Serverless and Peer-to-peer distributed interfaces for musical control

Andrés Cabrera*
Media Arts and Technology
University of California Santa Barbara
Santa Barbara, California
andres@mat.ucsb.edu

ABSTRACT

This paper presents the concept and implementation of a decentralized, server-less and peer-to-peer network for the interchange of musical control interfaces and data using the OSC protocol. Graphical control elements that form the control interface can be freely edited and exchanged to and from any device in the network, doing away with the need for a separate server or editing application. All graphical elements representing the same parameter will have their value synchronized through the network mechanisms. Some practical considerations surrounding the implementation of this idea like automatic layout of control, editing interfaces on mobile touch-screen devices and auto-discovery of network nodes are discussed. Finally, GoOSC, a mobile application implementing these ideas is presented.

Author Keywords

Peer-to-peer, decentralized network, serverless, distributed interfaces, Open Sound Control, OSC

ACM Classification

C.2.4 [Distributed Systems] Distributed applications, H.5.5 [Information Interfaces and Presentation] Sound and Music Computing, H.5.2 [Information Interfaces and Presentation] User Interfaces—Input devices and strategies

1. INTRODUCTION

The use of networks to transmit control data is an established practice for musical as well as many other uses. OSC (Open Sound Control) has established itself as the standard protocol for musical data exchange, and many projects have emerged that can interoperate through it [14, 15]. Using the OSC protocol greatly simplifies the work needed to implement control interfaces on mobile devices, which has opened the door to the creation of many mobile applications for control like TouchOSC [2], Control [9] and Lemur [3] among many others. Browser based control interfaces that produce OSC have also become common and practical with systems like interface.js [11] and nexusosc [12]. Systems to simplify the managing of OSC control networks have been proposed [13], to aid in service discovery and to handle state [6]. Different network topologies have been proposed for the exchange of OSC control data, like Mobile ad-hoc wireless networks [7]. Decentralized communication for musical interchange is common in situations like laptop ensembles, but synchronization and equality between peers is not a concern.

Peer-to-peer networks (often referred to as P2P networks) are defined as decentralized networks where the nodes participating in it are "equal" peer nodes. In essence, there is no longer a hierarchical structure where a server accepts connections from clients, but all nodes in the network can play both roles. P2P networking defines a network architecture and also implies that the processing load and resources are shared by the peer nodes [8]. This decentralization of tasks brings many design challenges like addressing the dynamic layout of the network as nodes join and leave it, the optimizing of resource allocation and distribution, and the synchronization of state and data across the nodes. P2P networks are often studied and designed around the notion that nodes are distant and large in number, which adds significant additional requirements. One of the most common applications of P2P networks is file sharing, which can involve millions of simultaneous nodes spread across a great variety of networks with varying performance and speeds. A well known example of P2P file-sharing networks is the Bittorrent protocol [1]. Most P2P networks provide two separate layers. One is typically used for the exchange of data between nodes, and the other handles metadata [8].

2. NETWORK ARCHITECTURE

P2P networking is typically implemented as an abstraction on top of conventional server-client protocols and physical networks, but designed in a way that all nodes are able to provide the same functionality. OSC communication by design implies a client-server model, where there is a listener and a sender of messages. If bi-directional decentralized communication is to be established using OSC, it needs to be done by making all nodes in the network be both server and client of each other as needed. The proposed architecture specifies the relationship between nodes in a network and their roles as creators and consumers of data. This concept does not deal with the actual network topology or its physical layout and medium, so it can be implemented using regular wired or wireless networks, as well as ad-hoc/mesh networks.

The main characteristics and design goal for this network architecture were:

- Having consistent state for graphical displays of the same information, i.e. widgets can be thought of as just a display of a single value shared within the P2P network.

- Dynamic creation and destruction of graphical widgets, that is both easy and targeted to specific nodes.

*

- Nodes in the network can exchange data with other nodes as well as networked devices that are not part the P2P network.

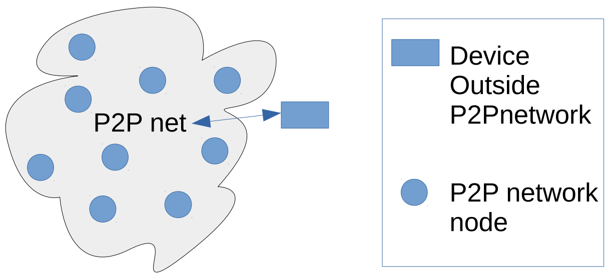


Figure 1: Abstraction of P2P network and connection to external devices

In practice, this P2P network is useful to deliver user interfaces to nodes in the P2P network from other nodes and non-P2P networked devices, as well as to exchange control information within and beyond the P2P network. Data from the control interfaces on the P2P network can reach what will usually be the desired destination: an OSC listener that is producing audio and/or video driven by the control data from the P2P network, but which is not itself part of the P2P network. This device that is not part of the P2P network could still send commands and values to nodes in the P2P network, for example to create/delete widgets in specific nodes or to set their values, but it won't relay data to other nodes, or receive control interfaces. This abstraction is shown in Figure 1, where an external device interacts with the network and sends and receives control values from it. Control data is synchronized among all the P2P nodes, so it is as if the external device is interchanging values with the network, no matter what specific node they originate from.

It must be noted that many details are deliberately left off from this proposal, like transport mechanism (UDP, TCP, etc.), as the goal is to define a generic P2P network that could be implemented using different technologies.

2.1 Network layers

A P2P network designed in this manner for the exchange of both control interfaces and control data, must inevitably be composed of two distinct layers:

1. A Command Layer: that handles network commands to create and delete widgets in a network node, as well as query states and other such global commands.
2. A Control Data Layer: that transmits actual control data between network nodes. This layer only connects specific nodes that need to share the control information, for example to link a control widget to a synthesis engine or to synchronize the value with another network node.

2.2 Command Layer

The command layer of this P2P network delivers commands across the network. There are two types of commands in this network: commands that are meant for all nodes, called *global commands* and commands that are meant to go to particular nodes called *node commands*. There are three possible scenarios that can occur in the command layer. Figure 2(a) shows a **global command** issued by one of the network nodes. The sending node must first query the network to find the network nodes and then send the command to all members. Each individual member if necessary replies to the original source of the message. Figure 2(b) shows a **node command** where one of the nodes sends a command to other specific nodes. Finally, Figure 2(c) shows a **node command** originating from a device not part of the network. Typically this will result in the node becoming a gateway node for the channel (see gateway nodes below).

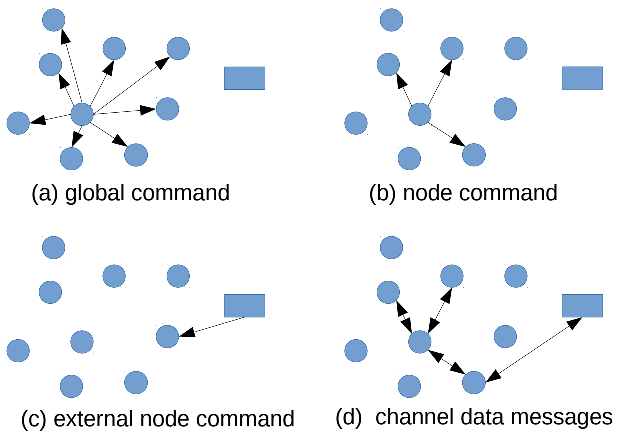


Figure 2: Commands and message exchange between and beyond the P2P network

2.3 Control data layer

The control data layer transports data values (usually generated from graphical widgets), to a listening application outside the network or to other nodes registered to the same channel within the P2P network.

2.3.1 *Channels*

The identifiers to data values within the network are called *channels*. The channel name or identifier corresponds to the OSC address. Note that there should only exist a single value for a channel within the network, since control data should be synchronized between all nodes. In practice, values in individual nodes might occasionally be inconsistent for a short period while a value settles (for example if two nodes are generating values simultaneously for the same channel, and there is some difference in arrival times). This means that all widgets on the same channel should reach a consistent state no matter where they are. The practical implication is that if a slider on one device is controlling a parameter called "filter frequency", any other widget that can control or display that parameter will reflect changes made on any another device or node.

There are two options to deal with the issue of data synchronization while still keeping the decentralized nature of the network. For simplicity, no synchronization mechanism could be implemented. In practice, it would be unusual for two nodes to end up with conflicting values. The case where this might happen is if two separate nodes generate a value for the channel and because of network latency this information reaches the nodes after the value is generated, so the widgets will display each other's last value. As this propagates to other nodes on the P2P network, because of varying network latencies, nodes might end up with any one of the values. In most practical scenarios, this might not be a significant issue, as inconsistencies would tend to be resolved fairly quickly as nodes generate new values, so it might just turn out to be a minor "cosmetic" issue. If robustness in channel data synchronization is required, an option would be to send additional timestamped messages to identify data. Timestamping needs to be done with a high resolution network clock, and could be sent using OSC timestamps within OSC bundles.

2.3.2 *Control data exchange*

To exchange control values, each node must know to what other nodes it needs to transmit the channel value to. A node must keep a list of all nodes that listen to that channel to be able to send values to them as they are generated (See discovery of peers below). An independent connection is made between the node generating the value and all other nodes that need to know about the change in the channel's value. Nodes do not propagate values to other nodes when they receive a value. The value is only sent if a node is the generator of the value.

A slightly different approach needs to be used when a device that is not part of the P2P network needs to listen to channel values from inside the network. A node can establish itself as a "gateway" node, or be ordered to be one when creating a widget. Gateway nodes send values to devices outside the P2P network both when they generate and when they receive values from inside the P2P network. The default behavior to determine gateway nodes should be that when a P2P network member node instantiates the widget, the node is not a gateway node for the channel and when the widget is created outside the P2P network, the node is treated as a gateway for that channel and that device. This could be overridden however as this is set by widget properties upon creation. The exchange of **channel data messages** is shown in Figure 2(d).

2.4 Commands

The command layer protocol is simple and requires only a few message types. The only limitation is that since both commands and data are issued using the OSC protocol, the separation between the layers is done by reserving the `/command/` OSC path prefix. In other words no other nodes or members of the network should use this OSC address prefix for anything other than commands. The peer node software needs to enforce this limitation by not allowing any channel name to be prefixed by `/command/`.

The available commands are:

/command/query A query command can be sent to specific nodes to find information about it. Details like protocol version, screen size and currently used channels can be obtained when the node responds to this command.

/command/reply A reply to a query command proving the information requested. Errors and warnings are sent as reply messages.

/command/create This message contains the code that needs to be executed in the node to create a widget. A node that receives this message should instantiate the widget contained in the message.

/command/destroy This message orders a widget to be destroyed. A widget can be identified through its channel or through its unique identifier.

/command/register A node sends this message to another one to let it know it wants to listen to value changes on a particular channel. When a widget is created, a node will query all the other nodes for their channels, and then register the channel with nodes that also use it.

/command/unregister When a node exits the network or a widget in it is destroyed, the node notifies other peers that it is no longer listening to that channel.

/command/relay The relay command tells the node if values for a channel not generated by it are to be relayed to an address provided in the command. This should generally not be used for nodes within the P2P network, as they would all receive the data directly from the source node. But it is useful to register listeners outside the P2P network to receive data from the node. This command effectively turns a node into a gateway node for a particular channel.

2.5 Discovery and removal of peers

The dynamic nature of a P2P network implies that nodes in the network can leave and enter it at any time, and a simple mechanism should exist to allow nodes to enter and leave the network without affecting its integrity. There are two stages involved here. First, nodes need to be able to find themselves and identify each other automatically on the network. Several strategies have been proposed for this like Zeroconf or SIP (RFC 3261) [5]. Any of them can be used effectively for this particular application. Once a node connects to a network and finds other nodes in the P2P network, it needs to announce itself, by registering any channels that are active in its interface. Whenever it creates a widget it must register the channel or channels with the other nodes in the network, and conversely must deregister once a widget is destroyed, or the peer application quits.

Nodes should also have some mechanism for unregistering nodes (for example using a time-out counter) in case nodes exited unexpectedly. A special case can occur when a gateway node exits unexpectedly, as this might mean communication between the P2P network and the outside might be severed. It is the outside node that must handle this and make another node a gateway node, by first querying the nodes for a particular channel, then registering itself as a listener on that channel on any particular node.

2.6 Preset server

Just as in file sharing P2P networks there is often the need to interact with a centralized catalog that is held independently of the P2P network, a similar model can be used on the control P2P network to handle preset saving and state management[1].

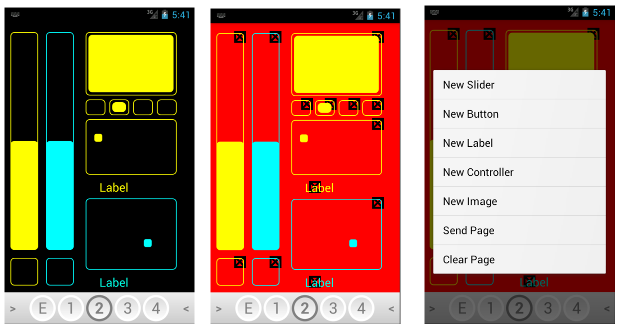
3. PEER SOFTWARE

Devices that wish to connect to the P2P network must be running peer software that implements all the discovery, query and registration mechanisms described in Section 2.4. Devices outside the network need not run this software, and only need to craft certain OSC messages in particular ways to create widgets on nodes on the network.

4. USE CASES

The use cases for a P2P control network are similar to others that have been proposed for distributed control of interactive performance. Apart from introducing the audience as a participant in performance, distributed interfaces can serve as a way to dynamically adjust the control interfaces of performers, to optimize for screen real estate or to add uncertainty to the controls the performers are given. The P2P network proposed here can cover all applications of other distributed systems plus it offers the following benefits:

[1]Preset in this context refers to the whole interface layout as well as the values for each graphical element



(a) GoOSC (b) Edit mode (c) Menu

Figure 3: Screenshots from the GoOSC application

- Synchronization of values for the same parameters in all interfaces, independent of how or where the data is displayed.
- Easy set up, as interfaces can be loaded from any device into another without a server.
- Control interfaces can be easily distributed when participants and/or control parameters and widgets are not known before the performance.

5. IMPLEMENTATION

GoOSC an implementation of a server-less OSC editor/controller that instantiates a node in a P2P network as described in this article. The application is written using the Qt toolkit [4], and is currently available in the Google Play store .

A few important features that are not part of the P2P design itself, but that provide useful functionality around it are implemented in the application and described in this section. A screenshot of GoOSC running on an Android device is shown in Figure 3a.

5.1 Editing control interfaces

Since peer nodes can exchange interfaces, it makes sense to make the mobile application be able to edit and generate the interfaces. A simple model was implemented in GoOSC targeting touch screens, where editing might be difficult. There are also actions available to send individual widgets as well as complete interface pages (see Figure 3c).

5.2 Interpreted language

Since the widget code that is transmitted from one device to another is interpreted by the QtQuick engine, there is the possibility of both extending the widget set to include new graphical representations of control, as well as the possibility of adding programming logic and behavior to the controls, for example to add physically modelled behaviour like bouncing, elasticity, swarming, etc. Because the new type of widget is now code on the device, if it is written properly, it can then be sent to other devices, or reused in other circumstances.

6. CONCLUSIONS AND FUTURE WORK

A peer-to-peer network design for the exchange of control interfaces and data was presented. The network is presented as an abstract model that can be implemented on top of different topologies and technologies. The P2P network model serves as a simple and consistent way to exchange control interfaces and widgets while maintining synchronization between members. The network has been implemented as part of the GoOSC mobile application.

7. REFERENCES

[1] The bittorrent protocol specification. Online. http://www.bittorrent.org/beps/bep_0003.html [Accessed: 26-Jan- 2015].

[2] h e x l e r . n e t | touchosc. Online. http://hexler.net/software/touchosc [Accessed: 26-Jan- 2015].

[3] Lemur – liine. Online. https://liine.net/en/products/lemur/ [Accessed: 26-Jan- 2015].

[4] Qt toolkit. Online. http://www.qt.io [Accessed: 26-Jan- 2015].

[5] SIP: Session initiation protocol (RFC 3261). Online. http://www.ietf.org/rfc/rfc3261.txt [Accessed: 26-Jan- 2015].

[6] I. Bergstrom and J. Llobera. OSC-Namespace and OSC-State: Schemata for Describing the Namespace and State of OSC-Enabled Systems. *Proceedings of the International Conference on New Interfaces for Musical Expression (NIME 2014)*, pages 311–314, 2014.

[7] S. Lee, G. Essl, and Z. Mao. Distributing mobile music applications for audience participation using mobile ad-hoc network (manet). In *Proceedings of the International Conference on New Instruments for Musical Expression (NIME 2014)*, pages 533–536, 2014.

[8] D. S. Milojicic, V. Kalogeraki, R. Lukose, K. Nagaraja, J. Pruyne, B. Richard, S. Rollins, and Z. Xu. Peer-to-peer computing. *HP Laboratories Palo Alto*, 2003.

[9] C. Roberts. Control : Software for End-User Interface Programming and Interactive Performance. In *Icmc 2011*, pages 425–428, 2011.

[10] C. Roberts, G. Wakefield, and M. Wright. Mobile Controls On-The-Fly : An Abstraction for Distributed NIMEs. In *Proceedings of the International Conference on New Interfaces for Musical Expression (NIME 2012)*, pages 474–478, 2012.

[11] C. Roberts, G. Wakefield, and M. Wright. The Web Browser As Synthesizer And Interface. In *Proceedings of the International Conference on New Interfaces for Musical Expression (NIME 2013)*, pages 313–318, 2013.

[12] B. Taylor, J. Allison, W. Conlin, Y. Oh, and D. Holmes. Simplified Expressive Mobile Development with NexusUI, NexusUp, and NexusDrop. *Proceedings of the International Conference on New Interfaces for Musical Expression (NIME 2014)*, pages 257–262, 2014.

[13] N. Weitzner, J. Freeman, Y.-L. Chen, and S. Garrett. massMobile: towards a flexible framework for large-scale participatory collaborations in live performances. *Organised Sound*, 18(01):30–42, Mar. 2013.

[14] M. Wright and A. Freed. Open Sound Control: A New Protocol for Communicating with Sound Synthesizers. *Proc. ICMC 1997*, pages 101–104, 1997.

[15] M. Wright, A. Freed, and A. Momeni. OpenSound Control: state of the art 2003. *Proceedings of the International Conference on New Instruments for Musical Expression (NIME-03)*, pages 153–159, 2003.