Networked Interfaces Workshop

Chad McKinney
C.Mckinney@sussex.ac.uk
www.chadmckinneyaudio.com
www.glitchlich.com
https://github.com/ChadMcKinney

First, Let's Make Some Noise

If you have Chrome installed go here:
http://labs.dinahmoe.com/plink/

Otherwise check out John Bischoff's Aperture using Phil Burk's PureJSyn
http://www.transjam.com/aperture/aperture_client.html

Ok now we need to grab some software if you don't have it already. We'll need:
Documents for this workshop:
Processing http://www.processing.org/
SuperCollider http://supercollider.sourceforge.net/

A Brief Introduction to networking concepts

Warning: Networking often means lots of programming. This isn't bad, but you just have to be prepared to put in more work than for a non-networked piece/system/interface/game/whatever.

IP Address – Address of a computer in a network.

Port – Application specific endpoint, think of it like a channel on a TV.

TCP – Transmission Control Protocol is reliable and error-checked delivery of packets

UDP – User Datagram Protocol is a minimal protocol that emphasizes overall speed instead of packet integrity.

OSC – Open Sound Control is a message format that can be implemented with either TCP or UDP and is often used in music software.

It's important to understand the structure of an OSC message. OSC messages begin with an address pattern indicating the type of message being sent. This is followed by a stream of values. At the lowest level these values are paired with type tags to identify their type, especially important in strongly typed languages. When using an OSC library this is mostly taken care of for you.

There's tons more to networking, but this is enough to get us going.

Beginning with OSC

First make sure you have the oscP5 library for Processing. Once you have that open up the oscP5sendReceive project in processing and run it. Look over the code and try to find a few things:

1. What IP address are we sending messages to?
2. What port are we listening on?
3. What address pattern is being used for the OSC message?

Once we've figured this out grab a neighbor. Networking is collaborative! Exchange IP addresses and change the code accordingly (this may take a little consideration). Set it up in such a way that you trigger not your own program to post, but the partners and vice-versa.

Extend your programs by adding functionality based on the address pattern of a received message. You

could respond by sending your own message, or even delay your response by using something like:

```
       delay(1000); // milliseconds
```

Next add a boolean to the project indicating that a message with a given address has been received. You can do this by having the oscEvent method test it to true. In the draw loop draw something, say a rect, but only if the message has been received, and subsequently set the boolean to false. For example:

```
boolean testMsg = false;

// setup code here
...
// finally

void draw()
{
    background(255, 255, 255, 200); // Fadeout background

    if(testMsg)
    {
        rect(20, 20, 50, 50);
        testMsg = false
    }
}
```

If all goes well you should draw a rectangle but only when a message has been received.

After you've played with that for a while we're going to move on and open up the oscP5plug example. In this example we're going to be adding arguments to our messages. Arguments are important because they give content to your messages. Note how arguments are added to an OscMessage using the add method. What kinds of objects do you think can be added to an OscMessage?

TANGENT!
Now would be a good time to check http://opensoundcontrol.org/
You'll find the available types here: http://opensoundcontrol.org/spec-1_0

So you're quite limited. No sending arbitrary objects of various classes. This means you have to think carefully about how to describe your communications in the network. Back to the example (future).

The plug method is important here because it allows you to easily link an address pattern with a callback function. This means that parsing is much easier. You don't have to match various strings oscP5 does all the hard work for you. Lets set up a similar drawing response in our previous program, but now being called in the test function (or you could make your own and call it something more apt, like drawRect).

Now lets add some information, such as position. At the top, next to your boolean we're going to add two more variables, this time integers. Then in the draw loop we'll use those two integers for our x and y values in the rect() method. Penultimately, declare those integers to the values of theA and theB in your callbackfunction (test by default). Finally, we'll need to replace the current values in msg.add with mouseX and mouseY. Something along these lines of this:

```
boolean drawRect = false;
int rectX, rectY;
// SETUP CODE HERE
public void test(int theA, int theB) {
  drawRect = true;
  rectX = theA;
  rectY = theB;
}

void draw() {
  background(0);
  if(drawRect)
  {
    rect(rectX, rectY, 50, 50);
    drawRect = false;
```

```
  }
}

void mousePressed() {
  OscMessage myMessage = new OscMessage("/test");
  myMessage.add(mouseX); /* add an int to the osc message */
  myMessage.add(mouseY); /* add a second int to the osc message */
  oscP5.send(myMessage, myRemoteLocation);
}
```

If all goes well you should be able to draw rectangles based on your mouse position. If you haven't done so already now is a good time to exchange ip addresses with a neighbor and draw on each other's screens with your mouse. How do you think you could make the rectangles act more like permanent objects that are create, moved, and removed with different commands? Can you think of a way to make it so that both players could control the same rectangle?

When you get down to it, this is about it when it comes to networking. It's just messages and data. The tricky part is what you do with it, and how you deal with all of the problems that networking introduces.

Issues and Idiosyncrasies Of Networked Interfaces

Network Topologies

Hub – Hub networks use a central server as a focal point of communication between the peers. Examples include The Hub (specifically the early era) and Glitch Lich. Note: servers are a point of failure. If a server goes down, everyone goes down with it.

Mesh – Mesh networks utilize peer to peer communication. The live coding band Slub uses simple duo mesh network. Larger groups such The League of Automatic Music Composers, Later Hub, and Laptop Orchestras use mesh networks. Mesh networks solve the problem of having a single point of failure, however they have their own problems, such as a lack of unifying archtecture and the requirement of deployment of updates on all machines where in a hub network some updates can be applied simply by updating the server.

Asymmetrical – This is somewhat of a catchall group and for that reason is hard to categorize. This group will include networks that mix peer to peer and server based communication. Also networks with multiple servers or where communication access is not symmetrical across the topology.

Distributed Systems are asynchronous

When you create programs or interfaces, sometimes you have to deal with some latency or behavioral non-conformity. When using networks, it's important that the system can function even when everything goes wrong. Spoiler alert: Everything can, *and will* go wrong. There is no guarantee about latency, or that a message will ever be received, or that if it is received you get the reply. Creating networked systems is not like putting finely crafted cogs together to create an intricate machine. It's much more like having several improvising musicians in a room where at any given time a solo could break out or the style could suddenly shift. This means that your programs will begin to form pockets of subsystems that send and receive information but that rely on neither.

Three common approaches to networking

Mapping – Create interdependence by mapping and cross mapping variables (of some kind) across the band. The League of Automatic Music Composers were the progenitors but groups such as BiLE use this technique currently.

Sharing – Often sharing code, could be other information or files. Power Books Unplugged, The Mandelbrots

Synchronization – Can be as simple as synchronizing clocks, but can also include video game style state synchronization. Slub, Glitch Lich, the Mandelbrots. When synchronizing across a network an important question is, Where's the state? Depending on the answer you will have different problems to solve. Maintaining state in the clients will be easier, but you'll get worse results if your goal is to achieve similar state among the network. Keeping the state on the server will unify your network more successfully, but at the cost of more work/code.

Latency – The delay between messages being sent and messages being delivered. You can't get rid of it, but there are many strategies to cope with it.

Some strategies for coping with latency:

Accept it, just let there be latency

Scheduling: By using a scheduler combined with Clock Synchronization you can schedule events to happen in the future, outside of the largest delay between users. This way the actual time execution of events is much closer among the peers, however you have to increase the time between intention and effect

Use the latency as a fundamental component of the interface: Global String, Network Harp

Firewalls – They increase network security but can often interfere with your networking efforts. NAT traversal and NAT hole punching such was what OscGroups uses. Alternatively you can have a server on the open web (itself not behind a firewall) and you will often get better results by routing traffic through that.

TCP vs UDP for Interface Development

Issues such as state, synchronization, speed, shape of gestures are impacted greatly by which protocol you use.

Space
How do you create a collaborate space?
How or should you even, create ownership?
Should your system allow for privacy in the network? Are all events broad casted?

Security
Networking opens up your users to potential security risk.

Strategies for dealing with potential security issues:
Use passwords
Use Encryption
Don't allow execution of arbitrary code
for example, this bit of SC code can be devastating (**WARNING: DO NOT RUN THIS**): "rm -rf /".unixCmd;

Blocking IO/Threading/Serialization of Data Access

Often this detail will be hidden away from you by whatever library you happen to be using, however network sockets are commonly run in a separate thread because they have to block execution while they wait for responses. This often crops up when testing some code, sometimes code that's worked quite well for some time, and then it just randomly crashes. And again in another place, but completely unrelated. Runtime threading bugs are some of the most difficult to track down and the best approach is to be proactive about the chain of custody of your data. The simplest approach is to relegate it to a single place and require access through a serialized interface via semaphores and mutexes. For now, if you're new to this it's best to understand some of the other concepts first, however once you get far enough into this type of development it will inevitably crop up

Sharing Code
A common problem with collaborating on projects is version control. Passing versions back and forth is tedious and problematic, especially as group size increases. Becoming familiar with a version control system such as git (git-scm.com) will prevent your projects from descending into the depths of Rlyeh, the realm of Cthulhu, the great old one. I recommend github.com for hosting your open source projects.

Abstracting Networks

OscGroups http://www.rossbencina.com/code/oscgroups
Republic Quark (In SuperCollider execute: Quarks.gui;)
Benoit Lib https://github.com/cappelnord/BenoitLib
OSCthulhu https://github.com/CurtisMcKinney/OSCthulhu

OSCthulhu FAQ:

What libraries do I need for Linux?
To run OSCthulhu you'll need the Qt Core,Gui, and Network libraries. In Ubuntu run this:
sudo apt-get install libqt4-network libqt4-dev libqt4-gui

What if my Internet connections cuts out?
OSCthulhu automatically reconnects.

What if my program crashes?
Restart your program, the OSCthulhu client will update you as soon as you send the /login message.

What if the OSCthulhu Client crashes?
Restart it, it will automatically update to the server and subsequently so will your program.

What if the OSCthulhu Server crashes?
You can try restarting it, but in the middle of a performance it may be best to finish offline.

How do I run a server?
cd /path/to/OSCthulhu
./OSCthulhu Server 32242 yourpass

How do I connect to a server running on my own machine?
127.0.0.1 is always the IP address for your local machine.

At this point if you have SuperCollider it would be a good time to open up SimpleOSCthulhuExamples.scd You'll also need to have the OSCthulhu client running, along with a local OSCthulhu server. In your OSCthulhu client preferences set the server's IP address to 127.0.0.1 and begin executing commands in SC.

Try extending the examples to allow for a wider range of control over the synth sound. Try adding a second sound. This will require adding more code in the /addSyncObject OSCresponder. Now would be a great time to find a neighbor, or more. Choose one person as the server and have everyone use their IP as the server's IP and then execute the commands in the example file. If everything is working the “Sync Objects” in the OSCthulhu server will update across the network.

Next open up the OSCthulhuClassExamples.scd and also take care to add the OSCthulhu.sc file to your SuperCollider extensions folder. Here I've just provided some convenience methods to clean up some of the sprawling code from before. Try adding a synth play back responder like in the other file.

After you've played around with that we're going to go back to processing and look at an OSCthulhu in there. Open up OSCthulhuProcessing.pde and take a look through the code, specifically the OSCthulhu tab. There's some new code here to make networking even easier. In the main file you'll see the OSCthulhu object, something call a SubGroupMapper, and SyncCircles and SyncRects classes. The OSCthulhu classes works very similar to the one before in SuperCollider, however the SubGroupMapper adds some interesting new functionality. You don't have to use it directly save for instantiating it correctly, but it allows for easy parsing of OSCthulhu messages filtered to various SyncMap subclasses such as SyncCircles and SyncRects. SyncCircles and SyncRects create and manage objects of the type SyncCircle and SyncRect respectively. Play with the project with some friends, again setting up a local OSCthulhu network. When you use left click and release a circle will be drawn where your mouse is. Click again and It goes away. Right click makes Rectangles of random colors. Once you have the network working try making your own SyncObject subclass and coordinating SyncMap subclass to create new functionality.

For these kinds of projects it's important that everyone in the network is on the same version of software. You may consider passing around dropbox links or even setting up a quick github account. Network development often means having to manage code in a group and code repositories such as github make this much easier.

Streaming

Audio/Video
Skype: Scot Gresham-Lancaster, Pauline Oliveros. Simple but it gets the job done.
JackTrip: Chris Chafe, Juan Pablo Caceres, etc...
https://ccrma.stanford.edu/groups/soundwire/software/jacktrip/
Ice Cast/ Dark Cast – Server based audio streaming over the web.

Web Development

Mobwrite – http://code.google.com/p/google-mobwrite/

Lich – http://chadmckinneyaudio.com/Lich.js/Lich.html
Lich source code – https://github.com/ChadMcKinney/Lich.js
Share.js – http://sharejs.org/
WebSockets – Allows for connections between browsers. Web based networking solution.
Socket.io – A popular WebSocket library, http://socket.io/

Plink by Dinah Moe uses Web Sockets
http://labs.dinahmoe.com/plink/

Misc

Clock Synchronization

Installations

Leech
Article written about it – http://cncptn.com/post/9379182946/curtis-mckinney-and-chad-mckinney-leech-i
Video of it running – https://vimeo.com/21603631

Random Things Worth Checking Out

Network Bands:
The League of Automatic Music Composers
The Hub

Power Books Unplugged	http://pbup.goto10.org/
Slub	http://slub.org/
Benoit and The Mandelbrots	http://www.the-mandelbrots.de/
Alo Allik and Yota Morimoto	http://tehis.net/
	http://yota.tehis.net/
BiLE	http://www.bilensemble.co.uk/

// Full disclosure, Glitch Lich is my band, so I may be somewhat biased putting it here...
Glitch Lich http://www.glitchlich.com/