Rāgam: Realtime Wireless Music Streaming

CS684: Embedded Systems
Autumn 2016

Bharath Reddy
Roll No:- 16305R012
Department of Computer Science and Engineering
IIT Bombay
bharathk@cse.iitb.ac.in

Diptesh Kanojia
Roll No:-154054002
Department of Computer Science and Engineering
IIT Bombay
diptesh@cse.iitb.ac.in

Vinod Saini
Roll No:-154016003
Department of Aerospace Engineering
IIT Bombay
to.vinodsaini@gmail.com

Contents

1 Introduction

There's no doubt that listening to your favorite music can instantly put you in a good mood. In today's day and age, almost everyone plays music on digital devices like PC's, mobile devices, Apple iPods, and iPads. Digital Music or any digital audio is basically audio signals that have been encoded in digital form. Nearly all such devices are equipped with a 3.5mm audio jack or stereo pin plug, sample shown in figure 1.

Figure 1: 3.5mm Audio jack and its Compatible Pin

These jacks are universally accepted but the wires for your speaker system (however good brand it may belong to), may not necessarily be. We aim to obliterate the need of this wire using the already available wireless capability of your device. Wireless network is available in both mobile devices, and PCs for using the internet.

Our project aims at building a device which is capable of receiving music streamed from the very same wireless protocol at very low latency. We target the medium range Wi-Fi protocol (WLAN) for our work, and try to ensure that you do not loose your internet connection while listening to your favorite music on your device. Wired devices have to be connected through wires all the time and a small disruption in connection leads to unpleasant experience. Wireless solutions (like Bluetooth, Wi-Fi) can be used to get rid of this issue. There are cheap Bluetooth based music streaming devices available in the market, but they downsample the audio, which eventually reduces the audio quality. There are Wi-Fi based solutions which are either not available in India or very costly. Some of the prior work in the very same direction is listen below:

- **Trippy:** A small Wi-Fi based dongle to connect 3.5mm jack speaker to any Wi-Fi based devices to stream audio. The product was available on Witworks for crowdfunding which did not raise enough to be launched in the market. The technical specification of the product was: Micro USB charging point, size: 85 mm diameter 20 mm height, Wi-Fi hotspot mode at 802.11 b/g/n, supported audio codec: MP3 FLAC, WAV, 1000 mAh LiPo battery which could run for 5 hours without charging. This product is an app based platform where the device is connected through the app, which can't be used to broadcast audio from PC / TV.
- **Chromecast Audio:** A similiar dongle but works only when connected to the Internet, it needs a very stable internet connection, but can play music only from the browser, the desktop audio sharing capability is still in beta mode. Also, this device has high latency, and listening to music streaming through the device can feel terrible, while watching the video.

- **Sonos Play:1, Bose SoundTouch 10:** They are enabled with Wi-Fi technology, but they are very costly.
- **Gramofon:** has been an interesting solution to turn existing music system into Wi-Fi music player, but it has been limited to the clouds (internet radio stations, online music services), and is unavailable in India.

2 Problem Statement

We set out with the problem of *transferring music over wireless protocol* to *an embedded systems device*. We divide our problem into these parts:

- To develop a device which can receive streamed audio from PC / Mobile to speakers.
- The device should make use of Wi-Fi wireless protocol to receive data.
- It should be able to stream higher bitrate audio (usually 320 Kbps or more).
- The device should be able to perform without any lags whatsoever.
- Additionally, the device can be made to transmit music over FM frequency thus making it accessible over a larger range (across rooms), thus omitting even the wires to the speaker.
- *Make Speakers great again!*

We choose to use PC / Mobile as client side devices as they are the popular choice for a music listening device. On Server side (where we try to receive music wirelessly, we choose to use a Raspberry Pi Model B+.

3 Requirements

3.1 Functional Requirements

The targeted functional requirement for our device is realtime wireless music streaming, and when we say realtime, we would like the latency of music playing to be 0. Although given the current wireless protocols, and OS architectures, this seems to be a tough task to achieve. Functionally, we also require music files to be played on the system, or Internet for music streaming.

3.2 Non-Functional Requirements

We require a power source for the Raspberry Pi device, the device itself, and accessories to connect like LAN cables etc.

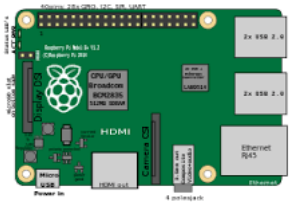

Figure 2: A Raspberry Pi Device

3.3 Harwdare Requirements

1. Raspberry Pi
2. Desktop / Laptop PC
3. Mobile Devices
4. LAN Cables (RJ45 CAT 6 Cable)
5. Speakers

3.4 Software Requirements

1. Client Side: MS Windows OS (Implementation 1, 2)
2. Client Side: iTunes (Free Music Player, Implementation 2)
3. Client Side: Linux (Implementation 2)
4. Server Side: Raspbian Jessie (Implementation 1, 2)
5. Libraries listed below

3.5 Libraries to be installed:

For Implementation 1:

- .NET Framework

- Microsoft Visual Studio 2015
- Advanced Linux Sound Architecture (ALSA) driver on Raspberry Pi server side.
- Window Audio Session API

For Implementation 2:

- OpenSSL or PolarSSL
- Avahi
- Advanced Linux Sound Architecture (ALSA)
- libdaemon
- autoconf
- automake
- libtool
- libpopt
- libconfig
- libsoxr

4 Implementation

4.1 Unsuccessful Implementations

Raspberry Pi with Music Player Daemon and MPDroid[1]

This implementation [1] worked really well, the only problem was music files have to be stored on the server first (in our case it was raspberry pi). Once the music is there, one can play music MPDroid android based app. It works other way around too, where music can be played from MPD to android phone. Raspberry pi acts as a music server. This implementation is very useful if music needs to be streamed online via online music channels.

[1] [https://www.lesbonscomptes.com/pages/raspmpd.html]

SoundWire Server[2]

This application [2] allows user to play audio from windows/linux based device to any android based device. The delay encountered in this implementation was 1-2 seconds. It works over wifi. Windows/Linux based machines work as a server and broadcast audio to an ip which is received by the application running on an android phone. We tried this implementation on Linux and to run this we also need Pulse Audio Control. This is a very nice idea where one do not need any other device for music playing, since a smartphone can act as a music streamer and convert wired speakers to wireless.

Pi MusicBox[3]

Pi MusicBox [3] is a Raspbian image for raspberry Pi to stream music based on Mopidy server. It is a complete solution to wireless music streaming where we don't require installation of any other software on RPi except Pi MusicBox image. It can be controlled by MPD Client. It allows music to stream from online music streamer as well as smartphones and computers. The pi music box can be accessed via url "http://musicbox.local/". A screenshot of the web interface is shown in figure-3[4]. This was one of the successful implementation, but the music was not streamed live, there was 4-5 second delay.

Stream What You Hear (SWYH)[5]

Stream What You Hear [4] is a MS Windows based software with open source code which helps transmit the Windows audio (complete system) to an internet stream. It was our first successful implementation which we presented during the basic implementation. SWYH creates an online server the local IP and transfers through a particular port (usually 5000). Internet streams can be listened on by using implmentations like PiMusicBox Server on a Raspberry Pi.

It worked for us though there was a 6 - 7 second lag between the audio being played, and the audio playing on speakers. The latency was very high, and hence we had to look for other methods. Although, this software did give the idea for looking towards .NET Framework for editing Windows Audio Session API (WASAPI) [5] , which is our implementation 1.

[2] [http://georgielabs.altervista.org/SoundWireHelp.html]
[3] [http://www.pimusicbox.com/]
[4] [http://www.codeproject.com/Articles/838501/Raspberry-Pi-as-low-cost-audio-streaming-box]
[5] [http://www.streamwhatyouhear.com/]

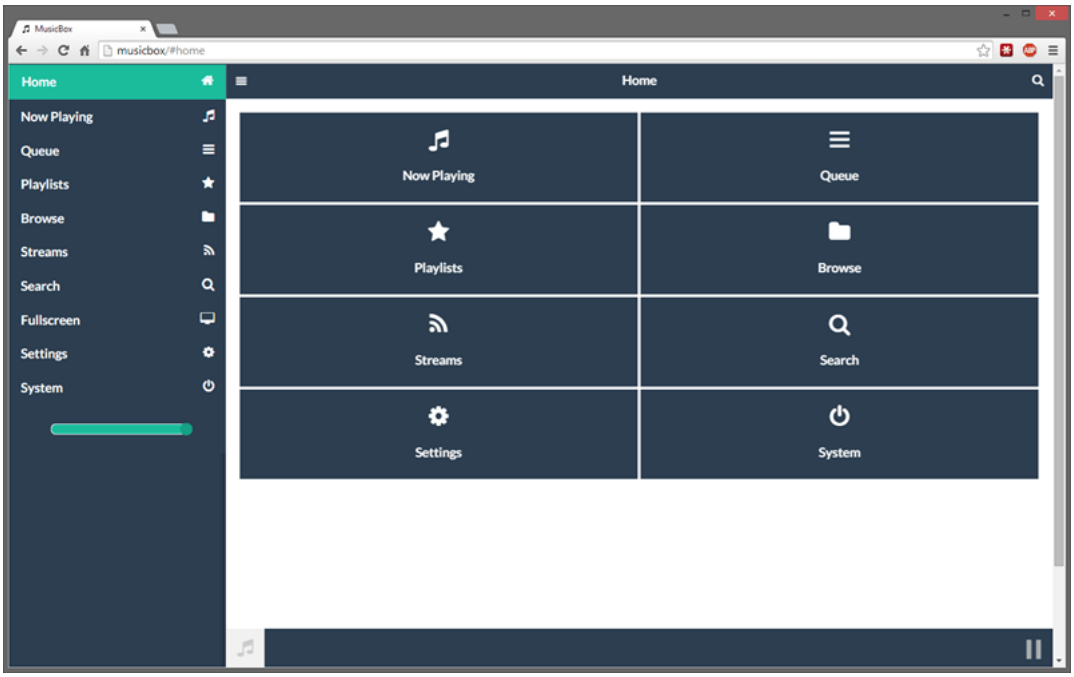

Figure 3: Pi Music Box Web Interface

4.2 Successful Implementations

1. Windows Audio Session API (WASAPI)[6] :

2. Shairport Sync[7] : Shairport sync [6] allows music to be streamed from iTune to different music player in sync. It uses timestamp information provided by iTunes player. It automatically deletes or add audio frames to be in sync with the sever. It is developed in Linux and uses ALSA (Advanced Linux Sound Architecture). We tried this implementation on Raspberry Jessie OS by following the instructions given on the webpage.

4.3 Implementation Methodology for Method 1

We implement our own code in .NET Framework using Visual Studio 2015 free version. Our implementation allows us to capture audio stream from Windows Client side, and stream wireless audio as packets through socket connection to our Raspberry Pi Server. Our code searches for sound output devices, and lists them on console. It later requests us to choose a device to capture audio.

After we enter the device number, the client side starts capturing audio buffer, and sends it to server.

[6] [https://msdn.microsoft.com/en-us/library/windows/desktop/dd371455(v=vs.85).aspx]
[7] [https://github.com/mikebrady/shairport-sync]

We start server side code to capture the audio from a port number as specified in the socket connection. We play the audio through Linux ALSA (generic sound driver in Linux).

4.4 Implementation Methodology for Method 2

Shairport-sync can be cloned from github and installed via the given instructions for Raspbian, and Ubuntu devices directly. Although they detail it in the README file, we will explain the commands, and installation procedure here.

Shairport Sync may already be available as a package in your Linux distribution (search for shairport-sync only). Packages are available on recent versions of Debian, Ubuntu 16.04, Arch, OpenWrt and possibly more. Otherwise follow the General Build Instructions. Then, when the program has been installed, refer to the section on Configuring Shairport Sync that follows.

Commands to run:

```
1. apt-get install build-essential git; - these may already be installed.
2. apt-get install autoconf automake libtool libdaemon-dev libasound2-dev
3. apt-get install libpopt-dev libconfig-dev

4. apt-get install avahi-daemon libavahi-client-dev
if you want to use Avahi (recommended).

5. apt-get install libssl-dev;
if you want to use OpenSSL and libcrypto, use PolarSSL otherwise

6. apt-get install libpolarssl-dev
if you want to use PolarSSL, use OpenSSL/libcrypto otherwise.

7. apt-get install libsoxr-dev; if you want support for libsoxr-based resampling.
This library is in pre-included many recent distributions, including Jessie and Raspbian Jessie.

8. git clone https://github.com/mikebrady/shairport-sync.git
9. cd <shairport-sync directory>
10. autoreconf -i -f
11. ./configure --sysconfdir=/etc --with-alsa --with-avahi --with-ssl=openssl --with-metadata
--with-soxr --with-systemd
(for Ubuntu 16.04 and Raspbian Jessie)

12. make
13. getent group shairport-sync \&>/dev/null || sudo groupadd -r shairport-sync >/dev/null
14. getent passwd shairport-sync \&> /dev/null || sudo useradd -r -M -g shairport-sync -s
```

```
/usr/bin/nologin -G audio shairport-sync >/dev/null

15. sudo make install
16. sudo systemctl enable shairport-sync
17. sudo update-rc.d shairport-sync defaults 90 10

(Uncomment some portion in the configuration file namely /etc/shairport-sync.conf)

The portion should look like:

general =
{
    name = "device_name";
    interpolation = "soxr";
    // ... other general settings
};

(Remember, anything preceded by // is a comment and will have no effect on the
setting of Shairport Sync.)

Also, Uncomment :
alsa =
{
  output_device = ‘‘hw:0,5’’;
  mixer_device = ‘‘hw:0’’;
  mixer_control_name = "PCM";
  // ... other alsa settings
};
```

5 Working of the System and Test results

Implementation Method 1:

We start the client side capturing using console, and run our code written in .NET on the Windows machine. It captures the audio from the sound card buffer, and creates a WAV file which is then sent through a socket connection to the server side Raspberry device. The music can then be played using alsa, which is the generic sound driver for Ubuntu / Debian / Raspbian.

Implementation Method 2:

Windows PC: The server system starts with the command “shairport-sync” in the folder where

the make install command was run. The client side PC (Windows OS) should be able to detect the device as an Airplay device in the iTunes Software. Select the device using "device_name" provided in configuration file, and play audio on iTunes. You will observe that music plays on your speakers attached to Airplay Raspberry Pi device without any lag.

Android Phone: Download the app called "AllConnect" from Play Store, and install it on your android device. Start playing any music on phone, and it will stream the audio on the Airplay Raspberry Pi device.

iPhone, iPad, Mac OS: Airplay is Apple's streaming protocol, and these device support Airplay device by default. Connect to the Airplay media device from settings, and start stream any media audio played on these devices to the Airplay Raspberry Pi device.

Test Results for our system: We are able to play the audio from Windows PC to Airplay Raspberry Pi device without any lag, pause music, play again, increase and decrease volume in realtime. Though on an android device we still face a 1 second lag, using the third party app - All Connect. This should be resolved by having our own native Android App. On the Apple devices we are able to play the music without any lag.

6 Discussion of System

- What all components of your project worked as per plan?

  We were able to stream music over Wi-Fi easily, but with a significant latency. There are various implementations available on internet using Windows/Linux/Mac/Android OS to stream music to RPI. Configuring RPi-2 was not much of a problem for Raspbian OS, though we faced difficulty in booting up RPI with OpenWRT.

- What we added more than discussed in SRS?

  We initially tried existing implementations, but most of the implementations didn't meet our requirement, except ShairPort Sync which can stream music from iTunes to Linux based OS with minimum changes in the current implementation.

- Changes made in plan from SRS?

  We made no changes in the SRS plan, but could not implement all the features that we planned to. We could not implement the Client Side Linux to Server side wireless streaming.

Problem with Raspberry Audio Card

Built-in audio card of raspberry doesn't support high quality audio. It is noisy and gets heated up after some time (5-10 mins). For high quality audio experience, we would need external audio card which can support high quality audio.

7 Future Work

With the current two way implementations, it looks cumbersome to use the device. We plan to improve the implementation method one, and expand it to be used for Linux based client side devices. We could also look at Real Time Transfer Protocol, and transfer time stamped audio packets over it, for low latency.

Eventually, we would like the device to be stripped of any extra components, include a better DAC (digital-to-analog converter), and use it to play any kind of audio on the Embedded device.

8 Conclusions

We manage to create two different implementations for a Raspberry Pi device which can act as a server to music streaming from multiple sources on the client side. The client side can be Microsoft Windows, Android based device, Apple Devices (iPhone etc.). Music can be streamed from Windows devices, and Apple devices without any lag, using both implementations, although we face a very minute lag on Android devices. We aim to better our approach by writing native system based code for Linux based client side devices, thus expanding our range of client side device types.

We completed the work in stipulated time but not without some hiccups. There were a lot of challenges, and failures along the way, but we are successfully able to stream the audio from one device to another wirelessly, and in realtime.

Links to Videos:

ScreenCast : https://www.youtube.com/watch?v=WzB7YsfObv4

Demo Video : https://www.youtube.com/watch?v=NbJbbAFp8L4

References

[1] "Raspberry Pi with Music Player Daemon and MPDroid," https://www.lesbonscomptes.com/pages/raspmpd.html, [Online; accessed October-2016].

[2] "SoundWire Server," http://georgielabs.altervista.org/SoundWireHelp.html, [Online; accessed October-2016].

[3] "PiMusic Box," http://www.pimusicbox.com/, [Online; accessed October-2016].

[4] "Stream What You Hear," http://www.streamwhatyouhear.com/, [Online; accessed October-2016].

[5] "MSDN: Windows Audio Session API (WASAPI)," https://msdn.microsoft.com/en-us/library/windows/desktop/dd371455(v=vs.85).aspx, [Online; accessed October-2016].

[6] "Shairport-sync," https://github.com/mikebrady/shairport-sync, [Online; accessed October-2016].