Real-Time High Fidelity Audio for Networked Music Performance

A Thesis Presented to

The Faculty of the Computer Science Program

California State University Channel Islands

In (Partial) Fulfillment

of the Requirements for the Degree

Masters of Science in Computer Science

by

Ray Alfano

May 2014

APPROVED FOR THE COMPUTER SCIENCE PROGRAM

[illegible] 8/6/14

Advisor: Dr Michael Berman Date

[illegible] 5/12/14

Dr Peter Smith Date

[illegible] 12 May 14

Dr Brian Rasnow Date

APPROVED FOR THE UNIVERSITY

[illegible]

Dr Gary A. Berg Date

Real-Time High Fidelity Audio for Networked Music Performance

Title of Item

Networking, VoIP, computer music, audio encoding, Opus

3 to 5 keywords or phrases to describe the item

Ray Alfano

Author(s) Name (Print)

[illegible] 5/14/2014

Author(s) Signature Date

Abstract

Audio transmission over the Internet is a vital component of modern communications, and digital formats have mostly supplanted physical media for the creation and delivery of cultural content by musicians. From the perspective of artists, access to computers and the Internet democratize the creation and distribution of media, allowing content creators much greater freedom. Just as the MP3 revolution and the growth of BitTorrent revolutionized the distribution of recorded music, increasingly fast Internet connection speeds lead us toward a future wherein musicians in different geographic areas will be able to record or perform music together in real time, bypassing the need for immediate physical proximity. This concept, known generally as networked music performance, is an increasingly viable means of creative expression for the general public, whereas previous implementations required specialized laboratory environments.

As the speed of data connections increases year after year and newer encoding techniques emerge, we approach a tipping point at which the general Internet-using public may collaborate at audio fidelity levels that effectively equal the quality of the original audio source. In the field of communications, Voice over Internet Protocol (VoIP) platforms continue to gain market share as a means of providing efficient and reliable real-time voice services by utilizing the Internet Protocol (IP) over a packet-switched network, and this has led to the development of increasingly mature encoding formats for delivering high-quality sound. As a result, live audio data encoded in real time for VoIP approaches audio fidelity levels acceptable for music without the onerous bandwidth requirements of transmitting uncompressed signals over a network. Until recently, the vast majority of encoding formats available to the general public failed to meet adequate performance requirements for delivery of high fidelity audio data with very low latency, but the emergence of the open source Opus audio format provides a viable mechanism to transmit live music at an acceptable quality of service.

In this study, I utilize VoIP network protocols and the Opus audio format to allow two (or potentially more) musicians to perform together at the highest possible level of audio fidelity. My experiments employ a variety of configurations and modifications to Opus and network transmission media with the goal of determining how best to deliver the highest quality audio possible given the speed of a network connection over time. The purpose of my study is to prove the viability of Opus and current technologies for networked music performance by establishing realistic constraints for transmitting audio via current Internet connection speeds, creating a program that utilizes network feedback to adjust the fidelity of encoding to ensure on-time delivery and qualitatively analyzing the fidelity of audio transmitted to the destination.

Acknowledgements

The author would like to thank the faculty at California State University Channel Islands for their continued support of high-quality education and research, the open-source community for their help and guidance, and family and friends for their support.

Table of Contents

Chapter 1: Introduction

1.1. The purpose of transmitting high fidelity audio in real-time

The intent of this project is to provide guidelines and a mechanism for transmitting real-time audio at the maximum level of audio fidelity based on network conditions. Considerable advances in recent years have enabled great increases in the quality, reliability, and availability of streaming video and audio from both live and file-based sources. Streaming video now constitutes large proportions of Internet traffic, primarily due to the rise of Youtube and Netflix Instant. Internet radio systems such as IceCast and streaming audio sources such as Spotify also have made great strides toward increasing the average quality of audio over the Internet in recent years. However, these services focus on one-to-many transmission of recorded media instead of participatory real-time performance. The same technologies have great promise for enabling the real-time transmission of live media in ways that open new avenues for creative collaboration. This project implements a method for two audio sources to communicate with each other in real time with the ultimate goal of providing a mechanism through which many musicians could perform together over a network.

Codecs and media streaming services for transmitting real-time audio over the Internet existed prior to the Opus format, but they have many important disadvantages such as arbitrarily-fixed bit rates, limited sample rates, delays in transmission, or a reliance on proprietary audio formats and hardware. In order for an audio transmission mechanism to be useful to a musician, a system needs to provide high availability, minimally compressed audio and a minimum amount of delay between the original audio source and playback at the endpoint. The means through which I address each of these objects are discussed in Chapter 2.

1.2. Definition of terms

QoS (Quality of Service): QoS is the subjective reliability and quality of the audio transmission. In this application, quality of service is objectively measured by the Opus bit rate and the sample rate. The quantification of the QoS in this thesis derives from measures of the average amount of packets received or missed as a function of time with consequent adjustment for lower bit rate audio. For the purposes of this research the acceptable QoS time limits are an end-to-end delay of 25ms for strict real-time responses, 60ms median delay for general real-time responses and 100ms for worst-case end-to-end delays.[1]

Opus: Opus is a lossy audio format created by the Internet Engineering Task Force that is intended for real-time audio purposes. Opus succeeds the Ogg Vorbis format. Opus is standardized by RFC 6716. [1]

RTP (Real-time Transport Protocol): RTP is used for the real-time streaming of audio and video. RTP is standardized by RFC 3550 and operates at the application layer.

RTCP (Real-time Transport Control Protocol): RTCP is the sister protocol to RTP and provides feedback for maintaining QoS over a session. The port number used by RTCP packets is one greater than that used by RTP. RTCP operates at the application layer.

RTSP (Real-time Streaming Protocol): RTSP is a network control protocol used to control streaming media servers. This often builds on RTP and RTCP, although often with the admixture of proprietary formats. The protocol handles the establishment and control of streaming sessions between a host and server. RTSP is standardized by RFC 2326 and operates at the application layer.

VoIP (Voice over Internet Protocol): VoIP technology comprises the implementation of digital telephony over a digital, packet-switched network using the Internet Protocol. This format emphasizes

1 The rationale for these delay values is explained throughout this document, but specifically in sections 3.2.3 and 3.2.5.

voice technology and, while it uses audio data, voice is a less complex signal to encode than music.

SILK: SILK is an audio compression format developed by Skype with a relatively low algorithmic delay of 25ms. This codec is primarily intended for speech, as the dynamic range of the audio does not extend to the limits of human hearing.

CELT (Constrained Energy Lapped Transform): CELT is a free software codec with low algorithmic delay developed by the Xiph.Org Foundation for high-quality audio. CELT has been subsumed by the Opus format.

ALSA (Advanced Linux Sound Architecture): ALSA is a part of the Linux kernel that provides an audio API to sound card drivers. This architecture provides reliably low-latency audio interaction between hardware and software.

SIP (Session Initiation Protocol): SIP is standardized in RFC 3261 and runs on the TCP, SCTP and UDP protocols as a text-based protocol for initiating and closing stream connections.

H.323: Recommended by the International Telecommunication Union, H.323 provides robust capabilities for video, audio, and text transmission over a packet-switched network, and as such it is widely employed in VoIP implementations. H.323 often implements a larger superset of media-providing technologies, frequently including RTP.

PCM (Pulse Code Modulation): PCM is the standard format for digitally representing a sampled analog signal. This format is a raw representation of an analog audio signal in an uncompressed digital format.

Jitter: Jitter is the variation of a signal from its expected periodicity. In audio over a network, this refers to the variation in latency of packet arrival, which must be compensated for to prevent audio dropouts or consistency of audio output. [2]

GStreamer: GStreamer is a pipeline-centric framework for transmitting multimedia streams, and it serves as the foundation for transferring data in the experiments for this project. The framework is written in C, uses a Gobject-like type system and provides facilities for a variety of plug-ins that dynamically load libraries for various codecs, containers, signal processing effects, and I/O drivers. GStreamer is cross-platform and is licensed for use by the GNU LGPL.

Chapter 2: Field Overview

2.1 Center for Computer Research in Music and Acoustics (CCRMA) Experiments

The primary work that inspired this thesis is a set of early experiments by Stanford's CCRMA foundation, which began conducting networked audio performances over short distances in 2000. In their publication "A Simplified Approach to High Quality Music and Sound over IP", Chafe et. al. implemented a TCP-based system called SoundWIRE that allowed them to transmit audio data and process signals over a network [3]. CCRMA's system provides robust facilities for adapting to changing network conditions and permits adjustment of packet size, fragment size for the audio I/O operations, buffer sizes and TCP-specific parameters such as fast acknowledgement. The software for CCRMA's experiments are written in C++ and reliably send low-latency audio data over the Stanford University core network at distances of 1km using unidirectional transmission of UDP and TCP packets. The CCRMA group performed multiple experiments, showing conclusively that institutional networks circa the year 2000 could reliably transmit audio at rates acceptable for networked musical performance.

The inspiration for my experiments stems from the fact that CCRMA was able to transmit very high quality audio (24-bit/96kHz) over 10 channels with maximum data rates of 23Mbps in their pioneering work. Stanford's experiments also yielded promising results with less complex data over 10Mbps networks, and that provided the impetus to apply more advanced codec technology to audio transmitted over current networks in the hopes of improving networked audio performances. Aside from the direct implications for my research, Stanford's experiments on audio that do not meet the typical 50-60ms threshold for live performance offer intriguing possibilities for networked audio performances that make use of delays approaching one second for novel artistic purposes such as ambient or experimental music.

2.2.1 JackTrip

JackTrip is a networked audio performance program released by Stanford's CCRMA in 2008 for Linux and Mac OS X as a freeware implementation their audio transmission mechanisms. It provides the simple configuration parameters described in the original CCRMA experiments to reduce latency, most importantly audio bit resolution, frame size, sample rate and buffering levels.[2] JackTrip utilizes the JACK audio routing facilities that exist natively in Linux and non-natively in Mac OSX. This application's capabilities are relatively simple compared to Opus and Gstreamer's capabilities, but it provides a baseline for comparison with the faster, lossy audio format. As with Gstreamer and Opus, this application can use UDP or TCP for audio signal transfers.

2.2 CobraNet

CobraNet is an early and successful application of Audio over Ethernet marketed by Cirrus Logic beginning in 1996. CobraNet is commonly deployed in large performance venues or other environments requiring high fidelity audio and real-time performance over many channels. This system provides 96kHz sampling rates at a bit depth of 24 bits and reproduces up to 64 channels over long distances at a minimal audio latency of roughly 1.3ms.[3] This product improves upon previous analog implementations for audio transfers in large venues, which suffered significantly from frequency clipping, voltage issues and electromagnetic interference. In its present form, this system requires a switched Ethernet network and provides digital audio at quality levels appropriate for faithful reproduction of the original source. CobraNet provides musicians with the ability to play together over a network, and as such it is widely deployed in many performance venues for the house sound system

2 JackTrip documentation: jacktrip-1.05/documentation/documentation.cpp

3 http://www.dspecialists.com/sites/default/files/publication/110126a_networking_paperembeddedworld2011_paperen_final_st_jc.pdf

connections.

CobraNet avoids several problems with typical Internet streaming arrangements due to its reliance on local area networks, but it is conceptually similar to what my project hopes to implement in that it facilitates a real-time high fidelity audio connection over a packet-switched network. This system does not utilize the Internet but rather a signal path through hubs, bridges, and switches. Consequently, CobraNet bypasses a large portion of the latency issues inherent to unpredictable connections over the Internet. There are obvious disadvantages to this sort of setup and similar proprietary systems, most importantly limitations to deployment and ties to vendor-provided hardware and unique encoding formats. Another important limitation is that the system is not intended for wireless networks aside from laser-based over-the-air transmission. CobraNet is further limited by Ethernet multicasting and a reliance on a large number of proprietary headers.

2.3 Internet2 Foundation Experiments

In late September 2012, the Internet2 Foundation demonstrated a technology that is conceptually similar to the one described in this paper for networked music performance, allowing musicians separated by long distances to collaborate in real time. Their system, called LOLA (Low Latency Audio/Video), has an average delay of 35ms and transmits uncompressed audio data.[4] According to Northern Illinois University music professors and multimedia specialists, this implementation meets all latency requirements for a real-time music performance system. In a 2013 demonstration of this system, the Internet2 Foundation used a 100Gbps network to demonstrate LOLA functioning over a distance of 1300 miles with 35 milliseconds of average delay, which is well within the QoS guidelines for a useful musical performance tool.

LOLA has several important differences from the methods described in this thesis, the most

4 http://www.internet2.edu/news/detail/2511/

important of which are that the system used by Internet2 operates on a proprietary, ultra high-speed network and transmits uncompressed audio data. The purpose of the Internet2 Foundation's system is to convey an exact representation of the source audio using redundantly high sample rates while taking advantage of uniquely favorable network throughput due to specialized hardware. The Internet2 test network provides a 100 gigabit Ethernet Layer 2 connection and several advanced bandwidth-enhancing features that are not available on the current Internet[5], such as an ability to optimize network connections for maximum throughput of high bandwidth data over distances.[6] These methods are unrealistic for typical packet-switched networks connected to the standard Internet of this day and age, although they provide a proof-of-concept for a future implementation of audio transfers in an environment where bandwidth is effectively unlimited.

While the Internet2 Foundation uses lossless audio in their demonstrations, the experiments for this thesis use a lossy format that accepts a reasonable compromise of audio fidelity while still emphasizing a very high level of representational accuracy. This constitutes a fundamental difference between the work described in this thesis and the LOLA experiments. Due to the relatively slow and variable speeds of the standard Internet compared to the LOLA experimental test network, a viable networked music performance system must take into consideration the limited bandwidth of current WANs. The experiments for this thesis emphasize quality in an innovative high fidelity format but utilize standard networks so as to provide as many musicians as possible with access to collaborative audio creation. The Internet2 Foundation's efforts are exclusive to their own dedicated fiber-optic networks, and therefore they will only benefit a select few researchers and demonstrators at a disproportional monetary cost compared to standard networks. So, while the LOLA experiments are useful as an example of a networked music performance system, their specific implementation is

5 http://www.internet2.edu/media/medialibrary/2013/09/07/Internet2-Innovation-Platform-FAQ.pdf
6 https://www.internet2.edu/vision-initiatives/initiatives/innovation-platform/

limited to the domain of academic research.

2.4 Low-Latency Audio over IP on Embedded Systems

A very recent experiment with networked music performance, published in October 2013 as a master's thesis roughly two months after formal research for this master's thesis began, utilizes the Opus format and embedded systems for transmitting live audio data. [4] This research, conducted at the Technical University of Hamburg-Harburg by Florian Meier, utilizes a Raspberry Pi embedded device and Linux to provide audio input and handle capabilities for sending Opus audio packets over a network. The device created for Meier's experiments minimizes latency by utilizing a lightweight low-latency kernel on a dedicated embedded Linux device. Meier's system achieves a steady latency of 40-50ms with a packet loss rate of approximately 2% and utilizes the built-in Opus facilities for adjusting bit rates automatically based on network conditions. [4]

Meier's experiments provide several insights into innovative methods for constructing a network-based audio performance device in that he proves that Opus is a capable medium for networked audio performance using a restricted amount of computational power. However, there are notable limitations to his work that stem from the nature of the Raspberry Pi embedded system. In particular, the packet loss of 2% in his experiments disrupts signal continuity because Opus's error-concealment algorithms, a key advantage of the format, are disabled due to their computational complexity on a low-power CPU such as his ARMv6k[7]. Also, the notion of utilizing a specialized embedded device does not meet my research goal for widespread availability. Additionally, the real-time capabilities of Meier's experiments are limited by the deliberately-simplified kernel employed by his Raspberry Pi device, which can be improved in the Ubuntu Studio real time desktop kernel.

7 While a capable processor in many respects, particularly as relates to audio digital signal processing due to upgraded SIMD instruction sets, the ARM1176JZF-S operates at 700MHz and is not able to handle the computational complexity of packet loss concealment. PLC would also require far more cache memory and system memory than Meier's system utilizes in order to provide a sufficient historical representation of signal state for acceptable gap-filling.

Chapter 3: Technical Details of the Work

3.1. High-level description of algorithms used for this program

At the most basic level, my experiments and demonstration programs utilize RTP packets to transmit an Opus payload to a destination endpoint and play back audio in real time. In my demonstrations, two audio sources communicate over a network to send audio to the other host with the goal of providing the highest-quality audio given the network conditions while considering the limits at which a human perceives a gap in signal. Each source receives feedback from the destination host in the form of an RTCP packet at variable intervals, providing data to notify the sender of the amount of packets lost and the round trip time of the network. In situations where the arrival time of packets on the destination exceeds the 60ms limit from sound creation to playback, the sender adjusts the bit rate of the audio sent to the receiver. Depending on network latency conditions, the system either increases the quality of audio upward to a theoretical maximum or downward until the size of the file transmits and is played back within an acceptable time window.

Essentially, the entire end-to-end process involves a regular series of steps, each of which can contribute delay. Depending on the source of the audio, the first step is optional and consists of converting live analog source audio to a digital format called a Pulse Code Modulated (PCM) signal, which is a raw format that represents a digital translation of analog waveforms. The second step is to encode the PCM data into a format that transmits (or plays back as audio) more efficiently than the raw digital data. The third step involves breaking the audio into frames that transmit as the payload of RTP packets sent over the network along with RTCP packets (at intervals) containing information about the individual audio file. The fourth step is transmitting the frames over the network to the recipient. The fifth step is deconstructing the RTP packet and processing the RTCP statistics (if applicable) on the receiver side while adjusting the encoding quality if necessary. The final step, conducted in parallel with the fifth step, is to play back the received frame of audio. This process continues until an RTCP "BYE" packet is received, ending the session.

3.2 Constraints of technology and audio

3.2.1 Input source latency

Input source latency depends on several factors which are discussed throughout this document. There are several potential sources of delay: the distance from the source of audio to the recording device (either through the air to a microphone or via a cable to an audio input), any signal processing delay from exterior sources (if using a direct-input connection and off-board processing) and device driver delay for audio I/O. Implementations of real-time systems must minimize delay contributed from each of the aforementioned sources, and the infrastructural problems involved in doing constrain high-quality networked music production in all but the most specialized of environments up to this day. The tests for my experiments include references to the delay incurred by the audio source and all of the signal stages.

3.2.2 Theoretical aspects of signal technology and common encoding standards

Signals travel over fiber optic cables at 5μsecs per kilometer.[8] Theoretically, this imposes a constraint of 5ms of delay for every 1000km for direct fiber optic signals. [5, 3.1] Additionally, there are constraints imposed by the nature of the file sizes for typical levels of audio quality. Also, audio devices connected to a computer tend to be USB 2.0 (USB 3.0 is rarely employed for audio devices as of 2014), IEEE 1394b (FireWire), or PCIe. These formats offer several advantages and disadvantages that contribute to latency which cannot be eliminated. Anecdotally, FireWire offers

8 Network Latency – How Low Can You Go? Light Wave Online. Volume 29, Issue 6. http://www.lightwaveonline.com/articles/print/volume-29/issue-6/feature/network-latency-how-low-can-you-go.html (Accessed March 26, 2014).

better performance than USB 2.0, as it is able to write directly to memory. PCIe is often fast, but requires significant investment in proprietary solutions, as no interoperability standards exist for audio over PCI.

As a general rule, one second of uncompressed audio at CD quality (44.1kHz sample rate, 16 bits per sample, stereo) requires 172KB of file space. For higher sample rates that same amount of audio requires 187.5KB at 48kHz and 375KB at 96kHz.[9] Considering an average frame size in my experiments' Opus-RTP packets of 5ms, these equate to roughly 172/200 = 0.86KB (CD audio), 187.5/200 = 0.9375KB (48kHz), and 375/200 = 1.875KB (96kHz). Opus is able to reduce audio bit rates from 768kbit per second to 72 kbit per second, which significantly improves the network throughput requirements. [4, section 7.1.3[10]]

3.2.3 Theoretical constraints of sounds transmitted through air

Musicians have always contended with latency incurred by sound traveling through air, and there are many sophisticated mechanisms for compensating for delays imposed by distance. For this reason, this project demonstrates a range of end-to-end latencies while trying to deliver audio packets within 25ms of round-trip time. The 25ms limit is based on the Ensemble Performance Threshold (EPT) value, which is a standard maximum delay for musicians to closely collaborate. [5, 7, 8, 9] However, this project assumes that additional delay is tolerable in certain circumstances, as latency in excess of 25ms is not necessarily debilitating to a performance. Musicians must manage delays from their instruments propagating sound waves through the air, which becomes significant

9 Thyagharajan, K.K., Performance Analysis of Media tranmission through Low Bandwidth Networks. SSN College of Engineering. http://www.academia.edu/1226940/Performance_Analysis_of_Media_Transmission_through_Low_Bandwidth_Networks (accessed April 12th, 2014)

10 Meier, Florian. Low-Latency Audio over IP on Embedded Systems. Technical University of Hamburg-Harburg. http://www.ti5.tuhh.de/staff/meier/master/meier_audio_over_ip_embedded.pdf (Accessed April 10th, 2014)

within larger performance venues. For example, on a large stage, two musicians performing 34 meters (~111 feet) apart experience 100ms of delay between producing a sound and hearing the other performer (assuming that the musician is also at the source of their sound).[11] Historically, latency from sound propagation has been tolerated by performers when possible or corrected by compositional techniques.[12] Because of the historical toleration of long delays and the potential for unique compositions resulting from out-of-sync reactions from performers, this project tries to minimize audio latency but does not constrain latency to an arbitrary guideline. Finally, delays incurred by distance between performers and microphones or signal input intermediaries in networked music performances must be included in the overall end-to-end latency calculations.

3.2.4 Problems inherent to encoding due to drivers

Drivers, defined as the programs that control devices attached to a computer, incur latency to audio input and output due to overhead from communication between hardware, the operating system, applications and users.[13] This project requires a UNIX audio system to minimize the amount of latency contributed by drivers. The experiments and source code for the experiments described for this thesis require a Linux-based system, preferably Ubuntu Studio, operating with a real-time kernel. The LAN and Internet-based experiments for this thesis utilize Ubuntu Studio running a hard real-time kernel[14] and simulate network latency over a local area network using the program netem. Experiments conducted via locally-hosted virtual machines described in this thesis are

11 These numbers are derived from calculations based on the speed of sound at sea level. Delay of sound through the air equates to an additional delay of one millisecond for each 334.29mm of distance between the live audio source and the recording mechanism if using a microphone. http://hyperphysics.phy-astr.gsu.edu/hbase/sound/souspe3.html#c1

12 An example is in medieval music, where choirs sometimes stood on opposite sides of cathedral congregations and had to synchronize with each other. This became a subject in early music theory, and techniques such as syncopation are used in the absence of more sophisticated synchronization mechanisms.

13 http://www.presonus.com/news/articles/The-Truth-About-Digital-Audio-Latency

14 A hard real-time operating system is differentiated from a soft real-time operating system by the following distinction: A soft real-time OS is capable or generally capable of meeting a time-defined deadline. A hard real-time operating system will deterministically meet a deadline that has a defined time.

conducted through a host Apple machine using Mac OS X v10.9 in order to utilize Xcode's Network Link Conditioner. Attempts to provide a Windows test system failed, as initial tests using a Windows 7 Professional platform suffered from encoding delays exceeding 100ms, which places the system well beyond the minimum requirements of a real-time audio system.

Given the impact of input and output latency, the test systems for network music performance in my experiments utilize several optimizations that improve audio latency by manipulating system settings and process scheduling. The processor-specific optimizations in this project are disabling Advanced Power Management (APM) in BIOS to prevent CPU speed-throttling and stabilize system timers and disabling hyper-threading in the system BIOS to improve real-time kernel stability. Also, there are several operating system-level optimizations that reduce latency overall by up to 7ms, as observed in my experiments. The primary Linux OS-level optimizations are enabling the threadirqs kernel parameters, using the Linux real-time kernel (linux-rt), using the noatime filesystem option, adding the current user to the audio group and setting the cpufreq setting to "performance." Lower-level optimizations include disabling/killing all processes and daemons that are not directly required for audio purposes, maximizing the PCI latency timer for PCI sound cards on all test devices and manually configuring IRQs to prioritize audio input, encoding and networking functions.

3.2.5 Perception of sound delays by humans

A major preliminary research priority for this project was to establish the delay limits at which musical performance and collaboration are realistically possible. Several studies concluded that 25ms from end-to-end is the functional limit for close collaboration between musicians [6, 7, 8], which is an unrealistic amount of latency for current networks. Other research concluded that 60ms is an acceptable upper bound for collaboration between musicians performing continuous sounds [9, 10, 11]. After personal investigation with several other musicians, both 25ms and 60ms upper latency boundaries provided ample opportunity for creative expression without presenting unusual delay. At latencies above 25ms musicians need to anticipate actions from the other performers, but this has been possible at up to 60ms in my experience. The experiments for this project use a 60ms soft upper bound for audio transfers, but 25ms latency limits are ideal.

Humans perceive delays of 5ms or greater in continuous sounds, with a mean gap detection threshold of 4.19ms, and therefore this project will attempt to use frame sizes within the 5ms guideline to minimize the perception of audio signal loss during audio transfers. [15] Practically, this project seeks to limit the round-trip time of audio to 60ms from the creation of audio on the sender's side to playback on the receiver side. Unexpected additional delays that are slightly more than 60ms should not interfere with the end user's experience, but keeping the median delay and standard deviation of delays near or below 60ms is a key requirement for this program to provide reasonable QoS. It is important to note that these latency requirements for end-to-end delivery of audio packets differ from the time requirements of typical VoIP arrangements, as the International Telecommunications Union (ITU) recommends a end-to-end delay of no greater than 150ms for acceptable VoIP QoS. [12]

3.2.6 Notes on Anti-Aliasing

Anti-aliasing filters serve to restrict the bandwidth of a signal to the range desired for sampling (such as 44.1kHz) in digital signal processing. Practically, these filters will allow some aliasing, causing two different signals to become indistinguishable from each other at the point of sampling. Less aliasing will occur when using a higher-quality filter. Such filters typically reside in the input stage of audio digital signal processors to digitize analog waveforms, but the effectiveness of these filters will always be imperfect. Oversampling of audio signals[16] and other design workarounds

15 National Institutes of Health: http://www.ncbi.nlm.nih.gov/pubmed/18465408

16 Oversampling is very common in audio, particularly at the output stage. Oversampling works by using higher digital sample rates such that a digital filter can make a clean cut off of aliasing at the original Nyquist frequency. A simpler

allow for effective anti-aliasing filters. In practice, anti-aliasing works as a low-pass filter, which effectively removes all frequencies above those necessary to reproduce an analog signal according to the requirements of the Nyquist-Shannon theorem.

3.2.7 Notes on Jitter

Jitter occurs when a signal deviates from its true periodicity, typically in relation to a reference clock source. For audio, this means that jitter occurs when a representation of an analog waveform is converted from a digital to an analog representation and the distances between transitions of the waveform are uneven, which causes signals to trigger at uneven intervals and introduces distortion. An illustrative example is drawing a sine wave on graph paper and then attempting to redraw the same sine wave on graph paper where the vertical lines are unevenly spaced. If the vertical lines are taken to represent time samples of the wave, that provides a conceptual model of audio jitter. In networking, jitter refers to the statistical delay of the arrival of packets throughout a session. My project is concerned with both the audio-specific and network-specific aspects and definitions of jitter.

3.2.8 Notes on Packet Loss Concealment

Opus provides built-in packet loss concealment to mask or reduce the effects of network packet loss. Packet loss techniques fall into three general categories: Insertion-based, interpolation-based, and regeneration-based schemes. [20] Insertion-based schemes insert fill-in packets for lost packets, typically consisting of noise or silence. Interpolation-based schemes utilize patterns to create a replacement packet that estimates the waveform of the incoming audio based on historical patterns. Regeneration-based packet loss concealment mechanisms derive codec parameters from the encoder and synthesize a replacement packet. Opus relies on insertion and regeneration, although in some circumstances the Opus utilizes the SILK codec which can insert noise or silence. The waveform

analog filter is able to make the same clean cut off of all frequencies above that new higher Nyquist frequency. This technique permits cheaper design of anti-aliasing filters.

differences from packet-loss concealment in various scenarios are well documented in *A Survey of Packet-Loss Recovery Techniques for Streaming Audio.* [20]

3.3 The RTP/RTCP Protocols

3.3.1 Introduction: The RTP/RTCP Protocols and Network Requirements

Multimedia applications typically use application streaming for multimedia streams. This type of system generally consists of three primary components: A streaming server, a client, and the network that links the two. [13] The rationale for using RTP and RTCP for this project derives from several basic considerations and an exhaustive set of tests performed on other potential options. The primary motivation for the transport format used in these experiments is that the RTP format provides a lightweight, real-time mechanism for transmitting Opus audio frames of variable size[17]. Also, RTP's sister protocol, RTCP, provides a network feedback mechanism that permits adjusting audio quality to increase or decrease the size of the audio file sent to ensure packet delivery within the median 60ms requirement for playback. In my implementation code and test cases, the RTCP feedback is used adjust the quality of encoding, typically by the Gstreamer application and Opus's built-in functions for maintaining QoS. The operations performed to ensure QoS comprise calculations on the round trip time for packet delivery, the ratio of lost packets, and the jitter contributed by the network. [14]

3.3.2 Utilization of RTP additions to UDP layer

The RTP protocol builds upon lower-level protocols, TCP or UDP, which operate at the transport layer.[18] My experiments utilize RTP built upon UDP exclusively since my objectives do not require guaranteed delivery of packets or an in-order delivery of packets. RTP, operating at the application layer, uses two destination address and ports along with a network address. These two ports consist of one for RTP and the next highest port number for RTCP. This sequential port allocation

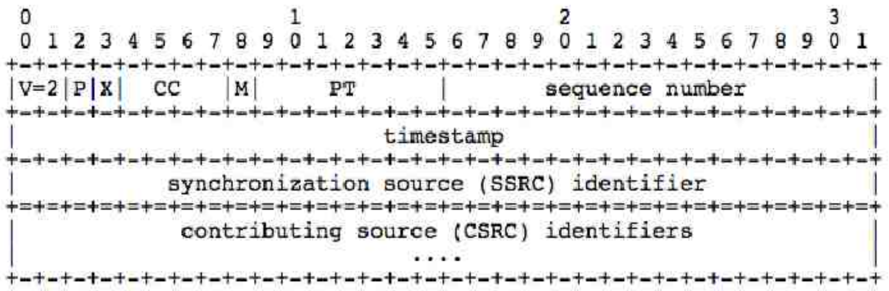
17 Draft Specification for RTP Payload of Opus data: http://tools.ietf.org/html/draft-spittka-payload-rtp-opus-03
18 http://www.ietf.org/rfc/rfc1889.txt Section 1.3

occurs by default according to the specification of RTP. This protocol can be used for many different purposes for multimedia transfer and grants much more flexibility than standard UDP for building robust applications. Critically, RTP also uses a low-level relay system which provides for degradation or enhancement of quality across the session as necessary.

Fundamentally, RTP provides for sender and receiver functions, and these are the functional basis for my experiments. Aside from senders and receivers, RTP defines another two vital roles: translators and mixers. Residing between senders and receivers, translators and mixers provide more advanced mechanisms for adjusting the stream quality and combining source streams respectively. The translator provides the ability to convert a stream to audio (or, if applicable, video) formats that use less bandwidth if the link is unable to handle the requirements of the transmission. The mixer provides the ability to combine multiple streams into one, which is useful for combining multiple network-connected musicians into a single consonant performance. The mixer also enables the quality adjustment functionality of the RTP protocol. [15]

The RTP packet has four necessary sections: The IP header (used to encapsulate the attached UDP packet), the UDP header, the RTP header and the payload type (Opus, in this context). The RTP header, as shown in Figure 1 below, consists of a version number, an extension bit, the CSRC count, a market, the payload type, the sequence number, the timestamp, the SSRC and the CSRC list. The receiver receives values for synchronization, the synchronization source, the ordered packets (identified by the sequence number) and the sampling instant. [16]

Figure 1: The RTP header



```
 0                   1                   2                   3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|V=2|P|X|  CC   |M|     PT      |       sequence number         |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                           timestamp                           |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|           synchronization source (SSRC) identifier            |
+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+
|            contributing source (CSRC) identifiers             |
|                             ....                              |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

3.4 Quality of Service (QoS) assurance through RTCP

3.4.1 RTCP Elements and Description of Usage in my Experiments

The RTCP protocol provides for two types of packets: SRs (Sender Reports) and RRs (Receiver Reports). These packets provide descriptions of the session state at a given time. Both types of packets contain an SSRC, defined as the Synchronization Source Identifier of the unique sender. Receiver Reports contain the SSRC of the first source. [2, 4.9] There are additional types of packets provided by RTCP, such as:

- SDES : Source descriptors, such as CNAME.
- BYE: A packet indicating the end of a transmission by a specific participant.
- APP: Custom field information for specific application functions.

SRs and RRs provide the information to determine the amount or percentage of lost packets and thereby indicate the quality of service (QoS) at a given interval. The most vital information that can be derived from SR and RR packets are the number of packets lost, the highest sequence number received and the delay since the last SR timestamp.

3.4.2 Derivation of QoS data through formulas

Metrics for round-trip time, transmission efficiency and jitter derive from several standard formulas. The round-trip time results from subtracting the time at which the last sender report (TSR) arrived from the time a source receives a given receiver report (TRR) and then subtracting the delay since the last sender report (ΔD). Therefore, RTT = TSR - TRR - ΔD. The RTCP transmission efficiency (E) is defined as the number of packets received (PR) compared to the ratio of packets expected (PE), or E = PR/PE. The ratio of lost packets (LP) derives from dividing the number of packets lost (PL) by the number of packets expected (PX) or LP = PL/PX. The number of packets expected by the receiver is the highest sequence number received subtracted from the first sequence number, or PX = SMAX – S1. [17]

3.5 Rationale for accepting only real-time audio without delays

It is important to note that the TCP protocol is inappropriate for this type of audio transfer because of a subjective decision that received audio should output in real-time or be discarded such that audio is continuously played back on the receiver's side. The TCP format incurs additional overhead and throttles the rate at which packets are sent based on feedback on the number of packets received in a window of time. TCP's purpose is to transmit packets of ordered data that must be received intact, as verified by a checksum operation. For example, an e-mail message transmitted via TCP would need to accurately represent the original source text material, and the end result does not need to transmit in real time. However, in a real-time audio performance, audio should play back on the receiver side within a median time of 60ms of its creation, or else the ability to collaborate degrades. As such, the on-time arrival of audio is more important than a perfect replication of the source material, even if packets are occasionally received with errors or out-of-order. The experiments in this thesis deliberately conceal or discard missing or out-of-order packets respectively via the aforementioned Opus packet loss concealment mechanism, which uses information about previously-received audio to conceal packet loss.

Ultimately, UDP-based RTP packets are preferable because it does not provide checksums or verify that the packet arrives intact (or at all), both of which are unimportant in real-time transmissions. RTP provides a compromise between TCP's robust QoS data and verification mechanisms and UDP's minimalist approach to packet transfer by sending RTCP packets with session data. Instead of using TCP's automatic transmission and retransmission mechanisms, RTCP packets provide transport layer data to the application layer to inform the host application as to how the audio encoder should degrade or improve the audio quality. That data allows for decreasing or increasing the overall file size of the audio packets to best fit the constraints of the network's capacity to transmit data within the time constraint of 60ms of round-trip delay. Additionally, the packet-loss information from RTCP responses provides the host with feedback to change the way in which it encodes audio if necessary to improve the consistency of data throughput.

3.6 Relative speeds of audio transfer

The base level encoding quality for these experiments is 16-bit at a 44.1kHz sample rate in stereo format, which is equal to CD audio quality. This subjective choice takes into consideration that most recorded music uses CD audio quality as a standard for distribution and audio recorded at higher sample rates and bit rates is ultimately downmixed to CD quality. Practically, that level of complexity equates to a data rate of 2 * 16 bits per sample * 44,100 samples per second, which equals 1411kbit per second for raw data. That figure does not include the additional bits required for packet headers. So-called lossless encoding formats, such as the Fully Lossless Audio Codec (FLAC), typically compress audio data to about half the size of PCM data, or about 700kbit per second for this example.[19] However, these data rates are considerably higher than acceptable data rates for network transfer of audio, given that a typical DSL connection speed has a maximum upload rate of roughly 1024kbit per second.

3.7 The Opus Codec Format

3.7.1 A history of Opus and VBR audio formats

Opus is a recently-developed format for audio created by the Internet Engineering Task Force Codec Working Group, and this codec succeeds the previous Ogg Vorbis format. Opus is standardized in RFC 6716, first released on September 11, 2012. Ogg Vorbis, AAC, and MP3 VBR each fulfill the requirements of variable bit rate (VBR) formats but each suffer from a number of disadvantages, in particular long encoding times and much larger file sizes than Opus.[20] Additionally,

19These figures are validated experimentally in section 4.3.
20 http://www.iis.fraunhofer.de/content/dam/iis/de/dokumente/amm/conference/AES116_guideline-to-audio-codec-

from an objective and subjective standpoint, the Opus format provides much better performance at lower bit rates, as is shown in chapter 4.

3.7.2 Rationale for using Opus in this context

Opus has many features that make it ideal for real-time audio streaming, in particular an extremely low algorithmic delay which can be as low as 5ms.[21] Opus also permits switching between bit rates without interrupting the signal flow and utilizes the most appropriate codecs (SILK or CELT) depending on the bit rate of input. Specifically, Opus uses a transform codec at high bit rates, a linear prediction codec at low bit rates, and a hybrid of the two when switching between codec types. These are explained in more depth in section 3.7.4. The combination of adjustable bit rate and other internal advantages make this codec ideally suited for transmitting audio signals at variable bit rates in real time. Also, the size of the individual Opus files tend to be quite small compared to other formats, which facilitates faster transfer of data over a network for my real-time purposes. Additionally, Opus allows for frame sizes as small as 2.5ms, which permits greater flexibility for transferring the audio files as RTP payloads. [22]

A crucial benefit of this format is the simplicity of creating and transferring Opus packets using the RTP protocol. Notably, the fact that the RTP payload will always equal one fully-formed Opus packet of a regular frame size is helpful, as is the lack of out-of-band signaling to decode a packet. In the latter case, there is no possibility for negotiation failure, all of the packet descriptors are usable and the necessary information regarding the audio itself is conveyed in-band. Also, the RTP Opus packet encoder and decoder are able to function at different rates, which allows for adjustment of the encoding

delay.pdf

21 http://www.opus-codec.org/presentations/opus_ccbe2013.pdf

22 Skoglund, Maxwell, Nguyet. “Opus Testing”. Proceedings of the IETF. http://www.ietf.org/proceedings/80/slides/codec-4.pdf

quality.[23]

3.7.3 Standards for audio quality for "high fidelity"

The standard for "high quality" in the context of this paper is that the original signal should be represented as faithfully as possible from both a subjective perspective and from an objective signal analysis. First and foremost, this requires an appropriately high sample rate so as to encompass the frequency bands of the source audio. Typical sample rates in audio are 44.1kHz, 48kHz, 50kHz, 88.2kHz, 96kHz, and 192kHz. [18] According to the Nyquist-Shannon theorem, a sampling frequency at least twice as great as the maximum frequency present in the original signal is necessary to accurately represent an analog signal digitally. [19] The range of human hearing (not considering hearing damage due to tinnitus or age) is 20Hz-20kHz, so sample rates of 40kHz are typical for encoding as a consequence of the Nyquist-Shannon theorem. However, there are reasons that an audio signal should be encoded at a higher sampling rate greater than simply double the range of human hearing. Musical instruments produce frequencies higher than the range of human hearing and, when these signals are compressed, the overall quality of the resulting audio is often reduced.

Standard quality levels of VoIP technology tend to be very low quality. This is because speech is considered an simple signal due to the limited tonal range of normal speaking voices, and also because the essentials of language can be easily determined even when highly compressed or transmitted at low sampling levels. Typically, digital telephones or VoIP sound sampling utilizes 8 bits per sample, or an 4kHz sample rate, and 8000 samples per second (64kbps), sent in raw pulse code modulated format. [17] [2, 4.11]

For most practical purposes, a sample rate roughly equal to double the range of human hearing is acceptable for audio, which is part of the reason the Compact Disc format uses the 44.1kHz sample

23 http://www.opus-codec.org/presentations/opus_ccbe2013.pdf

rate. Higher quality audio, such as those found on DVD-Audio disks and Blu-Ray Disc formats, are encoded at 96kHz or 192kHz sample rates respectively. During music recording, higher sample rates are often used to preserve the overtones and original quality of the original audio source, and reproducing at the same levels of quality during playback is ideal.

3.7.4 Modes in Opus for encoding

Opus extends two codecs: Skype's proprietary VoIP codec SILK, which utilizes lower bit rates, and CELT, which is better suited for higher-quality audio. The sample rates of 8, 12, 16, or 24kHz at the bit rate range 6-40kbps make this format unsuitable for music data where the preservation of audio quality is a requirement. However, SILK provides a higher-than-normal level of audio quality than is typical for speech transmission media. The version of SILK used in Opus is extensively modified to work with Opus, but it is only well-suited for speech (narrowband and wideband) at up to roughly 32 kB/s.[24]

The predecessor to Opus, the Constrained Energy Lapped Transform (CELT) codec, provides very low algorithmic delay and fullband frequency capabilities. CELT also contributes packet loss concealment, which permits gradual degradation of audio quality in the context of transmission errors without interrupting continuous playback. CELT functions most efficiently on 48kHz audio, and less efficiently on lower-quality audio.[25] This codec also provides robust capabilities for preventing cross-talk and ensuring full representations of stereo images in audio.

3.7.5 Technical aspects of encoding

There are valid reasons that a user may not want to use higher sample rates for recording or transmission, such as when very high frequency content contributes distortion or the endpoint is unable to play back higher sample rates. The tests described herein utilize 48kHz and 96kHz sample

24 http://www.opus-codec.org/presentations/opus_ccbe2013.pdf
25 http://www.opus-codec.org/presentations/opus_ccbe2013.pdf

rates, as they are typical sample rates for music production and are supported by Opus. The use of these common sample rates aims to prevent downsampling at the receiver end if the receiver is not configured to play back unusual or irregular sample rates, which would incur additional audio delay and produce distortion. However, my experiments also include some irregular sample rates, as the closed environments of my test suites allow for safe assumptions of correct playback configurations.

Additionally, Opus offers several capabilities that my project specifically disables or avoids due to irrelevance or unnecessary complications that they incur. Most importantly, all input sound is assumed to be two channels (stereo). Opus permits audio encoding in up to 255 channels, but the higher complexity of these signals creates additional latency for very little gain given that the vast majority of music is recorded in stereo format. While recordings of orchestras and larger bands may benefit from the deeper sound field image provided by additional channels, the purpose of this project is to provide at least two musicians with the ability to collaborate. Enabling additional channels is trivial through the use of the Gstreamer-Opus pipeline processes, but my experiments use two channels to reduce latency as much as possible.

Aside from basic channel constraints, lookahead is disabled in all tests. This is because delayed decision modes rely on looking ahead into buffered audio to anticipate the use of the SILK or CELT codec type (speech or music). Because this project requires minimal latency buffered audio is disadvantageous and as such the feature is disabled. Also, the experiments described herein automatically follow the recommended bit rates for Opus (6 kbits/second to 510 kbits/second) given in section 3.1.1 of the IETF draft specification for the RTP Opus payload format.[26] Discontinuous Transmission Mode, as described in section 3.1.3 of the draft specification of the RTP payload format for Opus, is enabled for some of my experiments to compensate for failures in network transmission of

26 Opus specification: http://tools.ietf.org/html/draft-spittka-payload-rtp-opus-03#section-3.1.1

audio.[27] Finally, jitter (defined in section 1.2) is concealed by internal mechanisms in Opus instead of external custom methods, as compensation for jitter is computationally intensive and creates additional delay. In the case where packets arrive out of order or malformed due to jitter, Opus's built-in error concealment activates or else the output is silence.

3.7.6 Bit rate adjustment during encoding before transmission

All well-formed Opus packets must contain at least a single byte called a Table of Contents (TOC) header that signals the configuration and modes used by a given packet, the structure of which is described in section 3.1 of RFC6716.[28] These can vary from packet to packet and permit continuous playback of audio with dynamic bit rates. The first five bits of the TOC byte define which codec mode is in use (SILK, CELT, or a hybrid of the two), as well as the bandwidth and the frame size. A required additional bit, indicating mono or stereo format, is always set to 1 to indicate stereo in my experiments. [1] Opus implements the SILK mode, which permits a variable bit rate (VBR) or a constant bit rate (CBR). In the constant bit rate mode, SILK will forcibly encode each packet with no more than the specified number of bits. In my experiments, these modes are hardcoded for comparison to their benefits to QoS.

Also, in line with media type registration as specified in RFC4288 and RFC4855, Opus offers several configurable parameters in the RTP payload header. These parameters are the means through which this project's code defines the sample rate and dynamically adjusts the bit rate. The parameter “maxaveragebitrate” varies based on the transmitted media, but the maximum average bit rate can be set dynamically during a session. When the maximum average bit rate is set dynamically, this signals

27 Opus specification: http://tools.ietf.org/html/draft-spittka-payload-rtp-opus-03#section-3.1.3

28 RFC5741: http://tools.ietf.org/html/rfc5741#section-3

that the quality of encoding has been adjusted according to the network feedback. [29]

3.8 Technologies Used for Experiments

3.8.1 Cython

Cython is a combination of the C and Python programming languages, allowing for the combination of C and C++ code in the context of Python code. Technically, Cython is a superset of Python that utilizes a foreign function interface to permit C and C++ code conventions. Unlike Python, Cython is a compiled language rather than an interpreted language, utilizing the CPython interpreter. Cython is primarily utilized for scientific computing, as it allows for optimization using C/C++ when necessary for critical and time-sensitive operations, while permitting the user the freedom and simplicity of Python syntax for other areas.

3.8.2 UNIX Audio Services and ALSA

The Open Sound System (OSS) is the collection of interfaces provided by UNIX systems for audio device functionality. It may also refer to the portions of the UNIX kernel that provide audio interface calls. On Linux systems, there are usually two audio systems offered, OSS and ALSA (Advanced Linux Sound Architecture), with ALSA being much more common. OSS provides more limited functionality than ALSA, but it provides highly advanced documentation and simplicity of processing with high-quality sound. This project utilizes the ALSA format.

ALSA is a unified system that facilitates functionality typically provided by specific audio drivers on other platforms, permitting reduced development time due to the use of common mechanisms for audio input, output, processing, and plug-in handling. ALSA exists inside the Linux kernel (with significant latency enhancements in Ubuntu Studio), and provides libraries for use by user applications and the system. This architecture provides notable advantages in that it can write directly

29 http://tools.ietf.org/html/draft-spittka-payload-rtp-opus-03#section-6.1

to output buffers continuously, providing lower latency and higher throughput for network applications. However, incidents where the stream is cut off or runs over the buffer will introduce audio artifacts that must be prevented by the programmer or tolerated by the user.[30]

3.8.3 GStreamer

GStreamer provides a lightweight framework written in the C programming language for piping audio or video playback. The format also provides lightweight transfer mechanisms in pipeline format for playback, streaming, and recording. My experiments utilize Gstreamer as the basis for transmitting audio data via RTP packets, due to the fact that it offers an effective method for implementing the Opus RTP payload format and providing for playback and statistics generation. This format also utilizes plugins to allow for different shared libraries and novel codecs on the fly, which allows it to be expanded and elaborated upon.[31]

3.8.4 Ubuntu Studio

Ubuntu Studio is a fork of the Ubuntu Linux distribution and it is intended for the creation of media. This operating system facilitates extremely low latency (below 5ms out-of-the-box) for audio signals through an available real-time kernel, which makes it ideal for purposes where end-to-end latency is critical. By utilizing the underlying ALSA sound framework and the real-time kernel for the encoding of audio, the RTP transfer of the Opus payload and the playback of received audio data, the overall latency of the system is greatly reduced compared to a system without real-time access to the processor.

30 https://www.kernel.org/doc/htmldocs/writing-an-alsa-driver/index.html
31 http://gstreamer.freedesktop.org/documentation/

Chapter 4: Experiments

4.1 Purposes of Experiments and Goals

This project comprises four primary experiments. The first is a analytical proof that Opus audio conveys an acceptable representation of the original source audio whereas previous popular formats do not. The second experiment demonstrates the process of isolating sources of latency in the end-to-end transfer of an audio signal, accompanied by a brief discussion of custom optimizations that reduce latency as much as possible. The third experiment describes the transfer of audio in controlled local networks, which serves to isolate the latency caused by hardware and permits controlled tests of control mechanisms to ensure acceptable QoS. The final experiment demonstrates the functionality of this system over the Internet in several test cases and discusses issues inherent to transferring audio over an uncontrolled network.

The first experiment, described in section 4.2, serves to justify Opus as a means of conveying high-quality audio for networked music performance as compared to previous formats. The objective of the second and third experiments, described in sections 4.3 and 4.4, is to provide a rich set of data to isolate the constraints of present-day hardware for transmitting live audio data in real-time over optimized systems and controlled networks specifically. The final set of experiments, described in section 4.5, demonstrates networked audio performance over the Internet and situations where geographic constraints or signal loss creates unavoidable delay beyond levels acceptable for real-time audio.

4.2 Set One: Comparison of the Opus Format vs. Previous Formats

The purpose of this experiment: Justify the use of the Opus format as a high fidelity format and demonstrate its superiority to other commonly-used formats for networked audio performance.

Methods used to establish limits: Spectrographic frequency comparison of identical raw PCM audio input encoded in different common formats, highlighting the differences in fidelity between the resulting file and the source as well as the complexity of creating each file. Careful analysis of these spectrographs show the loss and distortion of audio compared to the original source.

Resulting data:

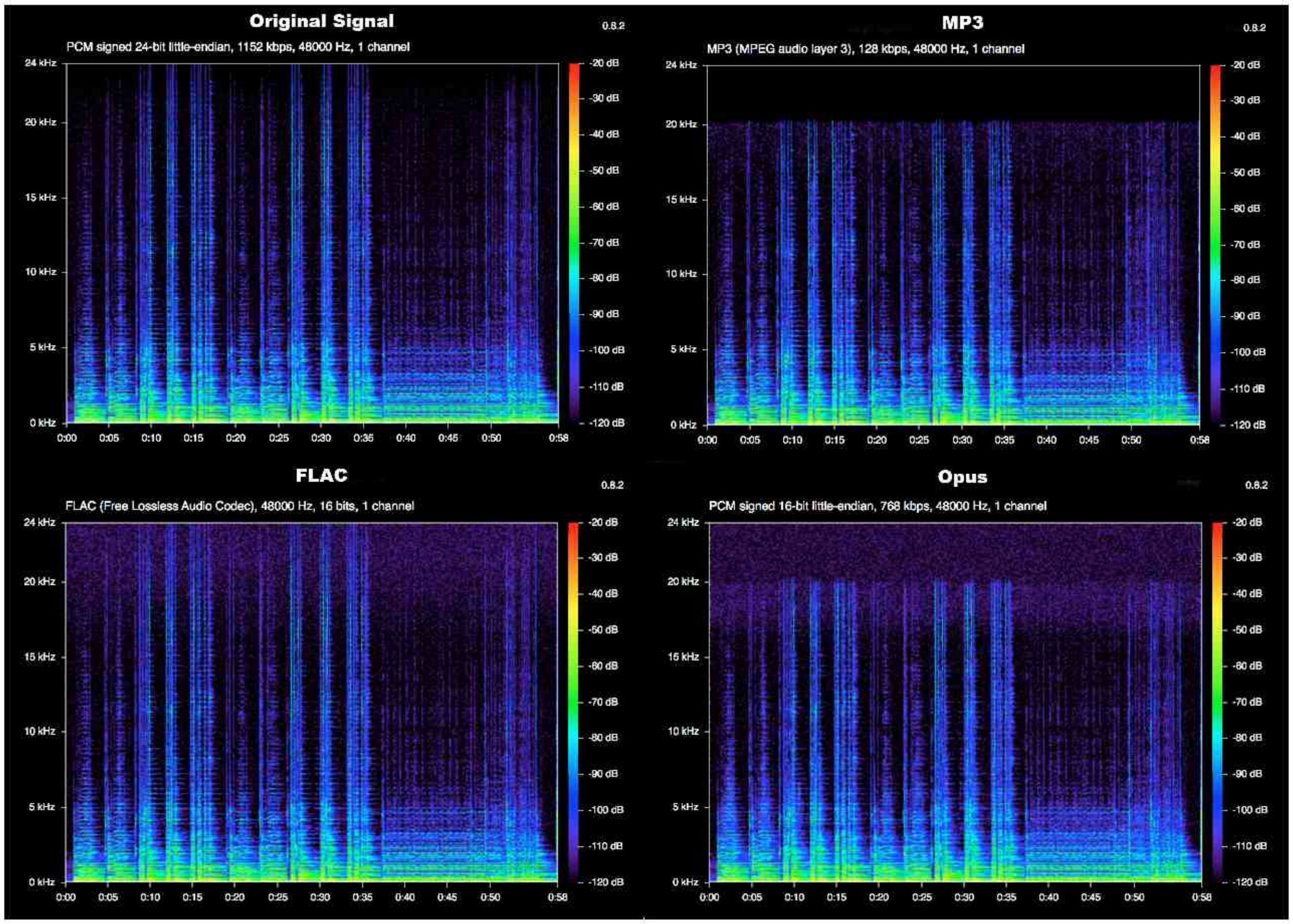


Figure 2: Spectrographic comparison of original signal, FLAC lossless audio format, MP3 format, and Opus (sampled at output). Spectrographs generated in Spek.

Format	Resulting File Size
32-bit AIFF (original PCM)	11.1MB

Format	Resulting File Size
24-bit WAV (uncompressed, original signal)	8.4MB
16-bit MP3	929.7kB
Ogg Vorbis VBR	590.2kB
Opus	531.8kB

Figure 3: Comparison of the file sizes of popular audio formats resulting from encoding a test file in variable bit rate modes for best quality.

Discussion of the data: The spectrographs show clear differences in the representation of audio compared to the original source, particularly between the typical lossless format and MP3, which is the most common format for recorded and streaming audio. The original source closely maps to the FLAC format output, as expected for a lossless audio codec. The MP3 file deviates considerably from the source audio, introducing artifacts and distorting the frequency response of the original source significantly, particularly in the higher frequencies. A qualitative analysis of the resulting Opus file shows that the encoded version strongly conforms to the original source and lossless audio file's structure, providing a much greater level of fidelity than the MP3 standard. Because Opus is a lossy format, it removes frequency information above the range of human hearing (20kHz) just as MP3 does.[32] However, a comparison of the two spectrographs clearly shows that the compression and data representation of the MP3 format distorts the original signal to a far greater degree than Opus. In listening tests, the difference in quality between the two files and the original is readily apparent.

32 The noise above the 20kHz frequency level stems from the fact that the spectrogram for Opus had to be analyzed at the point of output, whereas the MP3 file could be analyzed in place.

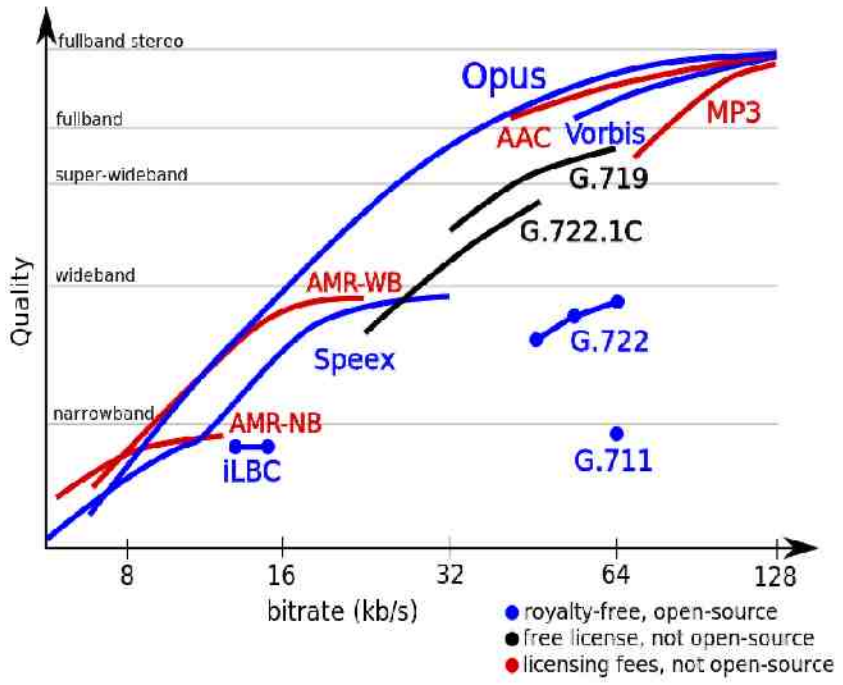


Figure 4: Generalization of quality and availability of codec formats as compared to Opus audio. (source: Xiph.org)

Conclusions: The spectrographs clearly show that the Opus codec delivers improved performance compared to its foremost competitors in streaming audio formats. Additionally, Opus encoding of test files resemble the original source audio much more closely that the competing formats, providing a good basis for a claim that the format provides high levels of audio fidelity for practical purposes. Given that the size of the resulting file (and individual frames) for Opus is also lower than the inferior competing formats, we can reasonably conclude that Opus provides improved capabilities for networked audio performance. By comparing the size of the generated files, we can also conclude that transferring lossless audio or the original source audio requires far more network bandwidth than the lossy formats. Therefore, although there is a compromise in quality from the lossless audio and Opus, the objective quality difference is not nearly as great as other lossy formats and the resulting file size is much smaller.

4.3 Set Two: Determining Practical Limits of Current Network Technology:

The purpose of this experiment: Establish sources of latency for audio transfers over networks to balance quality with the goal of delivering audio to an endpoint with a median delay of 60ms after performance. This involves: 1) Isolating latency from input to encoding, forming RTP packets with Opus payload data and decoding to output, and 2) establishing connection speed and geographic limits on transferring audio within the latency limits for real-time audio. The ultimate goal is to establish a basis for transferring audio over a network, as the latency for audio input and encoding/decoding must be less than the network transfer latency for the concept of networked audio performance to work.

Methods used to establish limits: For establishing connection speeds, a simple custom program recursively pings test machines located in different geographic locations with similar connection speeds. The test cases for this experiment describe the initial checks performed as diagnostics for this system. These test cases determine an average round trip time for the end-to-end connection in order to determine if high-quality real-time audio can realistically be transmitted and, if so, how much network latency to expect by default. The results of these tests, if performed, are provided as parameters to the next set of test cases instead of the default midrange values to determine if Opus's variable bit rate encoding (VBR) mode suffices for a consistent performance, or if the bit rate must be constrained to reduce the packet size and the network transfer latency in order to provide adequate QoS.

Practically, this set of code pings a given IP address at intervals over several minutes and writes the resulting worst-case, average case, best-case latency, and the average difference between the ping latencies to a parameters file that is loaded by the test software. The resulting parameters file provides seed values to the demonstration program which defines whether VBR mode or a constrained hard constant bit rate should be used to provide adequate QoS from the start of the program's execution. The average latency value is then combined with an arbitrary value for the expected additional latency (double the encoding latency by default) from decoding on the receiver side, which defines the expected initial mode and, if necessary, the expected limited bit rate value. The average latency variance values give the initial threshold values for estimating the bit rate during packet loss situations, and the worst-case latency defines the highest bit rate for audio that the program should increase towards. The equation for the latency estimate is: [Total latency] = [encode time] * 2 + [estimated transfer time to destination]. The total latency should be 60ms or below, unless otherwise specified in the parameters file, so as to stay within the QoS requirements for real-time audio.

Resulting data: For this experiment, I isolated the sources of delay and continually refined the process priority and OS-level optimizations until I minimized overall audio latency. The results on my local test machine using the internal microphone as an audio input are shown in Figure 5, and the results of using a Lightning/FireWire interface are shown in Figure 6:

Stage of Signal Path (48kHz PCM stereo audio)	**Average Latency (from 100 test cases)**
Sound propagation through air to microphone	1ms (measured)
Microphone delay to PCM encoding	Average of 2ms
Encoding in 2.5ms Opus frame at 256kbit/s bit rate	Average of 1.37% overhead = 0.03425ms
Formation of RTP packet from 80 byte frame	Average of 0.9% overhead = 0.0225ms
Decode time for 2.5ms frame at 256kbit/s	Average of 1.33% overhead = 0.03325ms
Total Latency	3.091ms

Figure 5: Latency without network delay

Stage of Signal Path (48kHz PCM stereo audio)	**Average Latency (from 100 test cases)**
Sound delay from guitar to Lightning/FireWire buffer	1.8ms + 4ms = 5.8ms
Encoding in 2.5ms Opus frame at 256kbit/s bit rate	Average of 1.37% overhead = 0.03425ms
Formation of RTP packet from 80 byte frame	Average of 0.9% overhead = 0.0225ms
Decode time for 2.5ms frame at 256kbit/s	Average of 1.33% overhead = 0.03325ms
Total Latency	5.89ms

Figure 6: (Apple) Lightning/FireWire interface

Additionally, the result of sending continuous pings during hourlong windows over several days to test workstation latency in different geographic areas yielded the results in Figure 7:

Location/Distance	**Average network latency (taken over multiple days)**
Danville (40 miles)	23.1ms
Monterey (77 miles)	29.9ms
Thousand Oaks (340 miles)	36.2ms
UCSD (San Diego, 480 miles)	106ms
University of Washington (Seattle, 840 miles)	38ms

Figure 7: Average latency values from UDP pings to various geographic locations

Discussion of data:

The results of generating these data show the base values for latency incurred by encoding, and the values of 3ms to approximately 6ms for encoding are well within acceptable guidelines. The estimate of 6-12ms for a total encode/decode delay allows for 48-54ms of headroom for network latency. Therefore, according to Figure 7, 4 of 5 destination hosts should reliably receive data within the 60ms QoS requirement while using variable bit rate encoding in Opus.

Initial versions of this experiment utilized UDP pings to domain controllers near the intended destination to determine estimated latency, expecting that the end-to-end latency would differ considerably due to unpredictable network traffic and differences of network latency over time. The latency to the destination was always greater than the ping to the domain controller, often considerably and unpredictably more. It was practically impossible to make an adequate latency estimation based on the region in which a test machine exists, so the test cases were updated to require the IP address of the destination for more accurate estimates. Realistically, pings to the domain controllers provide standards to determine whether or not the distance between the host and the destination will incur network latency beyond the acceptable 60ms guideline, but it is not useful for providing guidelines for encoding.

Conclusions:

The data collected for this experiment affirms that the goals of this experiment are possible, in that the test systems can accept the input audio, create Opus RTP packets, and decode them with enough time left for a realistic amount of network delay. Those results establish that the initial test system meets the basic requirements for keeping within the 60ms latency guideline and thereby provides a sufficient basis for the remainder of the network-based experiments. Another conclusion is that the utility of taking pings of regional domain controllers is only to provide rough baseline values for whether or not audio transfer can occur within the desired 60ms median window. The latency to reach the endpoint system is always greater than these regional domain controllers, and as such the ping of a regional domain controller can only provide geographic limits to the transfer of audio data.

Additionally, variability of connection speed was difficult to isolate in these diagnostic tests, which necessitated an additional experiment using a closed local network. In earlier versions of this experiment, the average latency to connect from my test system to my local Santa Clara, California Comcast domain server was 6ms.[33] On a slower connection (20mbps), the latency to the same Comcast domain server is 12ms during the same time period. On the same slower connection, the latency

33 This number is derived from the average of 150 pings at off-peak hours (11pm on a Sunday) with two extreme outlier latency values removed.

between a Comcast domain server in Los Angeles, 350 miles away, was 48ms. The faster connection was able to ping the same server at the same time with an average latency of 41ms. This underscored a critical challenge with live audio transmitted over the Internet, as many test runs of my experiments were derailed by insufficient connection speeds combined with long delays introduced by the geographic distance between the server and the endpoint machines. As a result of these early experiments, later tests were performed on standardized connection speeds or deliberately constrained local-area networks. In later versions of this experiment, latency between host and destination decreased greatly due to optimizations and more consistent network conditions.

4.4 Set Three: Experiments to Justify and Refine Threshold Mechanisms and Test Local Area Networks

Part 1: Experiments to Justify Threshold Mechanisms

The purpose of this experiment: These experiments isolate the difference between preset constant bit rates for encoding audio and the additional delay they contribute to the overall latency of the end-to-end transfer of audio data.

Methods used to establish the thresholds: Continuously transfer test audio encoded with different settings of Opus's hard-cbr (Hard Constant Bit Rate) setting and analyze the results over the time to transfer and decode the data over a local test network. Ideally, this system will use 2.5ms or 5ms frame sizes so as to keep within the 5ms period in which a human would perceive a gap in audio. Initially, this system increased the level of quality for an arbitrary number of periods (5 in my experiments), alternatively increasing the frame size and the quality with the frequency of the generated RTCP packets. The RTCP packets were transmitted with every third RTP packet until the system reaches the point of stably increasing quality, at which point the transmission of RTCP packets was reduced to be no greater than 5% of the total packet traffic (so as to align with the standard for

RTCP). This was later superseded by Gstreamer's automatic handling of RTCP feedback, which reduces computational complexity of the system and the possibility of interruptions. The test cases for this system simulate several situations where the network speed degrades rapidly or encounters service interruptions and the bit rate quality adjusts rapidly to increase quality and minimize service interruptions. [2, 2.10]

My experiments started with the basis of the TCP Reno mechanism for adjusting network throughput based on network feedback. My interpretation of the TCP Reno implementation in this context is to halve the bit rate of the Opus payload when more three or more consecutive packets are reported missing or exceed the median 60ms round trip time limit required for QoS. When the system reports a critical degradation of QoS, the initial experimental programs immediately attempt to salvage the performance and resume the real-time transfer of audio. This is done by drastically reducing the quality and then slowly and linearly increasing the bit rate over time while monitoring how quickly the system approaches the 60ms median latency for QoS. TCP Reno and TCP Tahoe provide the conceptual basis for this bit rate adjustment algorithm. In practice, this algorithm will decrease the audio quality to the median bit rate defined in the initialization process of Section 4.3 first. If packets are not delivered within the QoS guidelines after reaching the defined median bit rate, the algorithm will decrease the bit rate to half of the present bit rate until audio transmits within acceptable time guidelines. In the default base case, the audio quality will increase in increments of 10kbps per 2.5ms frame size after every second of audio transmitted with 95% of packets received successfully.

Resulting data: My experiment encountered complications after coinciding major updates to Opus and Gstreamer. These updates were released after my customized code was developed and delivered superior performance compared to my code, partly due to Gstreamer's C codebase and significant improvements to Opus's RTP performance in version 1.2. So, this experiment concluded with several different sets of results. The first set of results, derived from RTP/RTCP custom code based on the TCP Reno algorithm, is described briefly in Figure 8. The RTP/RTCP experiments function well in the unrestricted LAN environment, as latency issues are not a problem. When netem is used to introduce 25ms to 55ms of delay based on the output of random number generator every 5 seconds, Gstreamer recovered more rapidly than my RTP/RTCP code despite using 2.5ms frame sizes.

Experiment Type (120 seconds)	Average bit rate
RTP/RTCP with adjusted constant bit rate	172kbps
GStreamer	187kbps

Figure 8: Results of identical variable artificial latency increases starting from 49ms at maximum bit rate in Opus encoding

That said, in the short term, JackTrip performs somewhat worse than Opus over a local area network. In tests conducted over several hours, the results of running JackTrip (transmitting uncompressed audio) and Opus (transmitting a constant signal) reveal several differences:

Type of Test	**Latency (ms)**	**Samples per Packet**	**Packet Loss (%)**
Gstreamer (44100Hz)	5.38	64	2
JackTrip (48000Hz)	6.92	64	3
Gstreamer (48000Hz)	6.6	96	2

Figure 9: Latency values of recursive tests for uncompressed data vs. encoded Opus data

Discussion of the data:

Results of the transfer of audio between two systems showed that local area networks supported both the uncompressed transfer of data as well as the Opus-encoded RTP files well within the ideal guideline of 25ms round trip time. At the highest possible variable bit rate encoding, Opus averaged 6-12ms from end-to-end over the test systems. The JackTrip system averaged 7-16ms over the same test systems. However, JackTrip had noticeable anomalies in the reproduced audio that interfered with the overall quality. Some of the anomalies could not be easily reproduced and some occur cyclically for unknown reasons likely due to operating system issues or sound card problems. Ultimately, Opus outperformed JackTrip's round-trip performance slightly in short-distance isolated local networks, but both of the systems fell within the ideal amount of time for the round trip delivery of audio signals.

Conclusions:

Opus outperformed the uncompressed audio transfer from JackTrip over time in the local area network setting. This experiment proved that Opus performs sufficiently well for continuous audio signals, and the subjective audio quality was transparent between Opus and the uncompressed PCM data. Opus performed somewhat worse than the uncompressed audio signal when resampling was required due to the increased complexity of encoding. Audio errors in the uncompressed audio transfer were likely replicated in the Opus audio, but the errors were concealed by Opus's internal mechanisms for audio correction. Additionally, transfers of PCM data suffered from periods of silence during packet loss, whereas Opus's packet loss concealment algorithms were able to cover up nearly all instances of packet loss. These results establish that this format provides several critical advantages for networked audio performances.

Part 2: Experiments to Test Functionality of Data Transfer

The purpose of this experiment: Test bidirectional transfer of audio in real performance situations to verify the integrity of the audio transfer and ensure that QoS conditions are met over both typical local networks and a specialized test network.

Methods used to establish: Input live source audio to test systems and send it to a destination host while also receiving live audio and playing it back. These experiments eliminate the Internet delay and simulate a range of test cases using artificial network delay created using Apple's Network Link Conditioner and Ubuntu's netem to demonstrate the process of refining the bit rate adjustment for the best QoS when adapting to changing network conditions. The test cases for this experiment demonstrate the process by which the bit rate adjustment algorithm adjusts to a best fit for quality given the network conditions. These tests disregard the data provided by the first set of tests because there is no interaction with an outside domain servers, and a fixed average value is provided for the local 100MB router's latency contribution.

Resulting data: Over a large number of tests of the same audio input files, the resulting statistics normalize to:

Mode of Transfer	**Latency (artificially introduced)**	**Bit rate Average (for identical input)**	**Packet Loss (%)**	**Time Elapsed in Test**
RTP/RTCP	45ms	168kbps	3	20ms
GStreamer	45ms	180kbps	0	20ms
RTP/RTCP	46ms	130kbps	5	40ms
Gstreamer	46ms	135kbps	3	40ms
RTP/RTCP	42ms	196kbps	2	60ms
GStreamer	42ms	220kbps	0	60ms
RTP/RTCP	40ms	240kbps	1	80ms
Gstreamer	40ms	250kbps	0	80ms

Figure 10: Artificial latency and bit rate adjustments favoring highest quality

Discussion of the data: The Gstreamer tests performed better overall than the custom-built code that I originally used for early experiments, largely due to improvements in Opus RTP behavior in later versions supported in Gstreamer's updated C implementation. Ultimately, the adjustment of audio bit rates based on network conditions according to a target latency limit of 60ms was demonstrated in the resulting data. The overall effect of audio quality adjustment was subjectively transparent.

Conclusions: The data from setting target latency limits and introducing ranges of delay allowed for effective automation of quality adjustments to meet QoS conditions. The Gstreamer version of the audio transfer code benefitted from significant updates to Opus that provide enhanced automatic

QoS adjustment that outstrips the capabilities of my custom code. These results allow for a predictable adjustment of bit rates to prevent signal loss beyond set limits.

4.5 Set Four: Internet-Based Tests with Recorded and Live Source Data

The purpose of this experiment: Using data derived from the first set of tests and default midrange values, this set of experiments demonstrates refinements to the algorithm for adjusting quality levels and frame size over time. This experiment also demonstrates the behavior of the test system when participants have differing Internet access speeds and intermittent consistency problems with network speed. The ultimate goal of this experiment is to demonstrate networked audio performance capabilities with Opus and the advantages of using a lightweight, lossy codec compared to uncompressed audio data. These test cases comprise the proof of concept for my main project, which seeks to provide a means for musicians in disparate geographic locations to play music together with an acceptable level of quality.

Methods used to establish limits: Using real-world situations demonstrated in test locations located in different geographic locations. The audio transfer experiments occurred via relatively similar network connection speeds over interconnected academic networks from Danville, CA to Santa Clara, CA (40 miles), Thousand Oaks, CA to Santa Clara, CA (340 miles), Monterey, CA to Santa Clara, CA (77 miles), San Diego, CA to Santa Clara, CA (480 miles), and Seattle, WA to Santa Clara, CA (840 miles).

Resulting data:

Location/Distance	**Average network latency (taken over multiple days)**	**Total average encode/decode latency (including 2.8ms error concealment latency)**	**Average bit rate (averages of 50 tests)**
Danville (40 miles)	23.1ms	7.7ms	390kbps, packet loss 0.8%
Monterey (77 miles)	29.9ms	10ms	282kbps, packet loss 2.1%
Thousand Oaks (340 miles)	36.2ms	10.3ms	220kbps, packet loss 3%
UCSD (San Diego, 480 miles)	106ms	8.2ms	64kbps, packet loss 2.6%
University of Washington (Seattle, 840 miles)	38ms	10.2ms	171kbps, packet loss 1.92%

Figure 11: Opus audio tests

Location/Distance	**Average network latency (taken over multiple days)**	**Total average encode/decode latency**	**Average packet loss**
Danville (40 miles)	41ms	None	3%
Monterey (77 miles)	62ms	None	3.4%
Thousand Oaks (340 miles)	194ms	None	3%
UCSD (San Diego, 480 miles)	320ms	Variable due to resampling	5%
University of Washington (Seattle, 840 miles)	214ms	None	3%

Figure 12: Uncompressed Audio

Discussion of the data: The Internet-based tests showed clearly that the much smaller frame size and robust RTP capabilities for Opus permit superior quality for networked audio performance. Most crucially, the much larger size of the uncompressed audio files seriously degrades overall system functionality for longer distances, whereas the Opus-encoded audio allows for much faster transfer of audio frames. Also, the greater rate of packet loss in the PCM data was not compensated for by any built-in mechanisms due to the UDP transmission format. The Opus audio frames, which tended to suffer from packet loss less than the lossless audio frames, further benefitted from packet loss concealment and the ability to adjust bit rates to ensure delivery within an acceptable window for continuous playback in almost all circumstances aside from network connection loss.

Conclusions: Based on the experimental data, it is clear that Opus extends the distance by which musicians are able to collaborate well beyond the distances that would provide acceptable QoS for uncompressed audio, and the good performance of the system at low bit rates facilitates ready deployment of audio transfers. Furthermore, Opus enables setting a desired end-to-end latency and adjusting the bit rate accordingly when packet loss threatens to degrade QoS. This provides much greater flexibility than fully uncompressed audio with a minimal difference in quality. The resulting audio system based on Opus yields greater performance and reliability within time ranges acceptable for real-time or nearly real-time network audio performance. This proves that the Opus format, conveyed via RTP packets over a network, is a viable and effective format for networked music performance within the QoS guidelines for real-time live audio.

Chapter 5: Conclusions

5.1 Overall Conclusions

The foremost result of my experiments to date is a data-based verification that most typical Internet connections suffer from too much latency to be truly useful for networked audio performance using uncompressed audio. The second result is an experimental proof that the aforementioned problem can be sidestepped by using Opus format audio, which compromises audio quality in modest and mostly unnoticeable ways in exchange for vastly better packet concealment and bandwidth requirements, thereby permitting true shared network music performance. Furthermore, even at low bit rates, Opus audio provides subjective improvements over arbitrary bit rate shelving in uncompressed audio transfers. Faster Internet connections, such as fiber optic connections, allow for much greater bit rates and longer distances for audio transfers, but for current Internet speeds the modest compromises of a compressed audio format clearly are necessary. That said, despite being a compressed audio format, Opus provides a remarkably good level of quality at lower bit rates. But, the results of these experiments underscore that the efficacy of real-time high fidelity audio depends on increasingly high bandwidth Internet technologies that will become more widely available over time. Ultimately, these results are promising in that this is a proof of concept that musicians need not wait for major Internet infrastructure upgrades to collaborate in increasingly high-resolution networked audio performances with others over long distances.

5.2 Results of Failed Experiments and My Conclusions

My initial goals of providing dynamic sample rate adjustments in addition to bit rate adjustments proved overly ambitious to meet the goals of this project. In order to change the sample rate of a given connection, a new connection needs to be opened with a new “rate” RTP parameter, and the options become overly complicated. Sample rate adjustment, although possible, did not provide a significant advantage to the overall results of reducing latency and maximizing performance. The basic requirement of maintaining the original sound as much as possible while lowering the frame size to improving end-to-end speed is not helped by adjusting the sample rate, as Opus is only well-behaved in certain fixed sample rates and operating systems may need to resample the transferred audio in order to play it back. The possibility for errors in unsupported sample rates from Opus and the degradation of sound quality and end-to-end latency from resampling on the receiver side make this approach unhelpful.

Another inconclusive experiment involved attempting to elaborate on the RTP format's headers to convey target bit rate information and additional information that improves the adjustment of audio quality based on the network feedback, but this effort was abandoned because the existing RTCP feedback provides sufficient data for adjusting encoding bit rates. Along these lines, a separate effort to compare automatic adjustment of bit rates through RTP Opus streams compared to my system, which programmatically resets the bit rate to target values, was an important source of additional data. In particular, recent revisions to Gstreamer that implement new RTP improvements in Opus 1.2, became an important source of additional data. Ultimately, Gstreamer provides a simple and mostly “out-of-the-box” mechanism to get a best-fit latency-to-quality compromise for any given connection. The RTP/RTCP code used for my project, which can also be implemented via specific configuration of Gstreamer pipelines, provides more consistent bit rates at the expense of higher rates of lost packets.

Chapter 6: Future Work

6.1 Statement of problems for future consideration

There are a number of future problems that would improve the overall quality of this work. Further investigation of sample rate adjustments may open new possibilities for a best-fit frame size, but it would likely require an additional specialized encoder. Additionally, a more complex and fine-tuned incorporation of custom RTCP packet parameter feedback would reduce packet loss and increase sound quality when network latency increases beyond an unacceptable threshold. The relatively simple program based on the TCP Reno system is sufficient for most test cases, but a more complex scheme based on historical patterns and anticipated network conditions would provide more realistic anticipation of bit rate requirements.

Also, a real-time analysis of the type of sound transmission to limit the sample rate and bit rate to the appropriate maximum for the sound would be informative for predicting encoder robustness. For example, speech does not need to be encoded in very high quality forms. This would require a co-processor or additional set of monitoring system that could make inferences about the nature of the audio and change the Opus encoder's settings to favor voice or music data. Finally, it would be helpful to provide means for transmitting raw PCM data when the endpoint's processing capabilities are known to be sufficient for rapidly processing and playing back audio (when the local processor would incur additional delay). At the present time, this would likely be short-range fiber optic networks and local LANs.

Appendix A – Description of Test Environment

For the virtual machine to virtual machine tests, the test environment is a MacBook Pro with a 2.3gHz quad-core Intel i7, 8GB of DDR3 1333mhz RAM, and a solid-state hard drive to minimize latency. The two virtual machines used are each Ubuntu Studio 10.13 VMs run via VMWare Fusion with 2GB of RAM and 2 virtual cores allocated each. The input audio is performed via included audio files run from the individual VMs.

For the LAN tests, the test environments are a MacBook Pro (configured as above), utilizing a FireWire audio interface to capture live-audio audio without further system processing. The destination/secondary host is a similarly configured HP notebook with a conventional hard drive utilizing a USB microphone without further system-side processing. The MacBook Pro runs Mac OSX 10.9.2 and the HP notebook runs Ubuntu Studio 10.13. The transfer of audio is conducted over a 100mb Linksys router (model BEFSR41).

For the Internet-based tests, the test environments are described in the conclusions for the individual experiments, but the primary host is the MacBook Pro described above, and each of the connected systems meet the qualifications of being Intel Core i5 or i7 processors with at least 8GB of RAM and a 7200RPM hard drive or greater running Ubuntu Studio 10.13 or Mac OSX 10.8 or greater. In order to provide a representative sample of current capabilities of conventional networks for live musical performance, the audio interfaces and wired and wireless connections vary from host to host. All of these tests are conducted over the Internet without any artificial delay.

Bibliographical Materials:

[1] JM. Valin, K. Vos, T. Terriberry. RFC 6716. Mozilla Corporation and IETF. September 2012. http://tools.ietf.org/html/rfc6716

[2] Barreiros, Miguel, and Peter Lundqvist. QOS-Enabled Networks: Tools and Foundations. John Wiley & Sons. © 2011. Books24x7. <http://common.books24x7.com/toc.aspx?bookid=40960> (accessed March 20, 2014)
[3] Agbinya, Johnson I.. "Chapter 15 - RTCP: Real-Time Control Protocol", Section 3. IP Communications and Services for NGN. Auerbach Publications. © 2010. Books24x7. <http://common.books24x7.com/toc.aspx?bookid=26447> (accessed February 17, 2014)

[4] Meier, Florian, "Low-Latency Audio over IP on Embedded Systems." Master Thesis, Technische Universitat Hamburg-Harburg. Defended October 2, 2013.

[5] Carot, Alexander, Kramer, Ulrich, Schuller, Gerald. Network Music Performance (NMP) in narrow band networks. Audio Engineering Society. 120TH AES Convention. Paris, France. May 20-23, 2006.

[6] Audio for Wirelessly Networked Personal Devices. AES 43rd International Conference (September 29 to October 1,2011) http://www.aes.org/events/reports/43rdConference.pdf

[7] Weaver, Sarah. "Solutions for Perception of Synchrony in "ResoNations 2010: An International Telematic Music Concert for Peace." Steinhardy School, NYU. http://sarahweaver.org/files/weaver_masters_thesis_final.pdf (accessed April 9, 2014)

[8] Chris Chafe. Interactive Networked Music – Tapping into the Internet as an Acoustical / Musical Medium. CCRMA. Stanford University. http://www.fdis.org/05/Chafe_fdis05.pdf and C. Chafe, S. Wilson, A. Leistikow, D. Chisholm, and G. Scavone, "A simplified approach to high quality music and sound over IP," in Proceedings of the COST-G6 Conference on Digital Audio Effects (DAFx-00), Verona, Italy, Dec. 2000

[9]Pillay, Bipin. A Wide Area Online Music Collaboration Emulation Platform. Carleton University. 1998. https://dspace.library.uvic.ca/bitstream/handle/1828/1035/Thesis-Final.pdf?sequence=1

[10] LOLA (Low Latency Audio Visual Streaming System). Conservatorio di musica Giuseppe Tartini Trieste. Consortium GARR. http://www.internet2.edu/presentations/fall11/20111004-ALLOCCHIO-LOLA.pdf and Enabling remote real time musical performances over advanced networks. LOLA (Low Latency) Project. Conservatorio di musica Giuseppe Tartini, Consortium GARR. http://www.conservatorio.trieste.it/artistica/ricerca/progetto-lola-low-latency/lola-case-study.pdf?ref_uid=e98cac4a9c6a546ac9adebc9dea14f7b

[11] Chew, Elaine. Sawchuk, Alexander, et al. Distributed Immersive Performance. University of Southern California. http://eiger.ddns.comp.nus.edu.sg/pubs/dip-nasm04.pdf

[12] Szigeti, Tim, and Christina Hattingh. "Chapter 2 - QoS Design Overview". End-to-End QoS

Network Design: Quality of Service in LANs, WANs, and VPNs. Cisco Press. © 2005. Books24x7. <http://common books24x7.com/toc.aspx?bookid=35336> (accessed March 20, 2014)

[13] Zhu, Ce, Yuenan Li, and Xiamu Niu (eds). "Chapter 2 - Adapting Multimedia Streaming to Changing Network Conditions". *Streaming Media Architectures, Techniques, and Applications: Recent Advances*. IGI Global. © 2011. Books24x7. *<http://common.books24x7.com/toc.aspx?bookid=37941>* (accessed February 5, 2014)

[14] Agbinya, Johnson I.. "Chapter 15 - RTCP: Real-Time Control Protocol", Section 2. IP Communications and Services for NGN. Auerbach Publications. © 2010. Books24x7. <http://common books24x7.com/toc.aspx?bookid=26447> (accessed February 5, 2014)

[15]Arjan, Durresi and Jain, Raj. RTP, RTCP, and RTSP – Internet Protocols for Real-Time Multimedia Communication.

[16] [Agbinya, Johnson I.. "Chapter 17 - VoIP: Voice Over Internet Protocol". IP Communications and Services for NGN. Auerbach Publications. © 2010. Books24x7. <http://common books24x7.com/toc.aspx?bookid=26447> (accessed March 2, 2014] [Chapter 17, Overview].

[17] Agbinya, Johnson I.. "Chapter 14 - Real-Time Protocols", Section 2. IP Communications and Services for NGN. Auerbach Publications. © 2010. Books24x7. <http://common books24x7.com/toc.aspx?bookid=26447> (accessed February 17, 2014)

[18] Campbell, Roy H., Tan, See-Mong, et al. "Adaptation and Synchronization in Bandwidth-Constrained Internet Video and Audio". Department of Computer Science, University of Illinois at Urbana-Champaign.

[19] Definition of the Nyquist-Shannon Theorem. https://www.princeton.edu/~achaney/tmve/wiki100k/docs/Nyquist %E2%80%93Shannon_sampling_theorem.html

[20] Perkins, Colin, Hodson, Orion, Hardman, Vicky. A Survey of Packet-Loss Recovery Techniques for Streaming Audio. 1998. http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.49.9250 (accessed June 5, 2014).

Figures