Synchronized audio playback over WIFI and Ethernet

A proof of concept multi-room audio playback system

FREDRIK ABEFELT

Master's Thesis at ICT
Supervisor: Christer Eriksson
Examiner: Håkan Olsson

TRITA-ICT-EX-2015:112

Abstract

This thesis aims to develop an audio playback system, which can perform synchronized audio playback on multiple devices. Two different approaches for developing the system has been investigated, one using an already existing *off the self* product, and the other using an open source framework.

The system developed is a *proof-of-concept* that can perform synchronized playback five devices, connected by Wi-Fi or Ethernet. The system developed can use Bluetooth devices or common media players as the sound source for the system.

Referat

Huvuduppgift med detta examensarbete har varit att utveckla ett synkroniserat ljuduppspelningssystem, vilket kan spela upp ljud samtidigt på flera enheter, enheterna är anslutna med antingen med Wi-Fi eller Ethernet. Två olika tillvägagångsätt har undersökts för att utveckla systemet, ett redan färdigt system och ett system baserat på ett ramverk med öppen källkod.

Det utvecklade systemet kan utföra synkroniserad uppspelning på fem olika enheter och kan använda Blueetooth enheter och olika mediaspelare som ljudkälla.

Contents

Chapter 1

Introduction

1.1 Introduction

This thesis aims to develop an audio system, witch can preform synchronized audio playback on multiple devices located on a local network. The system collects an *audio stream* from common media players or a Bluetooth device and distributes it to other devices connected on the network, which preforms synchronized audio playback.

Today there already exist similar solutions, but whats differentiates this system towards the already existing systems is that this system can preform synchronized audio playback, from common media players or from a Bluetooth device (e.g. a mobile phone). The other systems are usually dependent on that a specific application is used as the audio source, but the developed system is less dependent on which audio source is used and therefor more versatile.

1.2 Benefits, sustainability

Benefits

The benefits from this thesis is a *proof-of-concept* system for achieving synchronized playback on multiple devices using standard hardware and open source software, which can be integrated with standard audio programs, Bluetooth devices and speakers.

The system developed can be used as a foundation for new similar systems, or further development of the system with increased functionality. The intention with this thesis is to provide a concept on how to achieve synchronized playback on multiple devices, which can be useful for systems in the future.

Sustainability

The system developed can be used together with existing speakers using standard outputs, which will reduce the need of buying new speakers for achieving synchronized playback on multiple units, which is the sustainable goal of this thesis.

1.3 Method and Ethics

Method

The method used for this thesis is a trial and error approach. The reason for using such method is based on the fact that the author never have done something similar, and has no prior knowledge in the subject.

Ethics

During this thesis the ethics described in the *IEEE code of ethics* [1] is used, and with no conflicts.

1.4 Problem description

Opalum [2] is interested in developing a solution for synchronized multi-room playback. A synchronized multi-room playback system is a system that can produce synchronized audio playback on multiple devices located on different locations (i.g. play the same audio in synchronization on different devices at the same time). The audio playback units in the system should be connected to a Local Area Network (LAN), and the possibility to both use wired and wireless connection approach using Wi-Fi and Ethernet.

1.4.1 Why?

This thesis is conducted because the company Opalum are interested in a concept on how to achieve synchronized audio playback on multiple devices, that both can use a common media player or Bluetooth devices as audio source. The system developed is a proof-of-concept system with limited functionality and is not intended to be a fully functional system.

1.5 Goals

The goals for this thesis are divided in two different categories: mandatory goal and main goal.

- **Mandatory Goal:** Demonstrate synchronized playback on at least three playback units, the synchronization should be within 50 ms difference between the sink devices. The system should be connected by either Wi-Fi or Ethernet and output the audio through a connected daughter card.

- **Main Goal:** Same as Mandatory, but with 10 sinks and 30 ms difference in synchronization between the sink devices. Use a Bluetooth device as audio source.

1.6 Outline of the thesis

The remaining part of this thesis is divided into four chapters. Chapter 2 contains the background information about the subjects used within this thesis. Chapter 3 contains the implementation of two different approaches of the synchronized audio playback system and a description about a sound card implemented. Chapter 4 contains the results of the thesis and the system. Chapter 5 contains the conclusions about the thesis and a future work section.

Chapter 2

Background

In this chapter the theoretical background used within this thesis is described, and is intended to give the reader knowledge about subjects that will be used in the implementation chapter.

2.1 Opalum

This thesis was preformed in collaboration with the company Opalum [2]. Opalum is a Scandinavian company specialized in speakers and sound.

2.2 Pulseaudio

This section describes some basic knowledge about the Pulseaudio sound server. Pulseaudio has been tested as a candidate platform for the synchronized multi-room playback system in this thesis.

Pulseaudio is a open-source network distributed sound server designed for Linux operating systems. Pulseaudio is integrated in many relevant Linux distribution, such as Ubuntu and Archlinux [3]. Pulseaudio runs as a deamon, so no interaction is usually needed for controlling the program.

A sound server is a program that manages access to audio devices, usually to the sound cards. Sound servers were originally designed to provide Unix-like operating systems the possibility to mix different audio streams and produce one output stream. The reason behind needing this kind of functionality is that unix-like operating system can usually only output one audio stream, therefore its not possible e.g. play an audio file and have a movie (with sound) simultaneously playing and produce an audio stream containing sound from both sources.

A typical workflow for Pulseaudio, Pulseaudio receives calls from different programs and mixes the sound together (to a preferred format) and produces one output stream containing raw audio.

Pulseaudio also contains the functionality of distributing audio streams across a network, Pulseaudio can redirect audio streams from one computer to other computers connected on the network.

Key features of Pulseaudio [3]

- Software mixing, mixing multiple audio streams to one audio stream
- Network transparency, allow playback on an other machine located on the network

2.2.1 Setup

The Pulseaudio daemon is built with a module based approach, so for using a specific functionality provided by Pulseaudio, the correct module must be loaded.

2.2.2 Pulseaudio over network

Pulseaudio provides the functionality of distributing audio streams across a network from one Pulseaudio device to other Pulseaudio devices. Pulseaudio provides the techniques: Direct Connect, Tunneling and Real Time transport Protocol (RTP) multicast [4] for distributing audio streams across a network, the technique tested in this thesis is the RTP multicast technique.

The RTP multicast module provides the functionality of transmitting an audio stream from one source device to multiple receiving devices connected on the network, the receiving devices will preform synchronized playback of the audio stream [4], which is the requested functionality of the synchronized multi-room playback system.

2.2.3 RTP Multicast

This section contains information about the techniques used by the RTP multicast module, provided by Pulseaudio.

The RTP Multicast is a module that can be loaded in the Pulseaudio daemon, which provides the functionality of synchronized playback of an audio stream on multiple devices.

Real-Time Transport Protocol

The Real-Time transport protocol (RTP) is a network transport protocol suited for transmitting real-time data, such as audio and video over a network. The RTP can both be used over the commonly used transport protocols, Transmission Control Protocol (TCP) and User Datagram Protocol (UDP), and can be used in unicast and multicast fashion. The RTP does not provide any guarantees for synchronized delivering of packages or delivering packages in sequence.

The idea of using RTP is to let the receiver deal with out-of-order delivery and losses of packages, with the provided information from the RTP package. [5]

Internet Protocol multicast

IP multicast is a technique used for sending data packages to multiple receivers without having to send the data packages specific to each receivers. IP multicast will only transmit data to nodes connected to a specific multicast group. A multicast group is identified by its IP address. In IPv4 there is a reserved span of IP addresses reaching from 224.0.0.0 to 239.255.255.255 [6].

All data packages transmitted will be addressed to the multicast group address, the responsibility of transmitting the message to the specific node connected to the multicast group lays on the router or switch.

A multicast datagram is delivered to its destination with *best-effort* reliability, there is no guarantees of delivering datagrams intact or in order. Therefor the underlying protocol is usually User Datagram Protocol (UDP).

For registration to a multicast group the Internet Group Management Protocol (IGMP) is used. The IGMP protocol operates between the connected node and its connected router or switch. The IGMP protocol provides the service of informing the router that the node wants to join a specific multicast group.

Internet Group Management Protocol

The IGMP is used by hosts (computers connected on the network) and connected routers on the network to establish multicast memberships to multicast groups. There exist two types of messages carried by the IGMP [7].

- **Host membership Query:** used by routers to search for members of a specified multicast group
- **Host membership Report:** used by host to declare that they are members of a specified multicast group

The router will send multicast traffic to all ports in the broadcast domain, which leads to host that are not members of the multicast group will receive data packages that are unwanted (which will be discarded), extra load will be put on the network and the host, since the host device must open every package, to se if it wants it. In order to reduce the problem with sending data on all ports, a technique called IGMP snooping can be used.

IGMP snooping

IGMP snooping is a techniques that is used in the Pulseaudio implementation of the synchronized multi-room system to reduce data traffic, within the network.

The switches connected in the network eavesdrops on the IGMP traffic, the switch records which host are a member of which multicast group, and only sends multicast traffic to the host under the circumstance that the host are a member of the specific multicast group. Which reduces the traffic on the network and the incoming traffic to the host.

2.3 User Datagram Protocol

The transport protocol User Datagram Protocol (UDP) is used by both the Pulseaudio implementation and the AllJoyn implementation in this thesis.

The UDP is a connectionless transport protocol, i.e. there exist no connection between the sender and receiver. UDP provides no guarantees that the package arrive at its destination or that the package arrives in the right sequence. Application suited for using UDP are application with low latency and if a package is lost or scrabbled isn't the end of the world [8].

2.4 WIFI and Ethernet

In order to connect devices to the network both WIFI and Ethernet has been used trough out this thesis.

2.4.1 WIFI

WIFI is a name for products using IEEE 802.11 standards given by the organization WI-FI alliance. The 802.11 is a group of standards for wireless local area networks (WLAN). WLAN uses radio waves for transmitting data. The frequency used by the IEEE 802.11 is 2.4 GHz, 5 GHz and 60 GHz, which in Sweden are *free to transmit* frequency bands, if the output power is smaller then 100 mW[9].

Since the frequency bands are *free to transmit*, the medium is shared with other devices, which leads to interference, which can lead to variation in network delay (jitter) and lost data packages.

2.4.2 Ethernet

Ethernet is a group of standardized data communication methods for communication over cable for Local Area Network (LAN), Ethernet is standardized as IEEE 802.3.

2.5 Pulse Code Modulation

Pulse code modulation (PCM) is a digital representation technique for analog signals, which is often the standard modulation of audio in computers. The audio transmitted in the synchronized multi-room playback system described in this thesis uses PCM.

PCM converts the analog signal to a digital representation by sampling the amplitude of the analog signal.

The properties of PCM is the sample rate and resolution, the sample rate specifies at which rate the samples are collected from the original signal, and the resolution specifies which possible amplitude the sample can take. Higher resolution gives more precision, since the sample can take a more precise amplitudes.

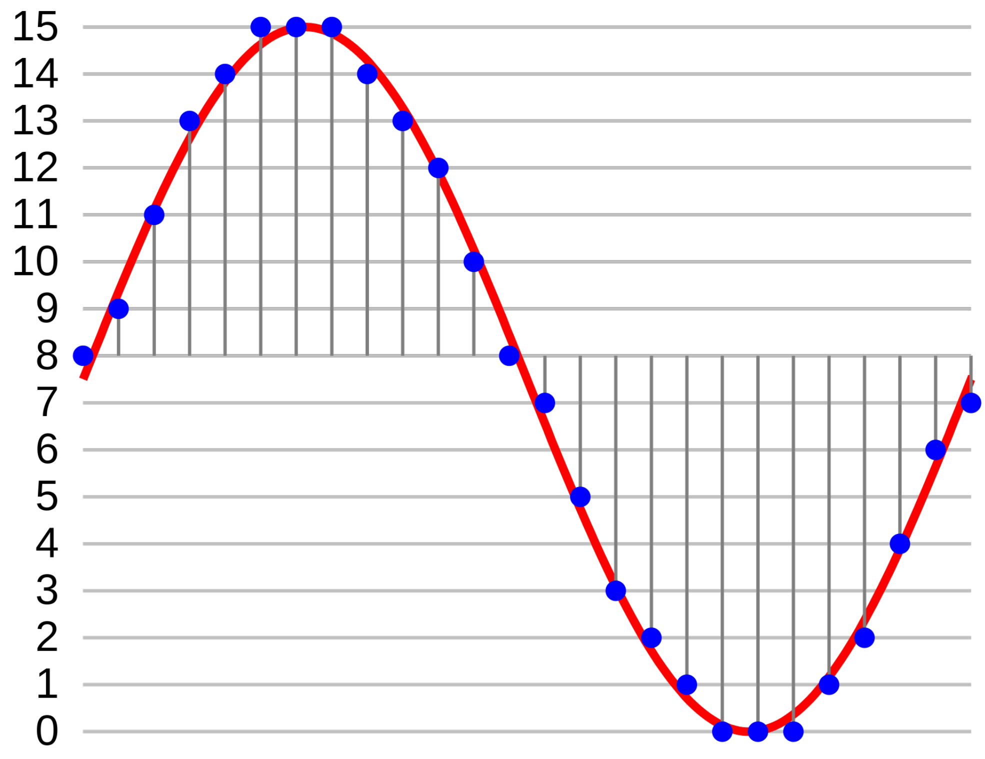


Figure 2.1. Pulse Code Modulation in the time-domain [10].

In figure 2.1 an analog signal has been coded with PCM with a resolution of four bits, which gives 16 discrete amplitudes.

In order to fully represent an analog signal digitally, the sampling rate must be greater or equal to two times the Nyquist frequency [11].

2.6 AllJoyn framework

This section describes some of the basic features of the AllJoyn framework. The AllJoyn framework has been used for the implementation of the synchronized multi-room playback system.

AllJoyn is an framework dedicated for developing systems for the *Internet of Things* (IoT). The IoT is a term used for describe the trend of connecting *things* to the Internet, the *thing* can be any type of electrical circuit, for example a lamp connected to the Internet. One of the major challenges with the IoT, is connecting devices together. The problem with connecting devices together is that different manufactures uses different technologies (WI-FI, Ethernet, Bluetooth etc) and operating systems(Android, Linux, Windows, IOS). The idea behind the framework is to provide a service to connect *things* together in a simple fashion regardless of the communication technology, manufacture and operating system.

The AllJoyn framework was originally developed by the company Qualcomm, but is now an open source framework hosted by The Allseen Alliance.

The Allseen Alliance is a nonprofit organization dedicated to drive services that supports the adoption of the IoT, trough the AllJoyn framework. Today the Allseen Alliance is supported by over 160 companies [12].

2.6.1 How does it work, cross platform and cross technology communication

The AllJoyn framework provides a distributed software bus as the communication channel. The bus is of type ad-hoc, no existing infrastructure exist, connected nodes participates in routing by forwarding messages. The transport protocol used on the software bus is network protocol independent, therefor a device using Ethernet can communicate with a device using Bluetooth.

The protocol that the distributed software bus uses is based on the D-Bus-wire protocol, which is used within Linux operating systems for inter process communication. The AllJoyn software bus extends it by supporting distributed devices on the bus.

Architecture

An AllJoyn network is composed of two different nodes, with two different roles.

- Routing Node (RN)
- Leaf Node (LN)

A routing node is responsible for managing the software bus, and routing messages across the network. A routing node can both be implemented as a standalone node or integrated in a application.

A leaf node can only communicate with routing nodes, a routing node can communicate with both routing nodes and leaf nodes. If two Leaf Nodes wants to communicate with each other they communicates through the routing node. Since the distributed messages bus is network protocol independent, two nodes can communicate even if they are not using the same transport protocol, by communicate trough the routing node (if the routing node supports both technologies) [13].

Device discovery

In order to establish communication between devices on a Network, the devices must first discover each other, since the network bus is of type ad-hoc, there is no *base station* keeping track on witch nodes are connected on the network. For discovery of other nodes on the network, an announcement message is used, there exist two methods for announcement.

- About announcement
- Well known name

2.6.2 About Announcement Message

The About Announcement message is the recommended method for announcing a device (and the method used in the implementation). The About message contains a set of meta-data which includes information about, model, name, manufacture, and supported interfaces. The most important information in the about message is the supported interfaces. When a device announces itself, the about message is broadcasted on the network, and if another device supporting the same interfaces, they have the possibilities to connect to each other and form a *session*, a *session* exist if at least two devices are connected on the same message bus (if a *session* already exist, the device joins the *session* instead) [14].

2.6.3 Audio API

The AllJoyn framework provides an audio API which offers the functionality of synchronized playback of a *local file*, the framework also provides functionality of controlling settings such as, volume, pause and mute [15]. The implementation made in this thesis adds the functionality of synchronized streaming of an *audio steam.* The difference between using a *local file* compared to using an *audio stream* is that an *audio stream* is not depended on having a local copy of the audio, but instead can be used together with a *audio program* or another source which generates a *audio stream.*

2.7 Raspberry PI

This section contains information about the hardware, operating system and media players used within this thesis.

The Raspberry PI 2 Model B has been used as the hardware platform for the synchronized multi-room playback system described in this thesis.

The Raspberry PI is a single board computer, with the size of a credit card, the specification of the Raspberry PI 2 Model B is [16]

- 900 MHz quad-core ARM Cortex-A7 CPU
- 1 GB RAM
- Full HDMI

- Ethernet Port
- 40 GPIO pins, (I2C, I2S, SPI, UART)

The Raspberry PI does not come with an operating system, it is up to the user to choose one.

2.7.1 Raspbian

This subsection contains the information about the operating system used within this thesis.

The operating system used within this thesis is the Raspbian. Raspbian is a POSIX LINUX operting system based on the Linux distribution Debian, optimized for running on the Raspberry PIs hardware.

2.7.2 Kodi

Kodi (previously known as XBMC) is an open source media center which is used within this thesis. Kodi possess the functionality of playing videos, music, pictures and games [17]. Kodi can run on the Raspberry PI using the Raspbian operating system.

2.7.3 VLC

VLC is an open source media player used within this thesis, which is capable of playing audio and video, and can run on the Raspberry PI using the Raspbian operating system [18].

2.8 Integrated Interchip Sound bus

This section contains information about the Integrated Interchip Sound (I2S) bus, which is a serial bus for transfer PCM-audio streams between integrated circuits in electronic devices. On the I2S bus, there exist two roles

- Master
- Slave

The master device, controls the clocks on the bus, and the slave device uses the clock provided by the master device [19].

The bus consists at least of three different lines, but can also include 2 extra lines.

Mandatory lines

- Bit clock line
- Word clock line

- Multiplexed data line

Extra lines (not mandatory)

- Master clock line
- Extra multiplexed data line

The bit clock controls the bit rate on the bus. The bit clock will pulse once for each discrete data bit on the data line. The Word Clock line determines which channel the incoming bits refer to. Multiplexed data line carries all data transmitted on the bus.

2.9 Daughter card

One of the goals with the thesis is to input and output audio trough a daughter card connected to the Raspberry PI GPIO pins. The daughter card uses the I2S bus to communicate with the Raspberry PI. The daughter card has the possibility to output the audio trough analog channel, S/PDIF and Bluetooth. The daughter card can also receive an input audio stream using Bluetooth.

The card is developed by Opalum, and the reason for using it is to provide better speaker connection possibilities and the ability to receive an audio stream from a Bluetooth device (e.g. a mobile phone). The daughter card act as a slave device on the I2S bus, and the Raspberry PI acts as the master.

2.10 Advanced Linux Sound Architecture

In this section the Advanced Linux Sound Architecture (ALSA) layer is described.

In Linux operating systems the ALSA layer is provided to simplify communication between user space programs and sound cards.

ALSA is an open source software framework and a part of the Linux kernel, ALSA provides an API for creation of sound card device drivers and a user space library.

The user space library provides functionality hosted by the sound card to a user space program without having direct interaction with the sound card. The API for sound card device drivers is used for writing sound card device drivers, used for communication with the hardware.

The main functionality of the ALSA userspace library is to provide the functionality of playback and capture of audio. Playback offers the functionality of playing sound on specific sound device. While captures offers the functionality of capturing sound from the sound cards input.

The main functionality functionality of the sound card device drivers API is to provide an interface for creating sound card device drivers. The sound card device drivers is used for creating an interface for sound cards, that can be used by the userspace library.

To improve ALSAs performance on embedded devices, the ALSA System on Chip layer (ALSA ASoC) is used for creating device drivers for sound cards.

2.10.1 ASoC

An ALSA ASoC device driver consists of three components for creating a sound card [20].

Codec driver - The codec driver is platform independent and contains audio interface capabilities (e.g. number of channels, sample rates and format), and codec I/O functions (e.g. determines who is the master on the I2S bus).

Platform driver - The platform driver is platform dependent and contains the information on how to retrieve the data, and which interface should be used (e.g. I2S).

Machine drivers - The Machine driver handles machine specific settings and controls, (e.g. enabling the Digital-Analog-Converter (DAC) on the receiving side of the system).

Chapter 3

Implementation

This chapter contains the implementation of the synchronized multi-room playback system, and an implementation of an ALSA sound card with the intended use of transferring audio from the Raspberry PI to the connected daughter card.

Two different approaches have been investigated to develop the synchronized multi-room playback system, one using the Pulseaudio sound server and the other one using the AllJoyn framework.

The Pulseaudio approach is based on the Pulseaudio sound server, Pulseaudio is an *off the self* product and no implementation has been done within this thesis.

The AllJoyn approach is based on the AllJoyn framework and the ALSA client library. The system is developed with the programming language C++.

The reason for testing the Pulseaudio approach is: if an already existing open source product existed on the market with the requested functionality, the scope of the thesis would lack significance.

3.1 How the Raspberry PI and the daughter card is connected

In order to output audio from the Raspberry PI to the daughter card, the I2S bus is used. The Raspberry PI acts as a master on the bus and the daughter card acts as slave device. To easily emit audio on the I2S bus, an ALSA soundcard that outputs the audio on the I2S bus is used.

Figure 3.1 shows the connection between the Raspberry PI and the daughter card.

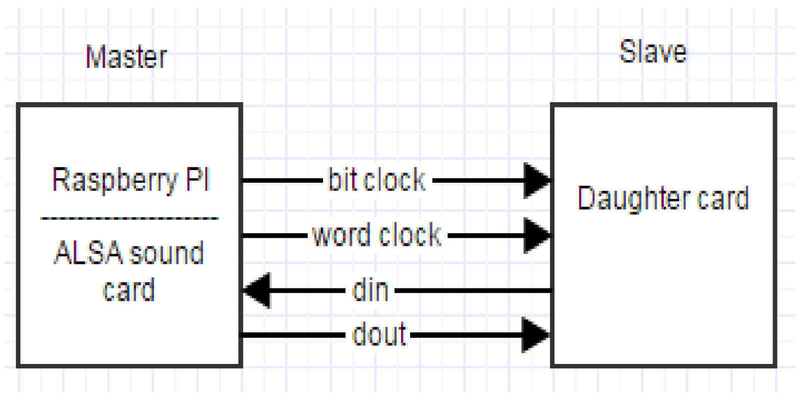


Figure 3.1. Basic Idea of I2S.

3.2 Overview of the synchronized multi-room system

The synchronized multi-room playback system consists of two different types of devices:

- The source device transmits the audio
- The sink device receives the audio

The *source device* is responsible for capturing *PCM-audio* from a program (some media player e.g.) and distribute the it across the network to each connected *sink device*. The *sink device* are responsible for receiving the audio and preform playback in synchronization with the other *sink devices*.

In figure 3.2 the basic idea of the synchronized multi-room playback system is shown with one *source device* and three *sink devices*. The *source device* captures the PCM-audio from an *audio program* and distributes it to the connected *sink devices* over the network, the *sink devices* preforms synchronized playback of the PCM-audio delivered by the *source device*.

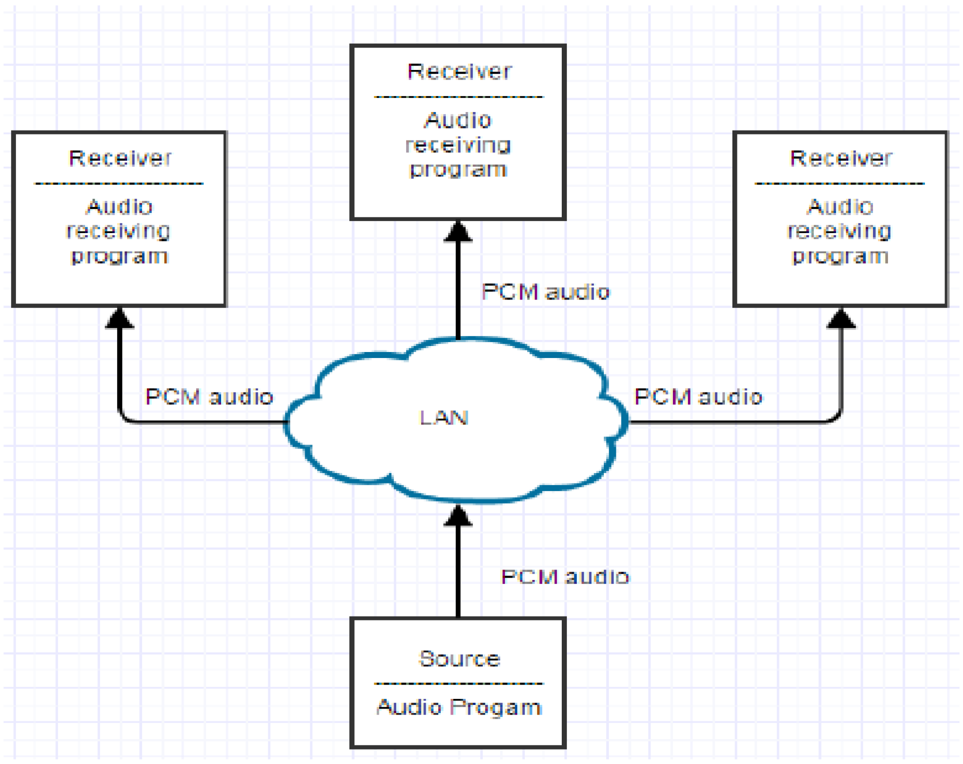


Figure 3.2. Basic Idea of the synchronous multi-room system .

3.3 ALSA sound card

This section contains the information about the sound card that is developed for emitting and receiving audio from the daughter card.

The ALSA ASoC library is used for creating a sound card, which can emit and receive audio trough the I2S bus located on the Raspberry PIs GPIO pins. The implemented sound card is based on the sound card for the Hifiberry DAC [21], which is a sound card that can be connected to the Raspberry PIs I2S bus.

Since the daughter card doesn't need any type of configuration, the sound card is a *dumb* sound card, that will emit audio onto the I2S bus without any knowledge what is connected to it.

ALSA ASoC uses three components

- Codec driver
- Platform driver
- Machine driver

3.3.1 Codec driver

The codec driver is based on existing driver named PCM5102a which is written by Florian Meier and is intended for the digital-analog-converter PCM5102a [22] from Texas Instruments.

The codec driver specifies the audio capabilities of sound card, this is the capabilities used for the sound card

Playback and capture capabilities

- Minimum number of channels: 1
- Maximum number of channels: 2
- PCM rate 8000-192000Hz
- Formats: 16bit, 24bit, 32bit

3.3.2 Platform and Machine driver

The platform driver specifies that the I2S bus should be used as output and that the PCM5102a codec driver should be used.

The machine driver controls the machine specific settings on the sound card (e.g. determine if the sound card connected, set up the initialization and hardware parameters). But since the daughter card does not need any configuration, no settings are set.

3.4 Synchronous multi-room playback using Pulseaudio

This section describes how Pulseaudio is setup, in order to use the RTP multicast module for synchronized playback.

The reason for using the Pulseaudio approach for synchronized multi-room playback system, is that Pulseaudio is a well known *of-the-self-product* that might solve the problem. Pulseaudio possesses the functionality of synchronized playback of an audio stream using the RTP multicast module.

Pulseaudio can capture a PCM-audio stream from an user space programs (under the circumstances that the user space program supports Pulseaudio). VLC is a media player that supports Pulseaudio, which was used as the audio source for the system.

If the RTP multicast modules is enabled on the source and sink devices in the system, the source device will distribute a PCM-audio stream to the connected sink devices in the system, which will preform synchronized playback.

3.5 AllJoyn framework for synchronized multi-room playback

This section contains information about the implementation of the synchronized multi-room playback system using the AllJoyn framework. First the general principle of the system is described, then a deeper description of the different parts in the system.

The reason for using the AllJoyn framework is that it is a well supported framework, has a big community and that it contains an audio API usable for this thesis.

The AllJoyn approach works in a similar way as Pulseaudio, there exist two types of devices

- Source device outputs the audio on the network
- Sink device receives the audio from the network

The source device is responsible for capturing the PCM-audio stream and distribute it to connected sink devices on the network. The source device captures the PCM-audio stream from the sound card, and distributes the PCM-audio as *data chunks* in an unicast fashion to the connected sink devices.

The sink device is responsible for receiving and playback of the *data chunks* provided by the source device, the sink device has no knowledge about the other sinks in the system. In order to achieve synchronized playback, the same *data chunks* must be played at the same time on all connected sinks, since the sink devices has no knowledge about the other sink devices in the system, it is the source device responsibility to make sure that the same data is played at the same time.

3.5.1 General idea of synchronized playback

The method for achieving synchronized playback on multiple sinks is based on that there exist one clock which is used both by the source and the sink devices. Every *data chunks* that is distributed is provided with a timestamp, the timestamp provides the functionality of determining that the *data chunks* is not to *old*. If a *data chunks* is to old, it will be discarded. The distribution uses the Internet protocol UDP with an unicast delivering approach. UDP is unreliable, package may not arrive to it's destination.

3.5.2 Source

The source device captures the PCM-audio *data chunks* from the sound card using the ALSA userspace library. The source device is independent of which program produces the PCM-audio stream (as long as the program outputs it's audio to the ALSA sound card, and that the sound card supports capture). Every *data chunk* captured is provided with a *timestamp*, the intended use of the timestamp is to prevent *old audio chunks* to be played at the sinks.

3.5.3 Sink

The sink is the receiving end of the system, which is responsible for receiving and playback of the PCM-audio provided by the source device. The job of the sink device is to receive *data chunks* and determine if the *data chunk* arrived within it's *timeframe*. The *timeframes* upper limit is set by the *timestamp* provided by the source. If the *data chunk* arrived within it's *timeframe*, the *data chunk* is added into a FIFO-buffer for playback, else the *data chunk* is discarded. The sink preforms playback of the data stored in the FIFO-buffer at a constant rate.

The System is based on that all sinks FIFO-buffers have the same data, and are played at the same rate, which will result in that the same data is played at the same time on all sinks.

3.5.4 Out of synchronization

Under optimal condition, all sink devices FIFO-buffer contains exactly the same data, all *data chunks* arrived within it's timeframe and no losses has occurred, but that is not necessarily the always the case. Since the network protocol used is unreliable, the data can arrive to late or not at all, which is a problem.

The out of synchronization problem arises when *data chunks* are not added into the FIFO-buffer for a specific sink device, this may happened due to packet loss or that the *data chunks* is too old (other sinks may have received *data chunk*, since the distribution is unreliable and unicasted). The data in the FIFO-buffer will be different between the connected sinks, if one or more sinks, but not all sinks lost a *data chunk*. When a new approved *data chunk* is added to the FIFO-buffer, the new *data chunk* will have a different FIFO-position compared to the same *data chunks* in the other sinks FIFO-buffers, because of the lost *data chunk*, which leads to unsynchronized playback.

In figure 3.3, two scenarios exist, the first scenario shows a situation were no losses has accrued and will lead to synchronized playback. The other scenario shows a situation were losses has accrued which will lead to unsynchronized playback.

In *scenario 1* at *time 0* all sinks FIFO-position are the same when the new *data-chunk* is added into the sinks FIFO-buffer. At *time 1* in *scenario 1* the *data-chunk* still have the same FIFO-position which will lead to that the *data-chunk* will be played simultaneously on all sinks sometime in the future.

In *scenario 2* at *time 0* the *sink B*s FIFO-position is different compared to the other sinks when the new *data-chunk* is added. At *time 1* in *scenario 2* the *sink B* FIFO-position still differentiates compared to the other sinks, which will result in that the *data-chunk* in *sink B* will be played earlier compared too the other sinks, which will lead to unsynchronized playback.

What this example shows is that if no losses has accrued and the sinks FIFO-position are the same it will lead to synchronized playback, since the data in the FIFO-buffers are played at the same rate. If the FIFO-buffers contains different

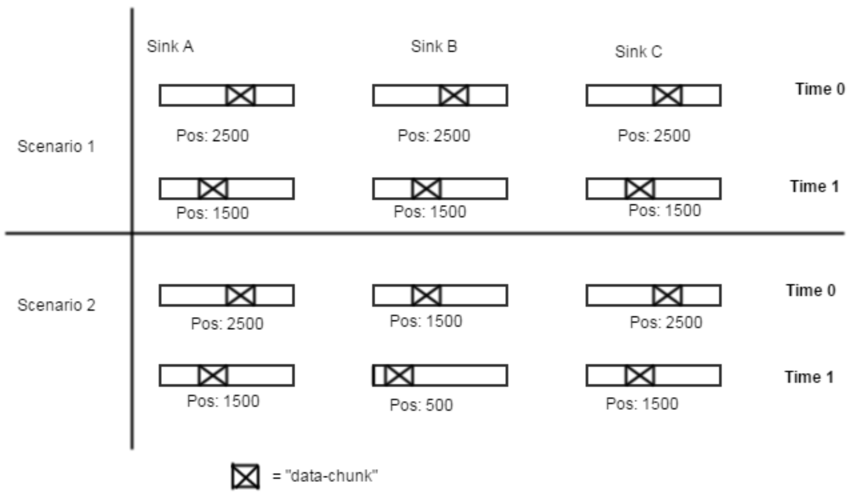


Figure 3.3. Out of synchronization scenarios

amount of data it will lead to unsynchronized playback.

The system is of type *best effort* for preventing sinks from getting out of synchronization i.e. no effort is invested in preventing sinks for becoming unsynchronized. In order to maintain synchronization, the system strives to maintain the same FIFO-positions on all connected sinks, which generates synchronized playback.

3.5.5 Out of synchronization, repair

The playback on multiple sinks will be out of synchronization if different *data chunks* are played on the sink devices at the same time. All sinks will have the same data addressed to them, but it is not a certainty that they all receive it. To detect sinks that are out of synchronization, their position in the FIFO-buffer is used. On a given interval, the source collects the sinks FIFO-position, and compare them, if the FIFO-positions differentiates to much over a time period, extra data is sent to compensate by the source.

3.5.6 Brief summary of the system

The source device captures and distributes the PCM-audio as a *data chunk* to the connected sinks. The audio is provided with a timestamp. The sink receives *data chunk* and determine if the data arrived within its timeframe. The sinks preforms playback with a given rate on data stored in the FIFO-buffer. At a given interval the source examines the sinks FIFO-positions, if the positions differentiates to much, extra data is sent to compensate. In figure 3.4 the block diagram describes which components exchange data to each other.

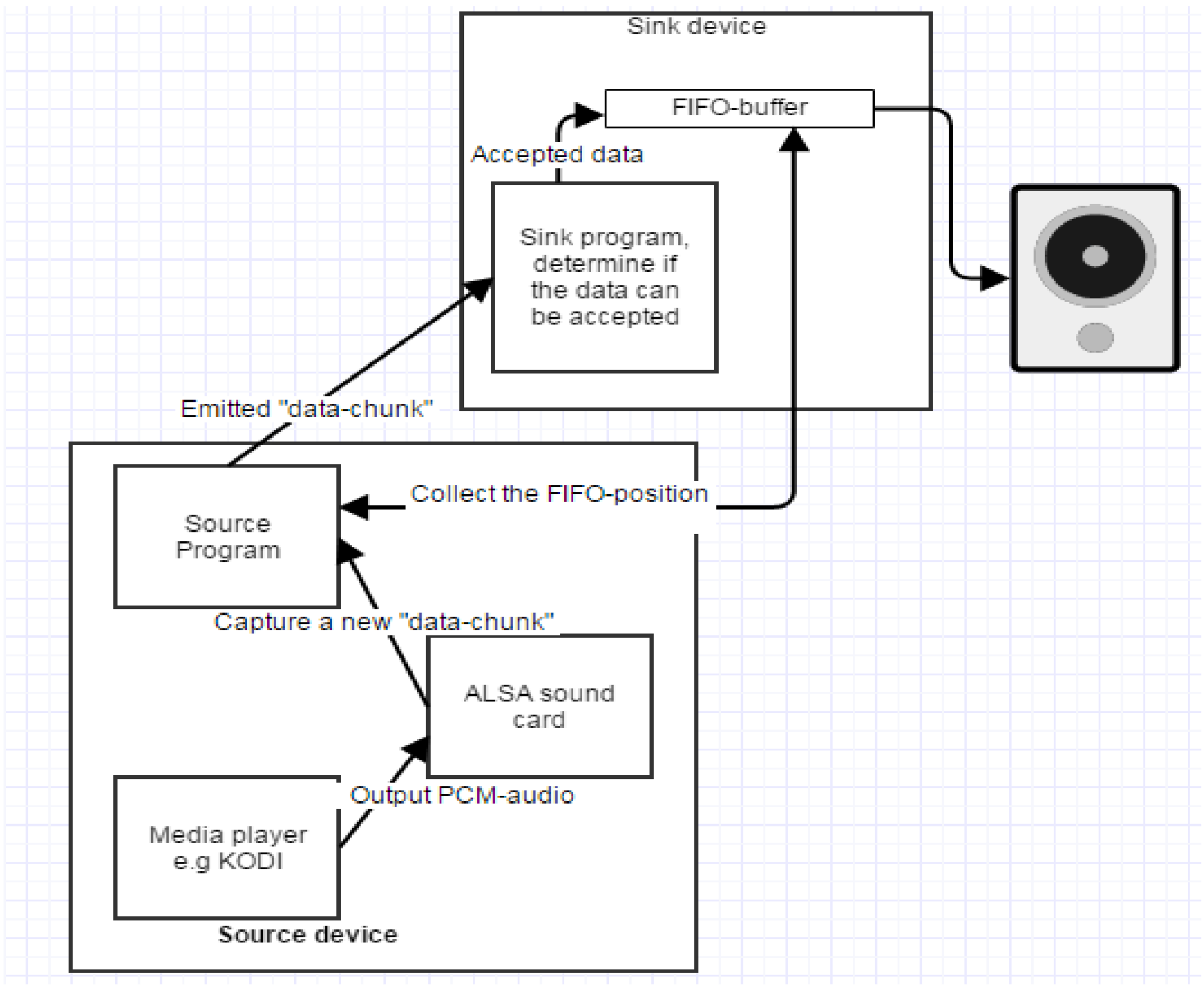


Figure 3.4. Block diagram for a source and a sink

3.6 Deeper description of the source device

The source device is responsible for capturing and distribution of PCM-audio. In order to distribute data, the source device must find and connect to sink devices on the network. The source device *searches* for new sinks on the network by looking for *about announcement* messages with matching interfaces.

When there exist a session (a session exist if at least one source device and one sink device is connected to the same message bus), an audio player is started on the source device. The main responsibility of the audio player is to provide the connected sinks with PCM-audio. The audio player starts an *emit audio thread* for each sink device connected, the thread is responsible for providing PCM-audio, and maintain synchronization (each sink device has an own *emit audio thread* in the audio player).

3.6.1 Handle a new Sink

When a new sink device is found, the new sink device internal clock must be synchronized with the source device internal clock. The clock is based on clock started at the time point of the UNIX epoch which started the year 1970 [23]. The sink device adjusts the internal clock with the source device internal clock plus the network delay in the system. The network delay is measured as the round trip time divided by two between the source and the new sink.

This will ensure that the clocks are similar, it's not necessarily for the clocks to be exact in this system.

3.6.2 Emit audio thread

The *Emit audio thread* is the *heart* of the source device, which is responsible for the main functionality

- Capture PCM-audio from the sound card
- Transmit PCM-audio to the sink device
- Repair synchronization mechanism

Extract PCM-audio from the sound card

To capture PCM-audio from a sound card (that supports ALSA capture), the ALSA userspace library is used. A sound card generally has a circular buffer that stores recorded samples. When extracting data from the sound card, the buffer keeps track of witch data has been read, which ensures that new data always will be read from the sound card (consecutive if read fast enough).

Transmit PCM-audio to the sink devices

The same *data chunks* captured from the sound card must be transmitted by every *emit audio thread* to it's respective sink device, because all sink devices need the same data. Only one of the *emit audio thread* will capture the *data chunk* from the sound card, and the *data chunk* captured will be shared with the other threads. The time of the capture will be used as a *timestamp* for the data, the *timestamp* is increased with some extra time to compensate for the time it takes for all threads to run and the delay in the system. Every thread can only run once in one session, a session is from when a new *data chunk* has been captured until all *emit audio thread* has transmitted it.

Out of synchronization, repair

On a given time interval the *emit audio thread* collects the sink devices FIFO-position. When all *emit audio thread* have collected their sink FIFO-position, the FIFO-positions are compared with the biggest FIFO-position collected, if the difference is to big, the sink is marked as *might be out of synchronization.*

If a specific sink has been marked as *might be out of synchronization* consecutive throughout a fixed time period, the sink is marked as *out of synchronization*, and will receive extra data next session to compensate for the difference in amount of data in their FIFO-buffer. The extra data is the same *data chunk* as it is suppose to emit (the same data is emitted twice).

In figure 3.5 the repair synchronization mechanism of the system is described.

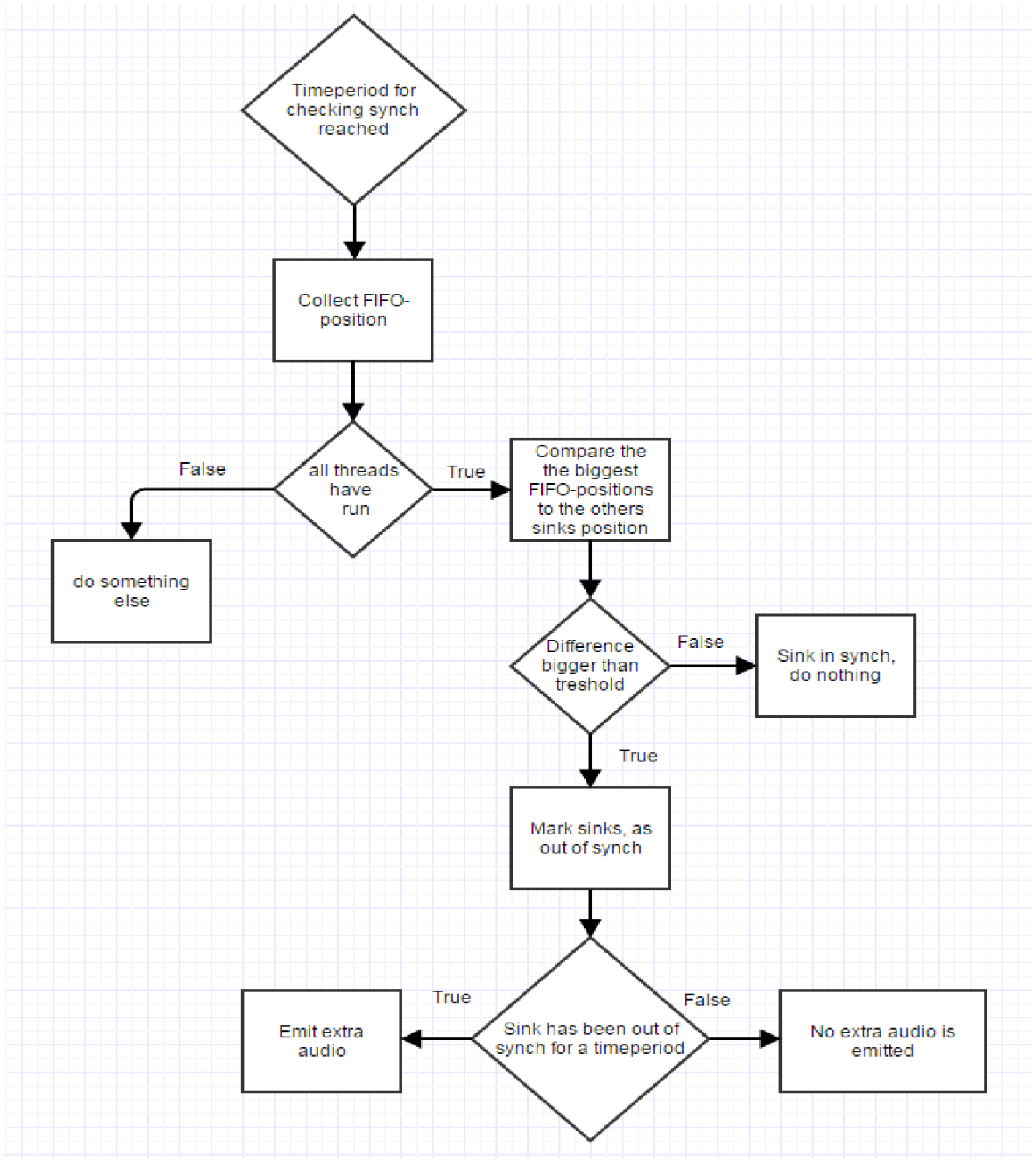

Timeperiod for checking synch reached
Collect FIFO-position
all threads have run
False
True
do something else
Compare the the biggest FIFO-positions to the others sinks position
Difference bigger than treshold
False
Sink in synch, do nothing
True
Mark sinks, as out of synch
Sink has been out of synch for a timeperiod
True
Emit extra audio
False
No extra audio is emitted

Figure 3.5. Synchronization repair mechanism

Chapter 4

Results

In this chapter the results of the implementation is discussed, and how the verification of it is conducted. The implementation discussed is

- Implementation of synchronized multi-room playback system using Pulseaudio
- Implementation of synchronized multi-room playback system using the AllJoyn framework
- Implementation of the ALSA sound card.

4.1 Result of the approach of using Pulseaudio for the synchronized multi-room playback system

In this section the results of the Pulseaudio solution for the synchronized multi-room playback system is described, two different setups of the system has been tested

- Devices connected trough Ethernet
- Devices connected both trough WIFI and Ethernet

The Pulseaudio implementation was tested using Pulseaudio 2.0 together with the media player VLC, playing a local audio file. The result of the implementation was not satisfying enough to be used in the system.

The test system contained two sink devices and one source device. The source device captured the audio-stream from the media player VLC. The synchronization within the system was measured by listing with a headphone from each sink device, and if the audio appeared to be played synchronously, the synchronization was *good enough.*

4.1.1 The sink and source devices connected trough Ethernet

The playback of the system appeared to be in *good enough* synchronization by listening with a headphone from each sink.

4.1.2 The source device connected trough WIFI and the sink devices connected trough Ethernet

The sinks couldn't preform playback, the sinks only emitted crackling sounds, it was clear that a lot of package was lost on the way. The reason for the crackling sound was because the router could not handle the rate of packages that was emitted from the source device. The bandwidth for transmitting sound onto the network is not high, but Pulseaudio emitted data package at a rate that could not be handled by the router used in the test.

In order to reduce the traffic used for the IP-multicast within the network, the technique ICMP-snooping was used but with no positive results, the router had the same problems as before.

The problem with using RTP-multicast module is a known bug, there exist bug reports tracing back to 2009 with the same bug and is not fixed today [24].

4.2 ALSA sound card

The ALSA sound card has been proven to successfully work together with daughter cards, it is possible both to use the daughter card as an input source and an output source. In order to verify the functionality of the sound card three test was preformed

- Analyzing the I2S data using a signal analyzer
- Listen to the audio, by connecting speakers
- Record input audio from the daughter card

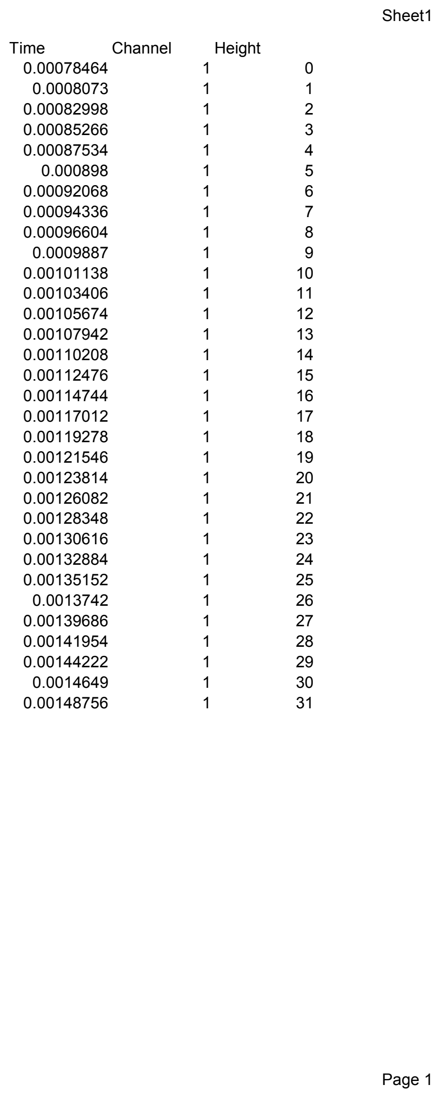
Analyze I2S data, using a signal analyzer

The idea for using a signal analyzer is to verify that the output of the I2S bus is correct. Given a known signal with expected output, the signal analyzer will analyze the signal and a human can compare the analyzed signal with the expected result. If the data matches the expected data, the emitted data is correct.

The signal analyzer used for this test is the Saleale 8 pro [25], which possess the functionality of analyzing I2S data. The Saleae 8 pro analyzes the I2S clocks and data line.

A ramp function has been used as the test signal, the ramp functions PCM-audio sample height starts from zero and is increased by one for each sample until it reaches 31 then it starts over again. The analyzer will sample the signal and displays the height of the sample.

The result of the test was a success as seen in figure 4.1 , the analyzed data matched the excepted output.



Sheet1

Time	Channel	Height
0.00078464	1	0
0.0008073	1	1
0.00082998	1	2
0.00085266	1	3
0.00087534	1	4
0.000898	1	5
0.00092068	1	6
0.00094336	1	7
0.00096604	1	8
0.0009887	1	9
0.00101138	1	10
0.00103406	1	11
0.00105674	1	12
0.00107942	1	13
0.00110208	1	14
0.00112476	1	15
0.00114744	1	16
0.00117012	1	17
0.00119278	1	18
0.00121546	1	19
0.00123814	1	20
0.00126082	1	21
0.00128348	1	22
0.00130616	1	23
0.00132884	1	24
0.00135152	1	25
0.0013742	1	26
0.00139686	1	27
0.00141954	1	28
0.00144222	1	29
0.0014649	1	30
0.00148756	1	31

Page 1

Figure 4.1. Analyzed data from the signal analyzer.

Analyze the audio

To analyze the sound, speakers were connected to the daughter cards analog output, and the outcoming audio sounded as expected.

Record input from the daughter card

To verify that the input functionality of the daughter card, a cellphone was connected to the daughter card using Bluetooth. The cellphone provided an audio stream which was recorded by the Raspberry PI. The recorded audio was played on Raspberry PI trough the analog output and sounded as expected, which verifies the input functionality of the daughter card.

4.3 Result of the implementation of the synchronized multi-room playback system using AllJoyn

In this section the results of the implementation using AllJoyn framework is discussed.

The AllJoyn implementation was tested with the Alljoyn framework version 14.12.00a together with the media player Kodi version 14.1. The test setup, consist of one source device running the Kodi media player as audio source, the source device is connected to network trough Wi-Fi, five sink devices is connected to the system either by WiFi or Ethernet (both techniques are used). In order to evaluate the synchronization, headphones are connected to the sink devices trough an analog output on the Raspberry PI.

The sound card used on the Source device is a sound card called snd-aloop [26]. Which is a virtual sound card with two devices. The audio from the media player Kodi is outputted onto first device as output, the data is mirrored to the second device as input, which can be captured by the source program.

The AllJoyn implementation also works together with the implemented sound card described in this thesis, with a cellphone as the audio source connected by Bluetooth.

4.3.1 Synchronization

The Synchronization is only measured by connecting headphones to each sink device and pairwise compare the sound. The synchronization appear to be in *good enough* synchronization, it is sometimes possible to here a small difference.

Since no effort is made to prevent the system for going out of synchronization, it is possible to loose the synchronization between sink devices, which must be repaired. The repair algorithm will emit extra data to the sink device with not enough data in their FIFO-buffers, the extra data is the same data transmitted twice, which will generate a small *crackling* sound that is audible, but will repair the out of synchronization.

4.3.2 Stability

The system has been tested that it can maintain synchronized playback on multiple sink device for several hours, with good results. The longest test preformed was eight hours test, the synchronization were manually checked sporadically through out the test, and if the synchronization was lost, the system recovered from it.

The overall stability of the system is not good, there exist several limitations that needs to be addressed. The system has a problem with recovering from a lost sink, if a sink is lost when the system is active, it either freezes or crashes. The reason for it is that there exist no mechanism for handling a loss of a sink.

4.4 Conclusions and future work

This chapter contains the conclusions about the thesis and future work of the system.

4.4.1 Goals

Mandatory goal

The mandatory goals of the thesis has not been verified completely. The mandatory goal was to achieve synchronized playback on at least three sinks using Wi-Fi and that the synchronization should be within 50 ms, and the possibility to output the audio trough the daughter card. The goal of achieving synchronized playback on three sinks was achieved, but the difference in synchronization has not been verified that it is within 50ms. The synchronization is according to me *good enough* for a proof of concept system, since it's hard to hear difference between different sinks. The goal of outputting the audio trough the daughter card has been tested with good results, since both the data is verified and it sounds correct.

Main goal

One of the main goals for this thesis was to achieve synchronized playback on at least 10 sinks, connected both trough WIFI and Ethernet, and that the synchronization should be within 30 ms. The system has only been tested with five sinks connected trough both WiFi and Ethernet. No time measurement has been done, but listening to the sinks reveals it is "good enough", no significant difference could be heard. The goal of receiving audio from the daughter card is tested with good results.

4.4.2 Implementation

In this section the thoughts about the two different approaches for the synchronized multi-room system is discussed.

Pulseaudio
The Pulseaudio sound server is not a suitable platform for a synchronized multi-room playback system, since Pulseaudio emitted data package at a rate that could not be handled by the router.

Alljoyn
The Alljoyn framework is a suitable platform for developing a synchronized multi-room playback system, since it has proven to function. The system developed in this thesis is not stable and lacks crucial functionality, but since it is a *proof-of-concept* it proves that is possible to develop a synchronized multi-room playback system based on AllJoyn framework.

4.4.3 Future Work

This section contains information about my vision for the future of the system and also suggestions on how to increase the number of sink devices and extra functionality.

My future vision for the system

My future vision of the system is

- Increased functionality
- At least 10 sink devices can be connected
- Stable enough to handle connecting and disconnecting sink devices, without loosing synchronization

My vision of the increased functionality is that the user is able to choose which sinks that should be connected to a session, the user should be able e.g. to choose the sink devices located in the kitchen and living room for playback, but not the bedroom.

Increase the amount of sinks

To increase the amount of sinks that can be connected to the system a *smarter* thread handling of the *emit audio thread* might be needed.

My Idea
The *emit audio thread* should only run once every session (a session exist from when a new *data-chunk* is collected from the sound card until all *emit audio thread* has transmitted the *data-chunk* to its sink device), and when a thread has executed, the thread shall remain blocked for the remaining part of the session and unblocked when a new session is started. That approach will guarantee that each thread only tries to executes once every session.

Current implementation

In the current implementation the *emit audio thread* it is only allowed to execute its functionality once every session, but it is possible for the thread to try to execute multiple times during a session. When an *emit audio thread* has executed, the thread will be blocked for a time period. After the time period ended, the thread is ready to execute again, which means that a *emit audio thread* can try to execute multiple times during a session, which is a waste of execution time.

Idea to add functionality for choosing witch sinks to include in a session

Every sink device can provide a name within its *about message*, the source device can inspect the *about message*, and can choose to accept or reject a sink based on the name provided.

Appendix A

Appendix

A.1 Run the AllJoyn applications

This section contains information about how to run the system, there exist two different programs, one source program and one sink program.

The sink program is called *SinkService* and the source program is called *SinkClient*.

Start the source and sink

To start the applications the environment variable to the library must first be set:

- ,**set the environment variable** export LD_LIBRARY_PATH=$LD_LIBRARY_PATH:/home/
14.12.00a-src/services/audio/build/linux/arm/debug/dist/audio/lib

A router node must be started at both the source device and the sink device. To start the router node:

- **change directory:**
cd /home/pi/WD/alljoyn-14.12.00a-src/services/audio/build/linux/arm/debug/dist/cpp/bin
- **Run the router application**
./alljoyn-deamon

To start the sink and source program:

- **change directory**
cd /home/pi/WD/alljoyn-14.12.00a-src/services/audio/ build/linux/arm/debug
/dist/audio/bin/samples
- **Run the source program:**
./SinkClient

- **Run the sink program:**
  ./SinkService

A.2 Compile the code

Compiling the source code:

- **change directory:**
  cd /home/pi/WD/alljoyn-14.12.00a-src/services/audio

- **Compile:**
  scons OS=linux CPU=arm WS= off OE-BASE= /usr BINDINGS=cpp CROSS-COMPILE /usr/bin/arm-linux-gnueabihf-

A.3 Cross-compile the Linux kernel for Raspbian operating system

This section contains instructions on how to recompile the Linux kernel, which have been used to build the ALSA sound card described in this thesis.

In order to cross-compile the Raspbian Kernel, a computer running Linux must be used, the operating system used in this thesis is Ubuntu 14.04 LTS.

Follow the steps described in [27].

Bibliography

[1] *Ieee code of ethics*, [accessed 16-June-2015], 2015. [Online]. Available: http://www.ieee.org/about/corporate/governance/p7-8.html.

[2] *Opalum — opalums homepage*, [Online; accessed 12-June-2015], 2015. [Online]. Available: http://www.opalum.com/.

[3] *Pulseaudio — information about pulseaudio, pulseaudio sound server*, [Online; accessed 12-June-2015], 2015. [Online]. Available: http://www.freedesktop.org/wiki/Software/PulseAudio/About/.

[4] *Pulseaudio — pulseaudio network setup, pulseaudio sound server*, [Online; accessed 12-June-2015], 2015. [Online]. Available: http://www.freedesktop.org/wiki/Software/PulseAudio/Documentation/User/Network/.

[5] *Real-time transport protocol*, [Online; accessed 12-June-2015], 2015. [Online]. Available: http://tools.ietf.org/html/rfc3550.

[6] *Ip multicast*, [Online; accessed 12-June-2015], 2015. [Online]. Available: http://tools.ietf.org/html/rfc1112.

[7] *Internet group management protocol, version 3*, [Online; accessed 12-June-2015], 2015. [Online]. Available: http://tools.ietf.org/html/rfc3376.

[8] *User datagram protcol*, [Online; accessed 12-June-2015], 2015. [Online]. Available: https://tools.ietf.org/html/rfc768.

[9] *Post- och telestyrelsens författningssamling [swedish]*, [Online; accessed 12-June-2015], 2015. [Online]. Available: http://www.pts.se/upload/Foreskrifter/Radio/ptsfs-2013_4-undantag-tillstand.pdf.

[10] *Pulse code modulation picture*, [Online; accessed 16-June-2015], 2015. [Online]. Available: https://upload.wikimedia.org/wikipedia/commons/b/bf/Pcm.svg.

[11] *Nyquist frequency*, [Online; accessed 12-June-2015], 2015. [Online]. Available: http://mathworld.wolfram.com/NyquistFrequency.htm.

[12] *Why use allseeen*, [Online; accessed 12-June-2015], 2015. [Online]. Available: https://allseenalliance.org/about/why-allseen.

[13] *The alljoyn architecture*, [Online; accessed 12-June-2015], 2015. [Online]. Available: https://allseenalliance.org/developers/learn/architecture.

[14] *The alljoyn discovery*, [Online; accessed 12-June-2015], 2015. [Online]. Available: `https://allseenalliance.org/developers/learn/core`.

[15] *The alljoyn audio service*, [Online; accessed 12-June-2015], 2015. [Online]. Available: `https://allseenalliance.org/developers/learn/base-services/audiostreaming/interface`.

[16] *Raspberry pi 2 model b specification*, [Online; accessed 12-June-2015], 2015. [Online]. Available: `https://www.raspberrypi.org/products/raspberry-pi-2-model-b/`.

[17] *Kodi media center*, [Online; accessed 12-June-2015], 2015. [Online]. Available: `http://kodi.tv/about/`.

[18] *Information about the media player vlc*, [Online; accessed 12-June-2015], 2015. [Online]. Available: `http://www.videolan.org/vlc/`.

[19] *I2s bus specification*, [Online; accessed 12-June-2015], 2015. [Online]. Available: `https://www.sparkfun.com/datasheets/BreakoutBoards/I2SBUS.pdf`.

[20] *Alsa asoc*, [Online; accessed 12-June-2015], 2015. [Online]. Available: `http://www.alsa-project.org/main/index.php/ASoC`.

[21] *Information about the hifiberry*, [Online; accessed 12-June-2015], 2015. [Online]. Available: `https://www.hifiberry.com/dacplus/`.

[22] *Information about the chip pcm5102a*, [Online; accessed 12-June-2015], 2015. [Online]. Available: `http://www.ti.com/product/pcm5102A`.

[23] *Information about the audio stream interface*, [Online; accessed 12-June-2015], 2015. [Online]. Available: `hhttps://allseenalliance.org/developers/learn/base-services/audiostreaming/interface`.

[24] *Bug in the pulseaudio rtp-multicast module*, [Online; accessed 12-June-2015], 2015. [Online]. Available: `https://bugs.launchpad.net/ubuntu/+source/pulseaudio/+bug/411688`.

[25] *Information about the signal analyzer saleae 8 pro*, [Online; accessed 12-June-2015], 2015. [Online]. Available: `https://www.saleae.com/`.

[26] *Information about the sound card snd-aloop*, [Online; accessed 12-June-2015], 2015. [Online]. Available: `http://www.alsa-project.org/main/index.php/Matrix:Module-aloop`.

[27] *Cross-compiling the raspbian kernel*, [Online; accessed 16-June-2015], 2015. [Online]. Available: `http://noiseisgood.com/howto-easy-i2s-audio-with-your-raspberry-pi/`.

[28] *Operating systems for the raspberry pi*, [Online; accessed 12-June-2015], 2015. [Online]. Available: `https://www.raspberrypi.org/downloads/`.

[29] *Information about the raspberry pi*, [Online; accessed 12-June-2015], 2015. [Online]. Available: `https://www.raspberrypi.org/help/what-is-a-raspberry-pi/`.

[30] *Information about the tas2552 evm*, [Online; accessed 12-June-2015], 2015. [Online]. Available: `http://www.farnell.com/datasheets/1853364.pdf`.