Cmpt433 Final Group Project Submission

Fall 2011

Nap Time Sound Appliance

Benny Yan

Cal Woodruff

Thomas Betz

System Overview

Our system is a multi-function sound appliance that is network aware. This means that if you have more than one they can be made aware of each other and be used to share mp3 files, memos and intercom connections. We achieve this by implementing a local multicast network that any node can use to find out what other nodes are available on the network.

Nodes are able to play mp3 files, record memos in wav format, send memos to other nodes and share their mp3 files with other nodes. In addition, any node can initiate an intercom session with another node on the network.

The components that make up a nap player node are divided into layers. At the top layer there is the boa web interface and Qt touch screen gui. These interact with an intermediate application layer that does the work of finding other nodes on the network, advertising a node's presence on the network, playing media, recording memos, connecting and breaking down intercom sessions and so on. The diagram below shows the basic architecture of the system.

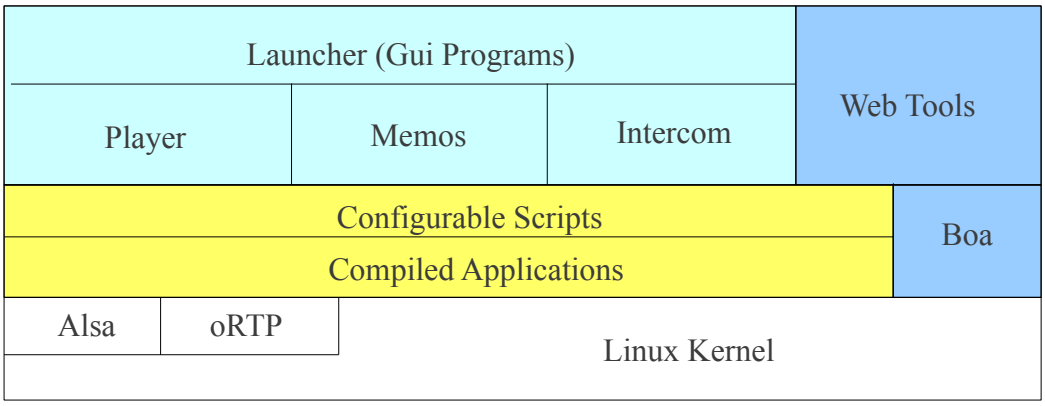

Fig. 1 System Diagram

Component	Installed location
Gui/ Scripts/ Applications	/bin or *{remote fs}*/bin
Web Tools (main directory for the website and shared data)	*{remote fs}*/p2p/shared
Other scripts	*{remote fs}*/p2p
Non-shared data from web config, memos (defined in /etc/nap.conf)	*{remote fs}*/napdata

A *remote fs* is used because storage requirements needed exceed the on board NAND flash.

The Launcher App

Fig. 2 The launcher app

The launcher gui program is the front end of the appliance. From there an end user can select the player, memos or intercom gui applications. Normally our gui applications do not interact with compiled programs directly. Instead the application start up logic is encapsulated in configurable scripts. This architecture was chosen because there were many unknowns in terms of what application software would work on the EM2440 and how the application software would need to be configured to work best on the hardware.

The launcher is meant to start with the system automatically. It takes care of starting up long term processes such as those that enable the node to be found on the network by other nodes (see below) and other periodic tasks that need to run all the time such as checking for new memo messages from other nodes.

Web Interface

In addition, a web interface using the boa web server is used to configure various aspects of the system and implement inter node control. A simple extensible CGI web interface was developed that can be easily extended to provide new configuration features and functionality. This web interface is also used for various tasks which involve communications between nodes in the network such as the transfer of memos between nodes in the network and connecting and disconnecting from an intercom session.

The Player App

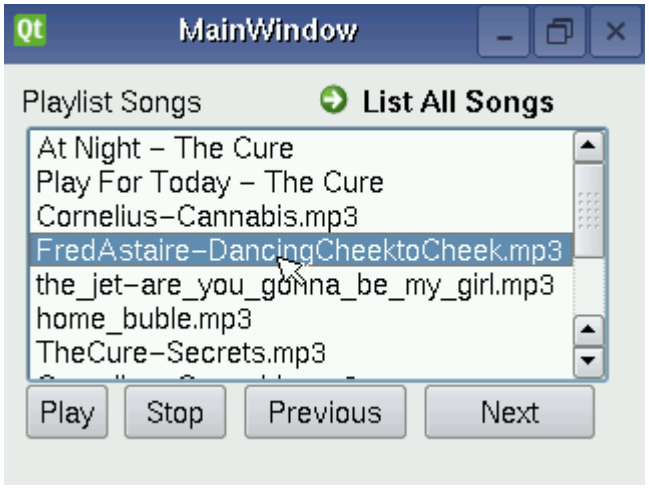

Figure 3a: The Player Control

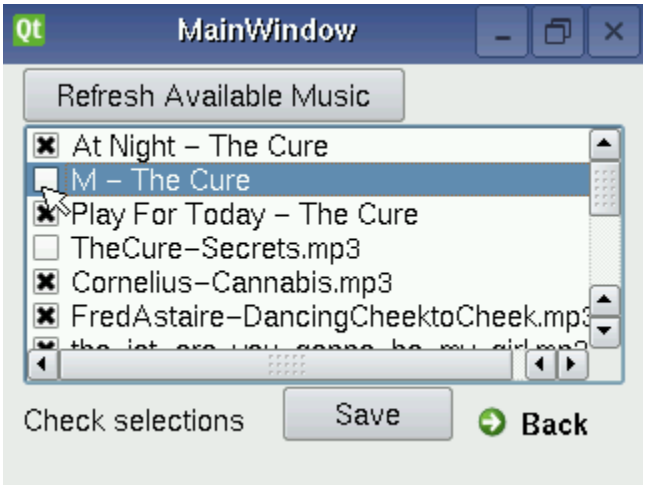

Figure 3b: The Playlist Editor

The player is comprised of two interface components: player control and playlist editor. The player control is initiated by the launcher. The player control provides a playlist of songs from local and remote sources, and the gui buttons to control playing, stopping, and movement up and down through the list. The gui also allows for double tapping a song name as the means to start it playing. The 'list all songs' button brings up the playlist editor, where users add or remove songs from the playlist by checking off their selections and clicking the save button, whereupon they are brought back to player control again. The 'refresh available music' button updates the list of available songs as well as the nodes from which they are served, (implicitly). QT QProcess calls to the madplay music player underlie the gui play control buttons, while song meta-data, (title and artist), are likewise obtained with

calls to madplay for local mp3 files.

The player application relies upon the napping / naplistener node discovery method, (described below), to identify peers. Additionally, a simple client/server model serves the task of sending lists of shared music generated by a shell script on the remote node (the server) upon request from the player (the client) which then populates the list of available music. Music added to the playlist is transferred to local SD card storage when played. The *musicscan* client/server and appropriate configurations of BOA allow for remote nodes which are either ARM boards or x86 PCs.

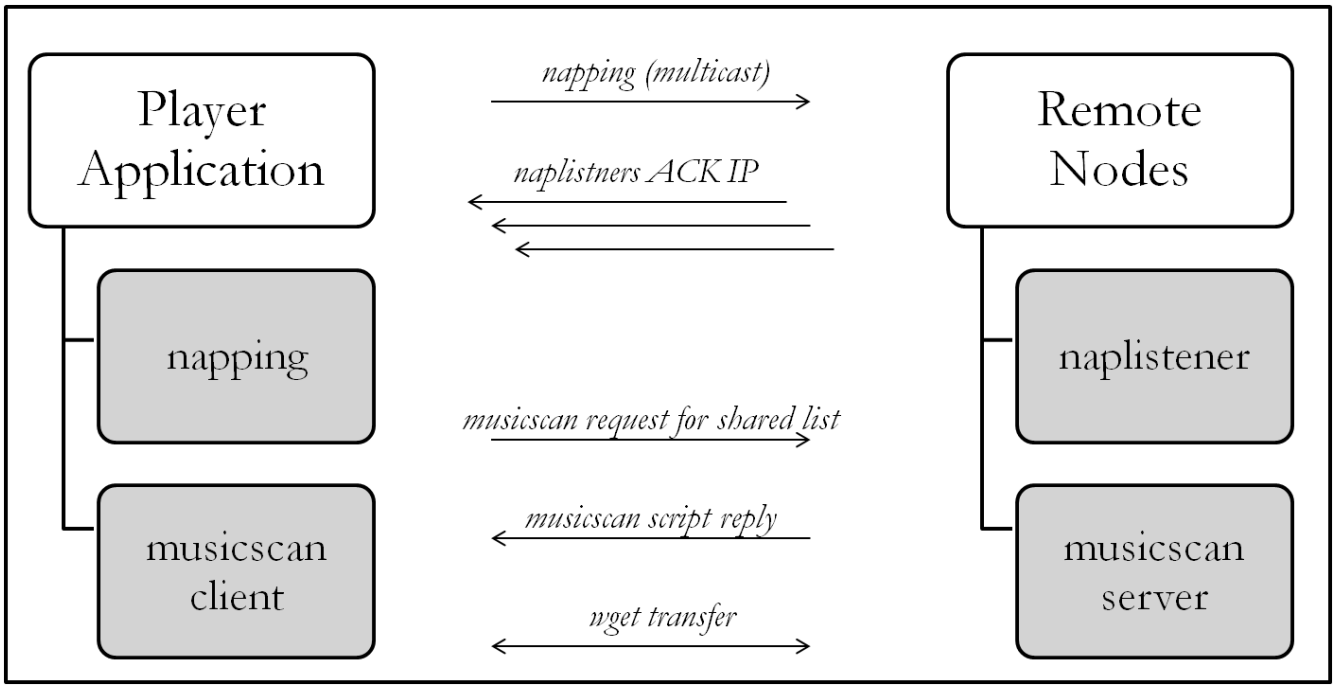

Figure 4: Player Application Process

Madplay Control

The player uses the madplay application to play mp3 audio files and playlists. Madplay is a command-line MPEG audio decoder and player based on the MAD library(libmad). With madplay an end user can control various aspects of play back via a keyboard. As this was not feasible with our touch screen interface, we looked into the madplay source code and decided to modify the source code so that madplay gets its control command from a named pipe. This way any other process has a way to

pause, stop, forward, back, increase and decrease volume, and quit madplay.

We programmed the hardware buttons to write to this named pipe. The mapping of buttons is:

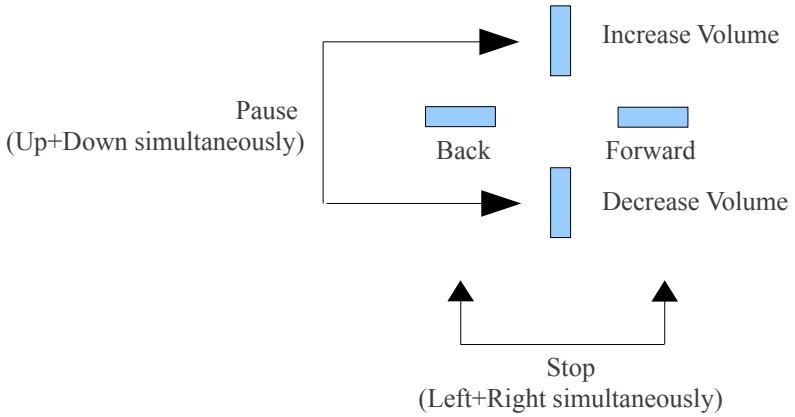

Fig. Button mapping for madplay control via /tmp/madplayFIFO

USB Auto Detection

Because of the limited storage on the board we assume anyone using the player application will be storing music on external media. We wanted to make it possible to support as wide a range of of usb storage devices as possible. However, we were limited because only FAT16 and FAT32 file systems were recognized by the EM2440 in its default configuration. In addition, auto mounting usb drives was not supported. One had to manually find out the file system and device parameters for the drive by either opening the kernel startup message buffer with dmesg or use the fdisk -l command to read through the available partition tables.

We decided to make it possible for our boards to read any standard file system. We configured the linux kernel via make menuconfig as follows. The ext2, ext3 and ext4 file systems were enabled as built-in modules:

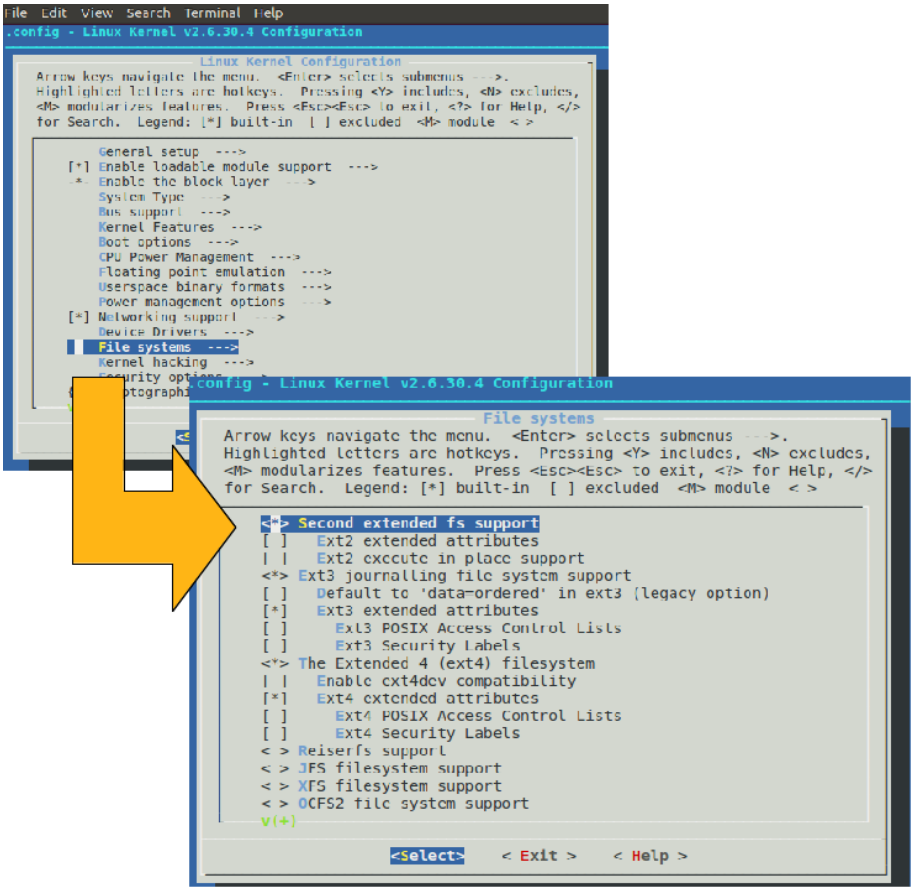

Fig. 5 Linux make menuconfig changes

We then compiled and loaded the kernel onto the boards, so our boards can recognize all the standard linux file systems as well as the windows fat32 and fat16 file systems included by default with the board configuration.

We also wanted to be able to auto mount a USB whenever it is attached to a board. After much investigation online we found there were two available standard options. One is autofs, the other is udev. As both were complex to cross compile and install we chose to create our own solution. One of us (Benny Yan) wrote a program which reads the partition table and auto mounts the flash drive when it

is inserted. The drive is automatically unmounted when it is removed.

The Memos App

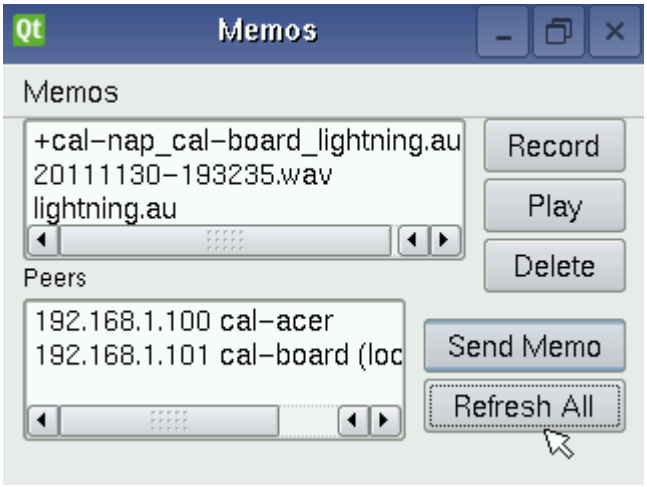

Fig. 6 The memos app

The memos app allows an end user to save, listen to and exchange short sound files called memos that are stored on the appliance itself. These files are named with a timestamp and saved in a node specific location and can be sent to other nodes. When a memo is sent to another node the leds on the target node blink with a series of short blinks equal to the number of memos received by that node. Memos from other nodes are indicated by the name of the node followed by the date and time the memo was recorded.

Node Discovery

How do apps find out about other nodes? We devised a very simple system that uses network multicast to allow a node to advertise its existence on the network. This system consists of two c applications (which can be found in the napscan directory in the code repository). The first *naplistener* is a multicast server it listens on a configurable port for multicast requests from other nodes. Nodes who wish to find out who else is in their multicast group use the *napping* program to find other nodes on the network. The script *getpeers* encapsulates this behaviour and returns a list of found peers. These

peers can then interact with each other via the web interface or other interfaces for various tasks.

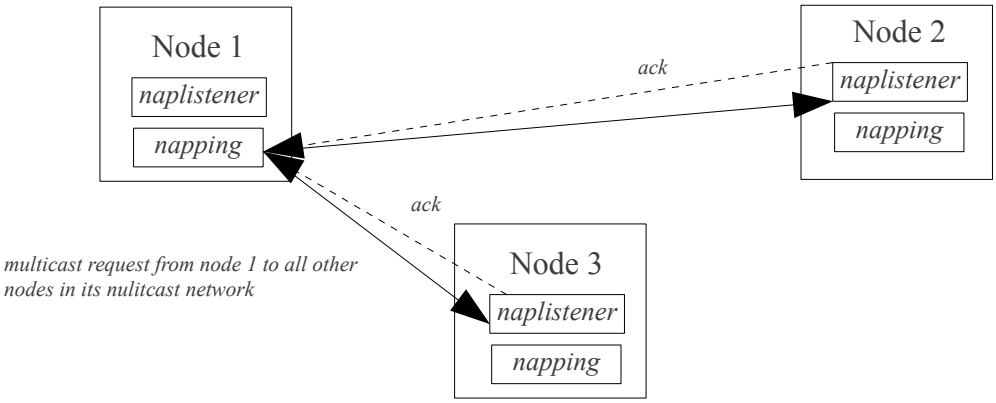

Fig. 5 The node discovery process

By default all nodes use the same multicast port and address but via the web configuration it is possible to partition a large network into smaller subnets by providing custom port and multicast ip addresses for specific groups of nodes. By default the multicast network can only communicate within its local subnet.

The Intercom App

Fig. 6 The intercom app

The intercom app lets an end user select another node initiate an audio conversation with another end user at that node via the network. They would select a peer from the gui form in the figure above and click connect. The target node will then beep to alert anyone nearby that they are in audio communication with the initiating node. The initiating user will then shut down the connection.

The connection is initiated via a web request tied to the “Connect” button in the gui. The request logic is encapsulated in the *naprtpconnect* script and the *connect.cgi* web script that is invoked on the target node from the initiating node. These, if successful, start an RTP server on each node. RTP, or Real Time Protocol, handles the network latency issues and quality of service of the audio network connections. The output from the RTP servers is then piped through the ALSA aplay program to send the audio to the sound card on the EM2440 board. At the same time audio sender processes on each node takes raw data generated by the ALSA aplay program from the sound card's microphone interface and pipes the raw data and pipe it through an RTP sender process that sends the data to the other node via the network.

The diagram below illustrates the intercom call process:

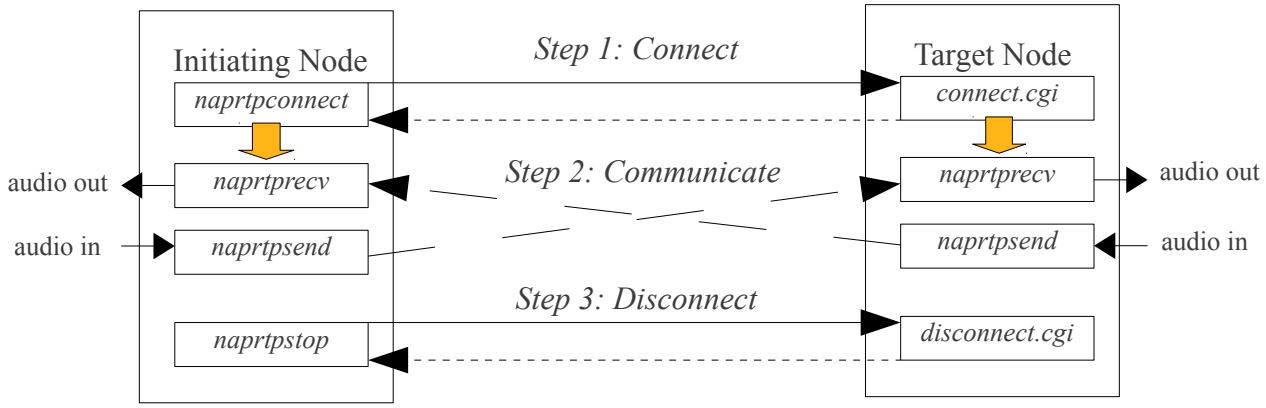

Fig. 7 The intercom process

In the first stage the *napconnect* script uses the built in wget program to signal the target node via the *connect.cgi* program via the web interface. The programs spawn the naprtpsend and naprtprecv scripts which connect the ALSA aplay and arecord programs to an RTP connection managed by oRTP. When the call is finished a similar process occurs when the user at the initiating node shuts the connection down and the *naprtpsend* and *naprtprecv* processes are shut down. The components to manage this process is almost entirely built from existing components with small changes to make it possible to use them together. The fact that these were developed as stand alone scripts that could be run independently of the gui and web interface was critical to successfully troubleshoot the system.

Conclusion/ Challenges

While this project did require a substantial amount of coding it was essential an integration project. Nevertheless it was quite challenging to find and build software that would work well with such a limited device. In particular the lack of a dsp or even a floating point unit on the device made any sound manipulation extremely time consuming. A number of features that we thought, naively, might be trivial to implement such as using sha1 checksums to verify files had to be dropped because of this limitation. The bitrate for the intercom sound capture is the lowest usable bit rate supported by our sound card. Anything higher would not reliably work.

The other major constraint was the small non-volatile storage. Between the Qt graphical libraries and libc there was little available space for the other libraries needed. This necessitated the use of external storage for basic system functions. While the boards were designed for this purpose with both sd card and usb slots, some of us had many challenges getting these to work reliably and had to rely on an nfs link to a host computer.

Our end product is useful as an edge case example of what a minimal system of this type would require but any consumer product would have to have substantially larger storage and a much faster CPU.

In our test environment we used external hosts to mimic a node in the network. Packaging the system to be used on larger servers would be a fairly easy change and would require no or minimal modification on the nodes. These larger hosts could then be used as more permanent file for the embedded network.

The multicast node identification system is not at the moment designed for security. While it is deliberately crippled to only work on a single local subnet. It is assumed that all nodes in the local network are allowed to participate in the multicast group. There is no way to unequivocally determine if a node is who they say they. Similarly, a partitioned network only partitions the naplistener/napping multicast communications they do not prevent nodes outside the local multicast group from interacting

with any other node on the local network. Password protection of the multicast groups and web interfaces would be essential in implementing an secure network. We believe this is achievable without an excessive amount of work but that it was outside of the scope of a proof-of-concept demonstration.

Going in the other direction we did start work on a bridging system that would allow the network to span firewalls. However, given the time constraints of the project we decided to emphasize local nodes over remote nodes. Nevertheless there may be instances where large installations would benefit from the ability to traverse firewalls.

What we do have is a more exact idea of what an ideal system of this type would look like. We know that we have only scratched the surface of what flexible, network aware, architecturally embedded computing could provide to consumers at reasonable cost.

Extra Hardware & Software Used

- ALSA – Advanced Linux Sound Library gave us raw connectivity with the on board sound devices.
- madplay – was used to play and navigate mp3 files.
- TCL – Tool Control Language was used to encode urls.
- oRTP – part of the linphone suite this software provides an “off the shelf” real time protocol stack. In particular we modified the rtpsend and rtpreceive test programs to work with our intercom system.

Acknowledgements

Dr. Brian Fraser went above and beyond the call of duty to help us with hardware and base linux software configuration problems.

We used a number of online guides extensively:

Qt

- http://doc.qt.nokia.com/latest/qprocess.html
- http://doc.trolltech.com/4.4/qhostaddress.html
- http://doc.trolltech.com/4.4/qnetworkaddressentry.html
- http://doc.trolltech.com/4.4/qnetworkinterface.html
- http://thesmithfam.org/blog/2010/02/07/talking-to-qt-threads/
- http://thesmithfam.org/blog/2010/02/07/talking-to-qt-threads/ - timers
- http://www2.cs.sfu.ca/CourseCentral/433/bfraser/solutions/qtthread/buttonthread.zip
- http://www.qtcentre.org/archive/index.php/t-14502.html
- http://www.qtforum.org/article/3079/howto-start-an-external-program-from-a-qt-application.html

Multicast

- http://ntrg.cs.tcd.ie/undergrad/4ba2/multicast/antony/example.html

How to find your IP address

- http://www.geekpage.jp/en/programming/linux-network/get-ipaddr.php

In addition to the abundant referencing of qt developer information at the nokia website, stackoverlow and hackchina.com provided helpful guidance for coding. Thomas also acknowledges the assistance of Jiahui Xu, a companion and frequent pair programmer – whatever small merit my effort may be credited with must be shared with her, while responsibility for errors and omissions is of course fully my responsiblity.