A TouchOSC MIDI Bridge for Linux

Albert GRÄF
Computer Music Research Group
Institute of Art History and Musicology (IKM)
Johannes Gutenberg University (JGU) Mainz, Germany
aggraef@gmail.com

Abstract

Mobile applications such as hexler's TouchOSC offer a cheap and convenient alternative to traditional controller hardware for computer music programs. TouchOSC is available for Android and iOS devices and supports both OSC and MIDI, two widespread standards for transmitting control data between computer music applications. On the host side the TouchOSC MIDI Bridge is required for MIDI support, which unfortunately is proprietary software and only available for Mac and Windows systems. This paper presents pd-touchosc, a library of Pd externals which aims to bring most of the functionality of the TouchOSC MIDI Bridge to Linux.

Keywords

TouchOSC, controller, OSC, MIDI

1 Introduction

Any reader familiar with the area of computer music will have heard of JazzMutant's Lemur controller [5], a big multitouch device with built-in OSC[1] and MIDI[2] support, which was fully configurable using a kind of GUI builder for control surfaces. Nowadays, the Lemur's place is taken by mobile apps running on modern (and much cheaper) devices such as smartphones and tablets. (It is no accident that the demise of the original Lemur hardware was brought about by the advent of the iPad.) The Lemur lives on as a mobile app on iOS[3], and there are other similar apps on both Android and iOS.

One of these is hexler's TouchOSC [4]. While it lacks some of the Lemur's more advanced features such as physical models and scripting capabilities, it certainly offers enough features to create fairly sophisticated interfaces and is also much cheaper. It comes with its own graphical layout editor (which is written in Java and thus runs on Linux just as well as on Mac and Windows). Like the Lemur, TouchOSC supports both OSC and MIDI and the layout of the controller elements is fully configurable, so that the user can tailor the graphical interface to the computer music application at hand. This sets it apart from applications like TouchDAW[4] which provide fixed interfaces usually inspired by existing MIDI controller designs. There are other apps similar to the Lemur and TouchOSC, such as OSCPad[5] which is more or less compatible with the TouchOSC layout format, and Charlie Roberts' open-source app Control[6] which features its own JSON format and is both scriptable and extensible using JavaScript. But among these TouchOSC seems to be the most mature and popular option right now, not least because of its graphical layout editor.

One of the downsides of TouchOSC for Linux users, however, is its MIDI support which requires either the TouchOSC MIDI Bridge program or an RTP-MIDI[7] interface on the host side, neither of which is readily available on Linux. (The TouchOSC MIDI Bridge is closed source software only available for Mac and Windows, and drivers for RTP-MIDI are hard to find for Linux these days. Moreover, the RTP-MIDI protocol doesn't seem to be supported in the Android version of the TouchOSC app anyway.) So the author set out to create a TouchOSC MIDI bridge replacement for Linux, which is what this paper is about.

Why would you want to use MIDI with TouchOSC anyway? It is true that MIDI is a much more limited format for control data than OSC, but you may find the conversion from/to MIDI convenient when interfacing TouchOSC to existing MIDI applications and hardware, such as synthesizers, algorithmic composition and music notation software, as well as DAW

[1] `http://opensoundcontrol.org/`
[2] `http://www.midi.org/`
[3] `http://liine.net/en/products/lemur/`
[4] `http://www.humatic.de/htools/touchdaw/`
[5] `http://burnsmod.com/software/oscpad.html`
[6] `http://charlie-roberts.com/Control/`
[7] `http://www.cs.berkeley.edu/~lazzaro/rtpmidi/`

(digital audio workstation) and sequencer programs. In particular, the conversion enables you to record control and automation data with DAW and sequencer programs, which typically offer good facilities for recording, playing back and editing MIDI sequences, but often provide only limited support for OSC, if at all.

So there are plenty of use cases for TouchOSC MIDI on Linux. Given that neither the proprietary TouchOSC Bridge protocol nor RTP-MIDI will work for our purposes, the most straightforward solution is to just take the MIDI mapping information available in TouchOSC layout files and convert OSC messages to/from MIDI in an automatic fashion using that information. This approach obviously has some shortcomings when compared to the "official" TouchOSC MIDI Bridge which connects directly to the TouchOSC app on the device. In particular, it requires a working OSC connection and that the TouchOSC layouts on the device and the host side match up. But this doesn't seem to be much of an impediment, and in any case it is better than not having any MIDI connectivity at all.

Our current implementation of the TouchOSC MIDI bridge for Linux uses Miller Puckette's Pd a.k.a. Pure Data[8], an interactive visual programming environment for computer music and multimedia applications. This makes it easy to create a working prototype of the software and also opens up the interface so that users can modify details of the implementation inside Pd. However, the core code of our solution (which is written in the author's Pure programming language [1]) could certainly be massaged into a stand-alone program which works outside the Pd environment.

2 TouchOSC Layouts

Let us begin with a brief overview of TouchOSC layouts. For further details we refer the reader to the documentation available at the TouchOSC website [4].

Layouts are created with the TouchOSC editor which can store them in zipped XML files or transfer them directly to a TouchOSC instance running on a device.

Figure 1 shows one page of a typical layout, as it is rendered on a device (an Android tablet in this case). When creating a layout with the editor, the user can choose from a built-in collection of various GUI widgets such as faders,

[8]`http://puredata.info/`

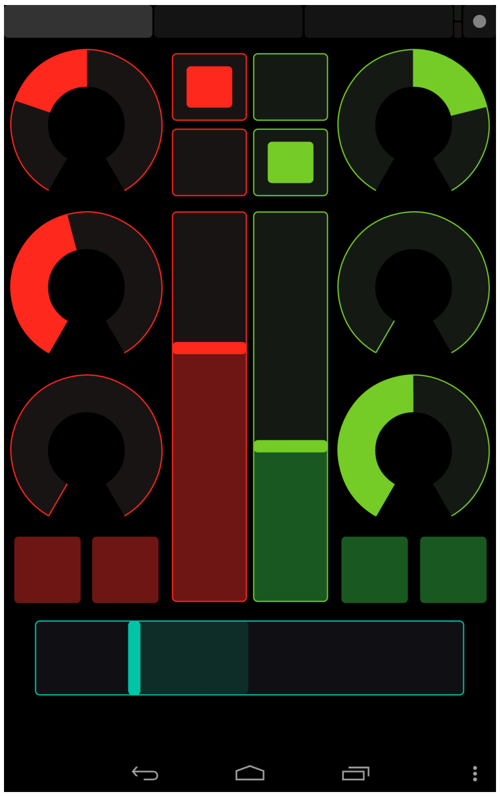
Figure 1: TouchOSC layout.

rotary controls, push and toggle buttons and XY pads. These can be placed freely on the screen. A layout may consist of multiple pages which can be selected using the tabs at the top of the screen.

Each TouchOSC widget has one or more OSC messages associated with it, which are emitted when the status of the widget changes in some way (button pressed, fader moved, etc.). Conversely, OSC messages can also be transmitted to the device in order to change the current value of a widget. The following status variables are supported by most widgets:

- x, y: x is the primary value of a control, such as the value of a fader or a rotary, or the status of a button (0 = off, 1 = on). XY pads have a secondary y value, so they encode two values x and y (position of the control along the x and y axis, respectively) at the same time. These variables are for both input and output, i.e., they are transmitted to the host in response to a touch event, but the host can also send them back to the device in order to change the value. The latter is useful for setting up presets or providing visual feedback for some host-side operations on the device.

OSC message	Meaning
`/1`	first page
`/1/fader1 0.1`	value of `fader1` on the first page
`/1/fader1/color red`	color of `fader1` (input only)
`/1/fader1/z 1`	touch variable of `fader1` (output only)
`/1/xy1 0.1 0.7`	x, y values of a XY pad
`/1/multifader1/1 0.1`	value of the first subcontrol of a multi-fader
`/1/multifader1/1/z 1`	the subcontrol's z value
`/1/multixy1/1 0.1 0.7`	x, y values of the first subcontrol of a multi-XY pad
`/1/multipush1/2/3 0.1`	value of the subcontrol in column 2, row 3

Figure 2: TouchOSC message examples.

- z is the touch variable which can be either 1 if the widget is being touched or 0 otherwise. In the case of a touch button this will be the same as the primary value of the widget, but this value is output-only (it cannot be changed by transmitting a corresponding OSC message to the device).
- c is the color variable. TouchOSC offers a built-in palette of nine different colors which are usually set when editing the layout. But the color can also be changed dynamically by transmitting a corresponding OSC message to the device. This variable is input-only.

The OSC address of a widget can either be assigned automatically by the TouchOSC editor (in which case it takes the form */n/widget-name* where n is the number of the page on which the widget is located) or the user can set it manually to any valid OSC address string. This address is used for the primary widget value(s) (x and y), whereas the z and c values are specified by tacking on `/z` or `/color` to the OSC address. The OSC ranges of the numeric values are usually 0–1 by default, but this can be adjusted in the editor. The color variables have symbolic values such as `red`, `green` etc. in the OSC encoding.

Faders, buttons and XY pads also have multi-widget variations which consist of multiple controls of the same type making up a single widget. In this case the individual controls have separate OSC addresses of the form */widget-addr/i* with an index i ranging from 1 to the number of subcontrols, or (in the case of multi-button widgets) */widget-addr/i/j* where i denotes the column and j the row index (note that the column index comes first, even though TouchOSC arranges the subcontrols in row-major order internally).

Layout pages themselves also have an OSC address (by default, this will be simply `/1`, `/2`, etc.). A message with just the OSC address (without any parameters) will be emitted whenever the page is clicked in the tab strip, and the host can also send a message of this form to change the page that's currently displayed on the device.

Figure 2 summarizes the syntax of typical TouchOSC messages and their meaning.

3 MIDI Assignments

The TouchOSC editor allows MIDI messages to be assigned to any status variable of a widget in a layout. The details are a little intricate at first because of the distinct characteristics of the various widgets and the different kinds of MIDI messages, but work in rather intuitive and straightforward manner once the user is familiar with the available configuration options. TouchOSC supports all the different types of voice messages MIDI has on offer, as well as the sequencer-related system real-time messages (start, stop and continue).

The start, stop and continue messages offer no further configuration options. They can only be assigned to on/off variables, i.e., the primary value of buttons, or the touch value of any control. In our implementation, this kind of MIDI message is triggered whenever the corresponding control variable goes to a non-zero value.[9]

Voice messages generally map a control variable to the *last* data byte of the message. This will be the note velocity or control value for

[9]Note that, in contrast, the official TouchOSC MIDI Bridge seems to emit the message for each status change, i.e., also when the variable drops back to zero. We do not consider this behavior very useful, however, as it causes a sequencer message to be sent *twice* when pressing and releasing a push button. Nevertheless, there's a compilation time option in our code which provides compatibility with the TouchOSC MIDI Bridge in this respect if this is needed.

No.	Type	Channel	Fixed Data Value	Mapped Data Range
0	control change	1–16	controller number (0–127)	controller value (0–127)
1	note	1–16	note number (0–127)	velocity (0–127)
2	program change	1–16	–	program number (0–127)
3	start	–	–	–
4	stop	–	–	–
5	continue	–	–	–
6	key pressure	1–16	note number (0–127)	velocity (0–127)
7	channel pressure	1–16	–	velocity (0–127)
8	pitch bend	1–16	–	pitch bend (0–16383)

Figure 3: TouchOSC MIDI mappings.

voice messages having two data bytes, and the single data byte of channel pressure (aftertouch) and program change messages. The pitch bend message gets special treatment; in this case the value of the control variable is mapped to the entire 14 bit range of 0–16383. (In MIDI this value is the combination of the two data bytes of the message, hence the 14 bit value range.)

Considering a variable with the source (OSC) range x_1–x_2 and the target (MIDI) range y_1–y_2, the variable (OSC) value x is mapped to:

$$y = y_1 + (y_2 - y_1)\frac{x - x_1}{x_2 - x_1}.$$

In the case of the default OSC value range ($x_1 = 0$, $x_2 = 1$), this can be simplified to:

$$y = y_1 + (y_2 - y_1)x.$$

The resulting value y is then rounded to an integer and clamped to the MIDI data byte range (or the 14 bit range for pitch bend messages). By default, $y_1 = 0$ and $y_2 = 127$ ($y_2 = 16383$ for a pitch bend message).

For voice messages the user may configure the (MIDI) value range for the control variable, the (fixed) value of the MIDI channel and the (fixed) value of the *first* data byte of the message, if any.

Figure 3 summarizes the MIDI conversions supported in the latest TouchOSC version. The MIDI message type numbers in the first column are as given inside the XML layout file. (TouchOSC uses its own encoding for the message types which has nothing to do with the actual MIDI status bytes of these messages.)

Note that while it's possible to map a variable to the velocity of a note or the value of a control change message, you cannot map it to the note number or MIDI controller number. While this kind of setup might occasionally be useful, TouchOSC doesn't allow it. Still it's possible to implement most kinds of controller configurations, such as mixer interfaces, DJ controls and even MIDI keyboards without much trouble.

The (OSC) source value for a MIDI control can be any of the x, y, z and c status variables. In a multi-control widget, each of the subcontrols has its own MIDI assignments. The TouchOSC editor allows you to pick those from a dropdown list in the MIDI properties (in the left side pane of the editor) after selecting a widget.

Note that it's possible to map the color (c) variable as well, so that you can change the color of a widget by sending a corresponding MIDI message. In this case the MIDI value range is fixed at 0–8, where 0 denotes `red`, 1 `green`, etc. Another special case that deserves mentioning are mappings of the page messages (`/1`, `/2`, etc. in OSC). You can map any of the MIDI voice messages to a given page, so that a MIDI message will be emitted if the user switches tabs on the device, and the current page will be switched when the MIDI message is sent to the device.

Our implementation fully supports all types of MIDI assignments described above. Note, however, that in order to receive touch messages (z variable), the corresponding OSC message type must be enabled in TouchOSC's OSC configuration dialog.

4 Interfacing TouchOSC and Pd

Our TouchOSC MIDI Bridge does its job by converting OSC messages from/to MIDI and thus a working OSC connection between Pd and the device running TouchOSC is required. While Pd doesn't offer any built-in OSC support, this can easily be added by means of corresponding Pd externals (plugins). Two well-known external libraries for this purpose are OSCx and mrpeach. These are both included in

Pd distribution packages such as Pd-Extended[10] and Pd-L2Ork[11]. The mrpeach externals offer additional features, such as the ability to access the source address of an incoming OSC message which is useful in order to set up bidirectional communication in an automatic fashion. Our sample patches employ this feature and are thus written using the mrpeach externals.

To make these facilities available, you only need to make sure that you have the mrpeach externals installed and on your Pd library search path. This should already be the case if you're running Pd-Extended or Pd-L2Ork. The only other required setup is to verify your TouchOSC configuration on the device. In the OSC setup, the host address should be set to the IP address of the computer running Pd (under Linux, you can find this by running the `ifconfig` program). You should also verify that the outgoing and incoming ports in the TouchOSC configuration are set to 8000 and 9000, respectively, since these are the default values our sample patches assume. These port numbers match the TouchOSC defaults, however, so chances are that you only need to enter the correct host address on the device.[12]

5 The MIDI Bridge

Our TouchOSC MIDI bridge is distributed in the form of a Pd external library called `touchosc` which implements two objects `tomidi` and `toosc`. Both objects take the name of a TouchOSC layout as their single creation argument. Thus, for instance, to have OSC messages converted to MIDI using the MIDI assignments in a layout named `sample.touchosc`, you'd create the object in Pd as `tomidi sample`. To make this work, the layout file needs to be in the same directory as the Pd patch containing the object. You can also use a full path name including the `.touchosc` extension, enclosed in double quotes, as in `tomidi "/some/path/sample.touchosc"`.

The operation of the Pd objects is fairly straightforward and doesn't require any additional configuration. Both objects provide a single inlet and a single outlet. The `tomidi` object takes OSC messages on its inlet and produces the corresponding MIDI messages on its outlet. The `toosc` object does the reverse operation, mapping MIDI messages to their OSC counterparts. The conversion is fully automatic once you've configured the MIDI mappings in your TouchOSC layout. No manual processing of OSC messages is required.

OSC messages are represented in the same symbolic format that's also used by the OSCx and mrpeach externals, so the output of `unpackOSC` (mrpeach) or `dumpOSC` (OSCx) can be piped directly into `tomidi`, while the output of `toosc` is ready to be used with mrpeach's `packOSC` or OSCx's `sendOSC`.

MIDI messages are also encoded in a symbolic format, i.e., as Pd meta messages. As Pd doesn't have a standard representation of MIDI messages other than as a numbers graveyard, we invented our own, but it's fairly straightforward if you're familiar with Pd's objects for MIDI input and output. A summary of the message syntax can be found in Figure 4. The format and, in particular, the somewhat idiosyncratic order of arguments has been designed so that it's easy to dispatch on the different message types using a Pd `route` object and pass the remaining data to the corresponding MIDI output objects. Conversely, MIDI messages can be received from Pd's MIDI input objects and converted to our format by just packing together the data and tacking on the proper message selector. Two helper patches `midi-input` and `midi-output` are included in the distribution to do this.

The distribution also includes a helper patch named `touchosc-bridge` (cf. Figure 5) which takes care of all the nitty-gritty details of set-

Message Type	Format
control change	`ctl` v n c
note	`note` n v c
program change	`pgm` n c
key pressure	`polytouch` v n c
channel pressure	`touch` v c
pitch bend	`bend` v c
start	`start`
stop	`stop`
continue	`cont`

Figure 4: MIDI representation of the Pd TouchOSC bridge. n denotes the note or controller number, v the velocity or controller value, c the MIDI channel number.

[10]http://puredata.info/downloads/pd-extended

[11]http://puredata.info/downloads/Pd-L2Ork

[12]TouchOSC also supports Zeroconf, which is implemented in the latest versions of our software as well. This makes it much easier to set up the network connections, see Section 6. But if necessary you can also configure the network connection by manually entering IP addresses and port numbers as explained above.

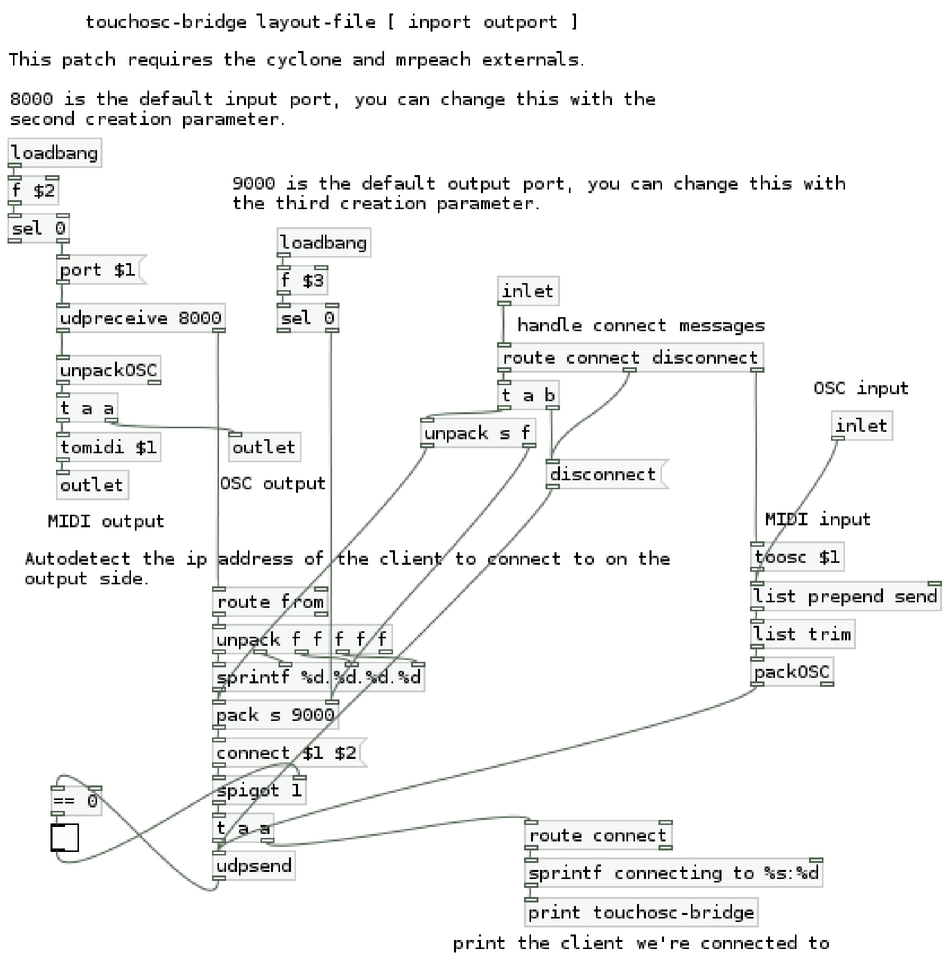
touchosc-bridge layout-file [inport outport]
This patch requires the cyclone and mrpeach externals.
8000 is the default input port, you can change this with the second creation parameter.
loadbang
f $2
sel 0
port $1
udpreceive 8000
unpackOSC
t a a
tomidi $1
outlet
MIDI output
outlet
OSC output
9000 is the default output port, you can change this with the third creation parameter.
loadbang
f $3
sel 0
inlet
handle connect messages
route connect disconnect
t a b
unpack s f
disconnect
OSC input
inlet
MIDI input
toosc $1
list prepend send
list trim
packOSC
Autodetect the ip address of the client to connect to on the output side.
route from
unpack f f f f f
sprintf %d.%d.%d.%d
pack s 9000
connect $1 $2
== 0
spigot 1
t a a
udpsend
route connect
sprintf connecting to %s:%d
print touchosc-bridge
print the client we're connected to

Figure 5: `touchosc-bridge` patch (simplified version).

ting up incoming and outgoing OSC connections, and provides a pair of `tomidi` and `toosc` objects to handle conversions in both directions. We describe this in the following section, where we cover the installation and usage of the MIDI bridge with Pd.

The latest versions of the `touchosc` library also include a third `oscbrowser` object which lets you discover available OSC clients, and can also publish its own OSC service using Zeroconf. This object is used to implement the Zeroconf support in the `touchosc-bridge` patch, but will also be useful in its own right for Pd users who wish to implement OSC applications; please check the pd-touchosc sources [3] for details.

6 Installation and Usage

Our TouchOSC MIDI bridge library for Pd is written in the author's Pure programming language [1], so you first need to install the Pure interpreter along with the pd-pure, pure-stldict and pure-xml addon modules. The Pure website will tell you how to do this. Binary packages for various popular Linux distributions such as Arch, Fedora and Ubuntu are also available, as well as ports for Mac OS X and BSD systems. Note that the pd-pure module is required to run any Pure externals with Pd, and needs to be enabled in Pd; please check the pd-pure documentation for details [2].

Next, to install our TouchOSC MIDI bridge, go find the pd-touchosc repository on Bitbucket [3] and clone the repository, or download it as a zip archive and extract it on your hard disk. At the repository website you can also find detailed installation instructions in the `README.md` file. Basically, you'll have to chdir to the source directory and run `make && sudo make install`. If you have Pd and all the other requisite software installed, this should build the external library and install it under your `/usr/lib/pd/extra` directory, along with some helper patches and examples.[13] Then fire up Pd and add `touchosc` to your startup libraries. Next time you start up Pd you should see a message in the Pd console showing that the `touchosc` library was loaded and registered with pd-pure.

Last but not least, you'll need TouchOSC, of course. You can grab the mobile app on Google Play or the iTunes Store and install it on your Android or iOS device. The TouchOSC editor can be downloaded for free on the TouchOSC website; it's a Java program, so you need to have a suitable Java runtime installed to use it.

The recommended way to run pd-touchosc is via the included `touchosc-bridge` helper patch. This patch sets up a pair of `tomidi` and `toosc` objects along with all the required OSC input and output machinery to connect with TouchOSC. The current version of the `touchosc-bridge` patch is depicted in Figure 5.[14] The patch is normally invoked with a single creation argument, the TouchOSC layout to use (in the same format as described in the previous section). Optionally, you can also configure the TouchOSC UDP ports by specifying these as the second and third argument.

The patch also detects the source IP address of incoming OSC messages and connects its output to it, so that after sending some data from the device the reverse connection should also work in an automatic fashion. Note that TouchOSC has an option which makes it send out OSC `/ping` messages in regular time intervals. If you enable this option then the `touchosc-bridge` patch will automatically set up its output connection as soon as it receives the first `/ping` message from the device.

Getting the network connections set up is even easier with Zeroconf. The latest version of pd-touchosc supports this via the Avahi Zeroconf daemon available for Linux and other Unix-like systems.[15] If you have Avahi installed and its daemon running, a client named `pd-touchosc` should show up in the host list in TouchOSC's OSC configuration dialog. Click on that to have the network address and port number filled in. On the host side, the full version of the `touchosc-bridge` patch offers the option to browse for available OSC services using Zeroconf. There's a toggle which lets you enable this; `touchosc-bridge` will then connect to the first OSC service available in the network (other than `pd-touchosc` itself). If there's more than one such service, you can cycle through the available services with the other GUI controls of the patch; see Figure 6. E.g., if you're running the Android version of TouchOSC then you'll have to look out for services named `Android (TouchOSC)` or similar (depending on which `ZeroConf Name` you chose in TouchOSC's OSC configuration) and pick the one that you want.

The left inlet/outlet pair of the patch is for sending MIDI messages to and receiving them from the device. In addition, the right inlet/outlet pair can be used to send and receive untranslated OSC messages. If you connect the left inlet and outlet to the `midi-input` and `midi-output` patches provided in the distribution, and route Pd's MIDI inputs and outputs to your MIDI devices and/or applications as needed, you should be able to set up Pd as a simple TouchOSC MIDI bridge with little effort. Or, if your computer music application is implemented as a Pd patch, you can use `touchosc-bridge` directly in your patch and hook it up to your existing control logic.

Figure 6 shows how `touchosc-bridge` can be employed in a simple test patch. Several sample patches and corresponding TouchOSC layouts are also included in the distribution. To get started, just download the sample layouts to your device, open the corresponding patches with Pd and kick the tires to see how things work.

7 Conclusion

We presented a TouchOSC MIDI bridge implementation which works on Linux, running inside Miller Puckette's Pd environment. This software allows you to convert between TouchOSC-formatted OSC and MIDI messages, following the MIDI mappings defined in a TouchOSC layout. It offers pretty much the same functionality as the official TouchOSC MIDI Bridge application, which is only supported on Mac and Windows at this time. The main differences to hexler's bridge are that it requires an actual OSC connection to the device and the TouchOSC layout file on the host side to work.

[13]This will work with vanilla Pd. For Pd-Extended and Pd-L2Ork you'll have to specify the Pd flavor using the `PD` make variable, e.g.: `make PD=pd-extended && sudo make install PD=pd-extended`.

[14]For the sake of clarity, the figure actually shows a simplified version of the patch, available in the distribution as `touchosc-bridge-simple.pd`, which doesn't include Zeroconf support. The full version of the patch can be found in the distribution as `touchosc-bridge.pd`.

[15]`http://avahi.org/`

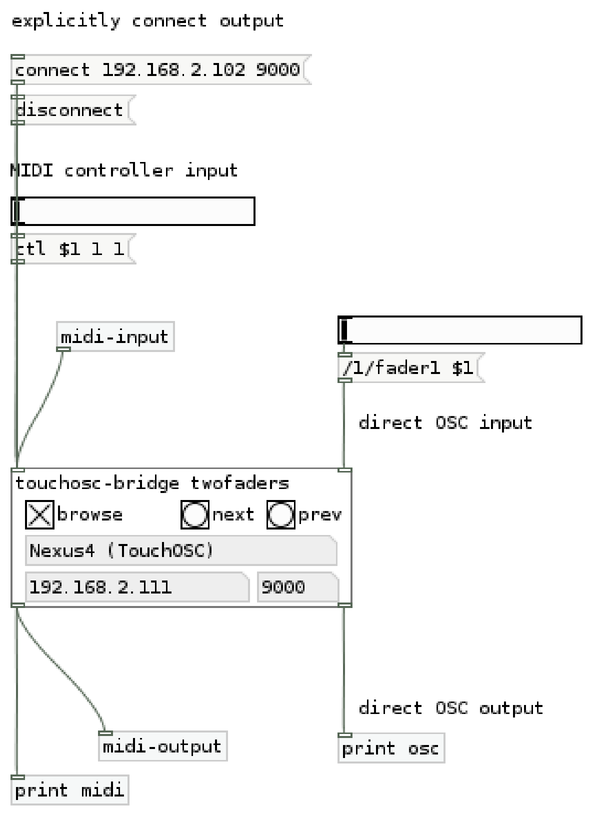

Figure 6: Sample patch using the `touchosc-bridge` abstraction.

To the author's knowledge, this is the first (and at the time of this writing, the only) fully automatic TouchOSC MIDI bridge application on Linux. Compared to the "real" TouchOSC MIDI Bridge, it also has the advantage that it is available under an open source license and doesn't rely on any proprietary and undocumented protocols, so it can easily be customized by the user.

Future work should be directed towards turning the software into a stand-alone program which can be run more easily by non-Pd users. In principle, it should also be possible to adjust our implementation to other OSC applications such as Control and the Lemur, although this will require some refactoring of the layout parsing code.

References

[1] A. Gräf. Signal processing in the Pure programming language. In *Proceedings of the 7th International Linux Audio Conference*, Parma, 2009. Casa della Musica.

[2] A. Gräf. pd-pure: Pd loader for Pure scripts. `http://puredocs.bitbucket.org/pd-pure.html`, 2014.

[3] A. Gräf. pd-touchosc: TouchOSC MIDI Bridge for Pd. `https://bitbucket.org/agraef/pd-touchosc`, 2014.

[4] hexler. TouchOSC: Modular OSC and MIDI control surface. `http://hexler.net/software/touchosc`, 2014.

[5] JazzMutant. Multitouch controllers for audio production, live music and media performance. `http://www.jazzmutant.com`, 2014.