WebPdL2Ork: Bringing the Linux Laptop Orchestra to the Browser

William Furgerson, Ivica Bukvic, Justin Kerobo, Bradley Davis

WEBPDL2ORK: BRINGING THE LINUX LAPTOP ORCHESTRA TO THE BROWSER

William Furgerson
Virginia Tech
Blacksburg, USA
williamfurgerson@vt.edu

Ivica Bukvic
Virginia Tech
Blacksburg, USA
ico@vt.edu

Justin Kerobo
Virginia Tech
Blacksburg, USA
justinkerobo@vt.edu

Bradley Davis
Virginia Tech
Blacksburg, USA
brdavis4@vt.edu

ABSTRACT

In the following paper, we introduce WebPdL2Ork, a new browser-based port of Pd-L2Ork. We present a brief history of Pd-L2Ork and the Linux Laptop Orchestra (L2Ork), the motivation for a Web-based implementation, and its benefits. We explain how we adapted Pd-L2Ork to run in the browser using Emscripten and WebAssembly and detail the technical hurdles we encountered along the way as well as our solutions. We highlight the L2Ork-Tweeter Pd-L2Ork app, a core component of the Linux Laptop Orchestra since 2020 and the most complex patch packaged with Pd-L2Ork, running in WebPdL2Ork with near-parity to its desktop counterpart. Finally, we list the remaining shortcomings of WebPdL2Ork as well as our plans for future work to address its shortcomings.

0.1. Funding and Copyright Notice

This work relates to Department of Navy award N00014-22-1-2164 issued by the Office of Naval Research. The United States Government has a royalty-free license throughout the world in all copyrightable material contained herein.

1. INTRODUCTION

The Linux Laptop Orchestra (L2Ork) [1], was founded in 2009. To meet the specific needs of L2Ork, Pd-L2Ork [2] was developed as a custom version of Pure-Data (Pd). Building on the defunct Pd-Extended, Pd-L2Ork has continued to evolve and expand alongside L2Ork, as well as finding use in K-12 education [3]. Although Pd-L2Ork was initially developed as a Linux-only application, it has since been ported to both Windows and macOS, broadening its reach, while also adopting a nw.js front-end. The introduction of libPd [4] [5], a lightweight embeddable version of Pure Data, has since shown the feasibility of integrating Pd-based processing into external applications. This approach gained traction in both commercial and experimental settings, including use in Electronic Arts' game Spore [6]. More recently, Web technologies have made it possible to bring Pd-based environments to the browser. A notable example is PdWebParty [7], created by Zack Lee during the 2020 Google Summer of Code (GSOC), which leveraged libPd and WebAssembly [8] to allow for basic execution of Pd patches within a Web browser.

Building on these foundations, this paper introduces WebPdL2Ork, a Web-based implementation of Pd-L2Ork which extends PdWebParty's capabilities. By implementing many of the features that PdWebParty lacks, WebPdL2Ork enables near-complete execution of Pd-L2Ork patches directly inside a Web browser. This development significantly broadens Pd-L2Ork's reach potential, reducing installation barriers, particularly in the K-12 sector, and increases its usefulness.

2. MOTIVATION

The motivation for a Web-based version of Pd-L2Ork stems from a desire to resolve compatibility and consistency challenges. The Linux Laptop Orchestra has historically required that users install and configure a full desktop version of Pd-L2Ork in order to run L2Ork Tweeter (A Pd-L2Ork patch used by L2Ork for real-time composition and performances). This friction has limited the diversity of platforms that can be readily used with it. By adding a Web-based option, we hope to eliminate many of these barriers and allow participants to use L2Ork Tweeter instantly without any installation process, from any device with a Web browser. L2Ork Tweeter has also served as a crucial benchmark due to its size and complexity. As such, it has become a stress and regression test platform for the ongoing development of both Pd-L2Ork and WebPdL2Ork.

Such an option would enable even broader participation in L2Ork by lowering the technical requirements in two ways. First, users would not have to worry about how to install and/or update Pd-L2Ork on their systems. Second, we would not need to worry as much about edge cases that only affect certain system configurations. Since Web browsers are fairly standard in how they behave, testing and fixing bugs becomes easier for browser-based applications than their native counterparts, where every system is potentially a unique environment.

Finally, this motivation aligned with another one of our research efforts focused on making Pd-L2Ork more accessible for Science, Technology, Engineering, and Math (STEM) K-12 education. For that particular research, we were interested in making Pd-L2Ork accessible for use in middle and high school curricula to teach wave physics, where schools face challenges installing and maintaining software due to limited IT resources. To address this, we were already exploring an online runtime platform, and given that the same solution would greatly benefit L2Ork Tweeter, it made sense to pursue a fully featured Web-based port of Pd-L2Ork.

3. IMPLEMENTATION

3.1. Foundation

Our new extension to Pd-L2Ork is based on two prior works by other authors. libPd [4, 5] serves as the foundation of WebPdL2Ork, allowing the use of Pd-L2Ork's core as a library. Using libPd's API, we can control Pd-L2Ork's core from the outside, tell it to open, close, or run patches, and add hooks to receive events when data changes, allowing us to instrument the state of Pd-L2Ork and perform additional processing. We make extensive use of these hooks for our new GUI and compatibility layer, WebPdL2Ork. We also make use of PdWebParty [7], a proof-of-concept bare-bones Web UI based on libPd, as a foundation for WebPdL2Ork. PdWebParty uses Emscripten [9] to compile Pd-L2Ork's core (and libPd) into WebAssembly [10],

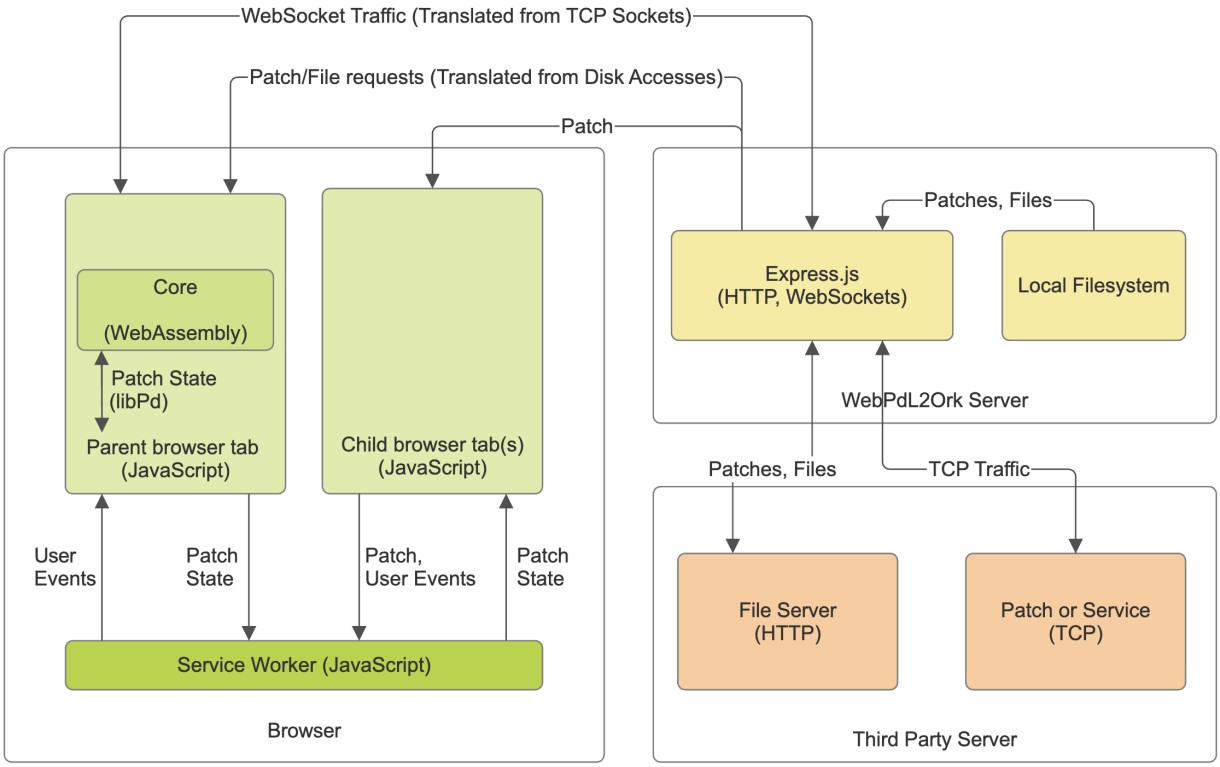

Figure 1: *WebPdL2Ork Architecture*

so that it can run inside a browser. It implements a very limited set of core GUI objects (Bang Buttons, Toggles, Sliders, Radio buttons, Canvases, and Comments), all vanilla non-GUI objects, and a subset of external libraries (externals), and as a result has significant limitations on what patches it can support. We based WebPdL2Ork on the foundation provided by PdWebParty, added the full set of GUI objects from Pd-L2Ork, added nearly all remaining Pd-L2Ork externals, and addressed almost all of PdWebParty's limitations on what patches are able to run.

3.2. Architecture

As a Web application, WebPdL2Ork is split into a client and a server. We serve WebPdL2Ork using Express.js [11, 12] and Node.js [13] behind an nginx [14] proxy. The server hosts WebPdL2Ork's static JavaScript and WebAssembly code, which is then run by the user's browser when they visit WebPdL2Ork. WebPdL2Ork's server also provides patches to the client. It hosts a small collection of patches itself and also supports loading patches from external URLs. In either case, the client will request that the server fetch the patch for it, and the server will serve that patch to the client for it to open in the browser. Any static files used by a patch are handled in the same way. We maintain a docker [15] image and sample docker-compose file in our GitHub repository [2] as well as documentation, so that others who are interested in running an instance of WebPdL2Ork can do so.

3.3. Limitations of PdWebParty

While PdWebParty offered a solid foundation, as a proof-of-concept implementation it has several major shortfalls which limit what patches it can run. It does not support many GUI objects, such as drop-downs, text boxes, links, images, arrays, and others. It cannot handle abstractions (additional patches loaded from another patch at run-time) or graph-on-parent (displaying GUI objects which live inside a subpatch or an abstraction), both of which are used heavily in L2Ork Tweeter. It is missing about 20 external libraries, many of which (cyclone, for instance) are used heavily in many Pd-L2Ork patches, like L2Ork Tweeter. It does not support networking, meaning that L2Ork Tweeter's main purpose as a collaborative telematic platform for musicking is lost. Finally, it suffers from a major limitation of libPd. In Pd-L2Ork, there are two kinds of connections between objects, so called wired and wireless. Wired connections consist of an explicit connection between two specific objects. Every wire connects exactly two objects together, bridging the output of one object with the input of another. Wireless connections, on the other hand, can be thought of as transmitters and receivers. Objects can transmit wirelessly using a specific address, and any objects that are listening using the same address will receive the data that was sent, as if on a wire. Any number of objects can transmit or receive data using the same address. Most objects in Pd-L2Ork only support wired connections, with the notable exception of GUI objects. Most GUI objects can be connected either using wires or using wireless connections, but libPd's hooks only work on wireless connections. Any data send over wired connections cannot be monitored by libPd, or PdWebParty by extension, since it receives all of its state information via hooks, Therefore, any GUI objects that use wired connections - the most common type of connection by far - will not work properly in PdWebParty.

3.4. GUI Objects

In WebPdL2Ork, we have added support for all non-deprecated GUI objects that are actively maintained in Pd-L2Ork and were missing from PdWebParty, including those introduced after its initial release. Since the core (which is compiled from C to WebAssembly) is responsible for audio processing and patch logic and does not handle any GUI operations, all user interaction must be managed externally with LibPd serving as the interface through which external code communicates with the core. As a result, we implemented all GUI object interactivity in WebPdL2Ork from the ground up using JavaScript (the native

language for web browsers) as a bridge between the user and libPd.

We implemented knobs (a.k.a. dials), added support for text and number boxes, including Pd-L2Ork's copy/paste and input exclusivity support. We added support for vu-meters to show audio peaks, and support for links, both to other patches and to external Web pages. We implemented drop-downs to select between multiple objects, and we added support for images, as well as raw mouse and keyboard event objects. Finally, we implemented arrays, which are effectively interactive point, line, or bar charts containing many values inside one object. L2Ork Tweeter uses arrays to allow the user to define the amplitude and modulation envelopes of their instruments, so we also support the modification of arrays via click and drag.

Given the increasingly mobile nature of the Web, throughout the process of implementing these GUI objects we kept mobile browsers in mind. All text entry fields support the virtual keyboards used by mobile browsers, and other fields support touch interactions in addition to mouse interactions. Because of this, Pd-L2Ork patches (including L2Ork Tweeter) can now be opened on mobile devices without installing any software, by using WebPdL2Ork in a browser, though the UI of many patches (including L2Ork Tweeter) is not very conducive to touch inputs because of small click targets.

3.5. Files

Many patches rely on reading files from the local file system. L2Ork Tweeter, for instance, allows users to load instrument and loop presets from preset files bundled alongside it. Emscripten provides a virtual file system for Pd-L2Ork to use, but that file system starts out empty with only Pd-L2Ork's executable code and the shared library code of any externals which were built alongside it. When a patch is fetched from an arbitrary relative or absolute URL and loaded into WebPdL2Ork, it is injected into Emscripten's file system, but there is no way to know at load-time what other files the patch will attempt to execute at run-time. These can be other patches, or just about any other file format supported by Pd-L2Ork, such as images, text files, sound files, etc.

To allow patches to find files stored in the same location as themselves, we replaced Emscripten's implementation of opening a file in the virtual file system with our own. When Pd-L2Ork attempts to open a file, we first check if it already exists in the virtual file system. If it does, then we open the file and give the reference to Pd-L2Ork in exactly the same manner as Emscripten's default function does. Otherwise, we try to find the file in the same location as the patch was loaded from. For instance, if the open patch's source is `http://example.org/some/patch.pd` and it tries to load "path/data.txt", we will ask the server to fetch data.txt from `http://example.org/some/path/data.txt`. If the file exists at that location, we inject it into Emscripten's file system and then return a reference to Pd-L2Ork. All of this is transparent to the core, since opening a file blocks until we either find or fail to find the file. From the core's perspective, it is interacting with a normal (albiet somewhat slow) file system.

3.6. Abstractions

Another core feature of Pd-L2Ork (and all other variants of Pure-Data) is the ability to embed patches (a.k.a. abstractions) inside other patches. In this way, the same patch can be re-used many times, similar to an object-oriented class in more traditional languages. PdWebParty did not support this, so we added support for it in WebPdL2Ork. While we could use the same method as mentioned above for allowing the core to read files, there are several major problems with that method because of how Pd-L2Ork loads patches.

The first issue is that Pd-L2Ork's core loads abstractions sequentially when parsing a patch. As mentioned in the previous section, the process of opening a file is entirely synchronous. The thread that attempts to open an abstraction is blocked until the network request which attempts to fetch that abstraction either succeeds or fails. Repeated accesses to the same abstraction can complete quickly since after the first request the file will be saved in the virtual file system, but as the number of unique abstractions adds up, the latency increases drastically. What takes a matter of milliseconds on a real file system may take seconds in WebPdL2Ork, since every new abstraction requires a synchronous network request. Another major issue is that Pd-L2Ork uses a list of search paths to find abstractions. For any given abstraction it will attempt to open it in each search path that it has registered, one after the other, until it finds the abstraction or ultimately fails, outputting an error. In WebPdL2Ork, each attempt at guessing the file path would require its own network request until the correct one is found, further increasing the latency.

For these reasons, handling abstractions at run-time is not feasible or scalable. Since all references to abstractions are easily found in the patch's code, unlike other types of file accesses, we decided to perform static analysis of the patch at load-time to make all the necessary network requests in a more efficient manner. To do this, we added a fetch stage to the patch processor before the main processing stage which handles GUI objects. When the patch is first received from WebPdL2Ork's server, the processor scans the entire patch to compile a list of possible references to additional patches. Once it has this list, it requests all of the patches from WebPdL2Ork's server in parallel. WebPdL2Ork's server then responds with all of the patches that it found, and the processor performs the same analysis on those new patches recursively until no more patches are found. Then, during the main processing, it inlines any references to the abstractions with the abstractions themselves. This has a side effect of increasing the total in-memory size of the patch, but the performance gains are well worth it. Using L2Ork Tweeter as an example, we perform the 281 searches necessary to resolve all abstractions in only 3 network cycles. Since processing time is negligible compared to network latency, this reduces the latency to just 1% of what it would have been if all requests were performed in sequence.

Another side effect of fetching all abstractions at load time is that we are able to handle graph-on-parent patches (patches which have GUI objects inside of abstractions). After the abstractions are inlined into the main patch, the patch processor finds their GUI objects in the exact same manner as it finds GUI objects that were in the main patch and creates DOM elements for the user to interact with.

3.7. Externals

We have added support for 20 of the 23 external libraries that Pd-L2Ork supports and PdWebParty does not. These include all externals used by L2Ork Tweeter, as well as others. Most notably, WebPdL2Ork now supports pdlua, pdjs, neuraln, flite, cyclone, and OSCx. Pdlua allows patches in WebPdL2Ork to run Lua [16] scripts in the browser for logic and audio processing, and pdjs does the same for JavaScript code. Neuraln provides support for patches to train and run neural networks and use them in audio processing. Flite provides text-to-speech support, cyclone is widely used in many patches for logic processing, and OSCx provides Open Sound Control [17] interfaces for patches to use. For most externals, the only work was to adjust their build scripts to support using Emscripten's compiler rather than gcc. For some patches this was straightforward, and for others it was very tedious. However, some externals required major changes in order to run in the browser. We had to make modifications to the flite external in order to get it to run in a Web-based environment, and we re-implemented pdjs from scratch. On the desktop version, pdjs bundles its own pre-compiled V8 [18, 19] engine.

There are no pre-compiled V8 binaries for browsers because the browser itself already necessarily contains a JavaScript engine. Because of this, we re-implemented pdjs in the client JavaScript rather than in Pd-L2Ork's C core, so that it could make use of the browser's JavaScript engine. We used dedicated Web workers to run scripts and set up a custom namespace inside them to emulate the namespace that desktop pdjs exposes to V8.

3.8. Wired Connections

Adding support for the missing GUI objects, files, abstractions, and externals goes a long way towards running L2Ork Tweeter, but libPd is still very limited in how it allows the core and the client to interact. Pd-L2Ork supports both wired and wireless connections, but libPd only allows the client to monitor wireless connections. Yet, wired connections are much more common in patches than wireless connections (in great part because wired connections make tracing execution order considerably easier and as such are considered a better coding practice, where possible), and L2Ork Tweeter is no exception. Because of this, we created a compatibility layer inside WebPdL2Ork to deal with this libPd's limitation. When the patch processor scans the patch for GUI objects, it maintains a list of all the GUI objects it has found. Then, when the it comes across a wired connection it checks its list of GUI objects to determine if either end of the connection connects to a GUI object. If so, it knows that the data over that connection won't be received through libPd. Thankfully, the Pure-Data ecosystem has two objects which we use to solve this problem: "send" (a.k.a. "s") and "receive" (a.k.a. "r"). "s" takes in a wired connection and passes through any data to a wireless connection, and "r" does the inverse. When a wired connection is found with a GUI object, the processor injects either an "r" or "s", connects it to the wired end of the connection, and gives it a unique wireless connection name to wirelessly connect to the GUI object. If the connection is between two GUI objects, the processor skips the step of adding an "s" or "r" and makes a direct wireless connection between the two objects. There are additional edge cases which add more complexity for full compatibility with wired patches, but they follow the same logic as these simple cases. As a result of our wired to wireless conversion, WebPdL2Ork is now able to load any patch regardless of the connection scheme it uses.

3.9. Networking

Another major compatibility issue when running patches inside a Web browser is networking. Many patches use TCP [20, 21] sockets to send and receive data over the network. L2Ork Tweeter, for instance, uses TCP sockets to connect to a Tweeter server which synchronizes the musical experience between multiple users performing or composing simultaneously. Browsers do not support TCP sockets, instead opting for HTTP-based WebSockets [22]. Emscripten has a built-in support for converting the TCP sockets created by C's standard library into WebSockets, but this assumes that whatever service the C code is trying to connect to is also capable of communicating using WebSockets. Since WebSockets use a completely different protocol than TCP sockets, this is rarely the case.

To solve this problem, we created a WebSocket bridge which sits in between Pd-L2Ork's core and the endpoints that it tries to connect to. Similarly to how we intercept when a patch tries to open a file, we intercept when the browser tries to open a WebSocket (as a result of Emscripten converting a TCP socket). When this happens, the interceptor redirects the WebSocket to WebPdL2Ork's server and immediately sends a packet containing the original IP that the socket was meant to connect to. WebPdL2Ork's server then opens a TCP socket on the client's behalf to that IP and forwards all data sent on either socket to the other.

Although Emscripten provides TCP to WebSocket conversion as a part of its compilation process, it does not provide DNS [23, 24] resolution. Because of this, any patches that connect to a server using its domain name rather than its IP address fail to resolve the IP address for that domain. To solve this, we replace C's low-level "gethostbyname" function (which all higher level DNS resolution functions use internally) with our own custom implementation. Instead of using the system's DNS resolver, which doesn't exist in WebAssembly, we open a socket to WebPdL2Ork's server, which resolves the domain on behalf of the client and returns the result.

By implementing transparent DNS resolution and WebSocket bridging, we have been able to implement networking without making any changes to existing Pd-L2Ork objects. This was a major goal of ours since there isn't one standard Pd-L2Ork object for networking, but rather a large selection of objects, each of which does things slightly differently. If we were to bake WebPdL2Ork's networking support into specific objects, we would need to maintain the functionality as objects evolve and add it to new objects as they are created. Instead, we have managed to avoid this by making all existing C networking code work without specific changes for WebPdL2Ork.

3.10. Multiple Open Patches

The last major area where we addressed PdWebParty's limitations is having multiple patches open simultaneously. Pd-L2Ork allows the user to open multiple patches simultaneously, as well as to communicate with each other using wireless connections. We have added similar functionality to WebPdL2Ork using a browser service worker. Service workers exist at the site level rather than the page level, so we used them to allow multiple tabs of WebPdL2Ork to communicate with each other and manage their relationships. When a WebPdL2Ork tab is opened, we perform dependency and GUI processing on the patch as detailed in previous sections, but there is one final step which we have not yet mentioned. Before installing the patch to the virtual file system and starting Pd-L2Ork, the page communicates with WebPdL2Ork's service worker to check if it is a parent page or a child page. Browser tabs that were opened manually by the user are defined as parents, and they run their own patches. When the user clicks on a link in a patch that goes to another patch, we create a new child tab on the user's behalf. Instead of running its own patch, it sends its patch to the service worker, who in turn sends it to the parent who created that child. The parent then opens the child's patch using the same libPd instance used to run the parent's patch. From the core's perspective this is the same as when multiple patches are opened on the desktop (all windows are shared between the single Pd-L2Ork application). Finally, instead of subscribing to events from libPd, child pages will subscribe to events from their parent via the service worker. In this way, multiple tabs of WebPdL2Ork can share the same libPd instance and function similarly to how multiple windows of Pd-L2Ork behave on the desktop.

4. OUTCOME

4.1. L2Ork Tweeter in the Browser

As a result of all these enhancements to PdWebParty, WebPdL2Ork is now able to run L2Ork Tweeter with near-parity to desktop Pd-L2Ork. All of the GUI elements in L2Ork Tweeter are supported by WebPdL2Ork, as well as all the externals that Tweeter uses. All of L2Ork Tweeter's presets are available because of our file interceptor, and its networking features work thanks to our networking compatibility layer. Even L2Ork Tweeter's multi-window features, like the loop and preset widgets (independent patches that supplement L2Ork Tweeter with additional functionality), are able to open in popups and properly communicate with the

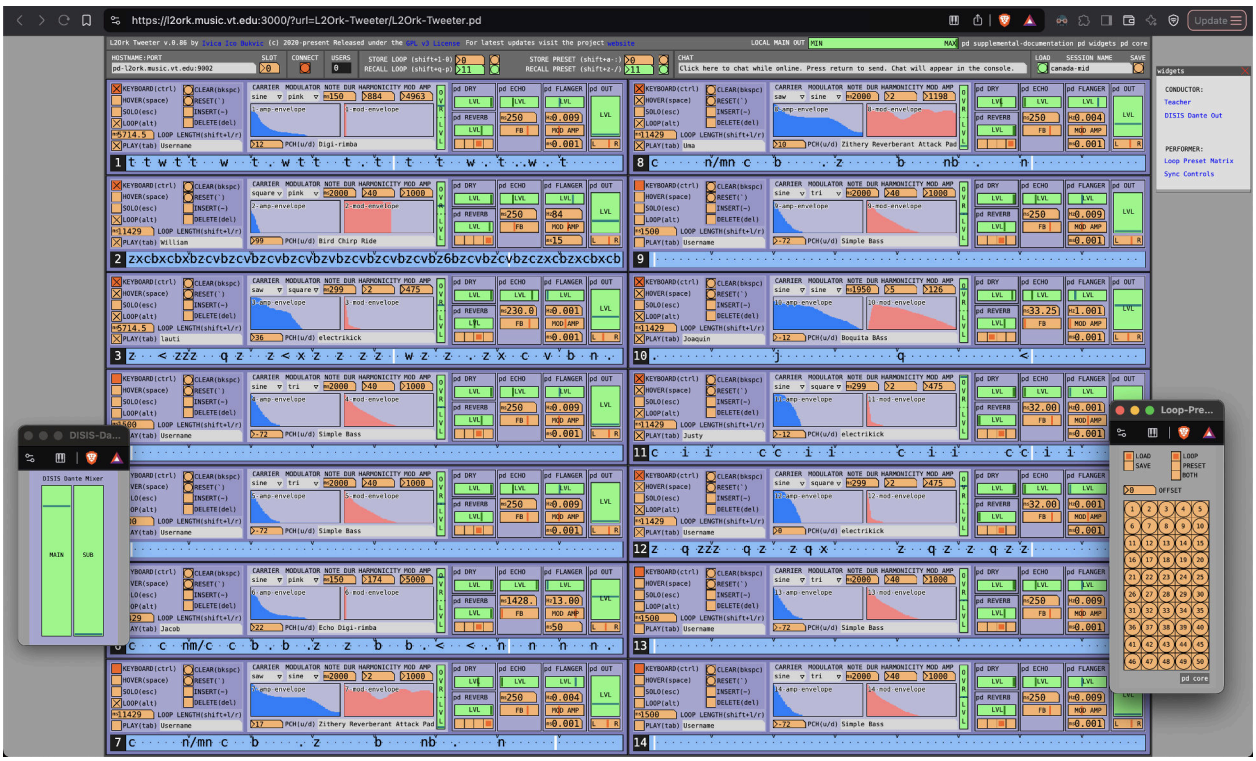

Figure 2: *L2Ork Tweeter running inside WebPdL2Ork (4.1).*

main window. We have deployed a WebPdL2Ork instance to `https://l2ork.music.vt.edu:3000/`. That instance includes L2Ork Tweeter, which can be accessed by navigating to `https://l2ork.music.vt.edu:3000/?url=L2Ork-Tweeter/L2Ork-Tweeter.pd`. To experiment with other patches hosted at external locations, place the URL to that patch in the url query parameter: `https://l2ork.music.vt.edu:3000/?url=https://example.org/patch.pd`. The following figure showcases L2Ork Tweeter running using WebPdL2Ork. 2

4.2. Limitations

Although we have achieved near-parity to desktop Pd-L2Ork, there are several areas where WebPdL2Ork still falls short compared to its desktop counterpart. First, while we are able to load external files into the virtual file system on demand, we currently do not support saving files from the virtual file system to an external source. In the case of L2Ork Tweeter, this means that any presets provided alongside the hosted L2Ork Tweeter patch can be loaded and modified, but they cannot be saved, as the virtual file system is lost when the browser tab is closed. Additionally, unlike desktop Pd-L2Ork, which allows the user to select an audio API (such as JACK or ALSA), WebPdL2Ork is limited to the audio interface which the browser exposes. Finally, we do not support editing patches via WebPdL2Ork. So, while L2Ork Tweeter is now available in the browser, we will continue to do all development on Tweeter via the desktop version of Pd-L2Ork. Finally, when opening multiple windows in WebPdL2Ork, once a window is closed it cannot be reopened. We have not yet determined why, but libPd does not function properly when reopening a previously closed patch. We hope to resolve this limitation in the near future.

4.3. Future Work

We plan to continue the development of WebPdL2Ork to address the limitations detailed above. In particular, we look forward to designing a mechanism to save and load data in the virtual file system. Once this functionality is implemented, both composers and performers will have the flexibility to use L2Ork Tweeter whenever and wherever they want, both on the desktop and on the Web. At some point in the future, we look forward to adding the ability to edit patches using a WebPdL2Ork editor. This effort will commence once WebPdL2Ork has addressed all runtime incompatibilities and limitations.

We are also interested in exploring the possibility of a central repository for Pd-L2Ork patches. Since WebPdL2Ork can load patches from arbitrary URLs, it lends itself to a repository where users could create and share patches for anybody on the internet to view and run.

5. CONCLUSIONS

Through our work on WebPdL2Ork, L2Ork Tweeter is now accessible directly inside a Web browser, eliminating the need for users to install Pd-L2Ork or configure their system. This significantly reduces the barrier to entry, allowing more people, students, musicians, and researchers alike, to experiment with L2Ork Tweeter using any device and any operating system. By broadening access to this tool, we also extend the reach of the Linux Laptop Orchestra, making collaborative live music production and sound design more widely available than ever before.

More broadly, WebPdL2Ork represents a step forward in the evolution of Pure-Data based environments. By filling the gaps in PdWebParty's prototype, we have demonstrated how powerful interactive audio tools can be adapted for greater compatibility while maintaining their core functionality and the features that set them apart. Though challenges and shortcomings remain, WebPdL2Ork's ability to run complex patches like L2Ork Tweeter in a browser shows the potential for further innovation in this field. We look forward to the future of WebPdL2Ork, Pd-L2Ork, and L2Ork Tweeter.

6. ACKNOWLEDGMENTS

This project is sponsored by the Department of the Navy, Office of Naval Research under ONR award number N00014-22-1-2164. Any opinions, findings, conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the Office of Naval Research.

We acknowledge Zach Lee for creating PdWebParty, which served as the foundation for our work on WebPdL2Ork.

7. REFERENCES

[1] Ico Bukvic, "L2Ork » Linux Laptop Orchestra," Jan. 2009.

[2] Ivica Ico Bukvic, "pd-l2ork," Mar. 2025, original-date: 2020-10-14T13:55:01Z.

[3] Ivica Bukvic, Liesl Baum, Bennett Layman, and Kendall Woodard, "Granular learning objects for instrument design and collaborative performance in k-12 education," in *Proceedings of the International Conference on New Interfaces for Musical Expression*, 2012.

[4] Peter Brinkmann, Peter Kirn, Richard Lawler, Chris McCormick, Martin Roth, and Hans-Christoph Steiner, "Embedding pure data with libpd," in *Proceedings of the Pure Data Convention*. 2011, vol. 291, Citeseer.

[5] Peter Brinkmann, Dan Wilcox, Tal Kirshboim, Richard Eakin, and Ryan Alexander, "libpd: Past, Present, and Future of Embedding Pure Data," *PdCon 2016*.

[6] David Kushner, "Engineering Spore," *IEEE Spectrum*, vol. 45, no. 9, pp. 36–40, Sept. 2008, Conference Name: IEEE Spectrum.

[7] Zack Lee, "PdWebParty," Mar. 2025, original-date: 2020-12-01T15:08:18Z.

[8] Andreas Haas, Andreas Rossberg, Derek L. Schuff, Ben L. Titzer, Michael Holman, Dan Gohman, Luke Wagner, Alon Zakai, and Jf Bastien, "Bringing the web up to speed with WebAssembly," in *Proceedings of the 38th ACM SIGPLAN Conference on Programming Language Design and Implementation*, Barcelona Spain, June 2017, pp. 185–200, ACM.

[9] Alon Zakai, "Emscripten: an LLVM-to-JavaScript compiler," in *Proceedings of the ACM international conference companion on Object oriented programming systems languages and applications companion*, Portland Oregon USA, Oct. 2011, pp. 301–312, ACM.

[10] Gerard Gallant, *WebAssembly in Action: With examples using C++ and Emscripten*, Simon and Schuster, 2019.

[11] Azat Mardan, "Using Express.js to Create Node.js Web Apps," in *Practical Node.js*, pp. 51–87. Apress, Berkeley, CA, 2018.

[12] Evan Hahn, *Express in Action: Writing, building, and testing Node. js applications*, Simon and Schuster, 2016.

[13] Alessandro Benoit, *NW. js Essentials*, Packt Publishing Ltd, 2015.

[14] Will Reese, "Nginx: the high-performance web server and reverse proxy," *Linux Journal*, vol. 2008, no. 173, pp. 2, 2008, Publisher: Belltown Media Houston, TX.

[15] Dirk Merkel, "Docker: lightweight linux containers for consistent development and deployment," *Linux j*, vol. 239, no. 2, pp. 2, 2014.

[16] Roberto Ierusalimschy, *Programming in lua*, Roberto Ierusalimschy, 2006.

[17] Matthew Wright and Adrian Freed, "Open soundcontrol: A new protocol for communicating with sound synthesizers," in *ICMC*, 1997.

[18] Georgiy Krylov, Maria Patrou, Gerhard W. Dueck, and Joran Siu, "The evolution of garbage collection in v8: google's javascript engine," in *2020 9th Mediterranean Conference on Embedded Computing (MECO)*. 2020, pp. 1–6, IEEE.

[19] Jan Kasper Martinsen, Håkan Grahn, and Anders Isberg, "Preliminary Results of Combining Thread-Level Speculation and Just-in-Time Compilation in Google's V8," in *Sixth Swedish Workshop on Multicore Computing (MCC-13)*. 2013, Halmstad University.

[20] W. Eddy, "Transmission Control Protocol (TCP)," Tech. Rep. RFC9293, RFC Editor, Aug. 2022.

[21] J. Postel, "Transmission Control Protocol," Tech. Rep. RFC0793, RFC Editor, Sept. 1981.

[22] I. Fette and A. Melnikov, "The WebSocket Protocol," Tech. Rep. RFC6455, RFC Editor, Dec. 2011.

[23] Paul Mockapetris, "Domain names: Implementation specification," Tech. Rep., 1983.

[24] P.V. Mockapetris, "Domain names - implementation and specification," Tech. Rep. RFC1035, RFC Editor, Nov. 1987.