Design of a real-time multiparty DAW collaboration application using Web MIDI and WebRTC APIs

Scott Stickland
The University of Newcastle, Australia
School of Creative Industries (Music)
scott.stickland@uon.edu.au

Rukshan Athauda
The University of Newcastle, Australia
School of Electrical Engineering and Computing (Information Technology)
rukshan.athauda@newcastle.edu.au

Nathan Scott
The University of Newcastle, Australia
School of Creative Industries (Music)
nathan.scott@newcastle.edu.au

ABSTRACT

Collaborative music production in online environments has seen a renewed focus as developers of Digital Audio Workstation (DAW) software include features that cater to limited synchronous participation and multiparty asynchronous collaboration. A significant restriction of these collaboration platforms is the inability for multiple collaborators to effectively communicate and seamlessly work on a high-fidelity audio project in real-time.

This paper outlines the design of a browser-based application that enables real-time collaboration between multiple remote instantiations of an established, mainstream and fully-featured DAW platform over the Internet. The proposed application provides access to, and modification and creation of, high-fidelity audio assets, real-time videoconferencing and control data streaming for communication and synchronised DAW operations through Web Real-Time Communication (WebRTC) and Web MIDI Application Programming Interfaces (APIs). The paper reports on a proof-of-concept implementation and results, including several areas for further research and development.

1. INTRODUCTION

Present online collaboration, facilitated by recent DAW-specific and -generic approaches, can be classified as either synchronous or asynchronous in design and operation. Synchronous approaches focus on sharing and editing music data in real-time. For effective and meaningful collaboration, this data consists of high-fidelity audio files and streams. However, current Internet bandwidths struggle to handle the real-time transfer of such data-intensive streaming and maintain the audio's high fidelity. Thus, these collaborations are limited to a few participants at most. Conversely, asynchronous approaches forgo real-time online interactions in preference for increased numbers of remote participants and maintenance of the audio assets' fidelity. Central to asynchronous collaboration is the uploading and downloading of lossless audio files to and from cloud storage linked to the DAW applications, or directly to and from collaborators' local storage drives.

Our previous research examined four widely-available approaches to DAW-integrated online collaboration [1]. Two synchronous collaboration platforms, Source Elements' DAW-generic *Source-Connect Pro* and Steinberg's *Cubase Pro*-specific *VST Connect Pro* and *VST Connect Performer* bundle, not only restrict the number of participants but also access to the DAW project itself [2, 3]. Synchronous collaboration necessarily requires compression of the project's audio streams to reduce the effects of latency and jitter inherent in Internet data transport. Applying lossy audio codecs, though, result in a degradation of the audio's fidelity. The two synchronous platforms do provide optional features to record, store, and stream lossless Pulse Code Modulation (PCM) audio files of a remote performer to replace jitter-affected files post-session.

Data- and bandwidth-intensive audio streaming, and the complication of varying latencies between multiple, simultaneous connections, obliged Avid and Steinberg to approach online collaboration asynchronously, adopting a similar concept to that utilised by Source-Connect Pro's, and VST Connect Pro's, automatic PCM audio upload features. The two asynchronous platforms, Avid's *Cloud Collaboration* and Steinberg's *VST Transit*, are included features of their respective DAWs, *Pro Tools* and *Cubase* [4, 5]. By eschewing real-time collaboration, these approaches significantly increase participant numbers and provide universal access to a DAW project. Employing lossless audio codecs to compress the project's PCM audio files before uploading maximises the available cloud storage capacity, reduces transfer times, and maintains the original audio's high fidelity.

Web browser-based platforms, such as Spotify's *Soundtrap*, AmpTrack Technologies' *Amped Studio 2*, and BandLab Technologies' *BandLab* also offer asynchronous collaboration methods for multiple participants [6-8]. All provide users with the ability to share a DAW project with others, issuing invitations by either co-opting existing social media applications, such as Facebook and Twitter, email and Short Message Service (SMS), or in-application messaging to distribute a link to the project. In Amped Studio 2 and BandLab, sharing a DAW project creates multiple versions, or forks, of the project, one per collaborator, so that the integrity of the original project is preserved [7, 9]. Soundtrap adopts a collaboration model similar to VST Transit and Cloud Collaboration, where users can contribute and save to a cloud version of the project, then sync their local project with the one in the cloud asynchronously.

Nevertheless, transferring large lossless audio files over the Internet takes time and bandwidth, meaning collaborators can only access, listen to, and assess project contributions post factum. Asynchronous collaboration can slow progress in this environment; contributions cannot be auditioned or modified in real-time, so the possibility of revisions, or indeed discarding material altogether, is significantly more likely. Soundtrap's approach to minimising these consequences is to integrate a videoconferencing feature using the WebRTC API [10]. Lind and MacPherson [10] explain that a video chat feature enables 'instant feedback with collaborators as they create projects together in real-time.' It is worth pointing out that while the video chat feature is indeed synchronous, sharing contributions to, and modifications of, a project remain asynchronous.

Web Audio Conference WAC-2019, December 4–6, 2019, Trondheim, Norway.

Georgia Institute of Technology's Centre for Music Technology's *EarSketch* is an amalgam of a browser-based Python and JavaScript Integrated Development Environment (IDE), and a rudimentary DAW for the playback and manipulation of audio samples via scripts [11]. Their recent developments focus on collaborative script editing through the use of an operational-transform algorithm via WebSockets, and time synchronisation utilising Network Time Protocol's (NTP's) clock-synchronisation algorithm [12]. It, however, lacks the requisite processing capabilities for professional mixing and mastering activities.

Currently, mainstream and web browser-based DAW platforms cannot facilitate synchronous, multi-party collaboration and communication, where all participants, irrespective of location, can simultaneously access, edit and contribute to professional, specialised activities such as post-production mixing and mastering. Repositioning such activities as an online collaborative undertaking presents several challenges and requirements.

Latency, Jitter, Bandwidth and High-Fidelity audio assets: Expert post-production activities require the use of high-fidelity audio files. In particular, professional mixing of multitrack recordings for commercial CD and streaming-service release require audio files with a sample rate of 44.1 or 48 kHz, at the very least, to avoid heavy anti-aliasing filtering. Similarly, audio file bit depths of 16- or 24-bit are necessary to reduce the noise floor and create more headroom. Streaming these assets in real-time is challenging considering high-fidelity audio is, by its very nature, bandwidth-intensive. Furthermore, streaming over the Internet utilising reliable transport layer protocols such as Transmission Control Protocol (TCP), increases latency, whereas, the use of 'best-effort service' protocols such as User Datagram Protocol (UDP) can reduce delays (latency) but does not guarantee reliable delivery, resulting in lost packets (jitter).

A benefit of existing asynchronous collaboration methods is the ability to preserve the DAW project's high-fidelity audio files; however, existing synchronous approaches rely upon lossy audio streams [2, 3]. Therefore, a way to access and modify high-fidelity audio files in synchronous multi-party collaborative environments is needed.

Access to, and synchronous operation of, a DAW platform: Existing asynchronous DAW-specific multi-party solutions grant each participant equal access to the music production project via their local DAW instantiation. Providing such equal access to a collaborative, specialised DAW platform in real-time, though, is currently lacking.

Real-time communication methods: Cloud Collaboration, VST Transit, Amped Studio 2 and BandLab, while catering for multiple collaborators, only provide asynchronous text messaging communication [4, 5, 7, 8]. While the VST Connect Pro/Performer bundle and Soundtrap include effective real-time video streaming, the communication is just peer-to-peer (P2P) [2, 6]. Videoconferencing is highly desirous in a multi-party, real-time collaboration environment.

The design of the DAW collaboration application presented in this paper aims to address the challenges and requirements outlined above.

2. THE COLLABORATION FRAMEWORK

Figure 1 illustrates the infrastructure and data flow of a proposed music production collaboration framework.

Each collaboration endpoint runs a local DAW instantiation. Every participant downloads and opens a previously uploaded collaborative DAW project in cloud storage to each collaborator's DAW instantiation before the collaboration session. The Signalling Server negotiates and instigates the WebRTC peer connections, while the Media Server is used to synchronise and manage the real-time communication and MIDI control data streams among multiple collaborators. The following sections discuss the architecture in detail.

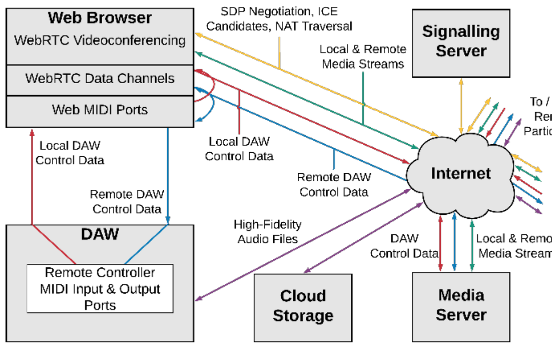


Figure 1. The proposed collaboration framework's architecture and data flow.

The proposed framework (see Fig. 1) avoids high-fidelity audio streaming for the collaboration project's playout and source of new audio material. The framework employs cloud storage and file sharing to distribute the DAW project and its audio assets to all participants before a collaboration session begins. The participants employ their local DAW instantiation's playback for monitoring the project's audio. Streaming lower-volume timecode facilitates collaborative synchronous operation and aligns the playback of remote DAW instantiations' locally-stored high-fidelity audio assets. While not eradicating latency, this approach dramatically reduces its effects by avoiding bandwidth-intensive, high-fidelity audio transmissions in real-time. Each participant is ignorant of the slight differences in timing across the collaboration.

2.1 Achieving synchronicity

DAW instantiations are capable of synchronising their playback utilising three mechanisms: machine control, clock source and timecode [13]. Together, they deliver transport commands, positional references in time, and speed references. The MIDI paradigm includes MIDI Timecode (MTC) and MIDI Machine Control (MMC) protocols. MTC has the additional benefit of functioning as a clock source; therefore, MTC and MMC together institute synchronised playback by establishing a master/slave configuration across the collaboration (see Fig. 2).

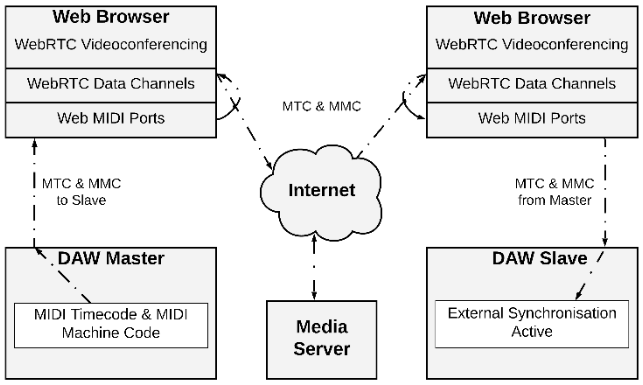


Figure 2. Synchronised playback of DAW instantiations through the streaming of MTC and MMC.

Incumbent on the framework is providing an equitable and inclusive ability to edit the collaboration project and its audio assets. All of the collaboration's DAW instantiations must also execute the on-screen modifications, made by any individual end-user to their local DAW project, in real-time. Doing so provides synchronised editing and navigation across the collaboration. We have chosen Cubase Pro 10 to be the framework's DAW platform, as discussed in the next section.

2.2 Cubase Pro 10

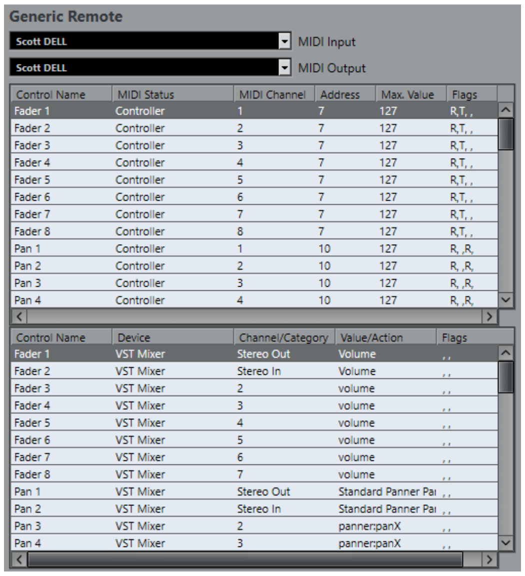
The rationale for choosing Cubase Pro 10, beyond its calibre as a professional, industry-standard DAW for post-production activities, includes the ability to create user-defined MIDI command maps implementing its *Generic Remote* feature, the ability to create user-defined keyboard shortcuts (*Key Commands*) and its MTC and MMC external synchronisation capabilities.

2.2.1 Generic Remote

The Generic Remote (GR) feature allows users to tailor the operation of a generic MIDI controller to most any of Cubase's functions [13]. For our purposes, the GR provides significant utility for transmitting and receiving MIDI control data mapped to the DAW's functions, navigation and transport. The GR creates bespoke MIDI maps, linking specific MIDI Continuous Controllers (CCs) and notes to DAW commands and operations (see Fig. 3). Users can assign the GR's MIDI input and output ports from a list of the computer's available MIDI devices.

2.2.2 Key Commands

Keyboard shortcuts are a common feature of many software applications, designed to enhance productivity by reducing the number of mouse moves and clicks. Cubase Pro includes numerous keystroke combinations linked to DAW operations, including an increasing number of homogeneous DAW-generic combinations. Cubase Pro's key commands can be tailored to map specific keystroke patterns to almost any DAW function or operation.



Generic Remote

Scott DELL — MIDI Input

Scott DELL — MIDI Output

Control Name	MIDI Status	MIDI Channel	Address	Max. Value	Flags
Fader 1	Controller	1	7	127	R,T, ,
Fader 2	Controller	2	7	127	R,T, ,
Fader 3	Controller	3	7	127	R,T, ,
Fader 4	Controller	4	7	127	R,T, ,
Fader 5	Controller	5	7	127	R,T, ,
Fader 6	Controller	6	7	127	R,T, ,
Fader 7	Controller	7	7	127	R,T, ,
Fader 8	Controller	8	7	127	R,T, ,
Pan 1	Controller	1	10	127	R, ,R,
Pan 2	Controller	2	10	127	R, ,R,
Pan 3	Controller	3	10	127	R, ,R,
Pan 4	Controller	4	10	127	R, ,R,

Control Name	Device	Channel/Category	Value/Action	Flags
Fader 1	VST Mixer	Stereo Out	Volume	, ,
Fader 2	VST Mixer	Stereo In	Volume	, ,
Fader 3	VST Mixer	2	volume	, ,
Fader 4	VST Mixer	3	volume	, ,
Fader 5	VST Mixer	4	volume	, ,
Fader 6	VST Mixer	5	volume	, ,
Fader 7	VST Mixer	6	volume	, ,
Fader 8	VST Mixer	7	volume	, ,
Pan 1	VST Mixer	Stereo Out	Standard Panner Pa	, ,
Pan 2	VST Mixer	Stereo In	Standard Panner Pa	, ,
Pan 3	VST Mixer	2	panner:panX	, ,
Pan 4	VST Mixer	3	panner:panX	, ,

Figure 3. Cubase Pro 10's Generic Remote page.

2.3 Control data streaming

One of the framework's significant efficiencies must be its ability to effectively and reliably stream MIDI control data to all participants over the Internet. Successful tests of Cubase Pro's GR MIDI mapping utilised the OSX MIDI network driver on Mac and Tobias Erichsen's *rtpMIDI* driver software [14] on Windows computers to establish remote network connections between MIDI ports. Employing the RFC 6295 Real Time Protocol (RTP) payload format for MIDI messages (RTP-MIDI), the drivers map MIDI 1.0 data onto RTP streams over UDP [15]. These ports were assigned to their corresponding GR's input and output ports (see Fig. 4), and each GR instance was configured identically by importing a mutual XML mapping file.

RFC 768 UDP [16] is an inherently best-effort transport service that is suited to real-time transmissions. UDP lacks reliable data transfer characteristics. Nevertheless, RTP offered a degree of reliability through error correction and concealment strategies to deal with lost packets when combined with the Real-Time Control Protocol (RTCP) [17].

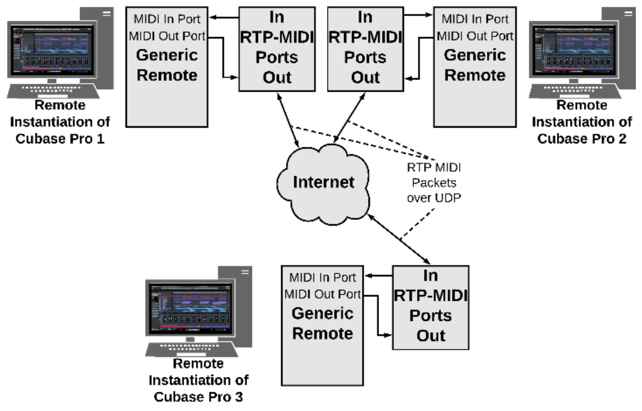


Figure 4. Using connected network MIDI ports over the Internet to test Cubase Pro's Generic Remote interactivity.

3. WEBRTC AND WEB MIDI APIs

This section outlines the use of Web RTC and Web MIDI APIs to implement the proposed architecture. In order to implement the framework, the application needs to: (a) access a collaborator's computer webcam and microphone; (b) create secure connections between the online participants; (c) provide a secure videoconferencing capability; (d) gain access to MIDI ports assigned to Cubase Pro's GR; (e) create secure, semi-reliable, configurable data channels between the online participants; and (f) route MIDI control data to and from Cubase Pro and the data channels for synchronous streaming over the Internet. Exploiting the WebRTC and Web MIDI APIs can achieve all of these requirements.

3.1 WebRTC

3.1.1 getUserMedia method

The `getUserMedia()` method is one of the most common ways to access local webcam and microphone media devices, thus creating a local media stream [18]. In the interests of privacy, only once a user gives permission, the browser can access the local media devices.

3.1.2 RTCPeerConnection API

WebRTC's `RTCPeerConnection` interface is its fundamental basis and establishes a connection between two endpoints, or peers, over the Internet. Once established, the connection provides direct bidirectional P2P communication without requiring an intervening server. Reducing the distance data needs to travel similarly reduces the latency it incurs. For the interface application, the `RTCPeerConnection` can facilitate both media and data flow between collaborators.

3.1.3 MediaStream API

A media stream comprises of two tracks, one each for video and audio, with each track comprising of one or more channels; for example, a stereo audio track consists of separate left and right channels. The `MediaStream` interface creates an object by grouping the local media tracks, thus defining each participant's media stream. The flow of `MediaStream` objects over an `RTCPeerConnection` facilitates the collaboration framework's videoconferencing.

3.1.1 RTCDataChannel API

The WebRTC `RTCDataChannel` interface creates an additional bidirectional channel over an `RTCPeerConnection` for the simultaneous transmission of arbitrary data with similarly low latency and high throughput [19]. The `RTCDataChannel` interface was modelled closely on the WebSockets API, and consequently, their methods (e.g. `send()`) and handlers (e.g. `onmessage`) behave similarly [18]. The transmission between participants of the framework's MIDI control data occurs over these data channels.

3.2 Web MIDI

3.2.1 MIDIAccess API

Web MIDI's `MIDIAccess` interface supplies methods to list the MIDI input and output ports available to the browser and provide access [20]. The `navigator.requestMIDIAccess()` method and `onMIDISuccess` handler allows the framework's interface application, and by extension the participants, to nominate input and output ports that correspond to Cubase Pro's GR ports, consequently establishing the crucial link between Cubase Pro and the interface application.

3.2.2 MIDIInput and MIDIMessageEvent APIs

The success of the browser-based application to act as an interface between Cubase Pro and the larger collaboration infrastructure depends upon bidirectional transfer of MIDI control data to and from a DAW's GR MIDI ports, to and from a created `RTCDataChannel`.

The `MIDIMessageEvent` interface achieves one direction when passing an event object, in this case, a MIDI 1.0 message, to a `MIDIInput` port's `onmidimessage` handler upon receiving control data from the GR output port. Each event object consists of a Uint8array comprising MIDI message data bytes and a high-resolution timestamp [20] and is transmitted via the `send(event)` method on the `RTCDataChannel` (see Fig. 5).

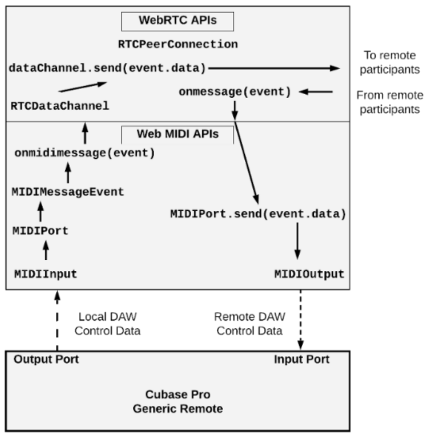


Figure 5. The interface application's use of WebRTC and Web MIDI APIs, methods and handlers.

3.2.3 MIDIOutput API

The Web MIDI `MIDIOutput` interface and its `send()` method realise the application's other directional interfacing, together with the `RTCDataChannel` interface's `onmessage` event handler. The data channel's `onmessage` handler receives a control data event object, which transmits the MIDI data bytes and timestamp over the `MIDIOutput` interface's MIDI port (see Fig. 5).

3.3 Virtual MIDI ports

The most productive way to link the GR and corresponding interface application input and output ports are via virtual MIDI ports. For the tests conducted so far, a third-party MIDI driver, *LoopBe30* by nerds.de [21], has created ports for the internal connections between Cubase Pro and the interface. However, plans are for the application to feature its own virtual MIDI driver and ports in the future.

3.4 Stream Control Transmission Protocol (SCTP)

WebRTC data channels utilise the RFC 4960 Stream Control Transmission Protocol (SCTP) for their implementation and delivery. Johnston and Burnett state that SCTP "provides useful features not available in TCP, including reliable or semi-reliable delivery, non-ordered delivery of packets, multiple streams within an SCTP association, and an ability to send messages." [18]. Of particular importance to the collaboration framework is SCTP's semi-reliable and non-ordered delivery of data packets, in addition to the `RTCDataChannel` interface's `maxPacketLifeTime` and `maxRetransmits` unsigned shorts and `ordered` Boolean attributes [22]. Further testing and tuning are needed to determine optimal values for such parameters.

4. CONNECTION ARCHITECTURES

While WebRTC is primarily designed to establish a P2P connection, three different approaches can create multi-party collaborations: (a) Mesh, (b) Mixing, and (c) Routing [23].

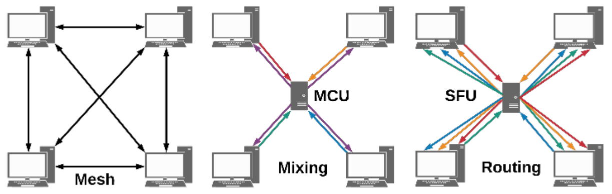


Figure 6. The mesh, mixing and routing architectures.

4.1 Mesh

As the name suggests, each participant constructs the mesh architecture by establishing a peer connection with every other participant (see Fig. 6). While it is relatively simple to implement and requires no backend infrastructure, it is limited in its ability to scale to a large number of participants and is CPU- and bandwidth-intensive as the number of participants increases [24].

4.2 Mixing

A mixing architecture requires the integration of a Multipoint Control Unit (MCU) into the multi-party architecture. Its construction involves each participant establishing a peer connection with the MCU only (see Fig. 6). It is the MCU's task to receive and mix the individual media streams, then send the mixed-media stream to the participants [24]. Each endpoint assumes it is interacting with another single endpoint.

4.3 Routing

A routing, or relay, architecture requires the integration of a Selective Forwarding Unit (SFU) where each participant establishes a peer connection with the SFU only (see Fig. 6). Unlike an MCU, an SFU forgoes transcoding of the media streams, instead deciding which of the media streams to forward on to the participants [23]. Each participant receives the routed media and data streams of all other participants in the collaboration.

5. PROTOTYPE IMPLEMENTATION

This section describes the results of a prototype implementation of the DAW collaboration application. The prototype implementation utilises a mesh architecture (which is the simplest to implement) to interact with peers. Although mesh architecture is not scalable to a large number of participants, the prototype demonstrates the feasibility of the proposed application. Table 1 summarises the resources used in the implementation.

Table 1. Resources for the P2P Mesh Test.

Computer 1 (Peer 1, Signalling Server)	i5-7300U 2.6 GHz CPU; 16 GB RAM; Windows 10 Enterprise OS
Computer 2 (Peers 2, 4, 6, 8)	i7-8700K 3.7 GHz CPU; 32 GB RAM; Windows 10 Pro OS
Computer 3 (Peers 3, 5, 7)	i7-3610QM 2.3 GHz CPU; 12 GB RAM; Windows 7 Home Premium OS
LAN Speed	1 Gbps
Browser	Chrome 75
DAW	Cubase Pro 10
Application-Layer Protocol	HTTPS
Signalling Server	Built using socket.io on node.js

The application's media and data streams were stable up to and including the addition of the sixth peer, though the time taken for the signalling process to establish each connection was noticeably longer with each new addition. While there was a perceptible increased latency in the playback and execution of functions across the DAW instantiations, this perception was only due to having all three computers in the one physical space.

The addition of the final two peer connections, however, saw a marked deterioration in stability, including frozen video streams, increasingly distracting audio stream jitter, and a progressive lack of DAW responsiveness and extended delays in executing data-heavy functions such as level fader operations. Figure 7 plots Computer 1's transmission and reception rates of WebRTC-related packets.

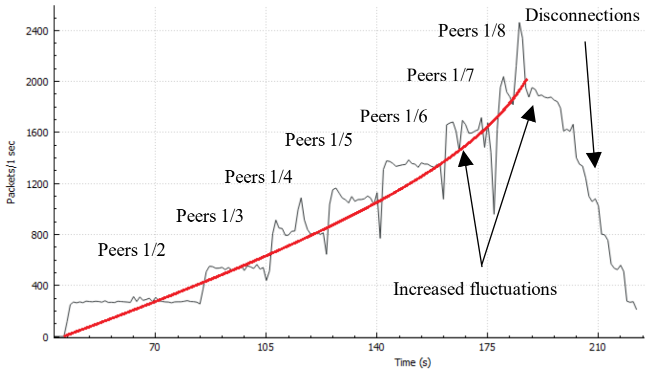


Figure 7. Peer 1 packets transmitted and received per sec.

Latency across the mesh architecture increased exponentially as each new peer joined the collaboration, as demonstrated by Figure 8's plot of Peer 1's packet delivery times. With a single P2P connection, the average delivery time was 0.47 ms and increased by 0.18 ms and 0.15 ms with the addition of the next two peers, respectively. However, the addition of connections to Peers 7 and 8 saw increases of 0.82 ms and 2.09 ms, respectively.

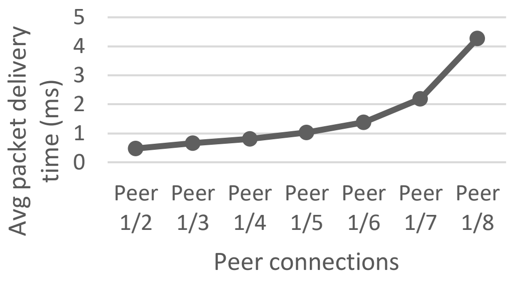


Figure 8. Peer 1 connections: average packet delivery times.

Figure 9 plots the percentage of Computer 1's overall CPU capacity utilised by the framework's three processes, namely Chrome.exe, Cubase10.exe, and Node.exe. At peak consumption, which coincided with 15 mesh connections and data-intensive Cubase Pro 10 operation, the processes accounted for 66.43% of the 2.6 GHz CPU. At the same time, the computer's overall percentage of CPU usage measured just over 93%.

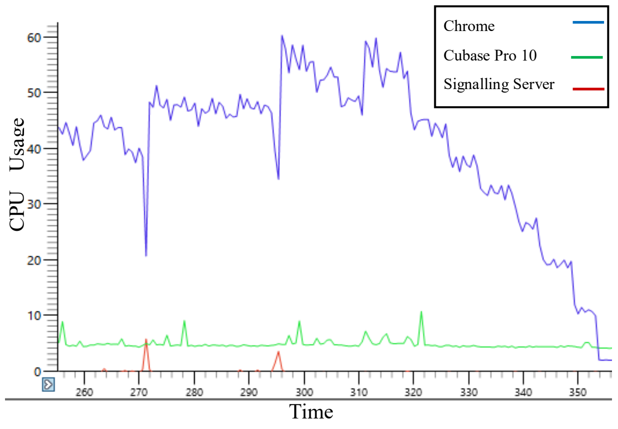


Figure 9. Peer 1 CPU usage percentage: Chrome Browser (blue), Cubase Pro 10 (green), Node Signalling Server (red).

6. CONCLUSION AND FUTURE WORK

This paper presented the design of a browser-based DAW collaboration framework that aims to provide multi-party real-time collaboration. The framework addresses many of the shortcomings of existing approaches – including access to high-fidelity audio assets by collaborators, equal access to a DAW project, multi-party real-time video-conferencing and others. The paper also outlined a prototype implementation using Web RTC and Web MIDI APIs as a proof-of-concept. The results are promising.

Future work will determine the most reliable and timely delivery methods, having demonstrated the ability to replace the transport of RTP-MIDI packets over UDP with MIDI bytes and timestamps over SCTP, via WebRTC data channels, successfully.

At present, Cubase Pro's GR feature limits the transmission of MIDI control data to commands and functions commonly featured on mainstream control surfaces. This restriction is due to control surfaces requiring feedback from the DAW reflecting on-screen execution operations, such as Mixer Console and Transport functions, Insert, Send and VST instrument plug-in parameters and channel-strip EQ settings. Future testing will encompass the use of keyboard commands with keystroke-to-MIDI translation to address the shortcomings of the GR's range of functions.

The architecture has been implemented only in a limited scenario and with mesh architecture only. Future work will implement and analyse results of mixing and routing architectures and their scaling capabilities through the inclusion of a media server. Testing will also expand to include implementation over the Internet to measure latencies and determine an acceptable delay threshold.

The MIDI 2.0 protocol, upon its release and mainstream integration, could provide enhancements to the transport and delivery of the application's MIDI control data. Information published by the MIDI Manufacturers Association (MMA) has signposted an increase in the resolution of control messages from 7 bits up to 32 bits, and MIDI packets will include a jitter reduction timestamp to improve timing accuracy [25]. Future work will integrate the MIDI 2.0 messaging protocol to determine the scale of improvement in the transport of data and the accuracy of received data streams.

7. ACKNOWLEDGMENTS

This research is supported by the Australian Government Research Training Program Scholarship.

8. REFERENCES

[1] Stickland, S., Scott, N. and Athauda, R. 2018. A Framework for Real-Time Online Collaboration in Music Production. In *Proceedings of the ACMC2018: Conference of the Australasian Computer Music Association* (Perth, Australia, 6-9 December, 2018).

[2] Steinberg Media Technologies GmbH. 2018. VST Connect Pro. Retrieved 6 May 2018 from https://www.steinberg.net/en/products/vst/vst_connect/vst_connect_pro.html

[3] Source Elements. 2018. Source-Connect. Retrieved 20 June 2018 from http://source-elements.com/products/source-connect

[4] Steinberg Media Technologies GmbH. 2018. VST Transit: The World of Music Cloud Collaboration. Retrieved 25 July 2018 from https://www.steinberg.net/en/products/vst/vst_transit.html?et_cid=15&et_lid=22&et_sub=VST%20Transit

[5] Avid Technology Inc. 2018. Avid Cloud Collaboration for Pro Tools: How It Works. Retrieved 5 June 2018 from http://www.avid.com/avid-cloud-collaboration-for-pro-tools/how-it-works

[6] Spotify AB. 2019. Soundtrap - Make music online. Retrieved 9 September, 2019 from https://www.soundtrap.com/

[7] AmpTrack Technologies AB. 2019. Amped Studio 2 | Online Beatmaker and Music Studio. Retrieved 9 September, 2019 from https://ampedstudio.com/

[8] BandLab Technologies. 2019. BandLab: Music Starts Here. Retrieved 9 September, 2019 from https://www.bandlab.com/

[9] Guitar Player Magazine. 2017. BandLab Collaborative App @ NAMM 2017. Video. (31 January, 2017). Retrieved 9 September, 2019 from https://www.youtube.com/watch?v=hYaOZll999g

[10] Lind, F. and MacPherson, A. *Soundtrap: A collaborative music studio with Web Audio*. Queen Mary Research Online, City, 2017.

[11] Web Audio Conf. 2018. Collaborative Coding with Music: Two Case Studies with EarSketch by Avneesh Sarwate. Video. (28 September, 2018). Retrieved 10 September, 2019 from https://www.youtube.com/watch?v=0qBVSCRpggg

[12] Sarwate, A., Tsuchiya, T. and Freeman, J. 2018. Collaborative Coding with Music: Two Case Studies with EarSketch. In *Proceedings of the Web Audio Conference 2018* (Berlin, Germany, 19-21 September, 2018).

[13] Bachmann, C., Bischoff, H., Harris, L., Kaboth, C., Mingers, I., Obrecht, M., Pfeifer, S., Schütte, B. and Sladek, M. (2018). *Cubase Pro 10, Cubase Artist 10 Operation Manual*, Hamburg, Germany: Steinberg Media Technologies GmbH., Retrieved 16 November 2018, from https://steinberg.help/cubase_pro_artist/v10/en/Cubase_Pro_Artist_10_Operation_Manual_en.pdf.

[14] Erichsen, T. 2016. rtpMIDI. Version 1.1.8. Computer Program. Retrieved 15 April, 2018 from https://www.tobias-erichsen.de/software/rtpmidi.html

[15] Lazzaro, J. and Wawrzynek, J. (2011). *RTP Payload Format for MIDI* (RFC 6295), The IETF Trust, Retrieved 29 April, 2018, from https://tools.ietf.org/pdf/rfc6295.pdf.

[16] Postel, J. (1980). *User Datagram Protocol* (RFC 768), Internet Engineering Task Force, Retrieved 18 February, 2018, from https://tools.ietf.org/pdf/rfc768.pdf.

[17] Schulzrinne, H., Casner, S., Frederick, R. and Jacobson, V. (2003). *RTP: A Transport Protocol for Real-Time Applications* (RFC 3550), The Internet Society, Retrieved 19 February, 2018, from https://tools.ietf.org/pdf/rfc3550.pdf.

[18] Johnston, A. B. and Burnett, D. C. 2014. *WebRTC: APIs and RTCWEB Protocols of the HTML5 Real-Time Web*. Digital Codex LLC., St. Louis, United States of America.

[19] Dutton, S. 2012. Getting Started with WebRTC. *HTML5Rocks.* Retrieved 24 July, 2018 from https://www.html5rocks.com/en/tutorials/webrtc/basics/

[20] World Wide Web Consortium. 2015. Web MIDI API. (March 17) Retrieved 30 July 2018 from https://www.w3.org/TR/webmidi/

[21] Schmitt, D. 2019. LoopBe30. Version 1.6. Computer Program. Retrieved 10 April, 2019 from https://www.nerds.de/en/loopbe30.html

[22] World Wide Web Consortium. 2018. WebRTC 1.0: Real-time Communication Between Browsers. (September 20) Retrieved 14 September, 2018 from https://www.w3.org/TR/webrtc/

[23] Levent-Levi, T. 2019. WebRTC Multiparty Architectures. *BlogGeek.Me.* (15 April, 2019). Retrieved 2 June, 2019 from https://bloggeek.me/webrtc-multiparty-architectures/

[24] Bernardo, G. G. 2014. WebRTC beyond one-to-one communication. *webrtcH4cKS.* (4 February, 2014). Retrieved 2 June, 2019 from https://webrtchacks.com/webrtc-beyond-one-one/

[25] The MIDI Association. 2019. Details about MIDI 2.0, MIDI-CI, Profiles and Property Exchange. *MIDI News.* (1 June, 2019). Retrieved 3 June, 2019 from https://www.midi.org/articles-old/details-about-midi-2-0-midi-ci-profiles-and-property-exchange