**Android를 통한 여러 기기의 Audio Stream 출력**

* 소 속 : 부산 소프트웨어 멤버십
* 작성자 : 고성민, 손민수, 김재원
* 작성일 : 2016.02.28

**S/W멤버십 과제 기획서**

|  |  |  |  |
| --- | --- | --- | --- |
| **과 제 명** | Android를 통한 여러 기기의 Audio Stream 출력 | | |
| **과제구분** | 창의과제 | | |
| **과제기간** | 2015.12.01 ~ 2016.03.01 (3개월) | | |
| **지 역** | 부산 멤버십 | **참여인원** | 3명 |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **회 원 명** | **학교** | **학과** | **학년** | **연락처** | **E-mail** |
| 고성민 | 부경대 | IT융합응용공학과 | 3 | 010-5090-9297 | wbfkrl1253@naver.com |
| 손민수 |  |  |  |  |  |
| 김재원 |  |  |  |  |  |

|  |  |
| --- | --- |
| **구분** | **내용** |
| **개발 목적**  **및 동기** | 한 디바이스의 소리를 듣기 위해서는 음향장치가 필요하다. 이러한 음향장치가 없는 경우 해당 디바이스에서 재생되는 소리를 들을 수 없으며, 간편하게 휴대할 수 있는 이어폰이 있더라도 유선 방식이므로 고려해야 하는 많은 불편한 상황이 발생한다. 이 때, 다른 디바이스의 내장된 음향장치를 통해 소리를 재생시켜 편리함을 제공하고자 한다. 또한, 이는 여러 디바이스에서 나는 소리를 하나의 디바이스에서 제어할 수 있도록 한다. |
| **개발 환경**  **및 일정** | - 운영체제 : Windows Vista 이후, Android KitKat/Lollipop  - 개발 툴 : Eclipse Luna, Android SDK, Visual Studio 2013  - 개발 언어 : Java, C, C#, PHP  - 개발 일정 : 2015.12.01 ~ 2016.03.01 (3개월) |
| **창의성/**  **우수성** | 간편한 인증 절차를 통해 사용자가 보다 손쉽게 해당 프로그램을 사용할 수 있도록 한다. 또한, Android에서 각 기기의 볼륨 조절을 가능하게 한다. 이를 통해 사용자는 원하는 기기의 소리만을 제어할 수 있다. Delay를 줄이기 위해 비동기화 방식을 사용하여 더욱 빠르게 받을 수 있고, 안정적이지 않은 부분을 보안하고자 작은 크기의 패킷으로 보내고자 한다. |
| **활용성/**  **사업성** | Bluetooth가 아닌 Wi-Fi 통신을 함에 따라 연결수의 제한이 없다. 따라서 모든 디바이스의 소리를 하나의 스마트폰으로 제어함을 가능토록 한다. 또한 더 나아가 스마트TV 등의 더 다양한 환경에서 소리 제어가 가능하다면 더욱 활용도가 높아질 것으로 예상한다. |

**목 차**

**- 내용 목차 -**

[1. 개발 목적 6](#_Toc436792461)

[1) 통합 스피커 환경 6](#_Toc436792462)

[2) 개인 스마트 기기 사용량 증가 7](#_Toc436792463)

[3) 오디오 장치의 제한 7](#_Toc436792464)

[4) 스마트 폰의 성능 향상 8](#_Toc436792465)

[5) 간편한 연결 9](#_Toc436792466)

[2. 개발 목표 10](#_Toc436792467)

[1) 인증 서버 구현 10](#_Toc436792468)

[2) 윈도우 오디오 캡쳐 10](#_Toc436792469)

[3) 안드로이드 오디오 캡쳐 10](#_Toc436792470)

[4) 안드로이드 소리 재생 10](#_Toc436792471)

[5) PC와 Android, Android와 Android 간의 통신 11](#_Toc436792472)

[3. 개발 내용 11](#_Toc436792473)

[1) 전체적인 시스템 아키텍처 11](#_Toc436792474)

[2) 인증 서버 구현 12](#_Toc436792475)

[3) 클라이언트 Android Audio 전송 13](#_Toc436792476)

[4) Android Audio 재생 14](#_Toc436792477)

[5) Loopback Recording 18](#_Toc436792478)

[6) Windows Volume Control 20](#_Toc436792479)

[7) 시나리오 21](#_Toc436792480)

[4. 개발 일정 24](#_Toc436792481)

[5. 용어 정리 24](#_Toc436792482)

[1) Android System Service 24](#_Toc436792483)

[6. 참고문헌 25](#_Toc436792484)

**- 그림 목차 –**

그림 1 다양한 IT 기기의 사용 6

그림 2 1인당 평균 사용 IT 기기 7

그림 3 개인 IT기기 사용의 중가 추세 7

그림 4 갤럭시 시리즈의 성능 향상 8

그림 5 네이버 페이의 간편한 인증 절차 9

그림 6 Audio Communication By UDP 11

그림 7 System Flow Chat 12

그림 8 웹 서버 15

그림 9 인증 절차 16

그림 10 서버/클라이언트 전송 방식 16

그림 11 Android Speaker 동작 Application Architecture 19

그림 12 AudioTrack Audio 출력 20

그림 13 Android Audio System 21

그림 14 JNI를 통한 Application AudioTrack과 Native AudioTrack 연결 22

그림 15 Audio Effect 구조 25

그림 16 Audio Equalizer 설계 26

그림 17 오디오 데이터 흐름 30

그림 18 어플리케이션 첫 화면 35

그림 19 스피커 디바이스 화면 35

그림 20 이퀄라이저 36

그림 21 Background 36

그림 22 Client 화면 37

그림 23 접속 진행 화면 37

**- 표 목차 -**

[표 1 Audio Format 10](#_Toc445229794)

[표 2 Android Audio Stream Type 18](#_Toc445229795)

[표 4 개발 일정 40](#_Toc445229796)

[표 5 Audio System Service 41](#_Toc445229797)

[표 6 반영되는 볼륨 종류 41](#_Toc445229798)

# 개발 목적

## 통합 스피커 환경



그림 다양한 IT 기기의 사용

우리는 다양한 개인 IT기기를 사용하고 이 기기들에서는 각자의 소리를 발생시킨다. 여러 대를 사용하는 경우 알림부터 일반 어플리케이션의 소리까지 하나씩 듣고 제어하기엔 어려움이 따른다. 혹은, 데스크탑의 경우, 스피커장치가 따로 장착되어 있지 않으며 책상 밑 본체에 이어폰을 연결하기에 불편함이 따를 수 있다. 본 과제를 통해 이런 모든 경우에 통합적으로 사용할 수 있는 스피커를 제공해주고자 한다. 한 대의 Android기기로 다른 여러 기기의 소리를 받아와 스피커로 출력해준다면, 선이나 장소, 장치의 여부에 구애받지 않고 각 기기의 오디오를 들을 수 있다. 또한, 여러 디바이스에서 나는 소리를 스피커로 연결된 하나의 Android에서 제어할 수 있도록 하여 편리함을 제공해주고자 한다.

## 개인 스마트 기기 사용량 증가

|  |  |
| --- | --- |
| 그림 1인당 평균 사용 IT 기기 | 그림 개인 IT 기기 사용의 증가 추세 |

우리의 생활이 더욱 개선되고 발전됨에 따라 IT 기기의 보급량도 증가하고 있다. PC, 노트북, 테블릿, 스마트 폰, 스마트 워치와 같은 개인 기기들이 계속해서 출시되고 사용자들은 필요에 따라 여러 IT기기를 사용한다. SOPHOS에서 조사한 위 그림에 따르면 평균 개인이 사용하는 IT 기기 수는 2.9에 달한다. Cisco IBSG에서는 2015년엔 3대가 넘고, 2020년에 무려 6대에 달하는 기기를 사용할 것이라고 예상하고 있다. 더군다나 사물인터넷(IoT, Internet of Things) 시대가 도래하면서 주변의 스마트 기기들의 점유가 굉장히 높아졌고 이러한 스마트 기기가 없는 곳을 찾기 힘들 정도가 되었다. 이러한 기기들은 각자의 소리를 발생시키고 개인 사용자가 이 모든 기기를 제어하고, 알림을 받기 위해서는 어려움이 따른다. 따라서 이 많은 IT기기에서 발생 하는 소리를 한 디바이스로 보내 단 하나의 기기를 제어하는 것만으로도 다른 기기들의 알림 소리를 들을 수 있는 환경을 제공하고자 한다.

## 오디오 장치의 제한

상황에 따라 오디오 장치가 없는 경우가 발생할 수가 있다. 데스크탑에 연결할 오디오 장치가 없거나 갑작스레 스피커가 고장 날 수도 있다. 혹은, 여러 대의 기기를 사용하고 있을 때 여러 기기에서 발생하는 소리를 듣기 위해 매번 이어폰을 번갈아 꼽은 적이 한번씩은 있을 것이다. 특히 카페와 같은 공공장소에서 사용 중이었다면 더욱 번거로울 것이다. 이어폰 선을 바꿔 연결하다 다른 것들을 흘려버려 난처했을 수도 있다.

이 경우 한 디바이스에서 모든 소리를 모아 제어한다면 더욱 간편해질 수 있다. 굳이 이어폰을 바꿔 꽂을 필요도, 그 때문에 다른 물건들을 쏟을 이유도 없다. 그냥 지금 연결되어 있는 기기로 다른 기기의 소리를 들으면 된다. 또한, 개인 공간인 경우, 불편하게 선이 늘어져있는 이어폰을 사용할 필요도 없이 스마트폰으로 듣고 싶은 디바이스의 소리를 들을 수 있다. 모니터에 멀찍이 떨어진 침대에서 바로 옆에 있는 스마트폰으로 소리를 들으며 영화를 즐길 수도 있다. 갑작스럽게 스피커가 고장 났을 경우에도 출력장치가 내장되어있는 개인의 다른 기기로 소리를 출력 할 수 있다.

본 과제에서는 좀 더 유용하고 융통성 있는 상황에 맞추고자 소리를 재생시킬 수 있는 기기는 확실히 스피커가 내장되어 있는 Android 기기로 제한하고자 한다.

## 스마트 폰의 성능 향상



그림 갤럭시 시리즈의 성능 향상

최근 스마트폰의 성능은 더 이상 좋아질 수 없다고 말 할 정도로 많이 발전했다. 스마트폰의 두뇌인 애플리케이션 프로세서(AP)은 퀄컴의 스냅드래곤 805까지 올라갔는데 2.65GHz에 달하는 클럭을 자랑한다. 이렇게 프리미엄 스마트폰이 아니더라도 많은 스마트폰이 스냅드래곤 800을 장착하고 있는데, 이 또한 2.45GHz의 클럭을 가진다. 갤럭시와 같은 안드로이드 계열은 스냅드래곤에 오디오칩이 내장되어 있는데, iOS에 장착되어 있는 ‘시러스 로직’사와의 비교에도 우세할 만큼 좋은 성능을 자랑한다.

하드웨어 스펙이 높아짐에 따라 전력 소비량도 늘고 있는 것이 사실이다. 이에 배터리 기술도 많은 관심을 받고 있다. 최근 나오는 스마트폰의 경우 꽤 높은 배터리 효용성을 자랑한다. 좋은 기종의 경우 2550mAh의 배터리 용량을 가지고 있으며, 평균적으로도 6시간 이상의 지속시간을 가진다.

## 간편한 연결

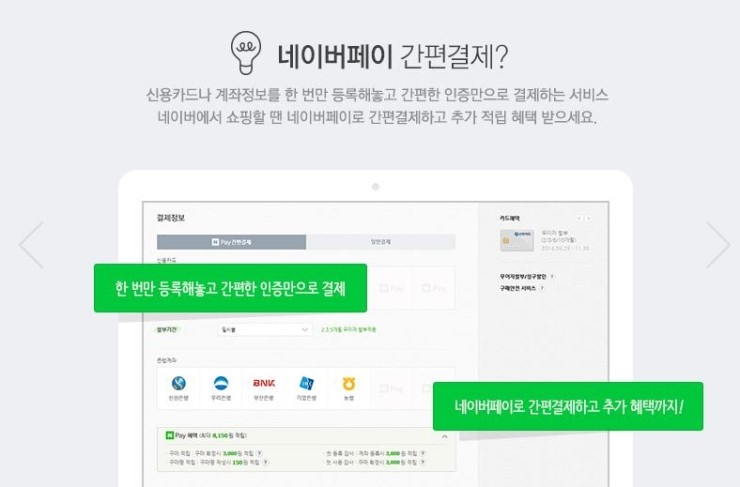


그림 네이버 페이의 간편한 인증 절차

사회가 발전함에 따라, 데이터를 전송하는 시간은 빨라지고, 그것을 안전하게 송수신 하기 위해서 보호하는 방법들은 좀 더 복잡하게 발전하고 있다. 이러한 안전한 연결을 보장하고자 복잡한 인증절차를 거치기도 하지만 그 많은 과정을 따라해도 완벽하게 안전한 프로토콜은 없을뿐더러 복잡하다는 부정적인 의견이 드세다.

때문에 굉장히 다양한 기능의 어플리케이션들이 쏟아져 나오고 있지만 사용자들의 반응은 냉정하다. 아무리 좋은 기능을 갖추고 있더라도 사용하기에 복잡하고 까다롭다면 사용을 포기해버리는 것이 요즘 사용자들의 추세이다. 큰 관심을 얻고 있는 핀테크(Fin Tech) 서비스들을 살펴보면 즉각적인 반응을 볼 수 있다. 네이버 페이의 경우 복잡한 인증절차의 간소화 이후 2위인 카카오 페이의 3배정도의 사용자를 보유할 정도로 인기를 받고 있으며 최근 출시된 삼성페이의 경우도 간편한 인증절차로 주목을 받았다. 또한 TV광고를 시청하다 보면 단 몇 초의 시간으로 원하는 일을 할 수 있다는 광고를 많이 볼 수 있다.

본 과제에서 연결을 위해서는 각 기기의 내부 IP주소와 보안 기능을 실현할 수 있는 패스워드가 필요하다. 그러나 사용자들이 내부 IP를 알고 있을 리 만무하고 패스워드의 경우 새로운 패스워드는 잊어버리기 일쑤에 기존 사용하던 패스워드는 위험에 노출될 가능성이 있다. 따라서 해당 시스템에서는 4자리의 각각 다른 난수를 발생시켜 주어진 4 자리의 번호 입력만으로 기기들간의 통신을 가능하게 하는 등 인증절차를 간소화 할 예정이다.

# 개발 목표

## 인증 서버 구현

윈도우에서 안드로이드, 안드로이드에서 안드로이드로 소리를 전달하기 위해서 소켓 통신을 하게 되는데, 소켓 통신을 하려면 server와 client 간의 ip 주소를 알아야 한다. 그러나 일반 사용자들은 이러한 전문적인 지식이 부족하기 때문에 사용자들에게 편리한 연결 방법을 제공 하고자 한다.

인증 서버에서는 안드로이드 Application의 Speaker가 자신의 ip를 인증 서버에게 알려 주고, 임의의 숫자 4자리를 Speaker에게 알려주고, Speaker는 이 숫자를 사용자에게 알려 주어, Client가 Speaker의 IP를 입력 할 필요 없이, 4자리 숫자를 이용하여 Speaker에 쉽게 접속 할 수 있도록 한다.

서버는 각 디바이스 간의 편리한 연결을 위해 이용되며, 서버를 통해 소리를 스트리밍 하지는 않는다. 소리 재생은 각 디바이스간에 udp 통신을 통해 이루어 진다.

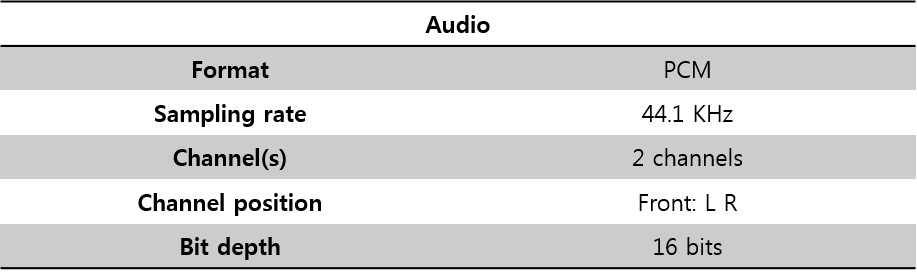
## 윈도우 오디오 캡쳐

윈도우에서 발생하는 출력 오디오 스트림을 캡쳐하여 네트워크를 통해 Android로 보내줄 수 있도록 한다. 윈도우의 오디오 동작을 제어할 수 있는 Core Audio API를 사용하여 Audio stream을 캡쳐 할 수 있도록 구현한다. 이 중 출력 Stream을 잡음 없이 캡쳐 할 수 있는 loopback recording 기능을 사용한다. Delay를 줄이기 위하여 보낼 수 있는 파일 크기를 최소화 하여 전송할 수 있도록 한다. Android에서 볼륨 제어 요구가 올 경우 ISimpleAudioVolume를 사용하여 볼륨 조절이 가능할 수 있도록 한다.

## 안드로이드 오디오 캡쳐

클라이언트 Android에서 발생하는 오디오 출력을 캡쳐하여 네트워크를 통해 서버Android로 보내줄 수 있도록 한다. Android 오디오 동작을 제어할 수 있는 Audio Recorder API를 사용하여 Audio stream을 캡쳐 할 수 있도록 구현할 것이다. AudioManager를 사용하여 사용자가 원하는 대로 볼륨 조절을 가능하게 한다. 파일 포맷에 대해서는 실시간 스트리밍 전송을 해야 되므로, 윈도우와 Android 모두 아래와 같은 포맷을 지정한다.

표 Audio Format



## 안드로이드 소리 재생

Client로부터 받은 PC 또는 Android의 Audio Data를 Android의 출력 장치를 통해 재생해준다. 독립적인 소리를 제어하기 위해서 새로운 Type을 추가로 정의하고자 하였으나 새로운 Stream Type을 추가하게 되면 Android Root 권한이 필요하므로 제한 사항이 따르게 된다. 따라서, 본 과제에서는 볼륨 조절을 클라이언트 Device에서 하고자 한다. 서버 Device에서는 수치 조절바를 두어 클라이언트 Device로 수치 값을 전달한다. 이를 통해 스마트폰에서 볼륨을 제어할 수 있는 효과를 얻는다.

또한, Audio Flinger를 이용하여 이퀄라이저와 같은 음질 조정이나 특수효과를 적용할 수 있도록 한다.

## PC와 Android, Android와 Android 간의 통신

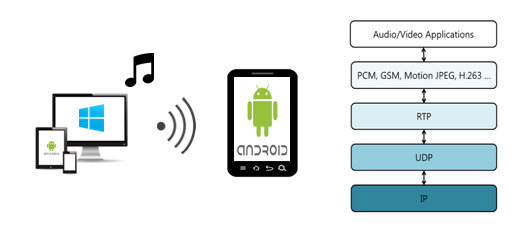


그림 Audio Communicatin By UDP

Windows 또는 Android 플랫폼을 갖는 Device에서 Android Device로 Audio Stream을 무선으로 전송한다. 이 때, 네트워크는 동일 망으로 제한하고자 한다. 또한 통신 프로토콜로서 TCP/IP 프로토콜을 고려해 보았을 때 동기화 방식이라 안정적이기는 하나, 동기화 방식의 프로토콜이므로 한번의 지연 시간이 발생하면 지속적으로 지연이 지속되기 때문에 Stream Service가 중요한 시점에서 적절하지 않다. 따라서 모든 데이터가 전송 되는 것을 보장 받지는 못하나, 데이터가 손실 되더라도 지연 시간이 없는 UDP 프로토콜을 사용 한다.

# 개발 내용

## 전체적인 시스템 구조도

1. 시스템 흐름도

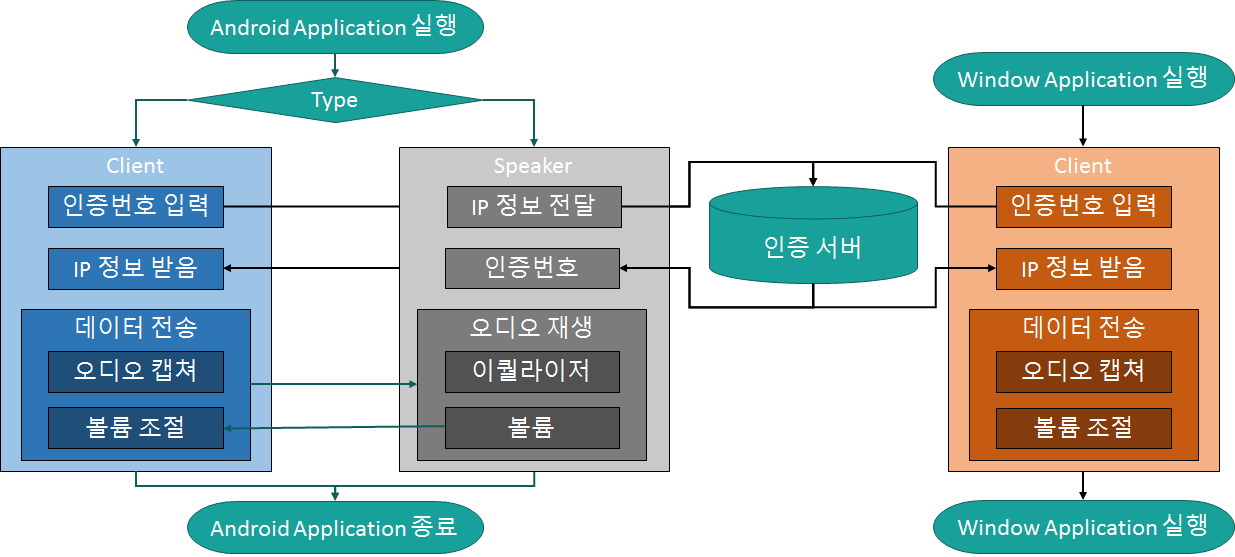


그림 System Flow Chat

[그림 7]은 시스템 흐름도를 나타낸다. 안드로이드는 Speaker와 Client 두 가지 타입이 있고, 윈도우는 Client 역할을 개발하였다.

먼저, 안드로이드 어플리케이션에서 스피커 환경을 만들기 위해 Speaker를 선택한다. 선택을 하게 되면 최초에 한번 Speaker 기기의 IP 정보를 인증 서버에게 넘겨주게 된다. 이를 받은 인증서버는 4자리 숫자로 구성된 인증번호를 랜덤 생성하여 Speaker에게 돌려주고, 해당 IP와 인증번호를 함께 저장한다. 인증번호를 받은 Speaker는 화면 상단에 인증번호를 노출하여 사용자에게 알린다. 다음으로 사용자는 안드로이드 기기 또는 윈도우 기기에서 인증번호를 입력한다. Client는 입력 받은 인증 번호를 인증 서버에 보내어 해당 인증번호와 연동된 Speaker의 IP를 얻어 Speaker와 연결하게 된다.

최초 연결 시에 Speaker는 소리를 재생하기 위해 한 개의 Thread를 생성하고, Client가 오디오 데이터를 보낼 PORT 번호를 할당해주며, Client가 해당 PORT로 오디오 데이터를 전송한다.

1. 안드로이드 시스템 아키텍처

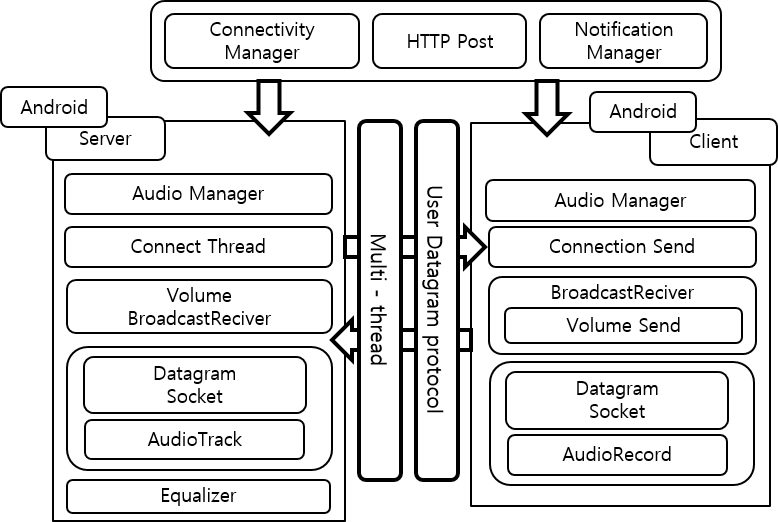


그림 Android System Architecture

안드로이드 시스템 아키텍처는 다음과 같다. 안드로이드에서는 스피커로서 소리를 전송받고 출력하는 서버 역할을 하거나, 소리를 출력해서 스피커로 보내는 클라이언트 역할을 할 수 있다. 두 부분은 동일하게 Connectivity Manager를 통해 Wifi 환경에서만 구동 할 수 있도록 제한하며, 서버에 접속 정보를 등록하거나 요청하기 위한 http Post 방식을 활용한다. 어플리케이션이 동작 중임을 사용자에게 항상 알려주기 위한 Notification을 제공하였다.

서버 부분의 아키텍처를 보면, 서버의 오디오 장치를 조작하기 위한 Audio Manager, Client와 연결을 하기 위해 Background에서 동작하는 Connect Thread를 구현하였다. 또한 서버의 Volume 제어는 BroadcastReceiver를 통해 어플리케이션과 Audio의 Stream\_Muisic 볼륨과 동기화시키도록 하였다. 스트림을 수신하기 위한 UDP Socket 통신을 각 클라이언트마다 독립적으로 스레드 및 재생트랙을 동작하였고, 이퀄라이저 구현을 위한 Equalizer를 구현하였다.

클라이언트 부분의 아키텍처를 보면, 서버와 동일하게 클라이언트 역할을 하는 디바이스의 오디오 장치를 제어하기 위한 Audio Manager를 사용하였으며, 서버와 연결하기 위해 인증번호를 통해 서버에게 연결 요청을 하도록 하였다. 클라이언트는 BroadcastReceiver를 통해서 Volume 변화에 대한 이벤트가 발생할 때마다 서버로 Volume 변경 정보를 전송하도록 하였으며 클라이언트는 AudioRecord를 이용하여 voice\_recognition 오디오 소스를 byte 형태로 서버와 연결되어 있는 Socket을 통해 스트림을 전송하도록 구현하였다.

1. 안드로이드 UML

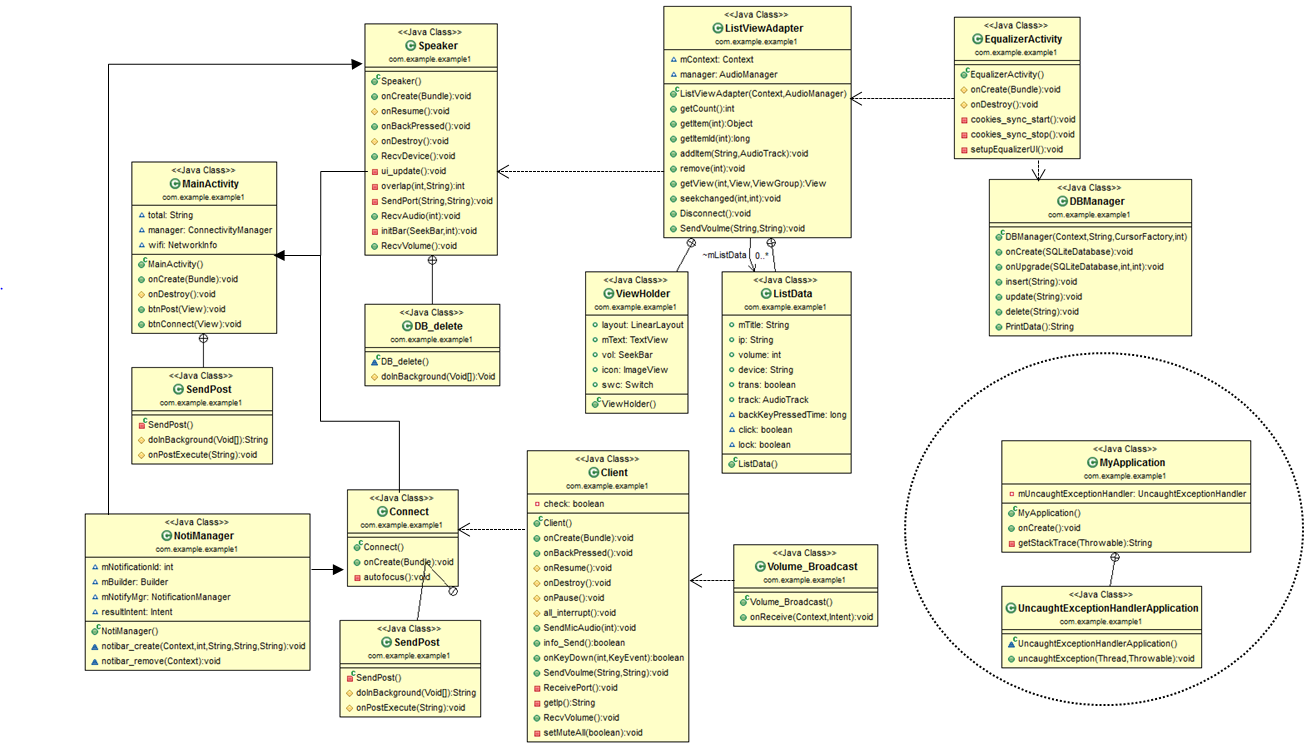


그림 Android UML

위 그림은 안드로이드 UML(Unified Modeling Language) Diagram 을 나타낸다. UML이란 소프트웨어 개발 과정에서 산출되는 산출물들을 명시, 개발, 문서화하기 위한 모델링 언어이다. UML Diagram은 소프트웨어 개발에 사용하기 위한 여러 다이어그램들을 정의하고 있으며, 또 다이어그램들의 의미들에 대해 정의하고 있다.

모든 Activity의 시작은 MainActivity Class에서 이루어진다. Type에 따라 Speaker 관련 Class 들과 Client 관련 Class들이 Dependency 한 관계를 갖고 있으며, Application 관련 Class는 타의 Class들과 관계없이 불특정 종료에 대한 예외처리를 담당한다.

1. 윈도우 시스템 아키텍처

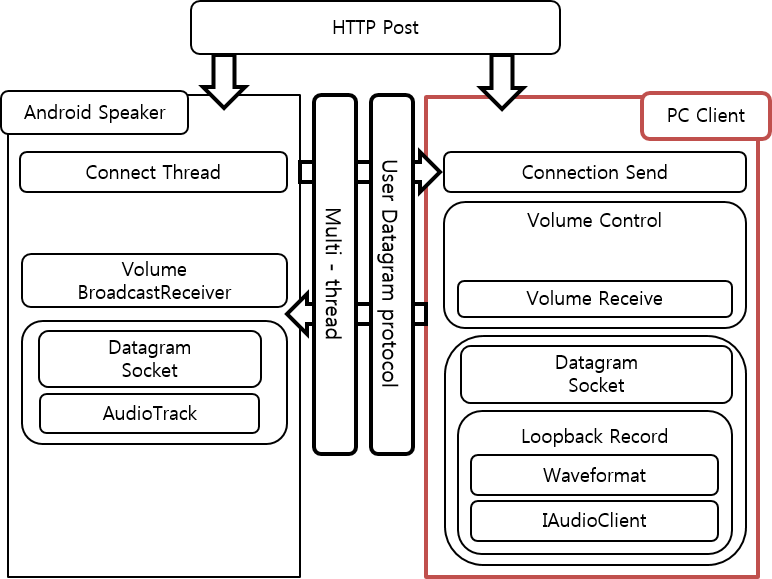


그림 Window System Architecture

윈도우 시스템 아키텍처는 다음과 같다. 윈도우 에서는 소리를 출력해서 스피커로 보내는 클라이언트 역할만 한다. 윈도우 에서도 Andriod가 Wifi 환경일 경우에만 해당 스피커에 접속 할 수 있다. 서버에 접속 정보를 등록하거나 요청하기 위한 http Post 방식을 활용한다.

PC 클라이언트는 Android 스피커와 연결하기 위해 인증번호를 통해 스피커에게 연결 요청을 한다. 클라이언트는 Volume\_Thread를 통해 스피커로부터 변경된 볼륨 값을 받을 때 마다 마스터 볼륨 값을 조정 해준다. Android 스피커에서 볼륨 바를 Lock 걸어 줬을 경우 마스터 볼륨 값은 변하지 않고 현 상태를 유지하므로 PC가 Mute인 상태로 PC 오디오의 볼륨을 조정하며 들을 수 있다.

Andoird 스피커에서 접속을 끊을 경우 -1값을 넘겨주며 PC 클라이언트는 -1을 받게 되면 새로운 연결 환경을 준비한다.

## 인증 서버 구현

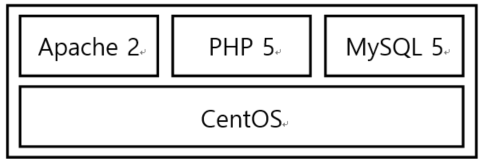


그림 웹 서버

인증 서버는 웹 서버를 이용하여 인증을 간편하게 한다. 웹 서버의 OS는 서버용 배포 판인 RedHat 계열의 CentOS를 사용하고, 웹 서버 프로그램은 Apache 2, 서버 언어는 PHP 5, 데이터 베이스는 MySQL 5로 구성한다.

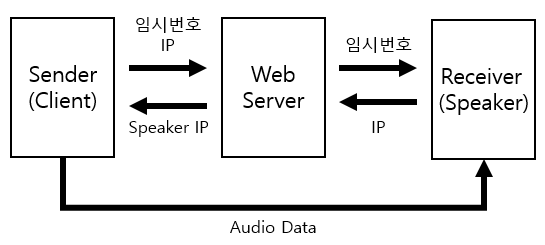


그림 인증 절차

Speaker Server는 서비스를 시작한다는 의미로 Web Server에게 Speaker Server의 ip주소를 보내게 되는데, 이 때는 특정 웹 페이지에 POST 방식으로 데이터를 전송하며, 이를 받은 Web Server는 난수를 발생시켜 임의의 숫자 4자리를 만들고, 이 숫자가 서비스에 사용 중인지 확인 후, 사용 중이 아니라면 Speaker Server에게 다시 알려 준다. Speaker Server는 이 임의의 난수를 화면에 출력하여 사용자에게 알려주고, 사용자는 이 난수를 Client에 입력하고 특정 웹 페이지에 POST 방식으로 데이터를 전송하고, 이를 받은 Web Server는 임의의 숫자를 발급한 서버의 IP 정보를 Client에게 알려주어 사용자가 ip와 같은 전문적인 정보를 모르더라도 두 기기를 간단히 연결 할 수 있도록 한다.

## 서버/클라이언트 Audio 전송 및 Volume Control

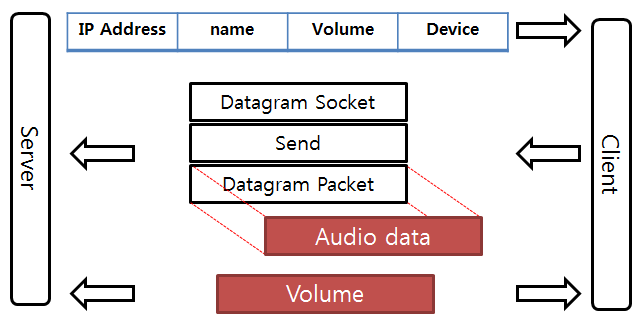


그림 서버/클라이언트 전송 방식

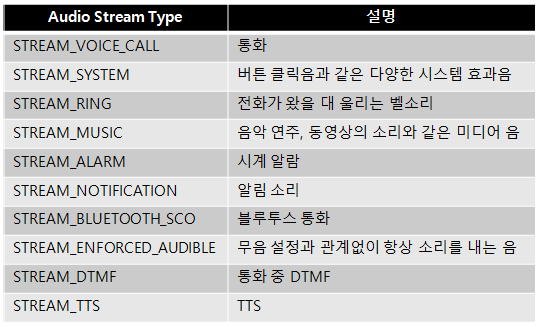
서버/클라이언트 Audio 전송은 [그림 10]과 같이 이루어진다. 본 과제에서 통신프로토콜로 UDP 프로토콜을 사용한다. 전반적인 통신은 모두 UDP 방식으로 이루어지며, Datagram Socket을 생성하여 Packet에 전송할 정보를 담게 된다.

우리는 최초 연결을 위한 연결 프로토콜(“IP주소/연결이름%volume%device”)로 정하였다. 연결이 성공하면 오디오 데이터가 packet에 담겨 계속해서 전송된다. 본 과제 시작 때에는 오디오 데이터가 전송되는 Packet의 포맷을 “접속 번호를 담는 2비트 + 순서 번호를 담는 6비트 + 오디오 데이터를 담는 8비트”로 정하였지만, 과제를 진행하면서 UDP임에도 불구하고 패킷이 순서적으로 잘 들어왔기 때문에 순서번호를 제거하였다. 또한, 서버는 최초 연결 시에 클라이언트마다 스레드를 생성해주는 멀티 스레드로 동작하도록 구현하였으므로 접속 번호에 대한 정보가 필요하지 않았고 접속 번호도 제거하였다. 결과적으로 생성된 Packet을 앞서 생성해둔 Socket을 통해 Send하여 전송한다. 여기서 패킷 생성에 있어서 audio data는 Audio Capture를 한 후 audio stream 버퍼에 write 한다.

또한, Volume Control은 Server의 경우 각 Client 마다 할당된 Seekbar를 통해서 조절 이벤트가 발생할 경우 Volume 값을 Client에게 전송하며 이를 받은 Client는 자신의 STREAM\_MUSIC 타입 Volume을 전송 받은 Volume 값과 동기화 시킨다. 반대로 Client에서 볼륨이 제어되는 경우 이는 어플리케이션에서 broadcast를 통해 이벤트를 받고, 이벤트가 발생한 경우에 Server에게 Volume 값을 전송하도록 구현하였다. 결과적으로 서버/클라이언트의 볼륨 값은 항상 동기화된 상태가 된다.

클라이언트가 PC인 경우, PC의 마스터 볼륨과 동기화되어 조절되지만 on/off 스위치를 두어 동기화에 대한 선택사항을 제공하였다. 만약, OFF 상태로 pc에 할당된 Seekbar를 조절하게 되면, 안드로이드 내에서 volume level에 맞게 audio data를 증폭시킨다. 즉, 스위치가 off인 경우 볼륨 제어는 가능하되 볼륨 동기화는 동작하지 않도록 구현하였다.

표 Android Audio Stream Type



[표 1]과 같이 Android 에서는 Volume을 Stream Type 에 따라 조절하기 때문에 본 과제에서는 STREAM\_MUSIC 타입에 제한하여 구현한다. 따라서 전송된 볼륨 사이즈를 AudioManager 클래스의 set 메소드를 통해 해당 타입에 적용한다.

## Android Audio 재생

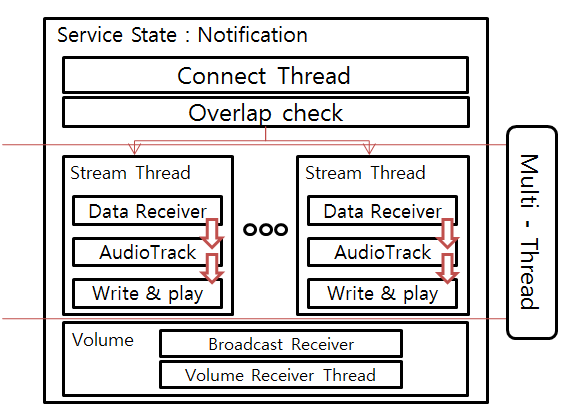


그림 Android Speaker 동작 Application Architecture

위 그림은 Android에서 Speaker로 동작할 때 Application의 구조이다. 본 과제에서 Application 이 Speaker의 역할로 동작하게 되면 이는 서버로서의 역할을 하게 된다고 명의 하였다. 서버로 동작하는 Application은 항상 백그라운드에서 동작해야 할 필요가 있으며 이는 Android 4대 컴포넌트 중 Service를 이용하여 구현하도록 하였다.

또한, 백그라운드에서 동작하고 있는 Application의 상태는 Notification을 통해 항상 상단 알림바에 위치하도록 하여 사용자가 실시간으로 현재 어플리케이션의 상태를 모니터링 할 수 있도록 추가하였다. 물론 어플리케이션이 백그라운드에서 동작하고 있는 동안 본 안드로이드 디바이스의 다른 기능을 작동하여도 충분히 동작할 수 있다. 즉, 다른 작업을 하면서도 다른 Device로부터 오는 소리를 들을 수 있다.

어플리케이션을 동작하면 최초 Device 연결을 대기하는 Thread가 진행된다. 연결을 시도한 Thread에 대해 중복검사를 시행한 후 적절한 경우 하나의 audio I/O Thread를 생성하여 제공한다. 각 연결에 대한 Thread들은 멀티 프로그래밍으로 동작한다.

결과적으로 우리는 Android 서버를 IOCP Server와 같이, 멀티프로세서 환경을 두고 I/O작업을 직접 하도록 설계 하였다. N개의 Thread와 DataReceiver, AudioTrack을 각각 생성하여 N대의 Device로부터 오는 Audio Stream을 독립적으로 제어하고 관리하게 된다.

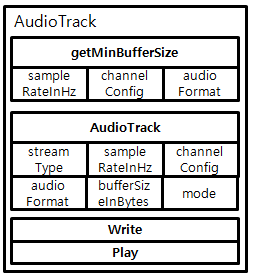


그림 AudioTrack Audio 출력

[그림 12]은 AudioTrack을 이용하여 stream data를 디코딩 한 후 출력하는 흐름을 나타낸다. Android 오디오 트랙을 재생하기 위해서는 가장 먼저 getMinBufferSize() 호출을 통하여 audio data에 대한 생성 버퍼를 설정해 줄 필요가 있다. 본 과제에서도 getMinBufferSize()를 이용하여 생성 버퍼의 크기를 잡아주고자 하였으나, 사실상 getMinBufferSize()는 SampleRateInHz의 영향을 상당히 받고 있었다. 따라서, BufferSize는 우리팀이 채택한 sampleRateInHz 44100과 맞추어 결정하였다.

sampleRateInHz는 44100과 48000 등을 모두 테스트한 결과 사실상 모든 샘플링 레이트에 대한 결과는 만족스러웠고 그 중 44100이 sync 및 음질 등에 대해서 적합하다고 판단하여 선정하였다. channelConfig는 Stereo 2 채널을 사용하였으며 2 채널에 적합한 포맷을 하고자 audioFormat은16bit로 선택하였다.

다음으로 위에서 설정한 BufferSize로 AudioTrack을 생성한다. 이때, 가장 먼저 해당 오디오 트랙의 StreamType을 설정한다. Android Stream Type에는 STREAM\_VOICE\_CALL, STREAM\_SYSTEM, STREAM\_RING, STREAM\_MUSIC, STREAM\_ALARM, STREAM\_NOTIFICATION 등이 있다. 스마트폰의 소리를 조절하는데 있어서 해당 타입의 모든 소리의 볼륨에 영향을 미치기 때문에 본 과제에서는 새로운 타입을 연결된 기기의 수만큼 추가하여 각각의 Device로부터 오는 소리를 독립적으로 제어하고자 하였으나 이는 Android Root 권한을 불가피하게 필요로 하여 각 클라이언트에서 제어하여 보내주는 방법으로 진행하려 한다. 사실상 StreamType과 관련 없이, 우리는 모든 오디오 스트림을 캡처 및 전송한다. 즉, StreamType은 STREAM\_MUSIC으로 하였다.

또한, Speaker에서는 볼륨 제어가 가능하다. 즉, 볼륨 제어를 위한 각 클라이언트에게 수치 조절바를 할당하여 이 수치 조절바와 클라이언트 디바이스( -> 본 과제에서 서버 디바이스와 반면 연걸을 시도하는 디바이스를 클라이언트라고 명의한다. )들의 STREAM\_MUSIC 타입의 볼륨이 항상 동기화되도록 구현하였다.

결과적으로 서버 Device에서 볼륨을 조절하는 효과를 볼 수 있다. sampleRateInHz, channelConfig 등은 위에서 설정한 옵션과 동일하게 선택하여 오디오 트랙을 생성한다.

이렇게 만들어진 AudioTrack을 write()를 통해 출력할 buffer를 넣은 후 play()하면서 사운드를 출력하게 된다. 이때, UDP를 통해 Receive 되는 audio data를 write()하며 바로 play()를 동작시킴으로서, 최소한의 delay를 갖고자 구현하였다.

1. **Android Audio Capture**

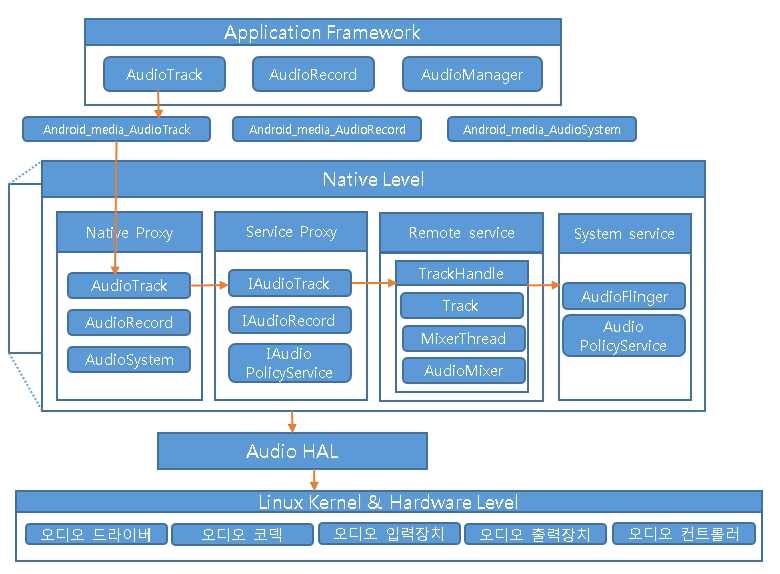


그림 Android Audio System

[그림 13]는 Android Audio System의 전체적인 구조이며 오디오 트랙으로부터 재생되는 오디오 스트림의 출력 흐름을 나타낸다. Android 오디오 시스템은 이어폰이나 스피커를 통해 소리를 출력하는 오디오 출력, 마이크를 통해 소리를 받아들이는 오디오 입력, 오디오 볼륨 조절, 출력 하드웨어 장치 선택 등의 오디오 제어와 같은 오디오와 관련된 특화 기능들을 전체적으로 다룬다. 본 과제에서 Android Audio Capture를 위해서 출력되는 오디오 스트림을 Android SDK API인 AudioRecord를 사용하고자 하였으나, 이는 Mic를 통해 녹음이 되는 것을 확인할 수 있었다. 따라서 Android NDK 의 Open SL|ES를 통해서 Audio Capture를 시도하였다.

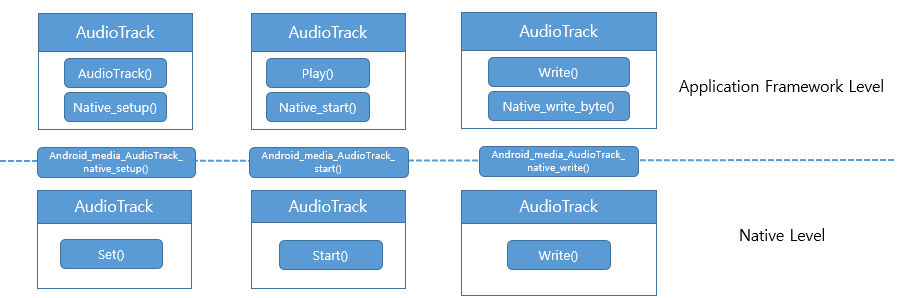


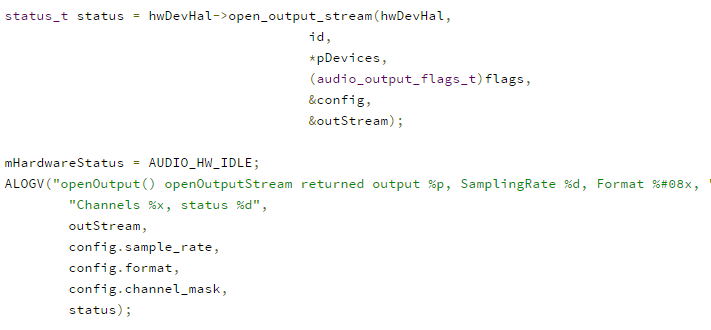
그림 JNI를 통한 Application AudioTrack과 Native AudioTrack 연결

[그림 14]와 같이, Application 계층에서 각각의 java 클래스는 Native 계층의 c++ 클래스와 JNI를 통해 연결되어 수직적으로 내려온다. 이후 오디오 서버를 구성하는 AudioFlinger부터는 서버/클라이언트 기본 통신 구조를 통해 데이터가 이동한다. Platform Kernel 의 AudioFlinger.cpp로 전달되기 전에 Native 계층에서 Audio Capture를 시도하였다.

|  |
| --- |
|  |
| 그림 Android NDK Open ES|SL |

Android NDK 계층에서 Open ES|SL 을 이용하여 I/O 오디오 스트림 데이터를 이용할 수 있었으나, AudioTrack 또는 AudioRecord java 클래스에 종속되어 있기 때문에 Application 이외의 Android 시스템 오디오를 가져오는데 있어서는 I/O buffer 관리를 할 수 없었다. 본 과제는 안드로이드의 모든 시스템 사운드를 목표로 하기 때문에 NDK 계층 방법 또한 적합하지 않다고 판단하였다. 결과적으로 본 목표를 달성하기 위해서는 모든 Application의 buffer stream을 관리하며 다루는 Android Kernel의 **/… /android/platform/frameworks/services/audioflinger/AudioFlinger.cpp** 을 수정할 필요가 있었다. Android Kernel 빌드 및 개발 환경을 위해서 Odroid\_xu3를 사용하여 안드로이드 커널을 빌드하였고, Android 4.4.4\_r3 소스를 바탕으로 커널 분석을 진행하였다.





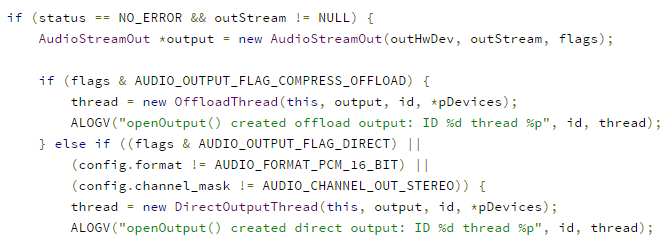


그림 Android Kernel AudioFliger.cpp

위 그림들의 순서대로 먼저 Hardware Out Device와 입출력 작업을 위한 io\_handle를 정의한다. 해당 Out Device와 입출력 핸들러를 통해 나오는 Output stream의 상태를 status로 정의하며 status와 outStream의 존재 여부에 따라 나오는 Audio Out Stream을 Capture 할 수 있었습니다. 해당 output을 본 과제의 USER APPLICATION으로 가져오기에는 한계점이 있으며, 이를 이용하기 위해서는 반드시 시스템 어플리케이션을 동작시켜야 하고 커널을 직접 빌드해야하기 때문에 사용자 측면을 고려하였을 때 사용성 측면에서 상당히 만족스럽지 못하다고 판단하였다. 따라서 우리는 android system audio capture를 하기 위한 보조적인 하드웨어를 사용하기로 결정하였다.

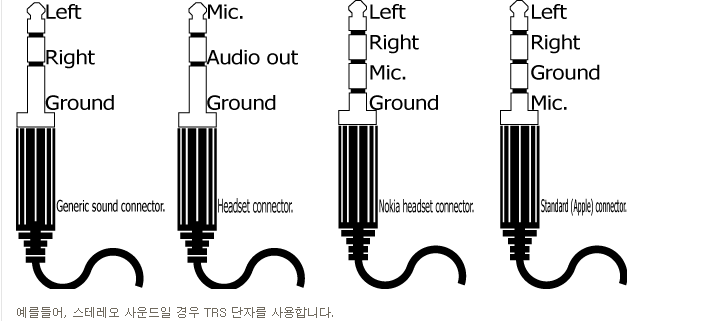


그림 Audio Jack 의 종류

위 그림을 참고하였을 때, 각 오디오 잭들은 극마다 각각의 의미를 부여한다. Left, Right의 경우 오디오 출력 스트림을 의미하며, Mic은 오디오 입력 스트림을 의미한다.

본 과제에서 추가하고자 하는 보조 장치의 원리는 다음과 같다.

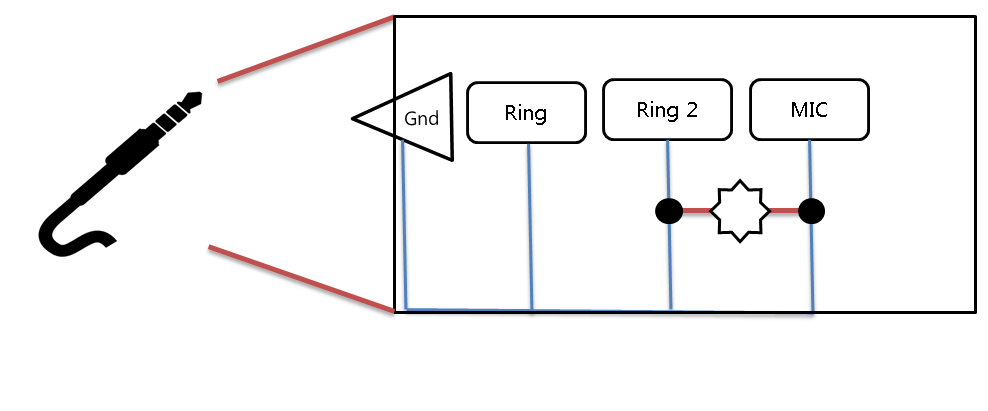


그림 보조장치의 설계 원리

안드로이드 디바이스는 본래 3극 또는 4극 등의 jack을 연결하면 이어폰으로 인식하게 된다. 본 과제에서는 Stereo를 위해 4극 jack으로 진행하였다.

안드로이드 디바이스에 4극 jack(dongle)을 연결하여 디바이스에서 나오는 소리를 4극 jack으로 출력하도록 하고 MIC 장치 또한 4극 jack(dongle)이 입력장치로서 역할을 하도록 하게 된다. 즉, 결과적으로 디바이스 장치는 오디오 입.출력 장치를 보조적 장치인 4극 jack 으로 인식하게 된다.

이때, 4극 jack의 MIC(입력)과 Ring(출력)을 연결시킴으로써 output 과 input 라인을 의도적으로 연결되도록 하며 결과적으로 android audio의 모든 output은 input으로 인식하게 되고 우리는 input을 voice recognition 오디오 소스로 capture함으로서, 잡음 및 에코 등을 최소화한 system audio stream을 Capture 할 수 있습니다.

## Android Equalizer

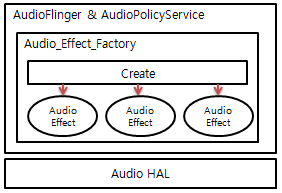


그림 Audio Effect 구조

Equalize, EnvironmentalReverb와 같은 Audio 효과를 주기 위해 AudioFlinger와 AudioPolicyService의 Audio Effect 처리 절차를 확인 할 필요가 있다. [그림 15] 처럼 AudioFlinger는 Create를 통해 다양한 AudioEffect 인스턴스를 생성한다. 이후 생성된 인스턴스들은 Audio Effect Factory에서 보관 및 적절히 적용된다.

우리는 Effect를 적용시키기 위해서 Android SDK Equalizer를 이용하였다. 각 AudioTrack에 대해서 모두 동일하게 오디오 효과를 적용 시킬 수 있도록 구현하였다.

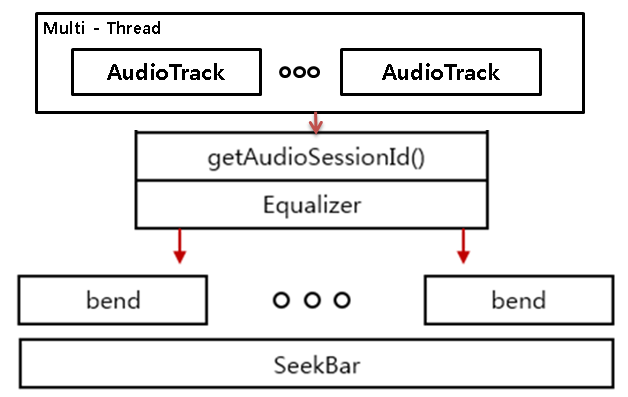


그림 Audio Equalizer 설계

Android Audio Equalizer 를 구현하기 위해 독립적으로 동작 중인 AudioTrack 클래스들로부터 SessionId를 얻는다. 얻어온 SessionId로부터 생성된 Equalizer 클래스로부터 해당 Audio Session을 제어한다. 이를 이용하여 주파수 범위에 따라 각각의 bend영역을 구분하고 값 조절을 위해 Custom SeekBar를 제공하므로 서, UI를 구성하였다. Equalizer를 통해서 주파수 별로 음의 느낌을 조절하여 Classical, Dance, Flat, Jazz, Pop 등과 같은 효과를 기본적으로 제공하였다.

## Window Audio System

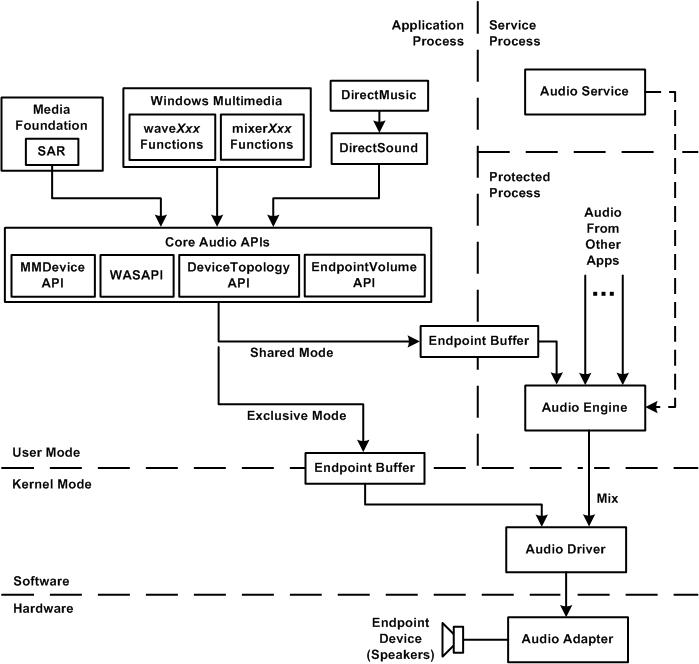


그림 Core Audio APIs

Window OS에서는 Window Vista 이후에 새로 도입된 Core Audio APIs를 통해 오디오에 관련한 동작을 제어 할 수 있도록 해준다. 해당 API는 Professional audio 어플리케이션이나 Real-time communication(RTC) 어플리케이션이 필요로 하는 low-level로의 직접적인 접근을 최소한의 latency로 가능하게 한다. 이 전 Window OS 에서도 waveXXX, MixerXXX 함수나 DirectSound 등으로 제어가 가능하다. 그러나 위의 방식으로 다룰 경우 100ms가 넘는 latency나 어플리케이션에 종속되지 않은 제어를 하기가 어렵다. 따라서 실시간으로 출력되는 오디오 스트림을 통신해야 하는 본 프로젝트는 Window XP 이 전 버전의 사용자들을 제한하고, Window Vista 이후 버전에서는 호환되는 Core Audio APIs를 사용하여 구현하였다.

COM 객체 기반의 Core Audio API는 MMDevice API, WASAPI, DeviceTopology API, EndPoint Volume API라는 4가지의 서브 API로 구성되어 있다. MMDevice API는 시스템 오디오 디바이스를 이용 가능한 인스턴스들의 목록을 관리하는 API이고, WASAPI 는 오디오 스트림들의 저장과 재생을 제어하는 API이다. 이 외에도 저음이나 고음 베이스의 기능들의 하드웨어 접근을 허용하는 API인 DeviceTopology API, 볼륨이나 Peak 미터를 제어하는 EndpointVolume API 가 있으나 본 프로젝트는 주로 MMDevice API와 WASAPI를 다룬다.

### MMDevice API (Window Multimedia Device API)

MMDevice API로 시스템에 연결되어 있는 오디오 장치에 대한 목록과 인스턴스를 다룰 수 있다. 오디오 Stream을 캡쳐하기 위한 맨 첫 번째 단계로 MMDeviceEnumerator에서 현재 Default 디바이스로 설정 되어있는 디바이스를 가져온다. 윈도우의 정책의 따라 Default 디바이스를 결정하는 우선순위가 다른데 이어폰이 꽂혀 있을 경우 우선 순위가 높은 이어폰이 출력 오디오 Stream을 출력하게 되는 Default 오디오 종단점 장치로 선택된다. 이를 거치게 되면 해당 종단점 장치로 가게 되는 출력 오디오 Stream을 WASAPI의 Loopback recording 기능으로 캡쳐 할 수 있게 된다. 후에 언급하겠지만, Loopback recording 기능은 설정된 오디오 장치로부터 오디오 Stream을 가져오는 형식이므로 중간에 오디오 장치가 바뀌면 출력 오디오 Stream이 다른 오디오 장치로 보내지기 때문에 원하는 오디오 Stream을 가져올 수 없다.

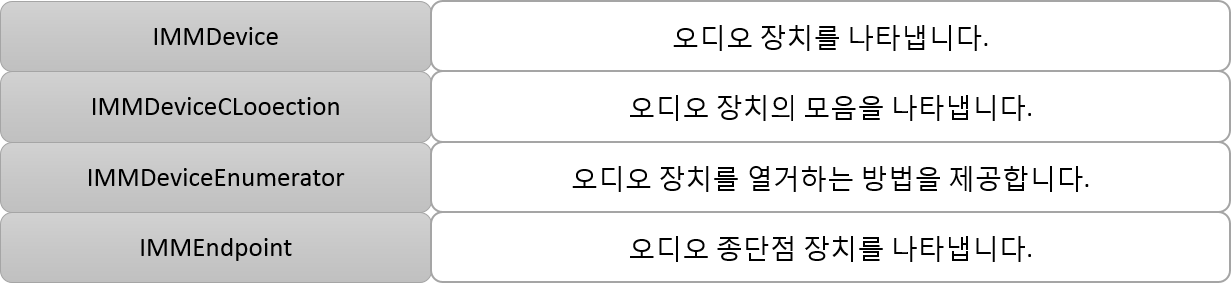


그림 MMDevice API의 인터페이스

### WASAPI (Window Audio Session API)

WASAPI는 Endpoint 장치로 가고 오는 오디오 Stream의 저장과 재생을 제어할 수 있다. WASAPI는 여러 인터페이스로 구성되어 있다. WASAPI의 기능들을 이용하기 위해서 클라이언트 프로그램에서는 몇 가지 처리 과정이 필요하다. 첫 번째로 필요한 것이 IAudioClient 인터페이스이다. WASAPI 인터페이스에 접근하기 위해서, 클라이언트는 먼저 IMMDevice::Activate 메서드를 호출하여 IAudioClient 인터페이스에 대한 참조를 얻는다. 위에서 말했듯이 현재 활성화 되어 있는 종단점 장치를 알아야 원하는 오디오 Stream을 출력하거나 가져올 수 있다. 클라이언트는 IAudioClinet::Initialize 메서드를 호출하여 종단점 장치에 대한 Stream을 초기화 해준다. 이 두 과정이 WASAPI를 사용하는데 기본적으로 실행해야 하는 과정이다. 이 이후에 클라이언트는 IAudioClient::GetService 메서드를 호출하여 WASAPI 인터페이스의 다른 레퍼런스를 사용할 수 있다. 즉 이 과정 후에 클라이언트 프로그램은 현재 오디오 출력 장치에 대한 IAudioCaptreClient 인터페이스를 할당 받아 Loopback recording 기능을 이용할 수 있으며 현재 출력되는 오디오 스트림에 대한 데이터를 다룰 수 있게 된다.

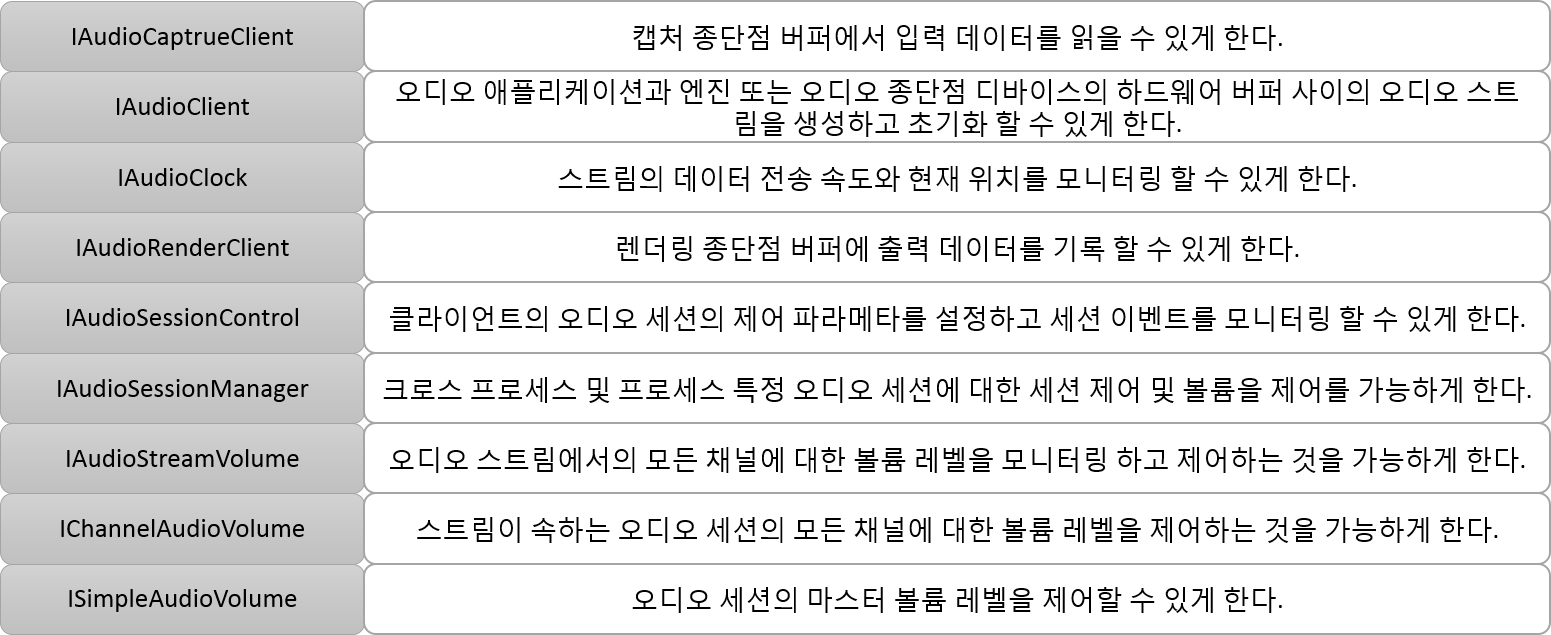


그림 WASAPI의 인터페이스

## Loopback Recording

Loopback 이라 하는 것은 전기 신호의 라우팅, 디지털 데이터 스트림(digital data streams)등이 의도적인 가공이나 수정 없이 언래의 장치나 장비로 돌아가는 것을 말한다. 이 것은 주로 전송이나 수송 기반 시설을 테스트 하는 수단으로 사용된다. 이러한 기능을 이용해 출력 오디오 종단점 디바이스로 보내는 출력 스트림을 다시 받아 해당 오디오 Stream을 Buffer에서 저장함으로써 Recording을 가능하게 한다.

Loopback recording 기능을 사용 하기 위해서는 몇 가지 설정이 필요하다. 먼저 MMDevice로부터 현재 종단점 오디오 장치를 선택하고 AudioCaptureClient 인터페이스를 접근한다.

먼저 AudioClientShareMode를 Shared 모드나 Exclusive모드로 설정할 수 있는데 Shared 모드가 초기값으로 각 어플리케이션 마다 사운드 컨트롤이 따로 있어, 오디오 엔진의 Endpoint buffer를 공유한다. 가 Exclusive 모드에서는 DMA에 직접적인 접근이 가능하며 오디오 모드를 사용하여 오디오 장치의 Endpoint buffer를 공유한다.

캡처된 오디오 스트림에 대한 데이터는 기본적으로 샘플 당 비트수가 32bit이고 샘플 주기는 48000Hz(DVD 음질) 이며 2 채널인 PCM 데이터이다. 원하는 WaveFormat으로 오디오를 캡쳐할 수 있는데 별 다른 설정을 해주지 않는 이상 현재 시스템에 Default 디바이스로 설정되어 있는 오디오 종단점 장치의 WaveFormat을 따른다. Window의 경우 최근 24bits까지 지원을 해주지만 Android에서는 48000Hz, 24bits의 HD 음질을 재생하기 위해서는 추가적인 인코딩 과정이 필요하다. 본 프로젝트 에서는 실시간 스트리밍 에서 Speaker와 Client의 전송 및 재생 속도를 최소화 하기 위해 사용자가 원하는 값을 설정 하기 보다는 가장 일반적인 44100Hz, 16bits의 WaveFormat을 따른다.

그리고 Loopback recording 기능으로 얻을 수 있는 오디오 데이터는 패킷 단위로 캡쳐 되며, 최소 단위 패킷을 얻을 수 있는 최소 주기 또한 클라이언트 프로그램에서 확인 할 수 있다. 본 프로젝트에서는 스마트폰으로 오디오 스트림 데이터 전송 후 출력시 지연 시간을 최소화하기 위하여 최소 주기 단위로 오디오 스트림 데이터를 캡쳐한다.

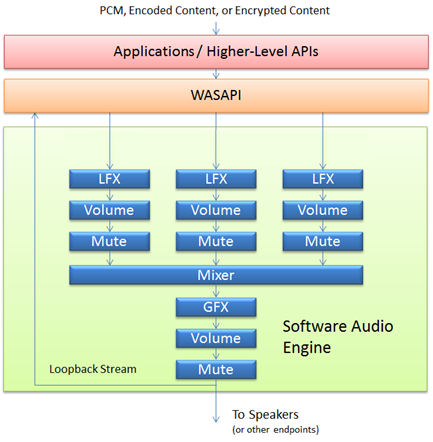


그림 오디오 데이터 흐름

[그림17]에서 볼 수 있듯이, 오디오 stream은 WASAPI 층에서 speaker로 출력될 때까지 LoopbackStream으로 존재한다. 소프트웨어 오디오 엔진에서의 높은 수준의 API를 통하면. 각각의 볼륨 조절 및 음소거 같은 기능도 가능해질 것이다.

WASAPI 시스템 모듈은 소프트웨어의 Loopback 모드를 구현한다. Loopback 모드에서, WASAPI는 오디오 엔진에서 어플리케이션의 캡쳐 버퍼로 출력 Stream을 복사한다. 이렇게 Loopback 모드에서, WASAPI의 클라이언트는 렌더링 Endpoint에 의해 디바이스에 재생되는 오디오 Stream을 캡처하기 위해 위에 기본적인 접근 과정을 거친다. 이 후, 렌더링 Endpoint장치에 IAudioCaptureClient 인터페이스를 얻기 위해 IAudioClient의 getService 메소드를 호출 할 수 있다.

주로 WASAPI는 음향 반향 제거 (AEC)를 지원하기 위해 사용되지만 다양한 오디오 어플리케이션에서는 Loopback 모드가 오디오 엔진에 의해 재생되는 소리를 혼합하는 데도 많이 사용한다. 이를 이용해 여러 디바이스의 소리를 듣고자 할 때도 사용 가능하다.

MMDeviceAPI 와 WASAPI를 사용할 때에는 자연스럽게 Stream이 옮겨가지 않으므로 수동으로 구현해줘야 한다. Stream이 초기화 되었을 때 그리고 Loopback 사용이 가능한 경우, pull 모드 캡처 클라이언트는 이벤트를 수신하지 않는다. 때문에, 이벤트 구동 모드에서 렌더링 Stream을 초기화해준다. 클라이언트가 렌더링 Stream에 대한 이벤트를 수신 할 때마다, Endpoint버퍼로부터 다음 번 샘플들을 읽는 스레드를 실행하는 클라이언트에게 알려줘야 한다.

또한 클라이언트는 공유 모드 Stream (AUDCLNT\_SHAREMODE\_SHARED)에 대한 Loopback 모드를 활성화 할 수 있다. 독점 모드 Stream Loopback 모드에서는 작동 할 수 없다.

이와 같은 과정으로 윈도우 디바이스 드라이버를 사용하지 않고도 PC에서의 출력 오디오 소리를 가져올 수 있다. 스테레오 믹스 기능을 사용했을 때와 달리, 음소거 상태 일 때에도 오디오 stream을 가져오는 것이 가능하므로 본 과제에서는 WASAPI의 Loopback recording 기능을 사용하여 오디오를 캡쳐하고자 한다.

## PCM (Pulse code Modulation)

PCM(펄스 부호 변조)는 아날로그 신호인 음성 신호의 디지털 표현이다. 먼저, 정보에 따라 연속적으로 변하는 선형 신호를 비선형 신호로 만들기 위해 등급을 균일한 주기로 샘플링 한 다음 디지털 코드로 양자화 처리한다. 샘플링(sampling)은 연속 신호를 이산 신호로 감소시키는 것을 말한다. 다음 그림에서 연속 값은 초록색으로 표현되었고 이산 값은 파란 색으로 표현되었다. 양자화는 아날로그 신호의 순간 진폭을 미리 결정된 등급들과 가장 가깝도록 매칭되는데, 등급의 숫자는 8, 16, 32 등 항상 2의 배수이다. 이러한 숫자들은 3, 4, 5개의 비트에 의해 표현될 수 있기 때문이다.

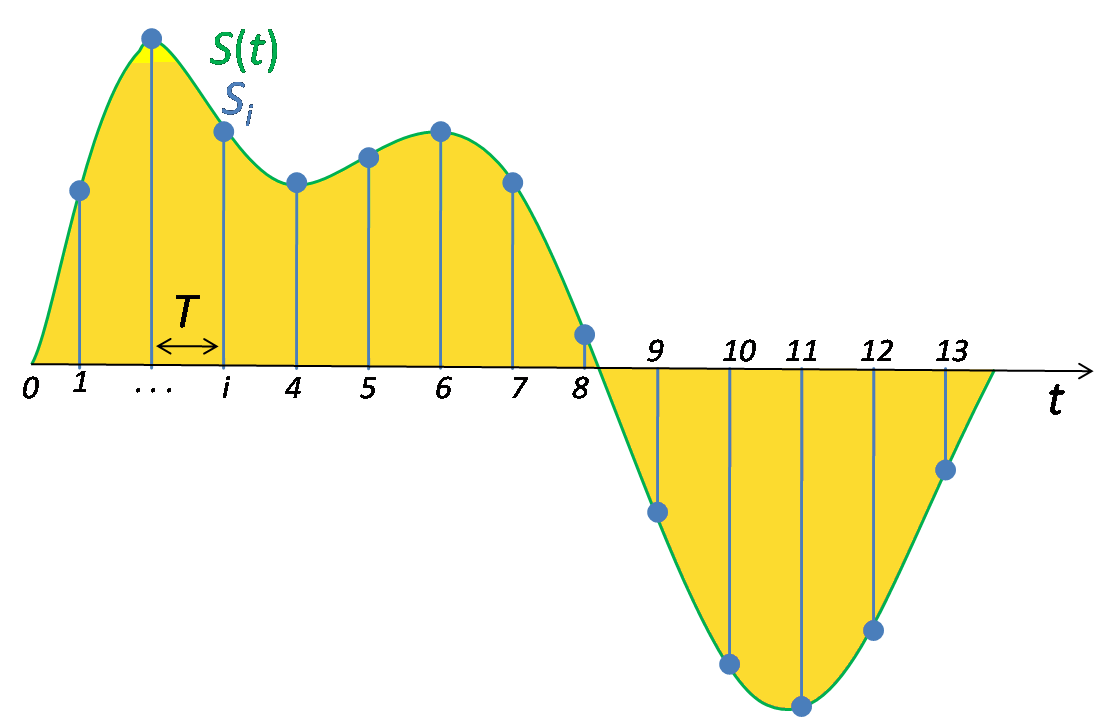


그림 신호 샘플링 표현

1. Sample Rate

실제 오디오는 파형으로 전송된다. PCM은 특정한 sample rate으로 오디오 wave를 디지털화한 것이다. 이 Sample Rate는 Hz 또는 KHz로 측정된다. 더 높은 Sample Rate는 더 잘게 디지털화 시켰다는 의미 이므로 더 좋은 음질을 가질 수 있다.

1. Sign

데이터는 Singed 일 수도 있고 Unsigned 일 수 도 있다. 일반적으로는 Signed 데이터를 사용하여 오디오를 데이터를 표현한다. Singed PCM stream 을 Unsigned 인 것처럼 다루는 것은 Clippling이 일어날 수 있기 때문에 처리과정이 필요하다. 다음은 Signed int형 PCM stream을 Unsigned으로 변환하는 과정이다.

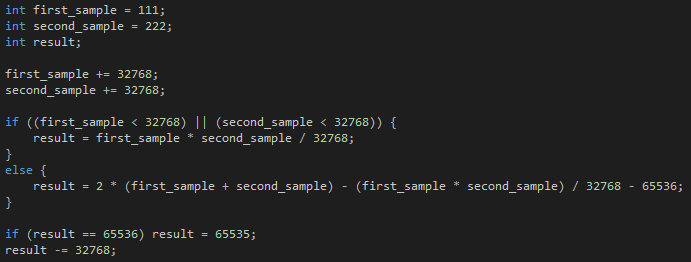


그림 Signed PCM 변환

1. Bit Depth

얼마나 많은 비트들로 하나의 샘플을 만들지 결정하는 것이 Sample Size 변수이다. 대개는 CD audio인 16bits를 사용한다. 요즘 음질은 24bits에서 32bits까지 지원한다. Window 8 의 경우 24bit의 DVD 음질까지 지원하지만 Android에서 HD 음질을 재생하기 위해서는 다른 처리 과정이 필요하다.

1. Byte Ordering

데이터가 Little-endian인지 Big-endian 인지를 뜻하는 변수이다. 대부분은 오디오 데이터들은 little-endian을 채택하고 있다.

1. Number of channels

1 channel의 경우 Mono, 2 channel인 경우 Stereo라고 부른다. 채널이 많을수록 더 입체적인 소리를 들을 수 있다. Stereo의 경우 channel mask는 Left와 right으로 구분된다. 멀티채널 PCM이라고도 있는데, 이것은 보통 PCM sample interleaving하는 것에 의해 다루어진다.

## Volume Control

본 프로젝트의 경우 LoopBack recording 기능을 이용하여 PC의 출력 오디오를 캡쳐한다. WaveWritter를 이용하면 오디오 코덱의 포맷을 씌워 볼륨 등을 추가로 인코딩 시켜 버퍼를 파일로 저장시킬 수 있지만, 바로 스트림을 보내고 재생시키는 어플리케이션 이므로 별도의 헤더를 포함하지 않는 Raw Data를 바로 Android로 전송한다. 때문에 볼륨의 경우 Master Volume 등의 아무런 가공도 되지 않은 상태로 Android로 날아가게 된다. 이 문제를 극복하기 위해서 PC에서 획득한 오디오 Stream을 다시 볼륨을 씌워 보내는 방법을 생각해 내었다.

Volume을 조절해주는 방법은 어렵지 않다. 원래의 파동에 원하는 값을 곱해 증폭시켜 주면 더욱 크거나 작은 소리를 얻을 수 있다. Volume을 조절해주는 API를 사용할 경우 float 값으로 볼륨 값을 받아 0에서 1사이의 값으로 볼륨을 조정해준다. 1을 곱해줬을 경우 현 상태에서 변함 없고 0을 곱해 줬을 경우 파동이 평평해지는, 즉 음소거 상태가 되게 된다. 그러나 현재 획득하는 오디오의 경우 Android의 볼륨과 비교했을 때 작았으므로 0부터 1까지 의 값을 곱하는 것으로 원하는 결과 값을 얻을 수 없었다. 따라서 Raw Data에 1이상의 값을 곱해 볼륨을 높여 줄 필요가 있었다. 이 경우, 한 샘플이 표현할 수 있는 주파수 범위를 벗어나 전혀 새로운 값으로 바뀌어 잡음이 되어 버리는 Clipping 현상이 발생한다. 이를 제어하기 위해서는 한 번의 처리 과정을 거쳐야 한다.

단순한 파형인 사인파(Sine Wave)를 예를 들어 설명 하자면 다음과 같다. [그림 6]과 같은 사인파를 5 만큼 증폭시킨다면 [그림 7] 로 표현되는 더 큰 소리를 들을 수 있다.

|  |  |
| --- | --- |
| 그림 원본 Sine Wave | 그림 5만큼 증폭 시킨 Sine Wave |
|  |  |

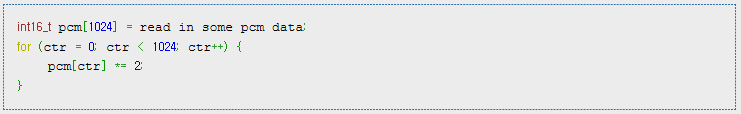


그림 단순히 증폭 시키는 경우

한 샘플의 크기가 2Bytes 인 1024개의 오디오 샘플을 증폭 시키기 위해서는 다음과 같이 쓸 수 있다. 그러나 이렇게 단순하게 구현했을 경우 샘플 처리 값이 범위를 초과하는 경우가 생긴다. 현재 Singed 16bits로 표현된 오디오 데이터를 다루기 때문에 한 샘플이 가질 수 있는 최대 양의 정수 값은 32767이다. 만약 값이 5000인 PCM sample을 가지고 10배로 증폭시키면, 그 결과값은 기대했던 50000이 아닌 -15536으로 표현 될 것이다. 따라서 증폭 결과가 Clipping을 발생시키는 지 확인하여 최대/최소 이내 값으로 표현해 주어야 한다.

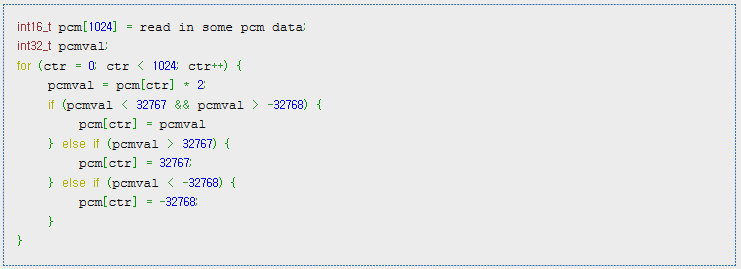


그림 Clipping 현상을 방지하는 경우

또한 float 형으로 오디오 스트림을 받아 처리해 주는데 본 프로젝트에서는 오디오를 Loopback 으로 받는 즉시 보내기 때문에 byte형인 상태이다. 따라서 볼륨을 씌우기 위해서는 float형으로 바꿔주고 다시 볼륨을 곱해줘야 하는 번거로운 작업을 거친다. 이 과정에서 지연이 발생할 수 있기 때문에 이를 생략하고 볼륨을 조절 할 수 있는 방법으로 구현하였다.

## 시나리오

1. Android Application 시나리오

|  |  |
| --- | --- |
| C:\Users\성민\Documents\카카오톡 받은 파일\KakaoTalk_20160308_125910780.png  그림 어플리케이션 첫 화면 | C:\Users\성민\Documents\카카오톡 받은 파일\KakaoTalk_20160308_125911692.png  그림 스피커 디바이스 화면 |

Android 첫 화면에 접속하면 ‘Speaker’와 ‘Connect’두 가지 버튼이 있다. 만들어진 스피커가 없는 경우 Speaker 버튼을 눌러 새로운 스피커 환경을 생성한다. Speaker를 누르게 되면 스피커가 될 기기의 접속 정보를 인증 서버에 보내고, 정보를 받은 서버는 해당 기기로 4자리 인증번호를 보낸다. [그림19]처럼 어플리케이션의 맨 상단에 네 자리 숫자 띄워주어 다른 기기들이 쉽게 스피커 환경에 접속 할 수 있도록 한다. 그 아래로는 본 어플리케이션에서 발생하는 소리, 즉 연결된 다른 모든 디바이스의 소리를 한번에 제어할 수 있는 볼륨바와 이퀄라이저 버튼이 존재한다.

새로운 Audio Type을 정의하지 않으므로 이 볼륨제어는 Speaker 디바이스의 STREAM\_MUSIC 타입의 볼륨 제어와 같다. 따라서, 해당 볼륨바로 볼륨을 제어할 경우, 스피커로 사용하고 있는 디바이스의 다른 어플리케이션의 볼륨이 영향 받을 수 있다. 그 아래로는 현재 접속되어 있는 디바이스들을 볼 수 있다. 연결된 디바이스를 서로 구분하기 위하여 장치 이름을 명시하였다. 또한, 연결된 다른 디바이스를 직접 제어할 필요 없이 이 화면에서 볼륨 조절이 가능하다. 이 경우, 볼륨 값을 줄이거나 높이면 Speaker 디바이스에서 각각의 볼륨 값을 변경하는 것이 아니라 그 값을 Client 디바이스에 요구하여 조절된 값을 제공받도록 한다.

|  |  |
| --- | --- |
| C:\Users\성민\Documents\카카오톡 받은 파일\KakaoTalk_20160308_125910188.png  그림 이퀄라이저 | C:\Users\성민\Documents\카카오톡 받은 파일\KakaoTalk_20160308_125907576.png  그림 Background |

세부적인 이퀄라이저 화면은 다음과 같다. 각 주파수마다 값을 변경할 수 있게 하여 사용자가 원하는 음향을 들을 수 있도록 한다. 기본값은 ‘사용자 설정’으로 되어 있으나 하단의 이미 효과에 따라 주파수를 지정해둔 버튼을 클릭하면 ‘일반’, ‘팝’, ‘클래식’, ‘재즈’, ‘락’등 다양한 모드를 선택할 수 있다. 이러한 이퀄라이저 모드를 선택할 경우 그 모드에 맞춰 자동으로 주파수의 값들이 변경된다. 원하는 값으로 조절바를 사용자에 알맞게 조절하면 해당 효과로 오디오를 즐길 수 있다. 이때, 한번 설정을 해둔 이퀄라이저 값은 Cookies로 저장을 해두어 어플리케이션이 종료 후 다음에 실행될 경우에도 별도의 동작없이 이전에 사용하던 값과 동일한 값을 제공받도록 하였다. Speaker와 Client 모두 어플리케이션이 실행되면 백그라운드 상태 정보에 대해서 사용자가 알 수 있도록 알림창에 제공한다. [그림 21]과 같이 Speaker가 실행되거나 Speaker와 연결이 시작되면 어플리케이션의 상태를 알림창을 통해 제공한다.

|  |  |
| --- | --- |
| C:\Users\성민\Documents\카카오톡 받은 파일\KakaoTalk_20160308_125909409.png  그림 Client 화면 | C:\Users\성민\Documents\카카오톡 받은 파일\KakaoTalk_20160308_125908741.png  그림 접속 진행 화면 |

이미 스피커로 작동하는 기기가 존재하는 경우, Connect 버튼을 누르면 4자리 인증번호를 입력할 수 있는 화면이 뜬다. 사용자가 해당 창에 연결하려는 스피커 기기의 4자리 인증 번호를 입력하면 인증 서버가 해당 인증번호를 가지고 있는 스피커의 접속 정보를 사용자의 디바이스에게 넘겨주어 서로 통신할 수 있도록 해준다. 사용자가 따로 해당 기기의 이름을 정해주지 않으면 기본값으로 이름이 매칭되어 Speaker 기기의 화면에 뜨게 된다.

1. Window Application 시나리오



그림 윈도우 프로그램 실행화면

Window Application을 실행하면 위와 같은 화면이 나타난다. ‘START’버튼을 눌러 Speaker에 접속하기 위해 인증번호 4자리와 장치 이름을 입력 할 수 있는 단계로 넘어 가게 된다.

|  |  |
| --- | --- |
| D:\Dropbox\문서\SSM\2015-2 창의과제\Android를_통한_여러_기기의_Audio_Stream_출력\디자인\Window_APP_Connect.png  그림 접속 화면 | D:\Dropbox\문서\SSM\2015-2 창의과제\Android를_통한_여러_기기의_Audio_Stream_출력\디자인\Window_APP_Connec2.png  그림 인증번호 입력 |

위 화면에서 Android Application에서 Speaker 환경을 생성하고 발급 받은 인증번호 4자리를 Client에서 입력하고 ‘CONNECT’버튼을 눌러 Speaker에 접속 할 수 있다.



그림 연결된 상태

위 은 Window Application에서 오디오를 캡쳐하여 PCM 16Bit, 44.1Khz 오디오 데이터를 Speaker로 UDP를 이용하여 데이터를 전송 하고, 이렇게 전송된 데이터를 Speaker에서 받아와 Window에서 발생하는 오디오를 재생한다.

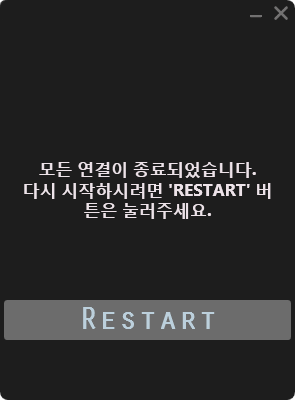


그림 연결 종료

위 화면은 Speaker에서 연결 종료를 요청 하였거나, Window Application에서 ‘DISCONNECT’를 누르면 Speaker와의 연결이 종료되고, 새로운 연결을 할 수 있도록 ‘RESTART’버튼이 나타나 새로운 연결을 할 수 있다.

# 개발 일정

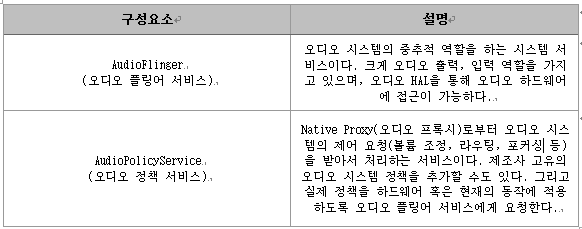
표 개발 일정

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Name | Development | 12月 | | | | 1月 | | | | 2月 | | | |
| 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 |
| 손민수 | 웹 서버 구축 |  |  |  |  |  |  |  |  |  |  |  |  |
| DB 설계 및 인증 절차 |  |  |  |  |  |  |  |  |  |  |  |  |
| Effect & Equalizer |  |  |  |  |  |  |  |  |  |  |  |  |
| 김재원 | WASAPI 분석 |  |  |  |  |  |  |  |  |  |  |  |  |
| 윈도우 Audio Capture 및 파일 전송 |  |  |  |  |  |  |  |  |  |  |  |  |
| Audio format Setting |  |  |  |  |  |  |  |  |  |  |  |  |
| 윈도우 service 및 UI 구현(Client) |  |  |  |  |  |  |  |  |  |  |  |  |
| 윈도우 Volume 제어 |  |  |  |  |  |  |  |  |  |  |  |  |
| 고성민 | Android Audio System 분석 |  |  |  |  |  |  |  |  |  |  |  |  |
| Android Audio recorder 및 파일 전송 |  |  |  |  |  |  |  |  |  |  |  |  |
| 보조장치를 이용한 Audio capture |  |  |  |  |  |  |  |  |  |  |  |  |
| Android service 및 UI 구현  (Speaker & Client) |  |  |  |  |  |  |  |  |  |  |  |  |
| Android Volume 제어 |  |  |  |  |  |  |  |  |  |  |  |  |
| Speaker Audio 재생 |  |  |  |  |  |  |  |  |  |  |  |  |
| Android capture를 위한 보조장치 제작 |  |  |  |  |  |  |  |  |  |  |  |  |
| Team | 1차 통합 테스트(.wav) |  |  |  |  |  |  |  |  |  |  |  |  |
| 2차 통합 테스트(PCM) |  |  |  |  |  |  |  |  |  |  |  |  |
| Testing and Debugging |  |  |  |  |  |  |  |  |  |  |  |  |

# 용어 정리

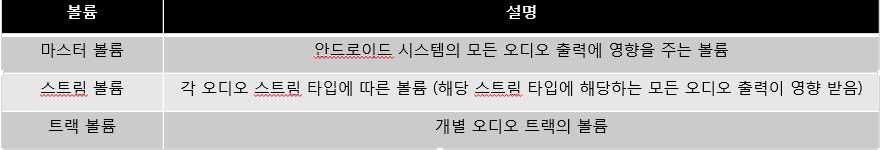
## Android System Service

표 Audio System Service



Android 네이티브 계층의 오디오 시스템 서비스에는 위 표와 같이 두 가지 요소를 가진다.

표 반영되는 볼륨 종류



Android의 소리의 크기를 제어할 수 있는 볼륨의 종류에는 위와 같이 세 가지가 있다. 본 과제에서는 한 트랙 별로 각각의 기기의 소리를 제어할 것이기 때문에 트랙 볼륨에 집중한다.

## Digital Clipping

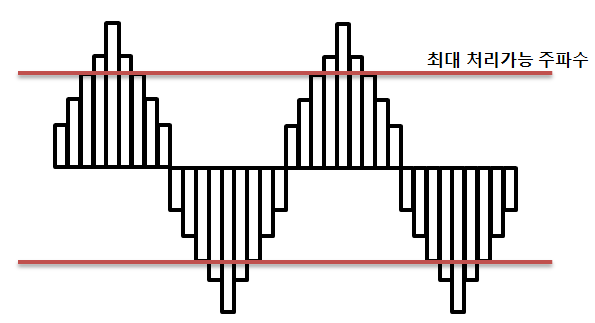


그림 디지털 클리핑

디지털 클리핑이란 위 그림과 같이, 기기가 최대로 처리할 수 있는 신호 수준을 넘어가게 되어 생기는 현상을 말한다. 클리핑이 일어나게 되면, 소리의 특성을 크게 왜곡(distortion)하게 되어 오디오 질을 상당히 저하시킨다. 주로 클리핑에서의 왜곡은 하모닉 디스토션(harmonic distortion)으로 발생하게 된다. 본 과제는 PCM 샘플링으로 인한 인터샘플 피크가 발생하여 클리핑을 처리해야 한다.

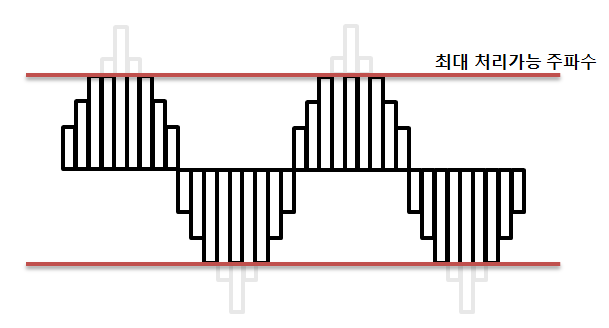


그림 클리핑 현상 방지

디지털 클리핑 현상은 위 그림과 같이 사인파가 사각파에 가까워지게 되는데, 사각파를 푸리에 해석하게 되면 특정 사인파의 배음들이 무한히 합한 것으로 표현된다

# 참고문헌

<http://blog.secmem.org/137> - RTP(Real-time Transport Protocol)의 이해

<http://www.ktword.co.kr/abbr_view.php?m_temp1=3394> - RTP(Real-time Transport Protocol) 구조

Android Component service 용어정리

<http://drcarter.tistory.com/162> - Android AudioTrack player

안드로이드 미디어 프레임워크 – audio system , volume control

강북 – stream type 및 volume control

Android ndk programming – android ndk development

<http://www.slideshare.net/namhyeonuk90/iocp> - iocp server의 구조

<http://www.101apps.co.za/index.php/articles/perfect-sound-using-the-equalizer-effect-a-tutorial.html> - android Audio Effect

http://blog.secmem.org/207 & http://mobileandlife.tistory.com/260#rp - system 권한 app tip