CMPO 385 | Major Project Write-up

Mikey Lemmon
25 October 2020

A Latency-Native Virtual Space For Networked Audio/Visual Performance Over The Internet

For my major project, I built a web-based browser application and a WebSocket server to create a three-dimensional virtual performance space where users can collaborate over the Internet to create live audio/visual performances. A primary goal of the project was to evaluate if quantising latency to a shared beat could be an effective way to enable rhythmic accuracy in networked musical collaborations in high-latency conditions. A set of preconfigured instruments are implemented in the browser-based client application that generate both audio output and visual elements in the

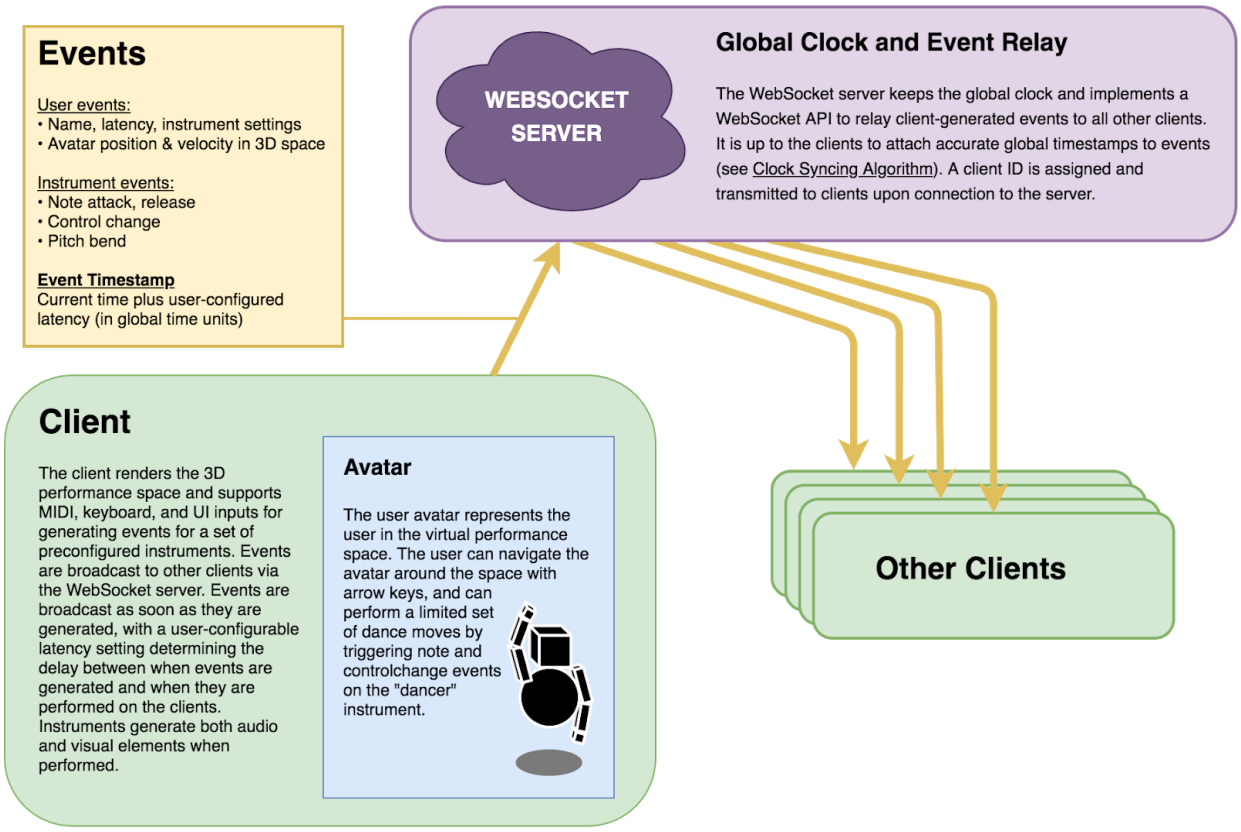

Figure 1. *High-level system architecture*

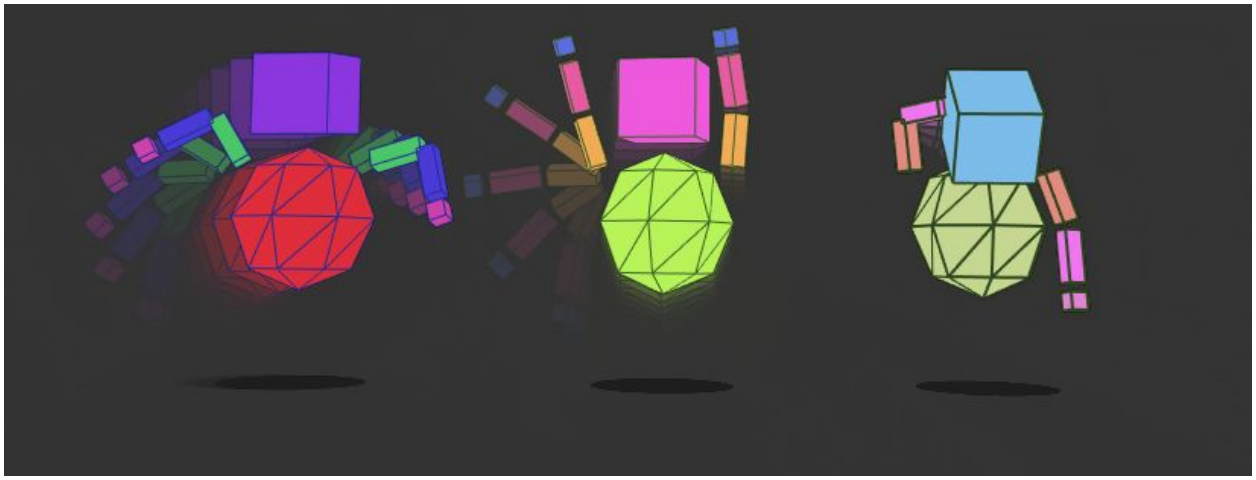

Figure 2. *Avatars performing movements controlled by the 'dancer' instrument*

virtual space, and support MIDI, keyboard, and UI input for triggering note events, as well as control signals events to modify instrument properties such as attack, decay, sustain, and release settings. Server-generated metronome events and a client-side clock syncing algorithm (see Appendix) are used to align clients to a shared global time context, combing with a beat-based latency control to enable performances to be synchronised to a shared beat even when distributed across the globe in high-latency conditions.

Users are represented in the virtual space by an avatar object that they can control with keyboard arrow keys to navigate around the space. A 'dancer' instrument (the default instrument for new users) responds to the same MIDI and keyboard events as other instruments and allows users to make their avatar perform a range of preset movements such as a body dip, head nod, spin, arm rotation, and elbow bend. When combined, even this limited set of controls can result in a wide range of expressive avatar movements (Figure 2).

To overcome the unpredictable latencies inherent to internet-based networking, the UI includes a latency setting that determines the delay between when an event input is received and when it is performed. To allow enough time for events to be received by all clients, latency must be set to a duration that is longer than the longest amount of time it takes for events to be distributed from any one client to all the others, which even under ideal conditions can take more than 200 milliseconds when users are distributed across continents. To make such large latencies predictable and musically relevant, the latency setting is measured in beats (based on a global BPM

setting), allowing users to more easily align their performance to the common beat.

The client application also includes support for recording events to loops, which automatically re-trigger events according to the loop's duration. Loops are represented as a circular dial, with 'clock hands' indicating both the position in the loop that is being recorded as well as the position that is being performed, allowing the user to better visualise the effect of their latency setting. Note and control events are drawn inside of the dial. A metronome dial, with clock hands but no looped events, is also provided to strengthen the user's orientation to the shared beat and time signature. Loop and metronome dials are shown in Figure 3.

All together, these elements combine to create an application that is something like an unholy combination of Ableton's Session View and Marshmello's Fortnite performance (Figure 4), and seems to have proved out the initial hypothesis that quantising latency to a shared beat can be an effective way to enable musical collaboration in high-latency scenarios. An in-class performance — a dance lesson followed by a short 'rave' — seemed to be enjoyed by the participants, and I hope to use the application for more performances and collaborations in the near future.

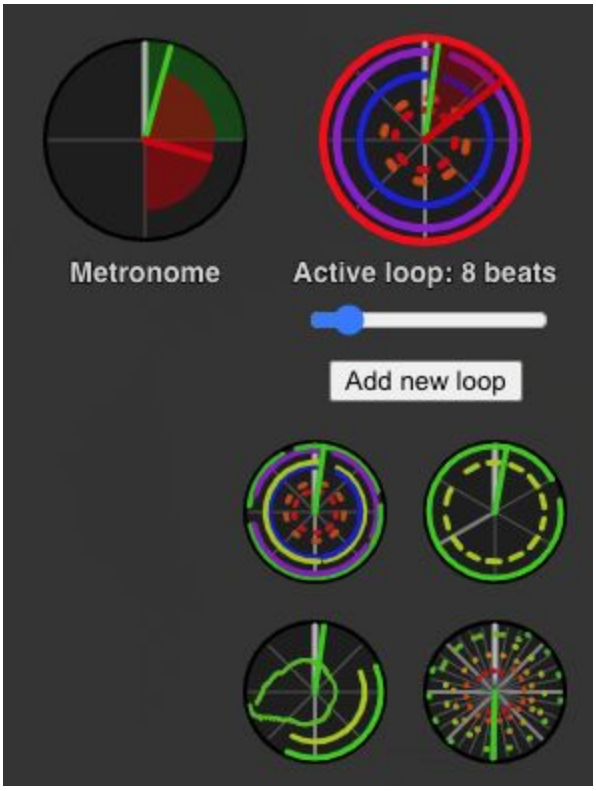

Figure 3. *Loop and metronome dials. Clock hands show the current recording position in red and performance position in green, with the difference between the two visualising the configured latency. Events are being recorded to the 'active' loop, indicated by a red outline around the dial. Each pie-slice in a dial represents one beat.*

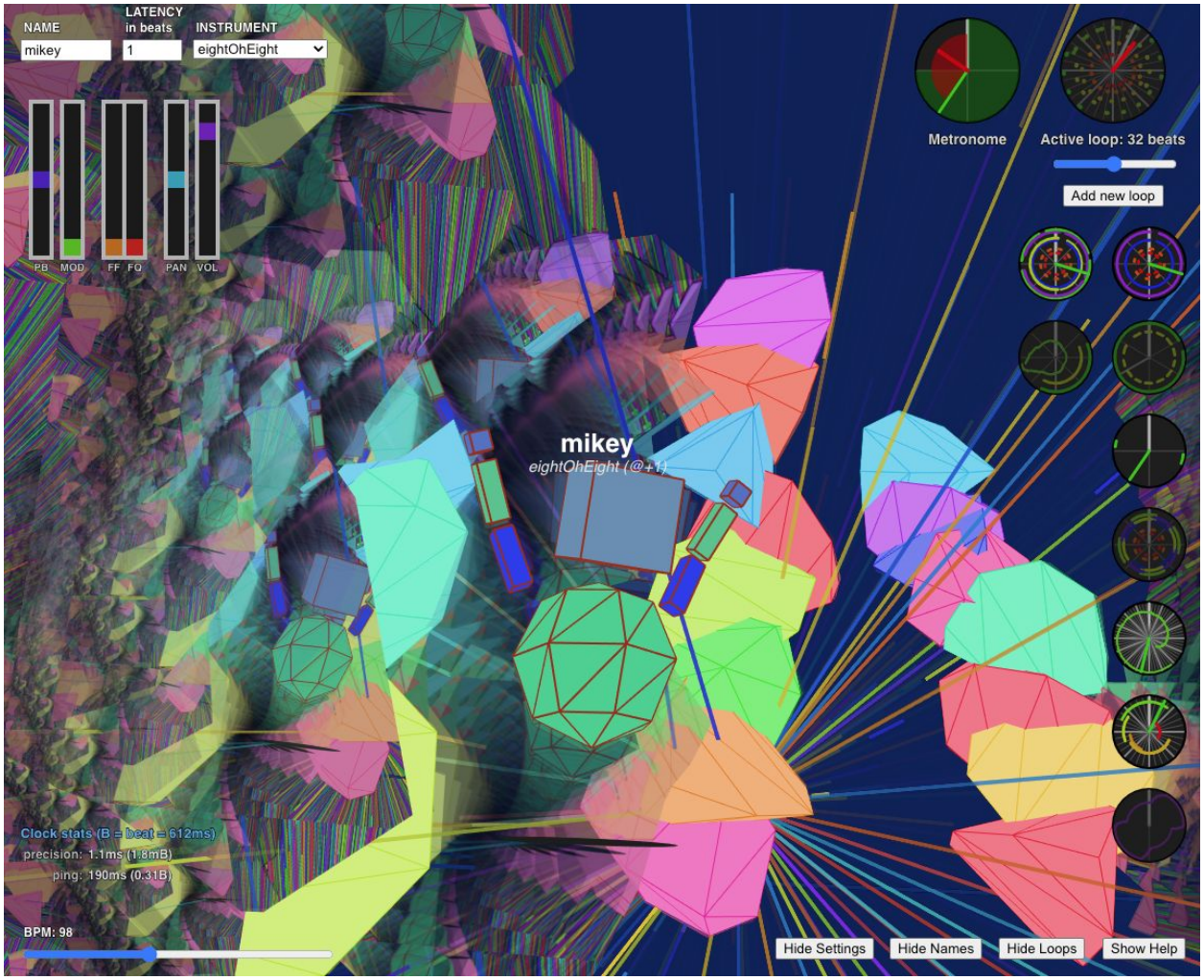

Figure 4. *An in-app 'rave'*

APPENDIX: Clock Syncing Algorithm

All time-sensitive system events (e.g. note attacks and releases) are transmitted to the server, and thus relayed to other clients, with a floating-point timestamp that is measured in milliseconds since the global time origin (the time the server started). A client-side syncing algorithm, loosely inspired by the Network Time Protocol, determines a correction to apply to the client's local time so that a best-guess estimate of the global time can be attached to client-generated events.

At regular intervals, the client requests the current global time from the server. Naively assuming that the time returned from the server represents the global time midway between when the request was sent and the response received, the client uses the difference between the local-time request midpoint and the value received from the server as a first-order correction to be applied to the local time (Figure 5). To account for variation and jitter in network conditions, a low-pass filter is applied to the first-order correction so that the correction used to locally generate best-guess global time estimates is a sliding average of past first-order corrections, weighted toward the most recent value.

If clock drift between the server and client clocks were not an issue, the first-order correction would be sufficient to generate precise global time estimates, but in practice a significant drift was observed (several milliseconds of drift per minute). The drift appears to originate from the browser's implementation of window.performance.now(), which is the only clock available in the main JavaScript context that has sub-millisecond precision. To account for this drift, a second-order correction parameter was added to the syncing algorithm by averaging the difference between subsequent first-order corrections over time.

With the second-order correction applied, testing shows that the global time can be reliably estimated locally to sub-millisecond precision when connecting over a broadband connection to a US-based server (average ping: ~150ms), though occasional transient events can cause precision to degrade momentarily to multiple milliseconds. Precision observed when testing over an LTE mobile connection was much worse, averaging around 20ms when connecting to the same US-based server. Further work is required to determine what adjustments could be made to the syncing algorithm to improve precision for mobile connections, as well as to improve the handling of transient degradation events.

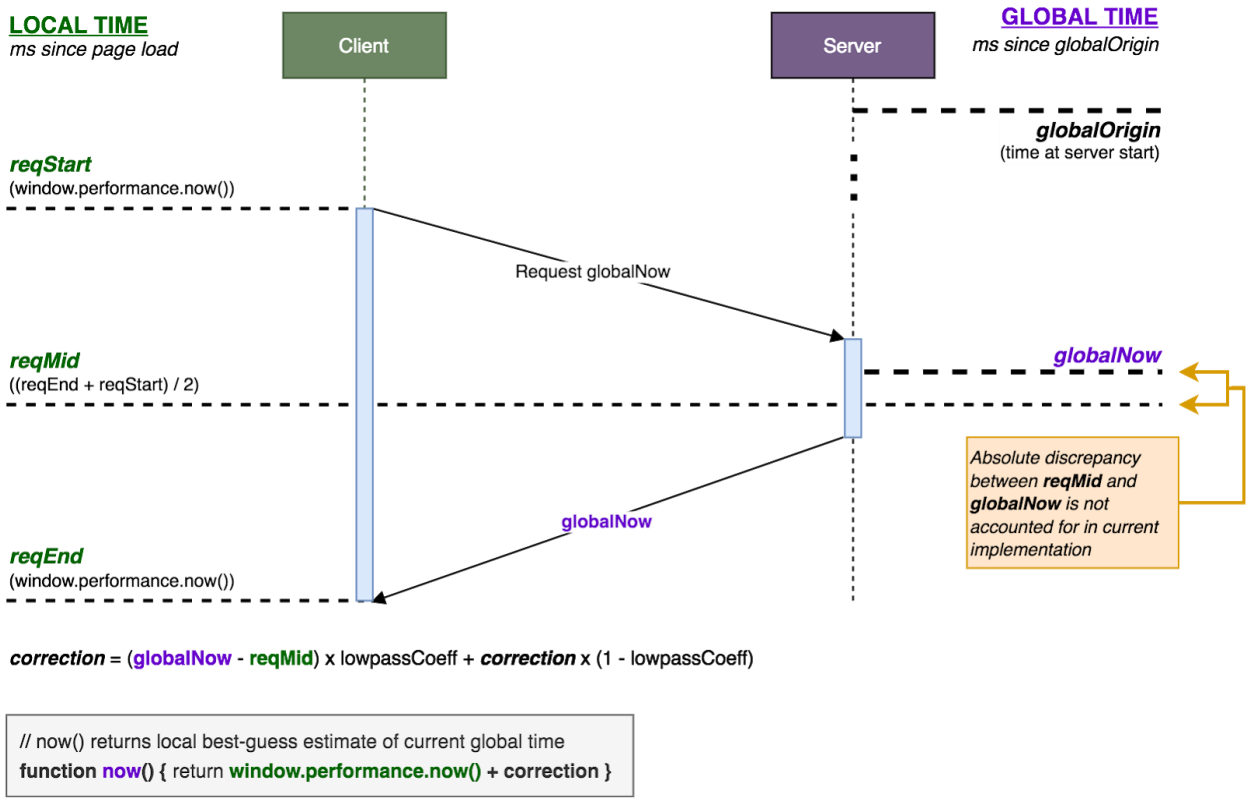

Figure 5. *Computation of a local best-guess estimate of global time based on first-order correction to local time*