Medusa - A Distributed Sound Environment

Flávio Luiz SCHIAVONI[1] and **Marcelo QUEIROZ**[1] and **Fernando IAZZETTA**[2]
Computer Science Department[1], Music Department[2]
University of São Paulo
Brazil
{fls,mqz}@ime.usp.br, iazzetta@usp.br

Abstract

This paper introduces Medusa, a distributed sound environment that allows several machines connected in a local area network to share multiple streams of audio and MIDI, and to replace hardware mixers and also specialized multi-channel audio cables by network communication. Medusa has no centralized servers: any computer in the local environment may act as a server of audio/MIDI streams, and as client to remote audio/MIDI streams. Besides allowing audio and MIDI communication, Medusa acts as a distributed sound environment where networked sound resources can be transparently used and reconfigured as local resources. We discuss the implementation of Medusa in terms of desirable features, and report user experience with a group of composers from the University of São Paulo/Brazil.

Keywords

Network music, Jack, SCTP.

1 Introduction

With the growth of the Internet and the rise of broadband home links, the association between music making and networked computers had a global acceptance. With music distribution via streaming, computers became the new sound players, and also the new way of distributing music on a global scale. This kind of musical application of computers is not concerned about latency issues because communication is totally asynchronous. The interest in synchronous audio communication came with the idea of putting this technology to new uses [Wright, 2005].

Synchronous networked music communication research started with music performance experiments years ago. Some early network music performances are reported for instance in Bolot and García [Bolot and García, 1996] as early as 1996, using TCP, UDP and RTP to route voice signals; see [Weinberg, 2002], [Renaud et al., 2007] and [Barbosa, 2003] for surveys on network music performance.

A tool for network music interaction might be used to promote interaction on a global or a local scale. In a wide area network such as the Internet, the main concern is the attempt to bring people together across physical space, whereas in a local area network context, where participants are usually in the same room, the network can be used to promote a rich range of interaction possibilities, by using the virtual communication link as an extension of the shared physical space [Wright, 2005]. Technologically mediated communication brings significant contributions to musical interaction even when people are face-to-face, for instance by allowing much more control in processing and combining sound sources within a room with almost no interference of room acoustics.

These new possibilities can be explored by musicians, allowing them to create new musical approaches to composition and performance, by exploring new ways of interacting that exceed physical proximity and maximize musical possibilities. There is some expectation about what could or would be done with music when this kind of free networked intercommunication is allowed [Cáceres and Chafe, 2009a]. As noted by [Chafe et al., 2000], "once [the delay issue] can be pushed down to its theoretical limit, it will be interesting to see what musical possibilities can be made of truly interactive connections".

1.1 Goals and related work

This paper introduces Medusa, an audio/MIDI communication tool for local networks whose design is based on a set of desirable features, which have been collected from several previous works in Network Music Performance, Interactive Performance and Distributed Systems.

The main goal is to unleash audio/MIDI communication between computers and software applications on a local area network without complex configurations or difficult set-ups. This is done by mapping each sound source (or sound sink) in the network to a local name that the user may connect to any input (or output) of audio/MIDI software applications. The focus on local area networks allows the mapping of musician's expectations based on local/physical/acoustical musical interaction to new desirable features of the system, and then the mapping of these desirable features to details of the software model.

Several audio processing platforms allow some form of network communication of audio and MIDI data. PureData, for instance, allows the user to send and receive UDP messages between several Pd instances using netPD [1]. SuperCollider [2] is implemented with a client-server architecture and also allows network communication. The goal of Medusa, on the other hand, is to allow communication also between different software tools and across computer platforms.

Some related work address the problem of synchronous music communication between networked computers, such as OSC [Lazzaro and Wawrzynek, 2001], NetJack [Carôt et al., 2009], SoundJack [Carôt et al., 2006], JackTrip [Cáceres and Chafe, 2009b; Cáceres and Chafe, 2009a], eJamming [Renaud et al., 2007], Otherside [Anagnostopoulos, 2009] and LDAS [Sæbø and Svensson, 2006], including commercial applications such as ReWire from Propellerhead [Kit, 2010].

Although some of the goals and features of these applications may overlap with those of Medusa, none of them addresses the issues of peer-to-peer topology for audio and MIDI communication in the specific context of Local Area Networks. The OSC standard, for instance, uses symbolic messages (e.g. MIDI) to control remote synthesizers over IP [Lazzaro and Wawrzynek, 2001]; Otherside is another example of a tool which works only with MIDI. While Medusa is based on peer-to-peer connections, NetJack works with a star topology and master/slave approach [S.Letz et al., 2009], and so do LDAS, SoundJack and JackTrip. Some of these tools allow WAN connections, which leads to a different application context with several other kinds of problems like NAT routing, package loss, greater latency and need for audio compression, and at the same time they do not fully exploit the specificities of LAN connections, for instance reliable SCTP routing. Besides, one of Medusa's central goals is to go beyond audio and MIDI routing, by adding on-the-fly remote node reconfiguration capabilities that may help environment setup and tuning.

[1] http://www.netpd.org

[2] http://supercollider.sourceforge.net/

1.2 Design based on desirable features

In this paper, we will discuss an architectural approach to the design of a local network music tool which is based on desirables features, either found in previous work from the literature or in actual usage with a group of volunteer musicians. We will also present a prototype that was implemented to support some of the features mapped so far. The current list of desirable features guiding the development of Medusa is the following:

- Transparency
- Heterogeneity
- Graphical display of status and messages
  - Latency and communication status
  - Network status
  - Input/Output status
  - IO stream amplitudes
- Multiple IO information types
  - Audio
  - MIDI
  - Control Messages
  - User text messages
- Legacy software integration [Young, 2001]
  - Audio integration
  - MIDI integration
  - Control integration
- Sound processing capabilities [Chafe et al., 2000]
  - Master Mixer [Cáceres and Chafe, 2009a]
  - Silence Detection [Bolot and García, 1996]
  - Data compression [Chafe et al., 2000]
  - Loopback [Cáceres and Chafe, 2009a]

Transparency and **Heterogeneity** are desirable features borrowed from the field of distributed systems. Transparency's main idea is to provide network resources as if they were local resources in a straightforward way. Heterogeneity means that the system should be able to run on several system configurations within the network, including different OS and different hardware architectures, in an integrated manner. These concerns also appear in related works [Wright, 2005; Cáceres and Chafe, 2009a], and helped in the choice of a development framework (including programming language, API, sound server, etc.)

The features listed under **Graphical display of status and messages** were collected via experimentation with potential users (volunteer musicians), in a cyclic process of update and feedback of early versions of the prototype. These features are directly related to the graphical user interface.

The need to work with both **MIDI** and **Audio** was also presented by volunteer musicians, as they frequently combine audio connections with the use of remote MIDI controllers. **Control Messages** are used to access a remote machine, for instance to reconfigure its audio connections during a musical performance. Also **user text messages** may be used for various purposes including machine reconfiguration and performance synchronization.

The need to integrate the system with **legacy softwares** is evident as every user is used to work with particular sound processing applications. Like Heterogeneity, this feature also determines the choice of a development API.

Sound processing capabilities include a set of tools that relate to the issues of latency, bandwidth and heterogeneity. These features will be further discussed in the **Sound Communication** section.

2 Architectural Approach

The system presupposes a set of computers in a local area network that may share audio and MIDI channels. In the context of this paper, the group of all machines connected to Medusa is called **environment** and every machine in the environment is called a **node**. A node that makes resources available to the environment, such as audio or MIDI streams, is called a **source**, and a node that uses environmental resources is called a **sink**; every machine can act simultaneously as source and sink. Every node has independent settings, and each user can choose which resources he or she wants to make available to the environment, and also which environmental resources he or she wants to use, and when. The following subsections discuss the architectures of each node and of the environment.

2.1 Node Architecture

The node architecture is a multi-layered model that uses the following components:

- GUI: used for configuring the node and interacting with the environment. Environment interaction includes adding/removing local audio/MIDI ports and environmental node search and connection;
- Model: used to represent the node configuration, including audio and network configurations and their current status.
- Control: is responsible for integrating sound resources and network communication.
- Network communication: used for data and control communication with the environment;
- Sound resources: used to map local and environmental audio resources.

The **GUI** is the user interaction layer. It is used to set up the system, to create audio channel resources, to connect to the environment and to remote resources. The GUI brings some level of **transparency** to the environment and makes the tool easier to use, by hiding the complexity of actual network connections and network and audio settings. [Cáceres and Chafe, 2009b] already noted that usually most of the time is spent adjusting the connections rather than playing music, and our GUI was designed trying to alleviate this problem. The feature of **graphical display of status and messages** is implemented by this layer. The status of the network and active communication channels are presented as indicators that provide visual feedback, such as Latency, Network status, Input/Output status and Signal Amplitude, which help the user in interacting with the network music environment.

The **Model** layer represents each node current status at any given moment. It contains the network configuration, sound resources and coding details such as number of channels, sample rate, local buffer size and other relevant information. The model is encapsulated in messages to preserve consistency between machines. The set of models of all nodes represents the current environment status. Messages will be further explained in section 2.3.

The **Control** layer is the main part of the system. It is divided in three components: **Sound Control**, **Network Control** and **Environment Control**. These controls hide the implementation details from upper-level components, by taking care of audio synchronization, sound processing and message exchange to keep the model representation up-to-date across nodes. The Environment Control maintains an environment node list with all details of the nodes known at each moment. The Sound Control encapsulates the Sound Communication layer, allowing the sound server to be changed at any time. The Network control encapsulates the network servers and clients allowing a server reimplementation without the need for major code rewriting.

The **Network Communication** layer is responsible for the low-level maintenance of the network infrastructure. It connects sources and sinks to audio and MIDI streams and manages control messages within the environment. Broadcast control messages can be used to sync all nodes in the environment. Plain text messages between users can help them to set up his/her node or to exchange any other kind of information in a human-readable way. The network communication layer has three servers:

- UDP server: send/receive broadcast messages;
- TCP server: send/receive unicast messages;
- SCTP server: exchange audio/MIDI streams.

The **Sound Communication** layer is responsible for interacting locally with the sound server in each node, creating a virtual layer that provides transparent access to remote audio and MIDI streams, integrating the tool with other **legacy sound softwares**, while hiding the details of network communication. The idea behind integration with legacy softwares is to avoid having any type of signal processing units within the communication tool, leaving those tasks to external softwares through a sound server like Jack or SoundFlower. The architecture can integrate, via external software, many other sound processing capabilities that may be applied before sending a stream to the network or upon receiving a stream and before making it locally available. Signal processing can be used, for instance, to translate streams with different audio codings, by adjusting sample rate, sample format, buffer size and other coding details between different user configurations [Chafe et al., 2000], thus providing heterogeneous and transparent access to remote data. Signal processing units may also include:

Master Mixer: allows the user to independently control the volume of network audio and MIDI inputs, and also to mute them. It allows groups of network channels to be mixed before being connected to a sound application, and to create mixed output channels consisting of several local sound streams. For added versatility the mixer has a gain that exceeds 100% (or 0 dB) with respect to the incoming signal level, allowing the user to boost weak signals or even distort regular signals up to 400% (or 12 dB) of their original amplitude level.

Data compression: in order to minimize transmission latency, data compression can be applied to the signal, reducing the amount of audio data transmitted. Codecs like CELT [Carôt et al., 2009] can be used to reduce the amount of data without significant audio loss. Audio compression also reduces transmission bandwidth, which allows more audio channels to be sent over the same transmission link. On the other hand, compressing a signal introduces an algorithmic latency due to the encode/decode cycle, which is why some systems prefer to use uncompressed audio [Cáceres and Chafe, 2009a]. We believe this decision is better left to the user, and the communication tool should have an option for turning compression on/off and also for tweaking compression parameters, allowing a finer control over sound quality and algorithmic latency.

Silence Detection: silence detection algorithms might be used to avoid sending "empty" audio packets to the network, using up transmission bandwidth needlessly. This feature introduces a non-deterministic element in bandwidth usage, and so its use is subject to user discretion.

2.2 Environment architecture

The environment architecture represents how node instances interact with each other within the client-server model proposed. Node interaction includes audio/MIDI streaming, and control communication via messages used to reset the environment or to change its current status. To do this, SCTP is used to deal with streaming and TCP and UDP servers deal with messages for environment control.

Control messages (see figure 1) are XML-based action commands that are managed by the control/model layer components; their results are displayed in the GUI. Messages are used to update and extend the local model, for instance by adding new information about remote machines and streams, removing streams for users that logged out, etc. The choice between UDP or TCP corresponds to sending a message to all nodes (broadcast) or to send a message to a specific node (unicast).

```
<msg bufferSize="512" ip="192.168.0.101" msgType="0"
      name="Flavio" sampleRate="44100" port="40000">
 <outputs>
  <audioOutput name="My Output 1"/>
  <audioOutput name="My Output 2"/>
 </ouputs>
</msg>
```

Figure 1: XML Message

2.3 Environment Messages

The tool has messages that inform the local node about the current state of the environment. A report is sent to all users whenever a new user connects to the environment, when a user connects to a remote output port, or when any kind of environment configuration is changed. Messages may be of Broadcast (B) or Unicast (U) communication type:

HI_GUYS (B): This message is sent when a node enters the environment. It is composed by the IP address, network port, audio ports, MIDI ports and name of the user. When a machine receives this message it will add a new node to the environment node list and send back HI_THERE and LOOP_BACK messages.

HI_THERE (U): This message is sent when a machine receives a HI_GUYS message. It sends information back in order to help the new node to update its environment node list. The fields of this message are the same of HI_GUYS. Whenever a machine receives this message, it will add or replace the corresponding node of the environment list, and send back a LOOP_BACK message.

LOOP_BACK (U): After receiving a HI-GUYS or HI-THERE message, the node uses this message to measure the latency between the corresponding pair of nodes. This message contains the sender and target node names and a time-stamp field with the local time at the sender node. Whenever a machine receives a LOOP_BACK message it will first check for the sender: if the local machine is the sender, it will calculate the latency to the target node by halving the round-trip time; otherwise it will only send the message back to the sender.

BYE (B): This message is used to inform all nodes that a machine is leaving the environment. When a machine receive a BYE message it will disconnect the corresponding audio sinks (if any) and remove the node from the node environment list.

CONNECTED/DISCONNECTED (U): This pair of messages inform a node that a sink is connected to one of its sound resources (passed as an argument), or that a sink just disconnected from that sound resource.

CHAT (B): Used to exchange human-readable messages within the environment. It may help with synchronization (of actions, for instance) and node setup.

CONNECT_ME/DISCONNECT_ME (U): Ask a node to connect to (or to disconnect from) a source. These are useful to allow configuration of the environment in a transparent way.

ADD_PORT/REMOVE_PORT (U): Ask a node to add or remove a audio/MIDI port. This message contains the sound port type (audio or MIDI) and the sound port name, and is used for remote management.

CONNECT_PORT/DISCONNECT_PORT (U): Ask a node to change audio connections in a local sound route. It may be used for remote configuration: with this message one node might totally reconfigure another node's audio routing.

START_TRANSPORT/STOP_TRANSPORT (B): The transport message in the Jack sound server is used to start all players, recorders and other software that respond to a play/start button. It is used for instance in remote playback/recording, or to synchronize actions during performance.

3 Implementation and Results

To allow for a multi-platform implementation, some choices regarding the development framework were made.

The Medusa implementation uses QT [Nokia, 2011] for the GUI implementation, XML encapsulation, and UDP/TCP communication. For streaming, SCTP [HP, 2008] [Ong and Yoakum, 2002] was used as a transport protocol alternative to the usual UDP/RTP protocols. The current implementation of Medusa uses Jack[JACK, 2011] as sound server, and one of the core issues is to extend the functionalities of Jack to enable multi-channel routing of audio and MIDI through a computer network. All these libraries are licensed under GPL and work in Linux, Windows and MacOS. The C++ programming language was chosen because of the object-oriented design of the framework, and also because it is used by QT and Jack.

The preliminary results with a prototype implementation used by a group of composers at the Music Department at the University of São Paulo presented some interesting possibilities in network music. Using a wifi connection we were able to route 4 channels of uncompressed audio at 44.1 kHz without noticeable latency. MIDI channels were used to allow for MIDI synthesizers using remote controllers. Control messages were successfully used for automatically setting up the environment, which lessened the burden of the users. Broadcasting node information allowed users to connect to remote resources, and a constantly updated GUI showed whether remote users were accessing local resources.

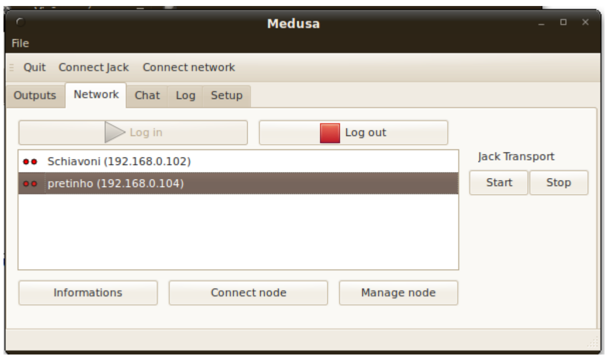
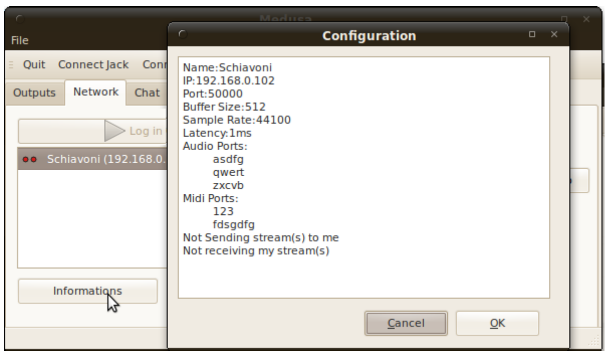


Figure 2: Network tab - connecting to/disconnecting from others nodes



```
Name:Schiavoni
IP:192.168.0.102
Port:50000
Buffer Size:512
Sample Rate:44100
Latency:1ms
Audio Ports:
        asdfg
        qwert
        zxcvb
Midi Ports:
        123
        fdsgdfg
Not Sending stream(s) to me
Not receiving my stream(s)
```

Figure 3: Setup tab - local node information

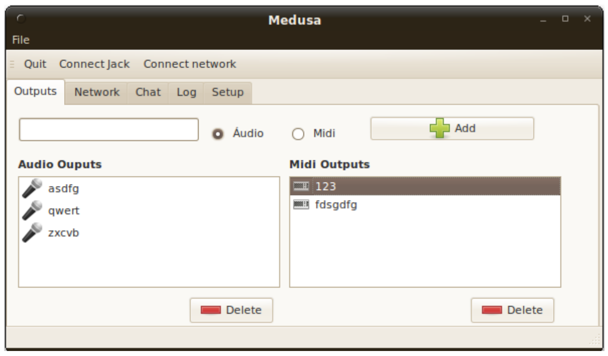


Figure 4: Outputs tab - creating/removing output ports

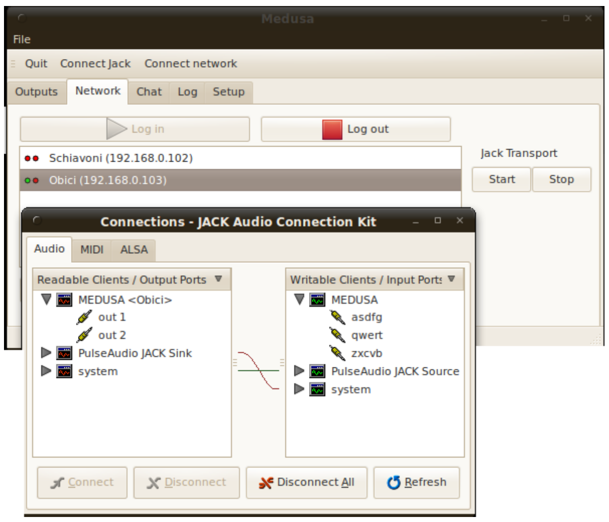


Figure 5: Interfacing with qJack Audio Connection

LAN nodes were associated to user names, making it easy for a user to identify peers and create connections (see figure 2). Each user is allowed to configure its audio settings independently from the others (see figure 3). Figure 4 shows the GUI that corresponds to the ADD_PORT/REMOVE_PORT messages. Connections between local and remote audio inputs and outputs are transparent and can be made using Jack's interface qJack as in figure 5.

4 Conclusions and future work

One relevant subjective conclusion at this point is the recognition that an user-friendly, graphical tool for network music may encourage musicians to experiment and play using networks. The possibilities of using a local area network for musical performance go beyond the common use of computers in live electronics, by allowing the distribution of computer processing and musical tasks among several performers and a heterogeneous group of computers and sound processing software. Network group performance on wireless connections is a fertile ground for musicians, composers and audio professionals. On the technical side, we observed that SCTP is a reliable protocol for sound exchange of a small number of audio channels, with unnoticeable latency and without packet loss on a local area network.

The next step in the validation of this tool is to measure latency, transmission bandwidth and network performance with different transmission links such as crossover cables, wireless connections, 10/100 Hubs and others. We would also like to have Medusa available to other platforms like PulseAudio, ALSA, PortAudio, ASIO and SoundFlower.

In order to allow remote connections outside of the Local Area Network (e.g. Internet), we would like to implement audio/MIDI communication using other transport protocols such as UDP and TCP in addition to SCTP. Since the SCTP protocol avoids packet loss, sticking to SCTP when going from LAN to WAN would make latency go way beyond an acceptable range.

5 Acknowledgements

The authors would like to thank the support of the funding agencies CNPq and FAPESP - São Paulo Research Foundation (grant 2008/08623-8).

References

Ilias Anagnostopoulos. 2009. The otherside web-based collaborative multimedia system. In LAC, editor, *Proceedings of Linux Audio Conference 2009*, pages 131–137.

Alvaro Barbosa. 2003. Displaced soundscapes: A survey of network systems for music and sonic art creation. *Leonardo Music Journal*, 13:53–59.

Jean-Chrysostome Bolot and Andrés Vega García. 1996. Control mechanisms for packet audio in the internet. In *INFOCOM '96. Fifteenth Annual Joint Conference of the IEEE Computer Societies. Networking the Next Generation. Proceedings IEEE*, pages 232 – 239 vol.1.

A. Carôt, U. Kramer, and G. Schuller. 2006. Network music performance (NMP) in narrow band networks. In *Proceedings of the 120th AES Convention*, Paris, France.

A. Carôt, T. Hohn, and C. Werner. 2009. Netjack–remote music collaboration with electronic sequencers on the internet. In *In Proceedings of the Linux Audio Conference*, page 118, Parma, Italy.

Chris Chafe, Scott Wilson, Al Leistikow, Dave Chisholm, and Gary Scavone. 2000. A simplified approach to high quality music and sound over IP. In *In Proceedings of the COST G-6 Conference on Digital Audio Effects (DAFX-00*, pages 159–164.

Juan-Pablo Cáceres and Chris Chafe. 2009a. Jacktrip: Under the hood of an engine for network audio. In *Proceedings of International Computer Music Conference*, page 509–512, San Francisco, California: International Computer Music Association.

Juan-Pablo Cáceres and Chris Chafe. 2009b. Jacktrip/Soundwire meets server farm. In *In Proceedings of the SMC 2009 - 6th Sound and Music Computing Conference*, pages 95–98, Porto, Portugal.

Hewlett-Packard Development Company HP, 2008. *SCTP Programmer's Guide.*

JACK. 2011. JACK: Connecting a world of audio.

ReWire Software Development Kit. 2010. Propellerhead software. Stockholm, Sweden.

John Lazzaro and John Wawrzynek. 2001. A case for network musical performance. In *In Proceedings of the 11th international*, pages 157–166. ACM Press.

Corporation Nokia. 2011. Qt Software.

L. Ong and J. Yoakum. 2002. An Introduction to the Stream Control Transmission Protocol (SCTP). RFC 3286 (Informational), May.

Alain B. Renaud, Alexander Carôt, and Pedro Rebelo. 2007. Networked music performance : State of the art. In *Proceedings AES 30th International Conference*, Saariselkä, Finland.

S.Letz, N.Arnaudov, and R.Moret. 2009. What's new in JACK2? In LAC, editor, *Proceedings of Linux Audio Conference 2009*, page 1.

Asbjørn Sæbø and U. Peter Svensson. 2006. A low-latency full-duplex audio over IP streamer. In *Proceedings of the Linux Audio Conference*, pages 25–31, Karlsruhe, Germany.

Gil Weinberg. 2002. The aesthetics, history, and future challenges of interconnected music networks. In *Proceedings of International Computer Music Conference*, pages 349–356.

Matthew Wright. 2005. Open Sound Control: an enabling technology for musical networking. *Org. Sound*, 10:193–200.

John P. Young. 2001. Using the Web for live interactive music. In *Proc. International Computer Music Conference*, pages 302–305, Habana, Cuba.