Breaking Links: Evaluating distributed time synchronization for musical applications using Ableton Link

Xavier Riley

Submitted for the Degree of Master of Science in

Distributed and Networked Systems

Department of Computer Science
Royal Holloway University of London
Egham, Surrey TW20 0EX, UK

August 7, 2018

Declaration

This report has been prepared on the basis of my own work. Where other published and unpublished source materials have been used, these have been acknowledged.

Word Count:

Student Name:

Date of Submission:

Signature:

Abstract

With the recent explosion of connected musical devices, the challenge of making these play "in time" with each other mounts against application developers and device manufacturers. Ableton Link[4] aims to provide robust, resilient musical synchronization using principles from distributed systems programming, in contrast to previous master/slave approaches. However the evaluation criteria for such a system are not well represented in the existing literature, with particular reference to a musical context. The following presents a system for empirical testing of the Ableton Link library using the Jepsen[6] testing framework, along with a set of criteria for evaluting similar libraries that may be developed in future.

Contents

1 Introduction

The synchronizing of events is fundamental to our perception of musical performance. Where performers are using networked connected devices, the challenge maintaining accurate synchronization incorporates the well studied problem of clock synchronization from distributed systems.

As network technologies continue to grow in usage and importance to musical performances[8], it becomes increasingly important to find common approaches to allow devices and applications to synchronize, without reliance on expensive proprietary technology or protocols which are difficult to implement and configure. The need to purchase or to understand such equipment creates an unnecessary barrier to entry, potentially impeding important contributions from musicians who lack the necessary financial or educational resources.

Music programming environments in the academic space are well catered for with regards to open source software, however applications in the consumer market have tended to lag behind some of these advances. Often either the production of music is limited to a single device, or additional devices are synchronized using specialized hardware dedicated to synchronizing frames e.g. MIDI, SMTPE - see Goltz[4] for a review of these methods.

1.1 About Ableton Link

Ableton Link aims to address some of these issues by supplying an open source, permissively licensed C++ library for integration with application code. As well as the popular digital audio workstation Ableton Live, implementations exist for a large number of mobile applications and for many of the popular music programming environments. In adition the following three design goals are stated[4]:

- Remove the restrictions of a typical master/client system
- Remove the requirement for initial setup
- Scale to a wide variety of music applications

It differs from existing approaches in that it does not rely on a master process to propagate timing information directly. Instead nodes will establish a session, using a reference to the start time of the oldest member of the group even if that member is no longer present.

Clock synchronization is performed using a Kalman filter[2] which adds a level of robustness to jitter introduced by the network along with a more

accurate reflection of the real delay. This approach is relatively sophisticated when compared with others using basic averaging algorithms.

Implementation is handled by application developers who are left to integrate a small API. C++ has widespread support for integration with many popular languages, making this viable for the majority of existing applications.

Setup for the end user is virtually transparent - network discovery takes place automatically on all interfaces. Clock synchronization is performed automatically for nodes joining a session and then at 60 second intervals afterwards. Simple transport commands (start, stop) and tempo changes are also propagated automatically by reliable broadcast.

Failures, drop outs and re-entry are all handled with a model of eventual consistency where last-write-wins takes effect for changes in tempo and transport state.

1.2 Evaluating synchronization for music systems

With the advent of distributed systems for music applications, there are natural questions around how these should be assessed to determine their effectiveness. Traditional criteria from distributed systems research may be useful, particularly around the bounds synchronization, however musical performance doesn't have the same strict requirements as traditional databases, for example.

This means that bounds for synchronization and notions of data consistency may be relaxed if doing so would benefit some other aspect, such as ease of setting up.

1.3 Fundamental problems of music synchronization

To characterize some of the challenges more specifically, the following list covers some of the key criteria for success:

- Getting clocks in sync [1, Chapter 6.3.2]
- Keeping them in sync (in the presence of drift or variable latency) [1, Chapter 13]
- Network bandwidth (ensuring scalability as number of devices grows)
- Fault tolerance
- Ease of setup and deployment

2 Background research

The use of networks in computer music is an active area of study, with much of the research being driven by "laptop orchestras"[11] centered around academic institutions. This has led to approaches centering around the Open Sound Control protocol[12][8][9] as the "lingua franca" of connected musical applications, although older methods include the use of MIDI, SMPTE and other standards (see [4]).

Ableton Link uses a custom network protocol but all networking is handled via private methods within the library, meaning that application developers choosing to implement it do not need to concern themselves with network level code.

2.1 Finding bounds - the limits of human perception

In order to determine some lower and upper bounds on the level of synchrony required, data gathered around musical perception will be useful. While the consensus is not complete[5], one can assume that musical performers can tolerate up to 40ms of latency between sources. This is of course dependent on the individual performer as well as other factors such as the frequency domain of the sound.

In addition to delays that impact live performers, any delay introduces the risk of comb filtering effects on sound from multiple sources. This occurs when the frequencies from once sound source reinforce or cancel out those from another sound source.

Finally, the accurate synchronization of multiple speakers is essential for the use of sound spatialization effects, such as stereo panning or surround sound.

With these in mind, the lower bound would ideally be zero (perfect synchronization) but this is unlikely in practice. Delays of 0.05ms could theoretically introduce comb filtering effects in the audible frequency range at 10kHz[7] so this may offer a more practical lower bound.

In terms of an upper bound, 40ms of delay would appear to be the upper limit in terms of the impact on the majority of listeners. Where human performers are involved, this is more likely to impact their output so a lower figure of 20ms may be more appropriate.

2.2 Prior testing approaches

LANdini[9] was designed for use with a specific laptop orchestra in mind and was "tested" in rehearsal and performance. Functional testing is also described in their paper regarding message delivery and bandwidth usage.

A more rigorous testing scheme is proposed in the development of PiGMI[10] (The Raspbery Pi Global Metronome) in which metronome pulses were produced by the synchronized device at 120 pulses per minute for a duration of 30 minutes. The output is then recorded into separate channels on a sound card and later analyzed to determine offsets. In addition to this, commercial drum machines were also measured, synchronized using MIDI time clock, to add a benchmark.

This method of testing allows for excellent accuracy measuring "time at speaker" which is arguably the most realistic metric available; However, the recording and analysis appear to be labour intensive and the reliance on physical hardware makes replication of results more challenging.

A similar result may be achieved using virtual audio inputs on a single machine e.g. using the Soundflower application on OS X, however these are subject to their own sources of latency and solution that relies on the system clock would not be able to detect drift relative to real time if additional reference time sources are not used also.

3 Criteria and design

The testing approach taken in this work chooses not to measure the audio output but instead analyzes the reported times from each of the application nodes for each beat at the current tempo. This allows for simple reproduction of results and a more flexible way to test different failure modes and network conditions.

This is achieved using the Jepsen framework[6], which was originally designed to test safety guarantees under fault injection, however it is also flexible enough to test systems with less strict guarantees.

The framework starts 5 instances running a simple Ableton Link application inside Linux containers using Docker. In addition to these, a control node is started which invokes and records operations against the existing nodes. Primarily this is concerned with read and write operations against the tempo parameter of the Link session. Each node is also logging it's status at each beat. These are then analyzed following the test to calculate the offsets and convergence of events following changes in tempo.

The Jepsen framework also allows the control node to introduce faults into the virtual network between the nodes. Under initial conditions they are all connected, however links between nodes can be cut (packets between nodes all dropped) to form different topologies. Variable or fixed latency can also be introduced. These take place via a process called a "nemesis" in the framework terminology.

3.1 Consistency in a musical context

The introduction of faults and pertubations to the network can cause nodes to receive updates late, resulting in divergence for the beat markers in the session. Provided the network remains connected these will recover eventually.

This leads to an interesting question of how to quantify the divergence and the resulting effect on the music. This work puts forward that Ableton Link and similar eventually consistent systems in future should measure "musical consistency" as the time spent in agreement relative to the length of the overall session.

Another key metric would appear to be the duration for periods of divergence. If these can be minimized the resulting adverse impact on the musical performance can be limited.

3.2 Synchronization Accuracy versus Bandwidth

In order to combat clock drift, some synchronization systems will allow a high frequency of synchronization events to ensure greater accuracy. This comes at the expense of more messages being sent over the network. For example, the LANdini project opts for 3 synchronization messages (pings) every second.

For comparison, recent work by Geng et al[3] achieves synchronization on the order of 10s of nanoseconds within datacenters, however around 5 MBits/s of bandwidth is used to achieve this result.

Given that Ableton Link targets a mass market, the use of consumer grade routers should be assumed. This makes it important to minimize the number of messages sent by the protocol to avoid overloading the hardware, while striving for the figures set out in tolerable latency above.

3.3 Use of Jepsen, Docker and log parsing

While Jepsen is primarily designed to exercise safety guarantees of distributed systems under partitions, the flexibility allows for different kinds

of tests to be performed. This work opts mainly to use Jepsen to handle the running of tests and fault injection, whereas the analysis of results takes place in separate scripts.

The use of Docker containers allows straightforward reproducbility on development machines. One possible application would be in a continous integration testing environment so that changes to the Link codebase could be exercised against these tests automatically.

As the Link protocol favours transparent setup over configuration, there is a limited about of information available regarding the state of the sesssion by default. For these tests the debug logging is enabled and these logs are then parsed following the test to produce the output in the results section below.

3.4 Topologies and latencies

As part of the stated aim of ease of setup, the Ableton Link protocol performs service discovery and message broadcast on all network interfaces to ensure that all connected nodes are able to join the same session. For example, nodes using a LAN may also see other nodes on a WiFi network provided that at least one node existed that was connected to both.

This feature introduces the prospect of topologies other than a connected network in practice. Jepsen allows different topologies to be defined and implemented during the test procedure. As Link operates using reliable broadcast (retransmitting the state of the session on all interfaces when a valid state update is received) the topologies may be ranked according to the length of the maximum path length for an update to be propagated.

- Connected (easy)
- Bridge (medium)
- Line (hard)

Another test condition concerns the performance of the protocol under small, constant network delays. In the tests below this is defined as 100ms delays, which for clock synchronization messages results in a round trip time delta of +200ms.

Finally the protocol is exercised in the presence of large, variable or uneven network delays. The timings for these are included on the charts below.

4 Experimental results

Results here, maybe detailed graphs in appendix

5 Analysis

Resilient to different network conditions. Usually converges.

Talk about indeterminate divergence

5.1 Identifying and triggering invalid states

6 Further work

Generalize the work to other solutions.

Create CI testing for the Link library.

Suggest improvements to Link - Google nanosecond precision paper. Potential uncertainty window around synchronization events. Configuration/visibility of session information.

Integration into Sonic Pi. Windows compatibility.

7 Conclusions

References

[1] Hagit Attiya and Jennifer Welch. *Distributed computing: fundamentals, simulations, and advanced topics*, volume 19. John Wiley & Sons, 2004.

[2] Aggelos Bletsas. Evaluation of kalman filtering for network time keeping. *ieee transactions on ultrasonics, ferroelectrics, and frequency control*, 52(9):1452–1460, 2005.

[3] Yilong Geng, Shiyu Liu, Zi Yin, Ashish Naik, Balaji Prabhakar, Mendel Rosenblum, and Amin Vahdat. Exploiting a natural network effect for scalable, fine-grained clock synchronization. In *15th USENIX Symposium on Networked Systems Design and Implementation (NSDI 18)*, pages 81–94, Renton, WA, 2018. USENIX Association.

[4] Florian Goltz. Ableton link–a technology to synchronize music software. In *Linux Audio Conference 2018*, page 39, 2018.

[5] Waldo Greeff. *The influence of perception latency on the quality of musical performance during a simulated delay scenario*. PhD thesis, University of Pretoria, 2016.

[6] Kingsbury, Kyle. Jepsen - a framework for distributed systems verification, with fault injection, 2018. [Online; accessed 6-August-2018].

[7] Michael Lester and Jon Boley. The effects of latency on live sound monitoring. In *Audio Engineering Society Convention 123*. Audio Engineering Society, 2007.

[8] Sebastian OH Madgwick, Thomas J Mitchell, Carlos Barreto, and Adrian Freed. Simple synchronisation for open sound control. In *Volume 2015*, page 218:225. International Computer Music Association, 2015.

[9] Jascha Narveson and Dan Trueman. Landini: a networking utility for wireless lan-based laptop ensembles. In *Proc. SMC Sound, Music and Computing Conference*, 2013.

[10] Reid Oda and Rebecca Fiebrink. The global metronome: Absolute tempo sync for networked musical performance. In *Proceedings of the International Conference on New Interfaces for Musical Expression*, volume 16 of *2220-4806*, pages 26–31, Brisbane, Australia, 2016. Queensland Conservatorium Griffith University.

[11] Dan Trueman. Why a laptop orchestra? *Organised Sound*, 12(2):171–179, 2007.

[12] Matthew Wright. Open sound control: an enabling technology for musical networking. *Organised Sound*, 10(3):193–200, 2005.

A Self assessment

B Running the tests

C Configuring Jepsen to support other systems and tests

D Code listings

E Professional issues

F Acknowledgements

Open source code Colleagues at Heroku Salesforce for funding Gregory and staff at RHUL Em