The Journal of The Institute of Internet, Broadcasting and Communication (IIBC)
Vol. 16, No. 5, pp.165-172, Oct. 31, 2016. pISSN 2289-0238, eISSN 2289-0246

http://dx.doi.org/10.7236/JIIBC.2016.16.5.165
JIIBC 2016-5-25

임베디드 라이브 오디오 스트리밍 시스템 구현

Implementation of Embedded Live Audio Streaming System:ESCatcher

황기태*

Kitae Hwang*

요 약 본 논문은 Raspberry Pi 3 임베디드 컴퓨터를 이용한 스트리밍 시스템의 구현 사례를 소개한다. 본 연구에서 구현한 스트리밍은 파일 전송 방식이 아닌 라이브 스트리밍 시스템이다. 오디오 입력 단자로부터 받은 아날로그 신호를 wav 형식으로 변환하여 서버에 접속한 다수의 사용자에게 동시에 방송하는 푸시형 멀티스레드 스트리밍이다. 스트리밍 서버 소프트웨어는 전체 자바 언어로 구현하였기 때문에 Raspberry Pi 3 가 아닌 다른 임베디드 컴퓨터에도 수정 없이 탑재될 수 있다. 계산과 실험을 통해 분석한 결과 최대 65여명을 동시 스트리밍 할 수 있다. 그리고 오디오 소스로부터 청취 단말기의 재생까지의 시간 지연은 40ms 남짓되는 것으로 평가되었다.

Abstract This paper presents an implementation of a live audio streaming system using the Raspberry Pi 3 embedded computer. This system is a live streaming system not file-based streaming. This is a push streaming system which converts the incoming analog audio signal to digital samples and broadcasts them to multiple connected users concurrently. Since the server software is developed in Java language, it can be installed on any other embedded computers without any modification. We concluded that ESCatcher can service live streaming about 60 users concurrently through calculations and experiments, And also we achieved the delay time of a little bit more than 40ms between arrival of audio source and play on the android device

Key Words : Audio Streaming, Embedded Streaming Server, Raspberry Pi, Streaming

Ⅰ. 서 론

은행이나 관공서, 기차역 등 많은 사람들이 모여 기다리는 곳에서는 지루함을 덜어주기 위해 TV를 설치하고 있다. 하지만, TV 소리가 주변에 소음이 되지 않도록 소리를 꺼 놓거나 아주 작게 틀어 놓는 경우가 대부분이다. 혹은 가정에서 다른 가족들에게 피해를 주지 않기 위해 TV를 소리 없이 보는 경우도 종종 있다. 본 논문은 그림 1과 같이 오디오 입력 단자로 받은 아날로그 신호를 무선으로 라이브 스트리밍(live streaming)하는 임베디드 스트리밍 시스템을 설계 구현한 내용을 소개한다.

멀티미디어의 등장 이래 오디오 비디오 스트리밍 기술은 지속적으로 발전되어, Windows Media Services, VLC media player, Subsonic, IIS Media Services, Darwin Streaming Server, Open Broadcaster Software, Unified Streaming Platform, Wowza Streaming Engine 등 현재 많은 스트리밍 솔루션이 있다[1,2,3,4,5].

과거 대부분의 스트리밍 솔루션은 PC나 서버 급의 컴퓨터 실행되며 대형 스트리밍을 위해 개발되어 Raspberry Pi[6]와 같이 임베디드 장치에 탑재되는 사례가 거의 없었다. 최근 들어 임베디드 장치를 개인용 스트리밍 서버로 만드는 사례들이 등장하고 있으며, Pi MusicBox[7],

*정회원, 한성대학교 컴퓨터공학부(교신저자)
접수일자 : 2016년 7월 22일, 수정완료 : 2016년 9월 2일
게재확정일자 : 2016년 10월 7일

Received: 2 August, 2016 / Revised: 2 September, 2016 / Accepted: 7 October, 2016
*Corresponding Author: calafk@hansung.ac.kr
Dept. of Computer Engineering, Hansung University, Korea

Subsonic[8], Cherry Music[9] 등이 대표적이며, 이들은 거의 모두 개인이 USB 메모리 등에 소장하고 있는 오디오/비디오 파일이나 인터넷 상에 있는 미디어 파일을 스트리밍하는 솔루션이다.

본 논문에서는 Raspberry Pi, Orange Pi, CHIP 등 상용 임베디드 보드 중 가격이 비교적 싸고 성능이 뛰어난 Raspberry Pi를 서버로 하여 라이브 오디오 소스를 스트리밍할 수 있는 임베디드 스트리밍 솔루션을 구축한다.

본 논문은 2장에서 시스템 설계를, 3장에서는 구현 내용을 소개한다. 4장에서 성능 분석에 대해 소개하고 5장에서 결론을 맺는다.

그림 1. 임베디드 오디오 스트리밍 시스템
Fig. 1. Embedded Audio Streaming System

II. 시스템 구성 및 설계

1. 전체 시스템 구성

본 논문에서 구현한 임베디드 오디오 스트리밍 솔루션을 ESCatcher(Embedded Sound Catcher)라고 부른다.

가. 하드웨어 구성

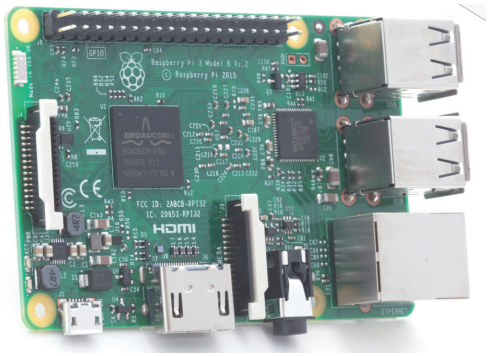

그림 2. Raspberry Pi 3 보드
Fig. 2. Raspberry Pi 3 Board

이 솔루션은 그림 2과 같은 모양의 Raspberry Pi 3 Model B[6] 상에서 구현한다. 이 보드는 1.2GHz Quad Cortex A53 CPU에 IGB SDRAM, 400MHz GPU, Ethernet, 802.11n 무선랜, HDMI 인터페이스 등을 가진 최신 임베디드 컴퓨터이다.

나. 스트리밍 클라이언트

ESCatcher는 2가지 종류의 클라이언트를 지원하도록 구현한다. 안드로이드 단말기에는 안드로이드 앱으로, HTML5가 지원되는 단말기에서도 웹 브라우저로 라이브 스트리밍을 받을 수 있도록 구현한다.

ESCatcher의 스트리밍은 오디오 파일을 스트리밍하는 방식이 아니라, 실시간으로 입력되는 아날로그 오디오를 클라이언트에 전송하는 라이브 스트리밍이다. 또한 여러 클라이언트가 동시에 라이브 스트리밍을 받을 수 있도록 설계하였으며 접속과 동시에 스트리밍을 받을 수 있도록 설계하였다.

다. 소프트웨어 구성

ESCatcher는 그림 3과 같이 라이브 스트리밍 서버(live streaming server), 안드로이드 오디오 재생기 앱, HTML5 웹 페이지와 자바스크립트 파일, 웹 서버의 4 부분으로 구성된다.

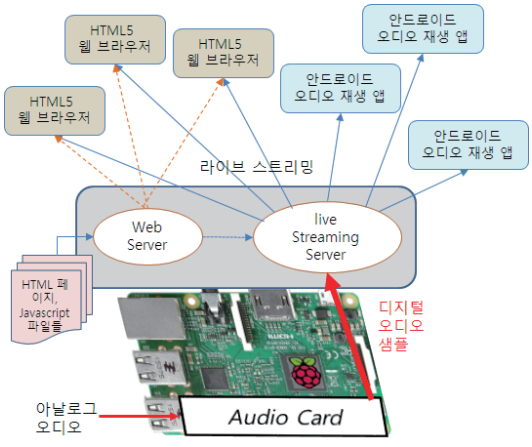

그림 3. ESCatcher 시스템 구성
Fig. 3. ESCatcher System

첫째, 스트리밍 서버는 임베디드 리눅스를 탑재한 Raspberry Pi 3 에서 실행되는 소프트웨어 모듈이다. Raspberry Pi 3에 Cirrus Logic Audio Card를 장착시키

고 외부 아날로그 오디오 신호를 디지털로 샘플링하게 하였다. 스트리밍 서버 소프트웨어 모듈은 Cirrus Logic Audio Card로부터 오디오 샘플을 주기적으로 읽어, 접속한 여러 클라이언트로 실시간 전송한다. 라이브 스트리밍 서버는 멀티스레드로 구현한다.

둘째, 안드로이드 오디오 재생기는 안드로이드 운영체제가 탑재된 장치에서 실행되는 클라이언트 소프트웨어 앱으로서, 사용자의 요청에 의해 스트리밍 서버에 접속하고 서버로부터 받은 오디오 샘플을 단순히 재생한다.

셋째, HTML5 표준을 따르는 웹 브라우저에서 스트리밍 서버에 접속하여 특별한 애플리케이션이나 플러그인의 도움 없이 오디오 스트림을 재생 가능하도록 하였다. 이를 위해 HTML5 페이지와 서버로부터 스트리밍 샘플을 받아 재생하는 자바스크립트 코드를 작성하였다.

넷째, 웹 브라우저의 접속을 받아 HTML5 페이지와 자바스크립트 파일을 전송하고, 웹 브라우저 사용자가 라이브 스트리밍 서버로부터 오디오 스트림을 받을 수 있도록 연결해주는 간단한 웹 서버를 작성하였다.

2. 시스템 작동 과정

시스템이 작동하는 과정은 다음과 같이 설계하였다.

가. 오디오 샘플링

Raspberry Pi 3에 장착된 Cirrus Logic Audio Card 카드는 아날로그 오디오 신호를 디지털 샘플로 바꾼다. 라이브 스트리밍 서버 소프트웨어는 Java API를 이용하여 오디오 샘플을 주기적으로 받아온다. 오디오 샘플은 wav 포맷이다.

나. 오디오 샘플 공급

라이브 스트리밍 서버는 스트리밍 버퍼를 가지고 있으며, Java 라이브러리를 통해 오디오 샘플을 주기적으로 스트리밍 버퍼에 옮겨 놓는다.

다. 안드로이드 클라이언트 접속

라이브 스트리밍 서버는 안드로이드 오디오 재생 앱으로부터 접속을 받으면 전담 서비스 스레드를 하나 생성한다. 이 스레드는 스트리밍 버퍼에 저장된 라이브 오디오 샘플을 전송한다. 버퍼에 새로운 샘플이 도착하면 즉각 앱으로 전송한다.

라. 웹 브라우저 접속

웹 브라우저가 접속해오는 경우, 본 연구에서 개발한 웹 서버가 접속을 받아 HTML5 페이지를 전송한다. HTML5 페이지에 적시된 대로 다시 웹 브라우저의 요청에 따라 자바스크립트 파일을 전송한다. 자바 스크립트 파일의 코드는 웹 소켓을 이용하여 라이브 스트리밍 서버에 접속하고, 안드로이드 앱의 접속 시와 마찬가지 방법으로 전담 스레드를 생성하고 스트리밍이 이루어진다.

3. 설계 원칙

ESCatcher 스트리밍 시스템을 설계 구현함에 있어 다음과 같은 원칙을 정하였다.

첫째, 개발 및 테스트, 디버깅이 용이하고 높은 이식성을 가질 수 있도록 한다. 이를 위해 하드웨어와 운영체제에 무관하게 실행 가능한 자바 언어로 개발하였다

둘째, 가능한 단순한 코드로 작성한다. 코드를 단순화시켜 여러 종류의 작은 임베디드 보드에도 탑재가 용이하도록 하고, 확장 및 수정이 용이하도록 한다. 이를 위해 간단한 웹 서버를 직접 설계 구현하였다. 또한 본 ESCatcher 솔루션이 사용되는 응용 환경은 원거리 인터넷이 아니라 가정, 사무실, 은행, 자동차 극장, 잔디밭 등과 같이 닫힌 공간이거나 오디오 소스와 최대 100미터 이내의 공간이 대상이다. 그러므로 RTP 등의 실시간 스트리밍 프로토콜을 사용하지 않고 간단히 UDP로만 전송한다.

셋째, 사용자가 라이브 오디오 스트리밍을 쉽게 서비스 받을 수 있도록 한다. 모바일, 탭, PC 등 거의 대부분의 장치는 HTML5 표준 웹 브라우저를 가지고 있다. 그러므로 ESCatcher는 HTML5 웹 페이지와 자바스크립트 코드를 통해 특별한 소프트웨어나 플러그인 없이 스트리밍을 받을 수 있도록 하였다.

Ⅲ. 구현

1. 라이브 스트리밍 서버

라이브 스트리밍 서버는 ESCatcher 솔루션의 핵심 코드이며, 그 구조는 그림 4와 같다.

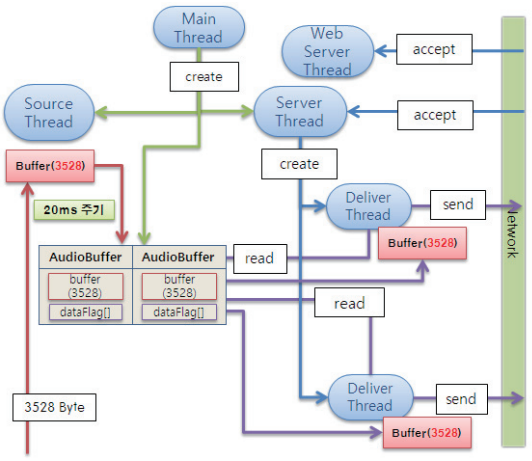

그림 4. 라이브 스트리밍 서버의 구조
Fig. 4. Architecture of Live Streaming Server

가. 멀티스레드 구성

라이브 스트리밍 서버는 다중 작업을 위한 Main Thread, Source Thread, Server Thread, Deliver Thread로 구성되는 멀티스레드로 구현되었다. Main Thread는 라이브 스트리밍 서버의 메인 스레드로서 서버의 시작 스레드이며, 처음부터 Source Thread와 Server Thread를 생성하여 실행시킨다.

Source Thread는 Audio Card로부터 오디오 샘플을 주기적으로 가지고 와서 AudioBuffer에 저장하는 작업만 전담한다. AudioBuffer는 이중 버퍼로 구성된다.

Server Thread는 안드로이드 앱으로부터 5000번 포트로 UDP 네트워크 접속을 대기하고, 접속을 받게 되면 Deliver Thread를 하나 생성한 후 다음 접속을 대기하는 접속 전용 스레드이다.

Deliver Thread는 생성되자 마다 바로 AudioBuffer로부터 오디오 샘플을 읽어 연결된 클라이언트로 전송한다. AudioBuffer가 이중 버퍼이므로 번갈아 읽으며 버퍼가 비어 있으면 대기한다. 클라이언트마다 하나의 Deliver Thread가 생성된다.

Web Server Thread는 웹 브라우저로부터 80 포트의 HTTP 접속을 대기하는 스레드로서 간단한 웹 서버이다. 웹 브라우저로부터 접속이 오면 HTTP 헤더를 분석하여 요청한 HTML 파일이나 CSS, 자바스크립트 파일 등을 전송하고, 웹 소켓의 연결 요청이 오면 Deliver Thread를 생성하여 웹 브라우저로 오디오 스트림을 전송하도록 한다.

나. AudioBuffer와 오디오 프레임

AudioBuffer는 이중 버퍼 구조로 만든다. 버퍼가 하나만 있으면 Source Thread와 Deliver Thread 사이의 충돌이 잦아져서 둘 중 하나의 대기 시간이 길어지게 되고 이에 따라 오디오 샘플을 잃어버리거나 클라이언트로의 전송 시간이 길어지게 되어 오디오 재생 지터(jitter)가 발생할 수 있기 때문이다.

ESCatcher에서는 44.1KHz로 샘플링하며, 하나의 샘플은 2 채널(스테레오)에 채널당 2 바이트로 설정하였다. 현재 Raspberry Pi 3에 장착된 Cirrus Logic Audio Card의 오디오 샘플링 과정을 검사해본 결과, Cirrus Logic Audio Cards는 여러 개의 샘플을 모아 3528 바이트 단위로 Java API에게 공급하였고 Source Thread는 3528 바이트 단위로 읽어 오게 되었다. 3528바이트를 시간으로 바꾸면 20ms의 시간이며, 20ms 동안 모은 오디오 샘플들을 한 번에 공급함을 알게 되었다. 그러므로 본 연구에서 AudioBuffer 한 개의 크기를 3528 바이트로 구성하였다. 본 논문에서는 3528 바이트 단위로 모은 오디오 샘플들을 오디오 프레임이라고 부른다.

다. 네트워크와 Deliver Thread

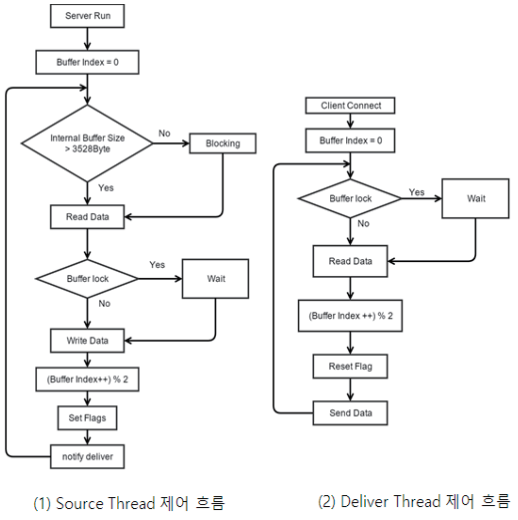

그림 5. Source Thread와 Deliver Thread의 제어 흐름
Fig. 5. Control flows of Source Thread and Deliver Thread

Deliver Thread는 UDP 통신을 통해 클라이언트로 오디오 프레임을 내 보낸다. UDP를 사용하는 이유는 서버

와 클라이언트는 라우터 없이 연결되는 근거리 통신이기 때문에 패킷을 잃어버리거나 패킷이 보낸 순서와 다르게 도착될 가능성이 거의 없고 통신에 따른 오버헤드가 TCP나 HTTP보다 적기 때문이다.

Deliver Thread는 AudioBuffer의 이중 버퍼로부터 오디오 프레임을 번갈아 읽어 UDP 소켓을 통해 클라이언트로 전송한다. 버퍼가 비어 있는 경우 Source Thread가 깨울 때까지 대기한다. 알고리즘은 그림 5와 같다.

2. 안드로이드 오디오 재생기

본 연구에서 개발한 안드로이드 오디오 재생기 앱의 구조는 그림 6과 같다.

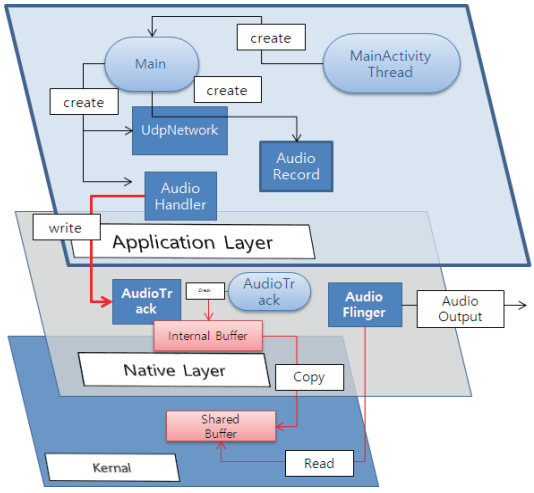

그림 6. 안드로이드 오디오 재생기의 구조
Fig. 6. Architecture of Android Audio Player

그림 6에는 안드로이드 커널 위에 오디오 재생을 위해 라이브러리로 제공되는 Native 계층이 있고, 그 위에 안드로이드 자바로 작성된 오디오 재생기 애플리케이션이 있다. 안드로이드 라이브러리의 native 영역에서 제공하는 AudioTrack 객체가 Application 계층으로부터 받은 오디오 프레임을 스피커를 통해 재생하는 모든 과정을 책임진다. 그러므로 Application 계층에서는 AudioTrack 객체에 주기적으로 오디오 프레임을 전달해주기만 하면 된다.

ESCatcher에서 다루는 오디오 프레임은 인코딩되지 않는 PCM 데이터로서 PCM 데이터를 지원하는 AudioTrack 객체를 사용하였다. 그림 7은 실제 구현한 안드로이드 오디오 재생기 앱을 캡처한 이미지를 보여준다.

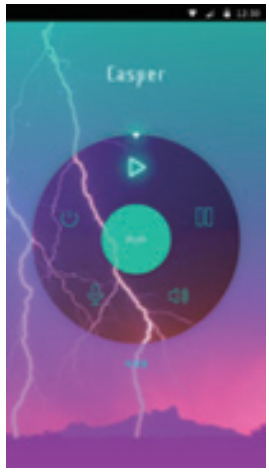

그림 7. 안드로이드 오디오 재생기
Fig. 7. Android Audio Player

3. 웹을 통한 라이브 스트리밍

웹으로 라이브 오디오 스트리밍을 받을 수 있도록 ESCatcher 서버에는 HTML 페이지와 하나의 자바스크립트 파일을 두었다. HTML 페이지를 웹 브라우저가 ESCatcher 서버에 접속하기 위한 목적이며, ESCatcher 서버에 접속하여 스트리밍을 받는 코드는 자바스크립트 코드로 작성되었다. 자바스크립트 코드는 HTML5에서 제공하는 WebSocket 객체를 이용하여 ESCatcher 서버에 접속하고, HTML5에서 제공하는 AudioContext 객체를 이용하여 오디오 샘플을 재생하도록 하였다.

웹 프로토콜에서 바이너리 정보인 오디오 샘플을 전송하기 위해 이용되는 Base64 인코딩에 따라, 서버는 오디오 데이터를 Base64로 인코딩하여 보내고, 자바스크립트 코드에서는 다시 디코딩하여 AudioContext 개체로 보내 재생한다. 오디오 데이터의 원활한 재생을 위해 3528 바이트의 오디오 프레임을 한 번에 여러 개 묶어 보내는 방식을 사용하였다.

IV. 성능 분석

1. 성능 분석 지표

본 논문에서 개발한 ESCatcher의 성능을 분석하기 위해, 동시 사용자 수 n과 소스와 단말기의 립싱크 오차 LipSyncError의 2 개의 성능 지표를 분석 대상으로 정하였다.

2. 동시 사용자 수 n

동시 사용자 수란 현재 Raspberry Pi에설 실행되는 스트리밍 서버 소프트웨어가 동시에 스트리밍 할 수 있는 사용자 수를 뜻한다. 동시 사용자 수 n의 최대 값을 계산하기 위해, ESCatcher의 아키텍처에 따라 스트리밍 동안에서 소요되는 각 시간을 표 1과 같이 표현하였다.

표 1. 스트리밍에 소요되는 각 시간
Table 1. Time Fractions in Audio Streaming

시간	설명
$T_{BufferRead}$	Deliver Thread가 AudioBuffer에서 스레드 내 버퍼로 복사하는 시간
$T_{SendData}$	Deliver Thread가 자신의 버퍼에서 소켓을 통해 클라이언트로 전송하는 시간
N_{client}	접속한 클라이언트 수. Deliver Thread의 개수와 동일
T_{α}	컨텍스트 스위칭 등 오버헤드 시간
$T_{consume}$	N_{client} 개의 클라이언트로 전송하기 위해 소요되는 전체 시간
T_{fill}	Cirrus Logic Audio Card에서 오디오 샘플을 3528 바이트 단위로 AudioBuffer로 공급하는 시간

스트리밍 서버가 N_{client} 개의 클라이언트로 3528바이트로 구성되는 하나의 오디오 프레임을 모두 전송하는데 걸리는 시간, $T_{consume}$은 다음과 같이 표현된다.

$$T_{consume} = (T_{BufferRead} + T_{SendData} + T_{\alpha}) \times N_{client} \quad (1)$$

그리고 $T_{consume}$은 T_{fill}보다 커서는 안 되며, 현재 T_{fill}은 측정하고 계산한 결과 20ms이다.

$$T_{consume} < T_{fill} \quad (2)$$

식 (2)의 조건을 충족시키는 N_{client}의 최대치, $Max(N_{client})$가 바로 수용 가능한 동시 사용자 수 n이다.

ESCatcher 시스템의 오디오 스트리밍 동시 사용자 수 n은 Raspberry Pi 3 하드웨어와 Cirrus Logic Audio Card의 샘플링 특성과 관계된다. 본 연구에서는 앞서 언급한 바와 같이 Cirrus Logic Audio Card에 44.1Khz로 16비트, 2 채널로 샘플링하도록 설정하였고, 그에 따라 20ms당 3528 바이트씩 Source Thread에게 공급된다. 물론 이 설정은 언제든지 변경 가능하다. 이 설정을 기준으로 하여 본 연구팀은 표 2와 같이 실제 각 요소 시간을 측정하였다.

표 2. 측정된 시간 값
Table 2. Measured Time Values

시간	평균 값
$T_{BufferRead}$	0.01 ms
$T_{SendData}$	0.2 ms
T_{α}	측정할 수 없는 값
$T_{consume}$	$(0.01ms + 0.2ms + T_{\alpha}) \times N_{client}$
T_{fill}	20ms

그러므로 식 (2)에 따라 표 2의 각 값을 적용하고 정리하면 다음과 같다.

$$N_{client} < 20/(0.21 + T_{\alpha}) \quad (3)$$

이 식에서 T_{α}의 값을 0.1ms로 하면 다음과 같다.

$N_{client} < 20/0.31$

$max(N_{client}) \simeq 65;$

즉 최대 65명까지 동시 서비스 가능하다고 계산된다.

이것은 이론적인 계산 결과이므로 실제 상황에서도 65명까지 서비스 가능한지 평가할 필요가 있으나, 실험을 위해서는 65개 이상의 모바일 단말기를 동시에 접속해야 하고 측정 가능하며, 무선 통신을 사용하기 때문에 전송 시간의 변동성이 높아 측정이 매우 어렵다. 또한 T_{α}의 값이 어느 정도인지 분명치 한다. 성능 검증은 추후의 연구를 통해 세밀하게 진행하기로 한다.

3. 립싱크 오차 LipSyncError

립싱크 오차란 영상을 시청할 때 비디오와 오디오가 불일치되는 오차 시간이다. 그림 8과 같이 립싱크 오차에 대한 미국 스탠포드 대학의 연구 결과에 따르면 45ms 이하로 유지하는 것이 바람직하지만, 90ms를 넘어가면 시청자들을 견디기 힘들게 한다[10, 11].

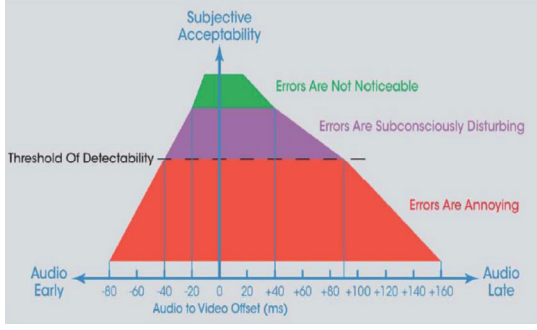

그림 8. 립싱크 오차에 따른 시청자의 감정
Fig. 8. Typical Viewer Perception of Lip Sync Errors

본 연구는 비디오와 오디오를 동시에 전송하지 않기 때문에 특별히 립싱크 오차를 따질 이유가 없지만, TV 앞에서 비디오를 보면서 오디오만 듣는 경우가 있기 때문에 이 오차에 대한 성능을 분석하였다.

사실상 립싱크 오차는 본 연구에서 오디오 신호가 입력된 시간과 클라이언트에서 재생되는 시간의 지연 시간이다.

버퍼 한 개 즉 3528 바이트로 구성되는 오디오 프레임이 서버의 소스에 입력되어 클라이언트가 재생하는 시점까지 걸리는 지연 시간은 다음과 같다.

$$T_{fill} + T_{BufferRead} + T_{SendData} + T_{\alpha} = 20.21 + T_{\alpha} \text{ ms} \quad (4)$$

새로운 클라이언트가 도착하여 Deliver Thread가 실행을 시작하면, 차 있는 버퍼를 전송하지 않고 현재 채워지고 있는 버퍼에서부터 전송을 시작하게 하면 Deliver Thread가 대기하는 시간은 평균 $T_{fill}/2$ 이지만, 최악의 경우 T_{fill}이므로 이를 바탕으로 (4)가 작성되었다.

만일 안드로이드 재생기에서 안정적인 스트리밍을 위해 이중 버퍼를 사용한다면 안드로이드 사용자의 경우 지연 시간이 다음과 같이 계산된다.

$$2*T_{fill} + T_{BufferRead} + T_{SendData} + T_{\alpha} = 40.21 + T_{\alpha} \text{ ms} \quad (5)$$

T_{α} 가 5ms 보다 큰 경우는 없을 것이라고 가정 하면, 결론적으로 ESCatcher 시스템의 립싱크 오차 혹은 지연 시간은 미국 스탠포드 대학의 연구 결과에서 바람직하다고 평가된 45ms 이하로 유지된다.

V. 결 론

가정이나 은행, 기차역 대합실, 자동차 영화관 등 라이브 오디오 스트리밍이 필요한 많은 경우가 있다. 본 연구는 Raspberry Pi 3 임베디드 컴퓨터를 라이브 오디오 스트리밍 서버로 하여 구현한 시스템 ESCatcher의 사례를 소개하였다.

이 스트리밍 시스템은 Raspberry Pi 3에 Cirrus Logic Audio Card를 장착하고 아날로그 오디오를 44.1Khz에 채널당 16비트, 2채널로 샘플링하여 동시에 최대 65명이 청취할 수 있는 라이브 오디오 스트리밍 시스템이다. ESCatcher의 스트리밍 서버는 Java 언어로 작성하여 PC에서 개발하고 컴파일한 클래스 파일을 별도의 포팅이나 수정 작업 없이 그대로 Raspberry Pi 3의 시스템에 복사하기만 하면 실행되는 높은 이식성을 가지고 있으며, 안드로이드 오디오 재생기를 제공하고 UDP로 단순하고 빠르게 통신하도록 하였다. 그리고 HTML5 웹 브라우저를 탑재한 어떤 장치에서도 별도의 플러그인 없이 오디오 스트림을 받을 수 있도록 구현하였다.

현재 구현된 시스템은 무선 통신을 사용하여 서버와 클라이언트 사이의 거리가 가까운 단점이 있다. 추후 원거리 인터넷에서 스트리밍 가능하도록 확대하고 그에 따라 실시간 스트리밍에 적합한 프로토콜을 구현할 계획이며, 보다 세밀한 성능 분석과 평가가 필요하다.

References

[1] https://en.wikipedia.org/wiki/Comparison_of_streaming_media_systems
[2] http://www.videolan.org/vlc/
[3] Hong-rae Lee, Hyung-suk Lee, Kwang-deok Seo, “Design and Implementation of Darwin Streaming system based on SVC”, Summer Conference, The Korean Institute of Communications and Information Sciences, 2016
[4] Sritrusta Sukaridhoto, Nobuo Funabiki, Toru Nakanishi, and Dadet Pramadihanto, “A Comparative Study of Open Source Softwares for Virtualization with Streaming Server Applications”, pp. 577-581, The 13th IEEE International Symposium on Consumer Electronics, 2009
[5] Jaegeol Yim, “Review of Streaming Server Management Systems”, Vol. 85, pp. 55-58, Advanced Science and Technology Letters, Information Technology and Computer Science, 2015
[6] https://www.raspberrypi.org
[7] http://www.pimusicbox.com
[8] http:/www.subsonic.org
[9] http://www.fomori.org/cherrymusic/
[10] Byron Reeves, David Voelker, “Effects of Audio-Video

Asynchrony on Viewer's Memory, Evaluation of Content and Detection Ability", Research Report Prepared for Pixel Instruments, Stanford University, Oct. 1993.

[11] http://www.lipfix.com/technical_details.html

저자 소개

황 기 태(정회원)

- 서울대학교 컴퓨터공학과 학사
  서울대학교 컴퓨터공학과 석사
  서울대학교 컴퓨터공학과 박사
- 경력 : University of Callifornia, Irvine 방문교수
  University of Florida 방문 교수

<주관심분야 : 모바일 시스템>

※ 본 연구는 한성대학교 교내 학술 연구비를 지원받았음