet can arrive at the client is 100 ms after the previous successfully delivered one was received. Even though 100 ms is beyond the 6 ms window allowed for the 50 ms rate, this issue pertains to dropped packets, and measurements of untimely delivery take this into account. Also worth calculating is the standard deviation for the latency a packet experiences travelling over the network and the intervals between when packets arrive – this will provide us with the variability of these two values which ideally should remain fairly low.

With this in mind, LOrkNeT takes the raw data it collects, calculates all the above, and then prints out a summary of this information for both multicast and unicast implementations. LOrkNeT also graphs this summary information, allowing one to visually assess the performance of the network for a particular test.

3. LOrkNeT Testing and Results

3.1 Procedure

Using LOrkNeT, trials were run for laptop orchestras ranging from the sizes of 1, 2, 3, 5, 10, and 15. Within each of these setups of varying sizes, the number of servers also ranged from 1 to the size of the orchestra being tested in the same intervals in which the orchestra grew (e.g. for an orchestra of five computers, tests were run with 1, 2, 3 and 5 servers). For each orchestra size with each particular number of servers, packets were sent at rates of every 50, 100, 200, 400, 800, and 1600 milliseconds using both multicast and unicast.

The first set of tests were run using the current PLOrk set up – all machines connecting wirelessly to the Apple Airport Extreme that is not connected to the Internet. These tests aimed to assess:

1. if there is a performance difference between multicast and unicast
2. how performance is affected by the number of nodes connected to the network
3. how performance is affected by the number of nodes sending traffic on the network
4. if performance depends on the rate at which packets are being sent
5. if all machines receive comparable levels of service from the network

Results presented here will discuss general trends, citing specific examples when appropriate. To access all the data collected for these trials, visit lorknet.cs.princeton.edu.

3.2 Results for Apple Airport Extreme

At all tested rates for varying orchestra sizes and servers running, there were noticeable differences in performance between multicast and unicast with machines connected to the Airport Extreme. The first and most major difference came in packet delivery success – when unicasting at any rate, nearly all trials yielded a 100% delivery rate, whereas multicast delivery

rates ranged from between 67.0% to 100%. Of the packets that do arrive, multicast generally provided more timely delivery – delivering a larger percentage of packets within the JND. Both unicast and multicast performed equally well in getting packets delivered within the grouping threshold.

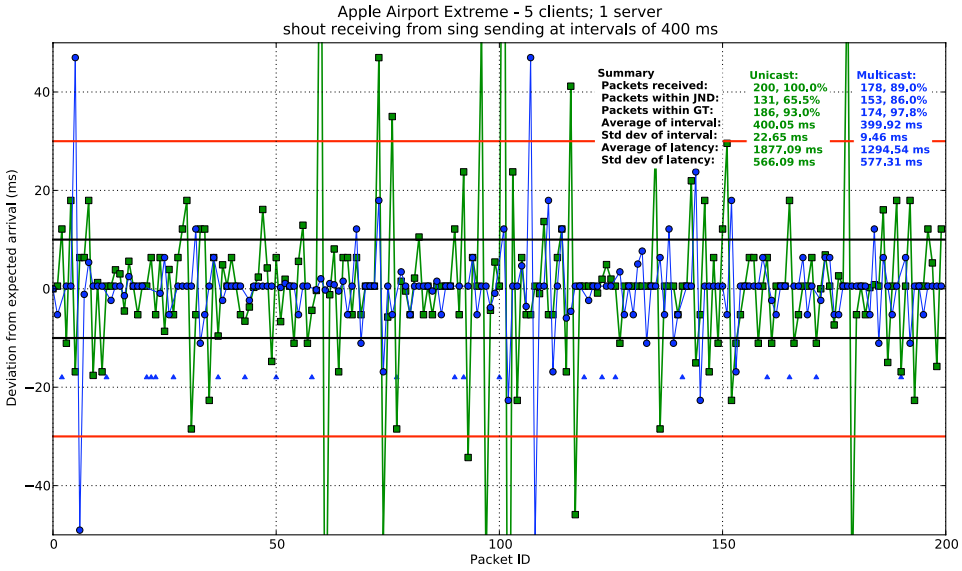

Figure 1

Figure 1 is the graphical output produced by LOrkNeT for a laptop orchestra with five machines running a single server connecting to the Airport Extreme where the server is sending a packet every 400 ms. Data for unicast is in green, and data for multicast is in blue. Sequence numbers for packets are graphed along the x-axis and deviation from the expected arrival is graphed on the y-axis. The black lines represent the JND limits – ideally, all packet deviations will fall within these lines. The red lines represent the grouping threshold. Dropped packets are represented by triangles graphed between the JND and grouping threshold lines.

As Figure 1 shows, there are multiple dropped packets when sending via multicast (represented by blue triangles), while there are no packets dropped when sending via unicast. Although multicasting results in dropped packets, more multicast packets fall within the JND

window when compared to unicasting. The reported latency a packet experiences travelling over a network, over 1000 ms for both unicast and multicast, appears to be abnormally high considering the last pulses played by both clients and servers perceptually occur simultaneously and the fact that a ping request can travel across the continental United States twice in well under 100 ms. This value indicates that there may be issues with our NTP configuration.

3.3 Source of the Multicast Issue

Using LOrkNeT, the performance differences between multicasting and unicasting in the PLOrk environment were verified. This performance difference can be caused by an issue in how the operating system handles multicast traffic, inefficiency in the ChucK programming language and its networking functionality, or in the router handling the network traffic. To test if issues exist in the router handling the traffic sent by ChucK programs, the router was eliminated from the testing environment. Tests were run on a single machine acting as both a client and server with its network interfaces disabled. With networking interfaces disabled, any packets sent use the loopback network interface, resolve to the localhost (127.0.0.1), and are sent back to the sender. Effectively, packets travel as if they were going to another machine on the network but bypass any networking interfaces and devices.

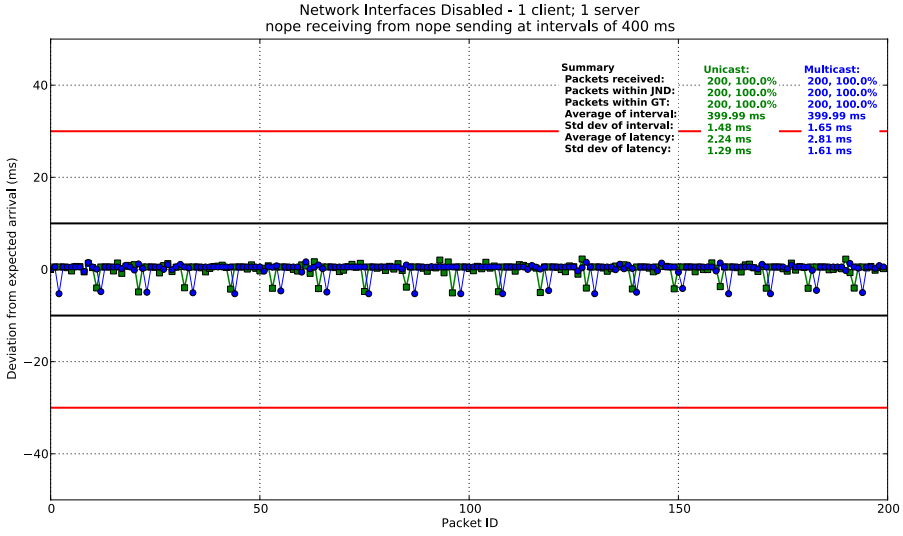

Figure 2

For all tested rates, both unicast and multicast performed optimally in delivering packets and delivering them in a timely behavior for all rates tested, delivering 100% of packets and having all packets delivered fall within the JND and grouping threshold windows. Figure 2 shows the results for the test running at packets being sent every 400 ms. While the measure of network latency measured over the network was abnormally high for the initial Airport Extreme tests, the measured latency for a computer sending packets to itself via the loopback interface was measured at about 2 ms, which seems reasonable and rules out implementation issues with how LOrkNeT measures latency.

Further exploration of the NTP setup for LOrkNeT yielded no positive results in improving measured latency. Originally relying on the NTP daemon running on machines yielded mixed results as the daemon does not force synchronization immediately, but rather slowly skews the system clock's frequency to synchronize time. Using the NTP daemon, tight synchronization comes when client and server are connected for a long time – on the order of days and weeks – which allows significant time to properly synchronize. Unfortunately, the desire to have computers synchronize over a local area network for tighter synchronization and the timing requirements of NTP as is are not compatible since PLOrk does not meet for days and weeks at a time. Fortunately, the NTP daemon has options to force synchronization. In the PLOrk environment, this resulted in machines getting within ± .0001 seconds (0.1 ms) of the server. Running tests immediately after achieving these very low offsets did not improve the reported latency. Subsequent results will eliminate the latency measurement.

Since the performance differences between unicast and multicast were eliminated by using the loopback interface, testing was done on different devices to see if the performance differences held across different devices, or were attributed to the Airport Extreme.

3.4 Further Testing

Using LOrkNeT, the same tests conducted using the Apple Airport Extreme were conducted with five other routers – the D-Link DIR-655 and DIR-825, the Linksys WRT54G, and the Netgear WNR2000 and WNDR3700. For all these routers, the multicast issue present in the Airport Extreme was absent – there was no significant difference among performances with unicast and multicast. Figure 3 shows results for the same test depicted in Figure 1, five computers with a single server sending packets every 400 ms, for the D-Link DIR-655. The differences are drastic – there are no significant performance differences between unicast and multicast, no packets are dropped, and nearly 100% of all packets arrive within the desired JND window for both protocols.

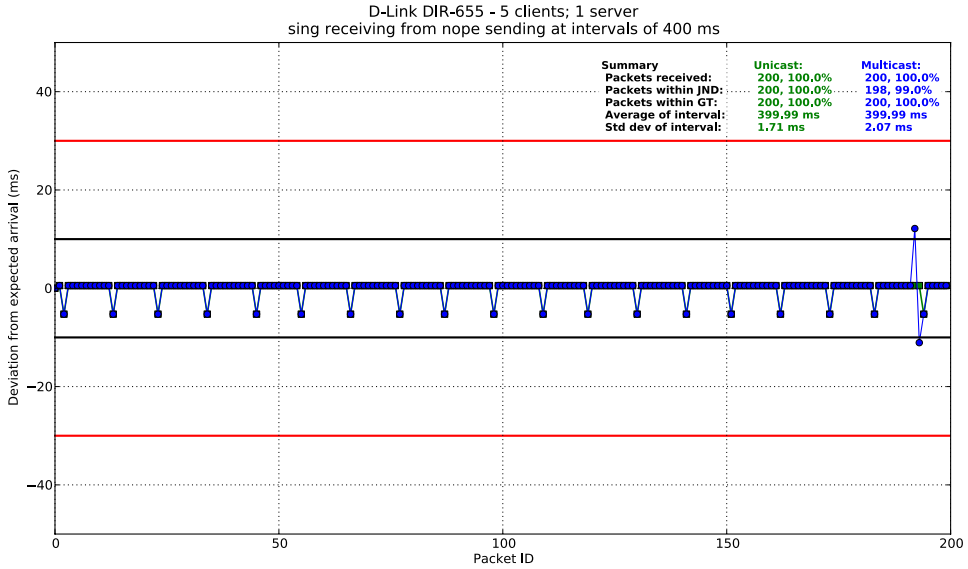

Figure 3

Table 1 shows a summary of the data collected for all routers evaluated for a laptop orchestra with five laptops and one server when packets are being sent every 50 ms. Information in the cells of the table shows the percent of packets delivered, the percent of packets that arrived that are within the JND, and the percent of arriving packets that are within the grouping threshold respectively. Subsequent tables will use this same type of setup.

5 computers, 1 server – packets sent every 50 ms		
Router	**Unicast**	**Multicast**
Apple Airport Extreme	100, 41.8, 91.0	74.0, 51.8, 93.2
D-Link DIR-655	100, 100, 100	100, 98.8, 100
D-Link DIR-825	100, 95.2, 99.0	100, 91.2, 93.8
Linksys WRT54G	99.8, 96.4, 100	99.8, 98.6, 100
Netgear WNR2000	100, 79.0, 98.0	99.6, 88.0, 97.6
Netgear WNDR3700	100, 98.0, 100	100, 99.6, 100

Table 1

Compared to the Airport Extreme – that shows quite a large performance difference in unicast versus multicast – all but one of the other routers performed well using either protocol. This performance difference may be attributed to how multicast is implemented on the routers – if multicast distribution is handled in hardware, it will be more efficient and faster. If multicast is implemented in software on top of the hardware, it may be slower than unicasting and may explain the performance differences present in the Airport Extreme and, to a lesser degree, in the Netgear WNR2000. While the same tests were conducted for all routers used in this study, subsequent analysis will focus on the routers that do not show performance differences between multicast and unicast (i.e. Apple Airport Extreme and Netgear WNR2000 are eliminated).

All routers were tested at rates ranging from sending packets every 50 to 1600 milliseconds, but there does not appear to be any significant performance difference when sending at varying rates within this range. Table 2 shows summary information for multicasting

on a five-laptop orchestra running one server for all tested rates. Except for the poor performance of the DIR-855 at the slower rates (highlighted in grey), varying the rate packets were tested at does not appear to affect performance. The poor performance of the DIR-855 at slower rates presented here did not hold for subsequent tests and should be considered a testing anomaly.

5 computers, 1 server				
	D-Link DIR-655	**D-Link DIR-855**	**Linksys WRT54G**	**Netgear WNDR3700**
50 ms	100, 98.8, 100	100, 91.2, 93.8	99.8, 98.6, 100	100, 99.6, 100
100 ms	100, 99.8, 100	100, 99.4, 100	99.4, 100, 100	90.6, 100, 100
200 ms	100, 94.6, 100	100, 99.2, 100	99.6, 97.2, 100	99.8, 99.6, 100
400 ms	100, 99.0, 100	100, 59.5, 89.0	99.5, 100, 100	99.5, 100, 100
800 ms	100, 100, 100	99.5, 41.2, 77.4	99.0, 100, 100	100, 100, 100
1600 ms	100, 100, 100	100, 41.0, 41.0	100, 100, 100	97.0, 100, 100

Table 2

Results presented so far have utilized a laptop orchestra with five computers. Tests were conducted for laptop orchestras ranging from a single computer to fifteen computers. Table 3 shows a summary of the data collected for multicasting at the 50 ms rate for varying laptop orchestra sizes with a single server. Aside from the anomalous behavior of the D-Link DIR-855 and Linksys WRT54G observed at this rate (highlighted in grey), routers did not show any significant decrease in performance as the number of nodes connected to the router increased.

Packets sent every 50 ms – 1 server				
Size	**D-Link DIR-655**	**D-Link DIR-855**	**Linksys WRT54G**	**Netgear WNDR3700**
1	100, 100, 100	100, 100, 100	100, 100, 100	100, 94.8, 100
2	100, 91.6, 95.0	99.8, 100, 100	99.8, 97.8, 100	94.8, 97.7, 98.3
3	100, 99.6, 100	100, 93.4, 95.6	98.2, 97.6, 100	100, 93.6, 94.0
5	100, 98.8, 100	100, 91.2, 93.8	99.8, 98.6, 100	100, 99.6, 100
10	100, 99.4, 100	100, 0.0, 0.0	99.2, 92.9, 100	100, 96.2, 100
15	100, 100, 100	99.8, 0.2, 0.2	94.0, 60.0, 64.2	100, 96.2, 100

Table 3

Tests conducted scaled not only the size of the laptop orchestra, but also the number of servers sending packets. Table 4 shows summary information for multicasting at the 100 ms rate for an orchestra of five laptops where the servers range from a single server (5, 1) to five servers (5, 5). Except for the two highlighted outliers, performance for configurations ranging through orchestras with 15 machines did not change as the number of servers increased.

Packets sent every 100 ms – 5 computers				
	5, 1	**5, 2**	**5, 3**	**5, 5**
D-Link DIR-655	100, 99.8, 100	100, 100, 100	100, 100, 100	100, 100, 100
D-Link DIR-855	100, 99.4, 100	100, 98.2, 99.6	100, 58.6, 94.4	100, 98.6, 100
Linksys WRT54G	99.4, 100, 100	99.2, 99.4, 99.8	100, 98.8, 100	84.2, 98.3, 99.8
Netgear WNDR3700	90.6, 100, 100	99.8, 99.2, 100	100, 99.2, 99.2	99.6, 100, 100

Table 4

All results presented so far have highlighted data collected on the same machine, but it is important that a router provide the same quality of server to all nodes connected to it. For the four routers being discussed, there was no significant difference among the performances reported across different machines. Table 5 shows summary of data collected for five different machines for a test in which there are five machines and a single server (nope) sending packets at a selection of rates via multicast when connected to the Netgear WNDR3700.

5 computers, 1 server				
Computer Name	**50 ms**	**100 ms**	**400 ms**	**800 ms**
nope	100, 100, 100	100, 100, 100	100, 100, 100	100, 100, 100
sing	100, 99.6, 100	90.6, 100, 100	99.5, 100, 100	100, 100, 100
shout	99.8, 100, 100	99.4, 99.8, 100	100, 100, 100	100, 100, 100
hollar	99.8, 100, 100	100, 100, 100	100, 100, 100	96.5, 100, 100
sniffle	99.4, 100, 100	99.2, 98.4, 99.2	93.5, 100, 100	99.5, 100, 100

Table 5

3.5 Aural Evaluation

As discussed earlier, the server and client programs that form LOrkNeT strike pulses when they send and receive packets. Thus, when running a test, it is possible to hear packets being sent and received, and perform an aural evaluation of the quality of service of the network. While this may be difficult when either packets are being sent at faster rates or there are many computers in the test, it is possible to mute all but a subset of the computers and then listen to pulses being produced by, for example, the server and a single client. While this method of testing is informal, it does allow for quick evaluation, and in the end, it is what we hear that matters most in this type of environment.

It is possible to make aural evaluation more than just an informal testing tool by capturing the audio output from devices being tested. Using a FA-101 Audio Interface by Edirol, up to eight machines can have their audio output redirected to this audio collector and then mixed on a single machine for analysis. Using an audio mixing program such as Digital Performer, it is possible to measure the time between positive impulses (produced when the server sends a packet) and negative impulses (produced when a client receives a packet). This allows us to measure the time a packet takes to travel over the network precisely since it does not rely on the NTP daemon running on the computers to synchronize the system clocks. It is also possible to compare the negative impulses produced across multiple clients, measuring if clients are receiving the same packet at different times. Both these measurements are useful, especially considering the difficulty encountered with configuring NTP properly for this study.

Figure 4 shows a screen capture of Digital Performer, after capturing the output of a five machine orchestra running tests on the D-Link DIR-655, receiving packets from a server multicasting every 50 ms. The impulse created by the server (in the topmost red track) is received by clients which then play a pulse. All the impulses played by the clients for this packet

fall within less than 200 samples or about 4 ms after the server sent the packet. All clients receive the packet and create an impulse within 100 samples or about 2 ms of each other. These values remain consistent throughout the entire recorded segment and fall safely within the grouping threshold of 30 ms. Running the same test with unicast yielded similar results.

Figure 4

4. LOrkNeT in PLOrk

After issues in the Apple Airport Extreme were found to be the issue plaguing many PLOrk compositions, further testing was done during a PLOrk rehearsal with thirty members in attendance connecting to one of the other routers tested – the D-Link DIR-655.

4.1 Large Scale LOrkNeT Testing

The previous tests stressed routers through laptop orchestras up to 15 machines running up to 15 servers. Recall that for these tests, performance was not impacted negatively as the number of machines and servers scaled. This rehearsal of PLOrk was first used to run the maximum stress test – 30 computers, all running servers. Due to time constraints, packets were sent at a single rate – every 50 ms – using both multicast and unicast.

One new element introduced when running tests in the PLOrk environment is the introduction of a variety of models of machines running different versions of the Mac OS X operating system. Results presented earlier were tested on the same model of Macbook running 10.5.8, which were configured identically. In addition to analyzing the data for the 30 computer, 30 server setup, the possible impact of computer model and operating system version was analyzed. Table 6 shows a summary of data collected for a range of clients listening to the same server.

D-Link DIR-655 – 30 computers, 30 servers – packets sent every 50 ms		
Model / OS Version	**Unicast**	**Multicast**
Macbook '06, 10.5.8	100, 100, 100	100, 100, 100
Macbook '06, 10.5.8	18.5, 21.6, 70.3	78.5, 8.3, 33.1
Macbook '06, 10.6.3	20.0, 17.5, 65.0	79.5, 16.4, 74.8
Macbook Pro '06, 10.5.8	18.5, 27.0, 67.6	79.5, 32.1, 88.1
Macbook Pro '06, 10.6.3	20.0, 27.5, 77.5	77.5, 31.6, 81.9
Macbook Pro '10, 10.6.3	16.5, 21.2, 60.6	76.5, 21.6, 68.6

Table 6

The data presented in Table 6 raises many different points of discussion. First, it appears performance does not scale well going from 15 machines running 15 servers to 30 machines running 30 servers. Also, there does not appear to be any performance impact among different models and operating system versions as all machines reported similar results. Furthermore, there is a appreciable performance gap between unicast and multicast, with multicast resulting in a significantly higher percentage of packets being delivered. Finally, the anomalous behavior of the first client (highlighted in grey) may be just that – a fluke. But it may also be offering some insight into how a router handles maintaining quality of service under extremely heavy loads: prioritizing the quality of service to one machine over the others.

In addition to the 30 computers/30 servers test, another test with 30 computers and a single server was conducted in hopes of separating performance impacts caused by a large number of nodes connected to the access point and impacts caused by a large number of nodes connected and all sending data over the network. Table 7 shows summary data for the same machines presented in Table 6 respectively.

D-Link DIR-655 – 30 computers, 1 server – packets sent every 50 ms		
	Unicast	**Multicast**
Macbook '06, 10.5.8	96.5, 49.2, 97.9	92.0, 48.9, 87.5
Macbook '06, 10.5.8	99.5, 58.8, 95.0	98.5, 58.4, 86.8
Macbook '06, 10.6.3	100, 61.0, 97.5	100, 69.5, 97.0
Macbook Pro '06, 10.5.8	100, 61.5, 97.5	100, 66.5, 97.5
Macbook Pro '06, 10.6.3	100, 79.0, 97.5	100, 57.0, 96.0
Macbook Pro '10, 10.6.3	98.0, 68.9, 98.0	97.5, 48.2, 87.7

Table 7

When analyzed with Table 6, Table 7 helps separate out what factors may be responsible for the poor performance observed with larger orchestras. Although performance with 30 computers and a single server is far from perfect, across all machines with both unicast and multicast, a large percentage of packets were delivered successfully. The success of packet

delivery for these tests correlates strongly with the number of servers sending traffic rather than the size of the orchestra. This is an intuitive conclusion – fewer servers sending packets will result in less contention among devices competing to use the router, increasing the chance packets will be delivered.

Regarding the timely delivery of packets, there is a noticeable improvement in the number of packets arriving within JND and the grouping threshold when the number of servers decreases to a single server, but the performance still falls significantly short of ideal. Timely delivery of packets appears to be correlated with both the number of nodes connected to and number of nodes sending data over the access point. Again this an intuitive conclusion – as the number of nodes scales, packets may still be getting delivered successfully, but queuing to a larger number of nodes at the router may result in untimely delivery. If the number of packets being sent increases when there is an increase in the number of servers, there could be two potential issues: a router may be unable to queue all the packets it is receiving and begin dropping packets completely, or a router may not be able to acknowledge all servers wanting to send packets and the packets of those unacknowledged servers will not even make it to the router.

The anomalous behavior exhibited by the first client in the first large scale test is not present in the second test. Again, the results for the first test may have been purely anomalous or quality of service may have changed under the less heavy network loads present in the second test. Further testing for larger laptop orchestras should be explored in the future to further explore the issues that emerged and the conclusions reached when tests scaled from 15 to 30 machines.

4.2 CliX Variations

The PLOrk classic, CliX, is a composition that relies on networking to function. Performers strike keys on their keyboards to produce clicking sounds. The ASCII value of the key pressed determines the frequency of the click. Performers connect to a network and a server sends out pulses that include information on the gain the following click will play at. Even though performers will be striking different keys and playing clicks of different frequencies – since all performers will be getting the same gain for their clicks from the server, a noticeable audible pattern emerges amidst the chaos that results from performers wailing away at their keyboards.

The natural solution to implement the CliX server would be to use multicast – since all clients play their clicks with the same gain, multicast would be the easiest way to get the same information to all machines on the network. With the traditional PLOrk setup, multicast was found to work improperly with earlier compositions so the CliX server was implemented using unicast – clients broadcast their presence periodically and the server opens up a direct connection to each particular client. This unicast implementation produced desirable results and CliX was performed many times successfully using this setup.

After LOrkNeT tests found performance differences among unicast and multicast to be attributed to the Airport Extreme, CliX was implemented using multicast. Both unicast and multicast versions of CliX were played during a PLOrk rehearsal and aurally evaluated by performers. While with the Airport Extreme used in PLOrk there was a noticeable difference between the unicast and multicast versions of CliX, with the D-Link DIR-655, performers were unable to notice a difference between the unicast and multicast version, noting that both versions performed identically, as far as they could tell.

A server-less implementation of CliX was also developed using NTP in ChucK – this variation was dubbed nCliX. Clients first synced their system clocks and then would sync to a 4 second interval. Once synced to a multiple of 4 seconds, clients would cycle through an array of 40 gains playing 10 clicks per second. Regardless of when someone begins playing nCliX, they will pause to that 4-second multiple and then join the other plays, cycling through the array of gains together. When nClix was performed, one performer described the performance of nCliX as sounding "watery," which can be attributed to different clients hitting different gains at the same time, which resulted from poor synchronization of system clocks. Future configurations of NTP for PLOrk may result in tighter synchronization that may make nClix work properly.

4.3 Compositions in Other Audio Synthesis Languages

As information from tests running on different routers revealed performance differences, current composers for PLOrk who had pieces that relied on networking were asked to try varying the router used for their pieces to see if the performance differences could emerge in compositions. Jascha Narveson has been composing a piece, Beepsh, written in the audio synthesis language, Super Collider. In Beepsh, performers compose their own beep melodies and drum machine rhythms, and a server cycles through performers. When a performer is selected by the server, their current melodies and rhythms play.

Narveson tested Beepsh on several machines while developing and debugging the piece. He found that the D-Link DIR-655 performed much better when compared to the Apple Airport Extreme. Narveson added that the D-Link DIR-825 was audibly more stable than the DIR-655 by a noticeable amount, citing that the DIR-655 dropped packets intermittently. This supports the finding that the ChucK programming language was not the source of the issues present in PLOrk, and also reveals that there may be performance differences among the better performing routers tested in this project.

5. Conclusion and Future Work

LOrkNeT was designed to identify problem areas in a network that can affect laptop orchestra performance. In a laptop orchestra environment, there are stringent requirements that a network must meet for a composition to produce a desirable sound. With LOrkNeT, issues in the current PLOrk environment were diagnosed and attributed to an issue in the Apple Airport Extreme router. Other routers tested were found to not have these issues and performed better – results confirmed by both LOrkNeT and the use of these other routers in the PLOrk environment. The effects of increasing machines connected to the network and the amount of traffic being sent over the network were tested and analyzed – increasing the number of machines connected to the router did not impact performance and increasing the number of servers producing traffic on the network did not impact performance significantly either when these numbers ranged up to fifteen. Performance for orchestras containing 30 members begins to show noticeable performance degradation.

There is still much ground to be covered in gaining a truly complete idea of how networks function for laptop orchestras. Packets sent with LOrkNeT always send a small, fixed amount of data. Varying the amount of data to higher levels would be an interesting factor to test and obtain performance data. Orchestras in size ranging from 15 to 30 computers should also be further explored as even the better performing routers begin to show performance degradation somewhere in this range.

While there were problems getting NTP to function properly for use in this project, the future use of NTP should be explored. The issues with NTP can be occurring at different points. Achieving synchronization via NTP across multiple ChucK threads on the same machine seems difficult as there exists some time delay when adding threads to the ChucK virtual machine that cannot be accounted for. Though, modifying LOrkNeT servers and clients run in a single ChucK

thread did not eliminate the odd values being reported by NTP-dependent values. Routers that block or delay packets on port 128, which is used by NTP, may also be the cause of the issue. Depending on the router being used, synchronization could take anywhere from only a few seconds to multiple minutes. A fully functioning, tightly synchronizing NTP distribution across a laptop orchestra environment would be a useful tool to have, as guarantees of timeliness could be incorporated into the networking functions present in ChucK.

LOrkNeT and the data collected for this project is hosted at lorknet.cs.princeton.edu; it is the author's sincerest hope that LOrkNeT will be of use to other laptop orchestras and that a collective and community-maintained database of information regarding networks for laptop orchestras can be created with LOrkNeT.

6. References

Cáceres, J.-P., Hamilton, R., Iyer, D., Chafe, C., Wang, G., “To the Edge with China: Explorations in Network Performance.” ARTECH 2008: Proceedings of the 4th International Conference on Digital Arts, pp. 61-66, Porto, Portugal, 2008.

Chafe, C., M. Gurevich, et al. (2004). "Effect of Time Delay on Ensemble Accuracy." Proceedings of the International Symposium on Musical Acoustics.

Chafe, C. and R. Leistikow (2001). "Levels of Temporal Resolution in Sonification of Network Performance." Proceedings of the 2001 International Conference on Auditory Display.

Chafe, C., S. Wilson, et al. (2000). "A Simplified Approach to High Quality Music and Sound Over IP." Proceedings of the COST G-6 Conference on Digital Audio Effects.

Clayton, M., R. Sager, et al. (2004). "In time with music: The concept of entrainment and its significance for ethnomusicology." ESEM CounterPoint 1.

Cook, P., P. Davidson, et al. (2005). "Interactive Network Performance: a dream worth dreaming?" Organised Sound 10(3): 10.

Dannenberg, R. and P. v. d. Lageweg (2001). "A System Supporting Flexible Distributed Real-Time Music Processing." Proceedings of the 2001 International Computer Music Conference.

Golmie, N. (2006). Coexistence in Wireless Networks.

Goto, M., R. Neyama, et al. (1997). "RMCP: Remote Music Control Protocol." International Computer Music Conference.

Smallwood, S., et al. (2008). "Composing for Laptop Orchestra." Computer Music Journal 32: 16.

Trueman, D. (2007). "Why a laptop orchestra?" Organised Sound 12(2): 171-179.

Wang, G. (2006). "ChucK => OSC." Retrieved 03-10-2009, from http://opensoundcontrol.org/implementation/chuck-osc.

Wang, G. (2008). The ChucK Audio Programming Language: A Strongly-timed and On-the-fly Environ/mentality. Computer Science, Princeton University.

Windl, U. and D. Dalton (2006). "The NTP FAQ and HOWTO." from http://www.ntp.org/ntpfaq/.