nmj

Network MIDI

for Android and desktop Java

What is this?

nmj is a pure Java, cross-platform MIDI over ethernet implementation compatible with

- RTP MIDI (as specified in RFC 4695 and implemented in Apple's Network MIDI in OS X)
- ipMIDI (and other “raw” solutions like Midi2Net)

nmj implements the javax.sound.midi Service Provider Interface (SPI) and also brings a small public API for platforms that do not support javax.sound like Android.
nmj will instantly supply additional “MIDI devices” to java applications on Windows, Linux and OS X soon as the library is placed on the classpath. It may be suitable for end-users to a certain degree. In principle though, it is in first hand intended for Java developers looking for an easy way to add network MIDI functionality to their applications.

What else do I need?

Depends on where you are running / whom you wish to talk to:

To interconnect two java programs that both have access to the nmj libraries nothing more is needed regardless of the underlying platform.

On Mac OS X 10.4 and greater RTP MIDI is supported through Apple's “Network” Midi implementation, so a program using nmj can talk to all native MIDI programs on Macs running Tiger or greater without further installations. (This is also available in iOS 4.2).
For raw multicast communication you will need to install ipMidi (from http://www.nerds.de).

On Windows you will need to install a native RTP MIDI driver (available from http://www.tobias-erichsen.de/rtpMIDI.html) or ipMidi (from http://www.nerds.de) to add ethernet MIDI ports to native applications.

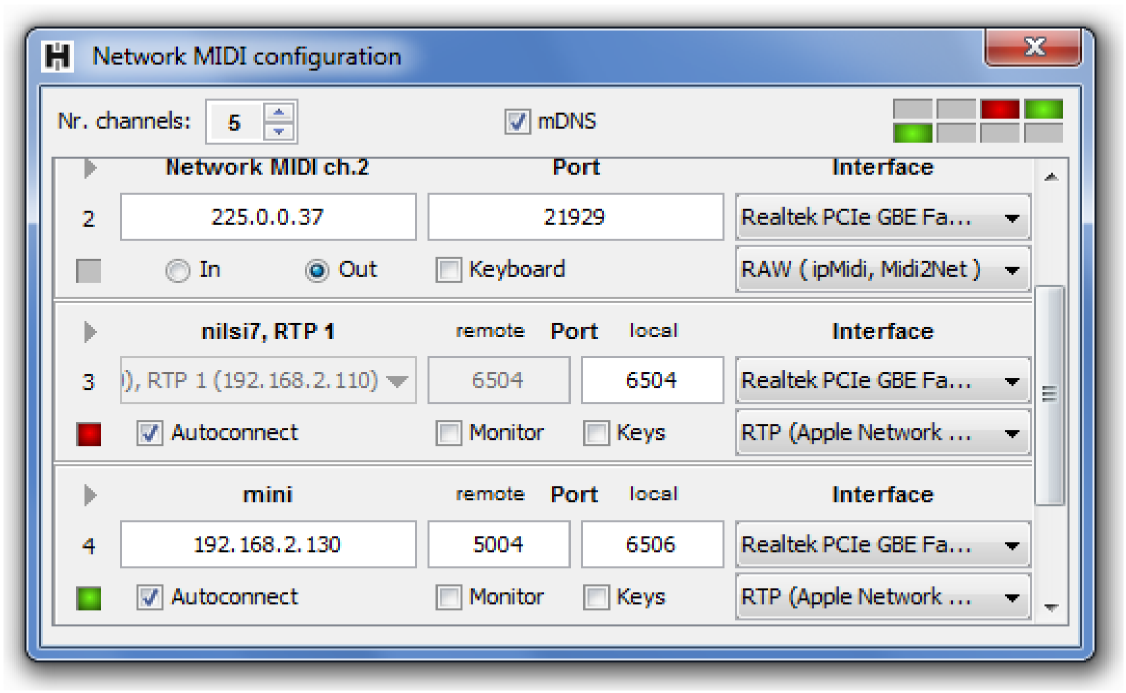
On Linux the multimidicast library (http://llg.cubic.org/tools/multimidicast/) adds multicast MIDI functionality to ALSA based software.

KissBox hardware interfaces (http://www.kissbox.nl/products_new.html#midi) should be directly addressable without further installations from any platform (not tested, though. Expensive stuff).

On all platforms it is relatively straight forward to create a simple “MIDI through” application to relay MIDI data from the network to native loopback devices (like MidiYoke on Windows or the IAC Bus on OS X). In principle though, this should not be necessary.

How to get started?

First thing you may want to do is looking at the library's built-in control panel. This is accessible by double-clicking the nmj jar file or - in desktop java - through the NMJConfig.showConfigDialog(...) API method, resp. invoking it as a separate activity in Android. It will bring up a dialog as shown below.



Network MIDI configuration

Nr. channels: 5 ☑ mDNS

Network MIDI ch.2 | Port | Interface
2 | 225.0.0.37 | 21929 | Realtek PCIe GBE Fa...
◯ In [✓ ◉ Out] ☐ Keyboard | RAW (ipMidi, Midi2Net)

nilsi7, RTP 1 | remote Port local | Interface
3 |), RTP 1 (192.168.2.110) | 6504 | 6504 | Realtek PCIe GBE Fa...
🟥 ☑ Autoconnect ☐ Monitor ☐ Keys | RTP (Apple Network ...

mini | remote Port local | Interface
4 | 192.168.2.130 | 5004 | 6506 | Realtek PCIe GBE Fa...
🟩 ☑ Autoconnect ☐ Monitor ☐ Keys | RTP (Apple Network ...

Before going through this one by one, please note that all this dialog does is calling public methods in the nmj API (namely the NMJConfig class) so there is nothing that will ever enforce you to use it or present it to users. If you don't like it (which would be totally understandable) you are free to roll your own or do all configuration in code alone. Every GUI element used here has a corresponding method in NMJConfig – the IP Address field calls NMJConfig.get/setIP(int channel, String IP), the remote port field uses NMJConfig.get/setPort(int channel, int port) etc. For more details see the javadocs.

The dialog nevertheless will be useful to explain the library's basic workings:

Starting top left, you see a spinner box that lets you set the number of devices that the library will provide to applications. As these are purely virtual you can add or delete them any time. When querying javax.sound for MidiDevice.Info objects you will see the corresponding ammount of nmj device infos in the returned array. In theory the maximum number of devices is only limited by available network ports.

Next comes the multicast DNS service discovery option. This is enabled by default and will use the standardized multicast DNS technology (aka Bonjour) to find and configure RTP Midi services on local networks. Discovered servers will automatically have additional channels created for. Note that multicast DNS has nothing to do with MIDI communication as such (or with nmj multicast channels), it is merely used for automatic device configuration.

The upper right hand holds an indicator showing which device is currently active / opened or present.

Below these come the individual device configuration panels. When first starting the library you will see two multicast channels configured to work with ipMidi's first two channels plus one local RTP Session. Additionally – as multicast DNS is active by default – newly discovered RTP channels may be

added.
Devices have some common and – depending on their operating mode – some controls that may vary. Common to all of them are settings for the remote host's IP address and port (i.e. the address of the machine or multicast group the device will connect to) and the network interface to use. The latter will be automatically preselected and should not normally need to be changed, but the option may be useful on multi-homed systems.
Underneath the interface selector you will find the channel mode switch, which lets you set a channel to either “raw” or RTP mode. Changing this will change some of the channel's controls, so let's look at these operation modes next:

“**raw**” mode has two further options (depending on the remote IP address):

Multicast:
In a multicast group all members will see all the data that any member sends to the group as well as being able to send to all other group members simultaneously. There is no individual addressing, data is sent to a virtual address (IP4 ranges higher than 224.0.0.1) where it will be picked up by everybody listening. The advantage of this is, that data does not need to be sent multiple times for every client involved and that you do not need any further configuration or additional handshaking between clients which makes it really easy to set up. On the other hand it also will not supply feedback when some client leaves the group, so it is kind of a “fire and forget” approach and it also will only ever work on the local LAN.
Multicast is used by the native ipMidi solution (and its Linux equivalent midimulticast).

Multicast channels in nmj are unidirectional, that is they are either inputs or outputs (otherwise MIDI feedback may occur), so you will see Input and Output selectors that are mutually exclusive (select Input and the Output switch will go off and vice versa). Depending on whether a channel is set to In- or Output there will be a “Monitor” or “Keyboard” selector on the channel. More on these later.

Unicast:
This creates a direct, hardwired connection to a specified machine. Unicast connections can be used between two nmj clients, between nmj and Toby Bear's Midi2Net and Net2Midi VSTs and probably with other solutions alike. Unicast connections can be bidirectional, given both parties connect to the correct ports. They will therefore show an additional “local port” field in the control panel., the “Input / Output” switches will both be selectable at any time and both “Monitor” and “Keyboard” switches will be available.
To set up a unicast device you would:
on the local machine:

1. enter the IP address of the remote machine
2. enter the port the remote machine lists under “local port” in the “remote port” field

and on the remote machine:

1. enter your local machine's IP address
2. use the port your local machine lists for “local port” under “remote port”.

RTP Midi:

This uses the “Realtime Transport Protocol” for MIDI data and an Apple proprietary session establishment and maintenance protocol. As for all other RTP payload types the way data – MIDI in this case - is to be packetiized into RTP has been standardized through an IETF effort, namely RFC 4695. The standard's first serious implementation appeared in Mac OS X 10.4 (Tiger) as the “Network” MIDI driver. These days RTP MIDI seems to become more popular with Windows drivers and proprietary hardware becoming available. In theory RTP MIDI is a lot more capable than the raw approach described before as it provides means to recover from packet loss. The protocol itself and these additional capabilities however also produce quite a large overhead, plus the session maintenance requires continuous additional handshaking on a separate connection, which altogether makes implementations a bit more complex. Whether or not these additional capabilities are really needed for local area network MIDI connections is somewhat questionable, although they probably make perfect sense when running over the internet. nmj 0_8_3 adds receiver side support for the RTP MIDI journaling mechanism for packet recreation. Please see javadocs for details..

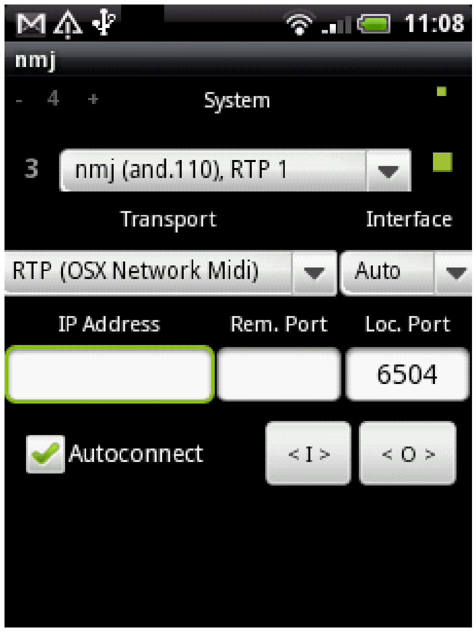
nmj's RTP MIDI channels are always bidirectional, so you will not see “Input / Output” switches with them. Instead there is a check-box labeled “Autoconnect” , which will make nmj initiate the connection process soon as the device is opened. With this option turned off the opposing party is expected to initialize the connection (the nmj device needs to be opened nevertheless).
As with raw unicast devices you will find both “Monitor” and “Keyboard” switches.

RTP MIDI works with a concept of sessions. A machine may offer one or more sessions to others on the network to connect to. Sessions are then established via an invitation / acceptance sequence where either side may initiate the process. The various aspects of establishing connections are probably best explained along with some screenshots. Please see below.

In order to establish a connection via RTP the opposing side must be fully available and ready to answer to “invitations” (which is different with multicast and raw unicast, where everybody can join the group or connection at any time). This will not be the case before either some program using the Network MIDI sessions (built in on OS X or rtpMidi on Windows) is running or the rtpMidi control panel / “Audio MIDI Setup” utility's MIDI page have been brought up. Both the latter will also let you create connections and monitor latency.

RTP connection processes illustrated:

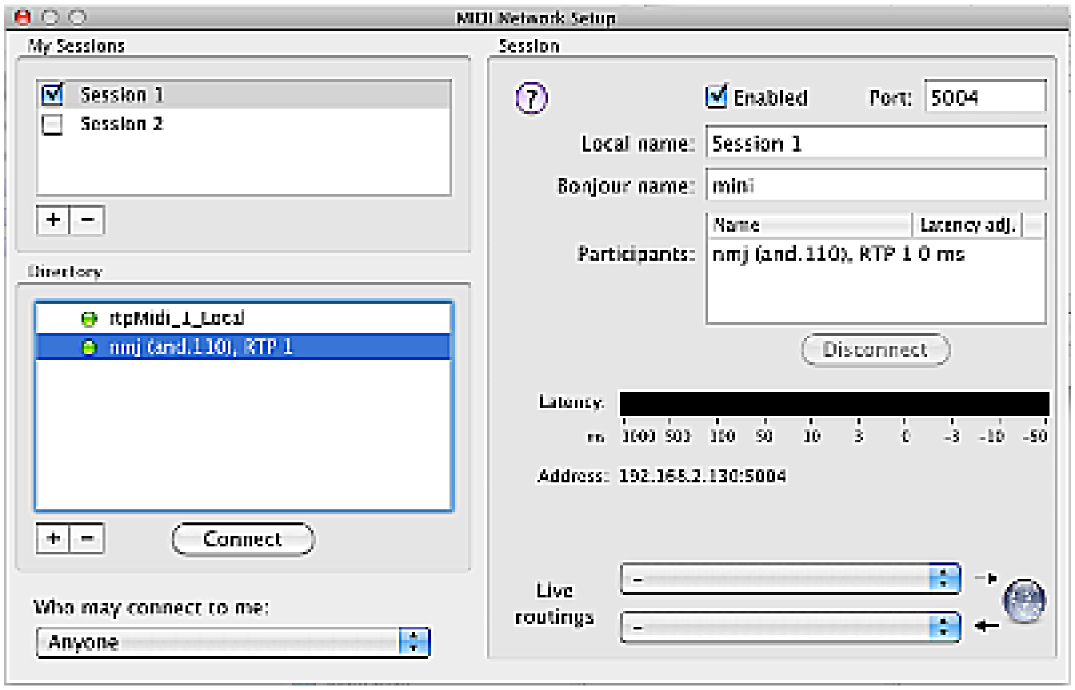
Scenario 1, connecting a local session (shown for an Android device, but similar on desktop OSs)**:**


nmj
System
3 nmj (and.110), RTP 1
Transport: RTP (OSX Network Midi)
Interface: Auto
IP Address:
Rem. Port:
Loc. Port: 6504
✓ Autoconnect

Step 1:

This shows the first RTP Session that nmj will create by default. The session will be announced to other machines on the network via DNS, but will not have entries for remote IP and port (NMJConfig.getIP(int channel) will return null). Sessions like this – with a null remote IP - are considered local sessions. If set to “auto-connect”, they will be activated at startup and then be listening for invitations.

At this point no MIDI client (on the Android device) may ever have touched the channel, but it is equally possible that it has already been opened MIDI-wise. However there won't be any data reaching the MIDI client long as the RTP session is not connected anywhere.


MIDI Network Setup

My Sessions

- [✓] Session 1
- [ ] Session 2

Directory

- rtpMidi_1_Local
- nmj (and.110), RTP 1

Who may connect to me: Anyone

Session

[✓] Enabled Port: 5004

Local name: Session 1

Bonjour name: mini

Participants:

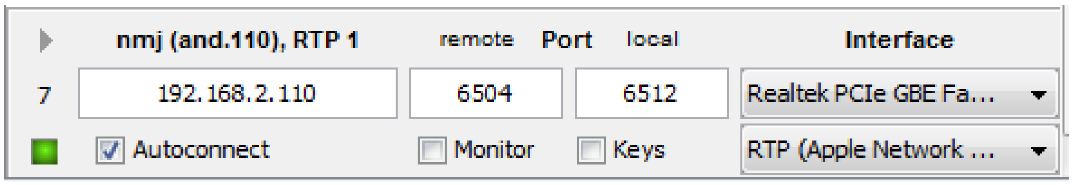
Name	Latency adj.
nmj (and.110), RTP 1	0 ms

Address: 192.168.2.130:5004

Step 2:

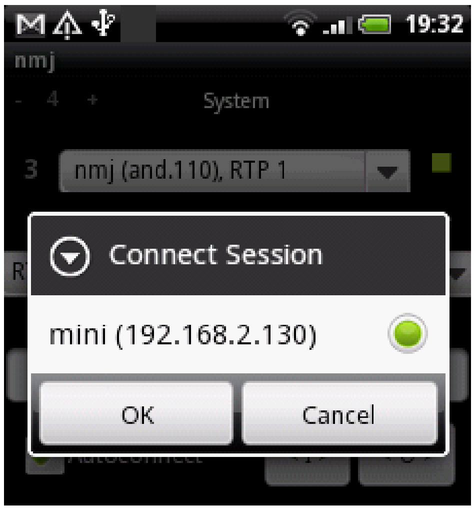
Variant A – remote client initiating the connection:

A second machine on the network has picked up the session announcement and now lists the session in its “directory” listing (this is exactly the same in OS X's “Audio MIDI Setup” utility and rtpMIDI's control panel. The next image shows the discovered session in nmj's control panel).


	nmj (and.110), RTP 1	remote	Port	local	Interface
7	192.168.2.110	6504		6512	Realtek PCIe GBE Fa…
🟩	[✓] Autoconnect	[ ] Monitor		[ ] Keys	RTP (Apple Network …

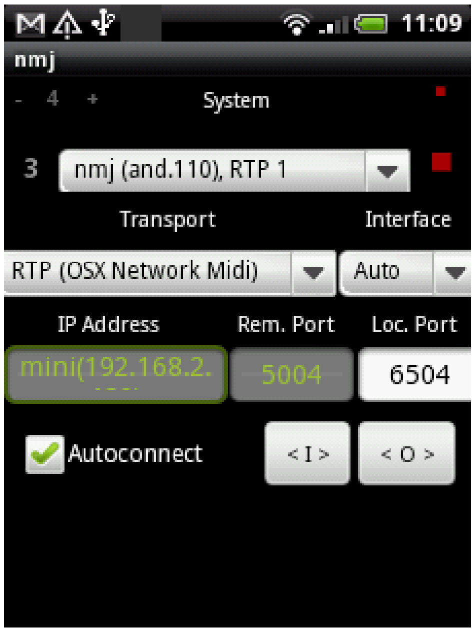
To create the connection you will then need to select the session in the directory listing and click the “connect” button or – when using nmj and the default auto-connection option is enabled– just open the channel corresponding to the discovered session for MIDI. In rtpMIDI and OS X you will then see the session listed as a “Participant” on the right hand side. In nmj's control panel the state indicator would turn to red (in reaction to a SystemListener callback).

Step 2, Variant B – initiating the connection from the device:



In order to be able to send out invitations, the device needs to know about remote sessions to invite, so this requires that DNS has found services on other machines or channels have been added manually.

Given this is the case a local session can be connected and disconnected by long-clicking the (empty) IP Address field and selecting a client from the upcoming dialog. In the desktop version one would use the pulldown menu on the right hand side of the address field. In both cases the NMJConfig.connectLocalSession(..) method will be called to do the job.


11:09

nmj

System

3 nmj (and.110), RTP 1

Transport: RTP (OSX Network Midi)

Interface: Auto

IP Address: mini(192.168.2.

Rem. Port: 5004

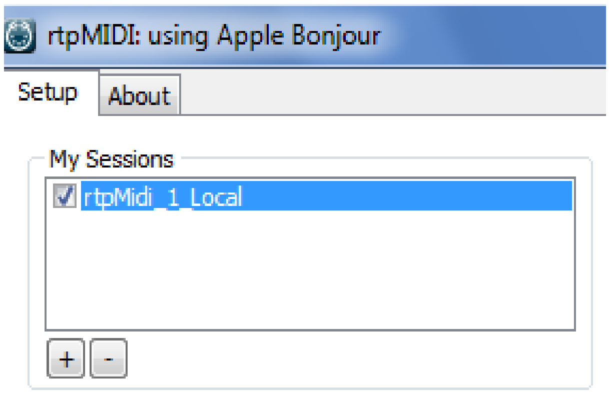
Loc. Port: 6504

Autoconnect ✓

Step 3:

No step 3. At this point the session is established and ready for MIDI communication. The Android device that originally announced its local session, would look like this now, with IP and port fields showing the connected client.

Scenario 2, connecting nmj to a session offered by rtpMIDI / OS X:

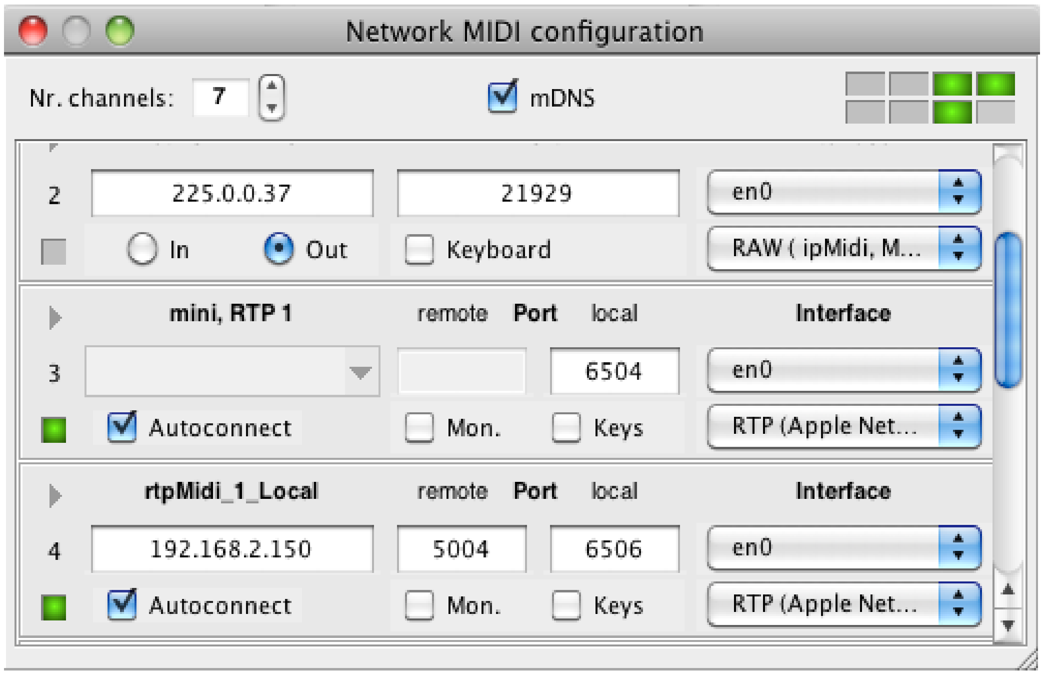

rtpMIDI: using Apple Bonjour

My Sessions

☑ rtpMidi_1_Local

Step 1:

Create a session by clicking the “+” button underneath the “My Sessions” field, give it a name (optionally) and make sure it is enabled.


Network MIDI configuration

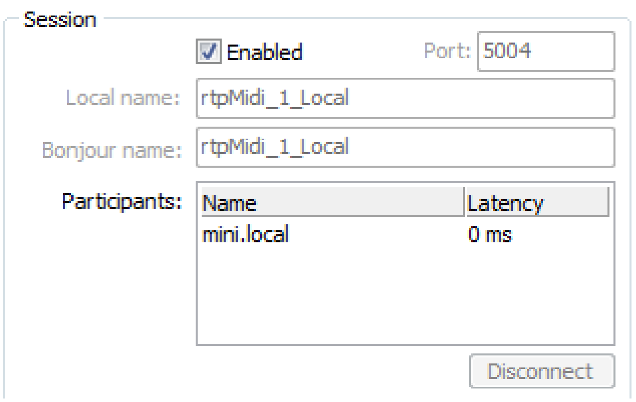
Nr. channels: 7 ☑ mDNS

2	225.0.0.37	21929		en0
	○ In [✓ ● Out]	☐ Keyboard		RAW (ipMidi, M...
	mini, RTP 1	remote	Port local	Interface
3			6504	en0
🟩	☑ Autoconnect	☐ Mon.	☐ Keys	RTP (Apple Net...
	rtpMidi_1_Local	remote	Port local	Interface
4	192.168.2.150	5004	6506	en0
🟩	☑ Autoconnect	☐ Mon.	☐ Keys	RTP (Apple Net...

Step 2:

After a short moment the session will have been discovered by DNS and a channel been added for it (or - if it had been present before and only got rediscovered - the channel's state would change to PRESENT and SystemListeners would receive a callback).
If you are hit by Android multicast problems, then you would need to create the session entry manually: Use the spinner box to increase the number of channels by one and edit IP and port settings.

Step 3:
Open the corresponding MIDI channel from an application or by checking the Monitor / Keyboard boxes and – with “auto-connect” enabled – the session will be connected. The client will again be listed under “Participants” in rtpMIDI and OS X and nmj will turn its indicator to red.
Given “auto-connect” is disabled then the other party would need to send the session invitation (which would work as described under step 2 of the previous scenario).


Session

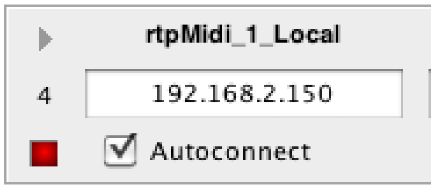
☑ Enabled Port: 5004

Local name: rtpMidi_1_Local

Bonjour name: rtpMidi_1_Local

Participants:

Name	Latency
mini.local	0 ms

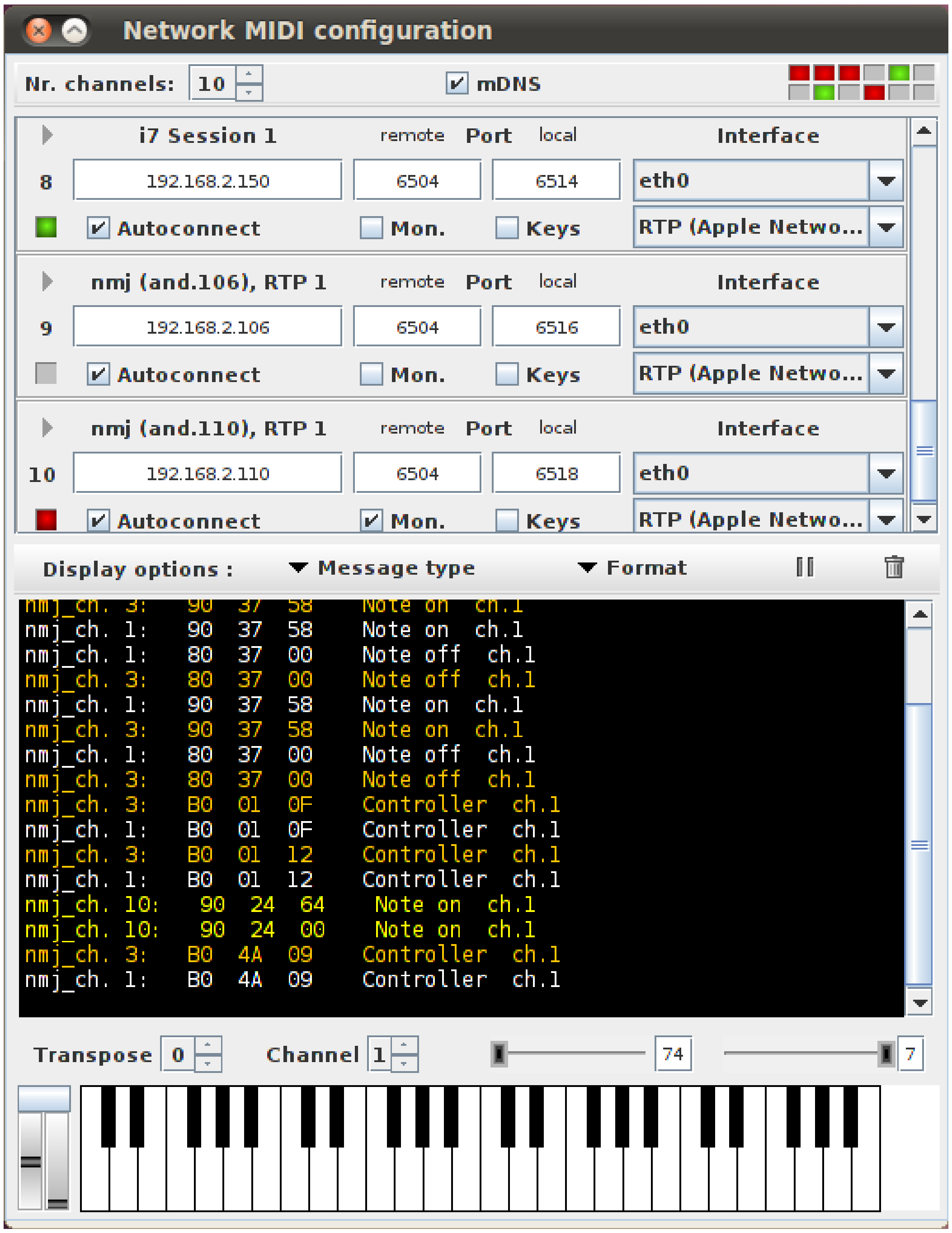

rtpMidi_1_Local

4 192.168.2.150

🟥 ☑ Autoconnect

Keyboard and Monitor (I & O on Android)

While the control panel is in first hand made for quick changes in device settings it can also open devices and attach to devices already opened by the application. The “Keyboard” (O) switch will open a device for output and let you play MIDI notes and some controllers to the device, while the “Monitor” (I) switch opens an input or attaches to one already opened to look at incoming MIDI data. On Android Keyboard and Monitor are separate views accessible via horizontal wipe gestures.
When using any of these options from inside an application make sure to close what you opened as devices will not exit long as they still have listeners attached and can prevent an application from shutting down properly.


Network MIDI configuration

Nr. channels: 10 ✓ mDNS

	Session	remote Port	local Port	Interface
8	i7 Session 1			
	192.168.2.150	6504	6514	eth0
🟩	✓ Autoconnect	Mon.	Keys	RTP (Apple Netwo...
9	nmj (and.106), RTP 1			
	192.168.2.106	6504	6516	eth0
▨	✓ Autoconnect	Mon.	Keys	RTP (Apple Netwo...
10	nmj (and.110), RTP 1			
	192.168.2.110	6504	6518	eth0
🟥	✓ Autoconnect	✓ Mon.	Keys	RTP (Apple Netwo...

Display options : Message type Format

```
nmj_ch. 3:   90  37  58   Note on  ch.1
nmj_ch. 1:   90  37  58   Note on  ch.1
nmj_ch. 1:   80  37  00   Note off  ch.1
nmj_ch. 3:   80  37  00   Note off  ch.1
nmj_ch. 1:   90  37  58   Note on  ch.1
nmj_ch. 3:   90  37  58   Note on  ch.1
nmj_ch. 1:   80  37  00   Note off  ch.1
nmj_ch. 3:   80  37  00   Note off  ch.1
nmj_ch. 3:   B0  01  0F   Controller  ch.1
nmj_ch. 1:   B0  01  0F   Controller  ch.1
nmj_ch. 3:   B0  01  12   Controller  ch.1
nmj_ch. 1:   B0  01  12   Controller  ch.1
nmj_ch. 10:   90  24  64   Note on  ch.1
nmj_ch. 10:   90  24  00   Note on  ch.1
nmj_ch. 3:   B0  4A  09   Controller  ch.1
nmj_ch. 1:   B0  4A  09   Controller  ch.1
```

Transpose 0 Channel 1 74 7

Known issues and limitations

- On Android there are known problems with receiving multicast communication that may arise depending on both the Android version (It “should” be all fine in Froyo, 2.2) and the device's firmware (HTC seem to disable inbound multicast as a “battery saver”, actively fragmenting the platform more than it already is. This is still the case in Froyo). Sending multicast data is not a problem, however not being able to receive multicast also means that DNS functionality will not work and you will need to manually add sessions provided by other machines on the network.

- nmj's local RTP sessions will only accept a single client at the time being.

- nmj's multicast DNS implementation is likely not fully waterproof so far and may fail with large setups, lots of session etc.

License

Contact

Send feedback, questions, suggestions, bug reports etc. to
nmj at humatic de

Version history

0.83
Added additional output methods for sending multiple MIDI messages in one go
Fixed bugs with large sysex dumps
Implemented receiver side support for RTP packet loss recovery.

0.82
Fixed bugs in multicast implementation that truncated CoreMIDI packet lists and did not handle some non 2 databyte message types correctly.

Desktop: added option to store settings under application specific nodes

0.81
Android: Tested on more hardware. Overcomes some assumptions on interface naming conventions that proved to be too naïve. Throws exceptions on network setup errors, that were previously caught.
No changes in the desktop version

0.8
initial public release