http://dx.doi.org/10.7236/JIWIT.2012.12.5.83
JIWIT 2012-5-11

스마트폰을 이용한 소규모 실시간 라디오 방송 시스템

A Small Real-Time Radio Broadcasting System by Using Smart Phone

이재문*

Jae-Moon Lee

요 약 본 논문은 안드로이드 기반 스마트폰을 이용한 소규모 실시간 라디오 방송 시스템 설계 및 구현에 관한 연구이다. 서버-클라이언트 구조로 설계하였으며, 시스템을 간단히 하기 위하여 데이터 전송 방식으로 HTTP에 의한 점진적 다운로드 기법을 사용하였다. 실시간 방송을 실현하기 위하여 원음을 짧은 시간 간격으로 잘라서 캡쳐하여 압축/저장하였고, 이들을 순차적으로 클라이언트에서 재생하는 방법을 사용하였다. 이 방법은 서버에서 캡쳐 시 원음을 잃음과 클라이언트에서 재생 시 파일과 파일 사이 재생의 끊어지는 두 가지 문제를 발생 시킨다. 서버에서는 캡쳐 스레드와 압축/저장 스레드로 분리하고, 이중 버퍼링을 사용하여 해결하였으며, 클라이언트에서는 안드로이드에서 제공하는 미디어플레이어를 사용하고, 하나의 파일 큐에 다수의 음원 파일을 저장하여 해결하였다.

Abstract This paper is a research on the design and implementation of a small real-time radio broadcasting system by using smart phone based on Android. It was designed as the server-client structure, and used the progressive download of HTTP as methods of transferring data to further simplify the system. In order to realize the real-time broadcasting, the original audio source was divided with a short interval and captured to be compressed and stored into files. Then the client receives and plays the compressed files sequentially as it is downloaded. However, this method occurs two problems each of which is the loss of capturing the original source in the server and the discontinuity of playing the files in the client. We solved the problem in the server by separating the thread into two parallel threads of which is each captured and compressed/stored, also by using the double buffering method. The problem in the client was solved using MediaPlayer in Android and the file queue to store the multiple files.

Key Words : Radio broadcasting system, Progressive download, Smart Phone, Android, MediaPlayer

Ⅰ. 서 론

인터넷 방송 시스템은 방송뿐만 아니라 청취자의 방송 참여가 쉬운 관계로 인기가 높다. 이러한 관계로 최근 대부분의 공중파 라디오 방송들이 인터넷 라디오를 동시에 제공하고 있다[1, 2, 3, 4]. 인터넷 방송의 활성화[5, 6, 7, 8]과 달리, 소규모 유선 라디오 방송[9]은 급격히 쇠퇴하고 있다. 유선이라는 시설의 한계와 대체 미디어의 활성화로 그 존재 이유가 사라지고 있는 것이다. 이러한 대표적인 시스템이 사내 방송, 중고등학교, 대학 등의 교내 방송

*정회원, 한성대학교 멀티미디어공학과
접수일자 : 2012년 8월 22일, 수정완료 : 2012년 9월 28일
게재확정일자 : 2012년 10월 12일

Received: 22 August 2010 / Revised: 28 September 2012 / Accepted: 12 October 2012
*Corresponding Author: jmlee@hansung.ac.kr
Dept. of Multimedia Engineering, Hansung University, Korea

등이다. 본 논문에서 연구는 개인 방송 또는 교내 방송등과 같은 소규모 실시간 라디오 방송에 적합한 인터넷 라디오 방송 시스템 개발에 대한 연구이다. 소규모란 방송 시스템 구축 간단하여 유지 보수가 매우 쉬우며, 동시 청취자의 수가 제한적인 환경을 의미한다.

인터넷 라디오 방송은 일반적으로 데이터를 다운받으면서 재생하는 스트리밍 방식을 사용한다. 스트리밍 방식을 지원하는 대표적인 방법에는 RTSP(Real Time Streaming Protocol)[10]을 이용하는 방법과 HTTP(Hyper Text Transfer Protocol)[11]을 이용한 점진적 다운로드(Progressive Download) 방법[12]를 이용하는 방법이 있다. 전자는 연결 지향형 네트워크를 사용하여 네트워크의 성능이 일정 이상인 경우 실시간 방송을 보장할 뿐만 아니라, 사용자의 다양한 명령에 따라서 방송 서버가 능동적으로 동작한다. 이 방법의 단점으로는 소프트웨어가 복잡할 뿐만 아니라 네트워크가 불안정한 상태에서는 방송 품질을 보장하지 못한다. 후자는 전자의 단점을 보완하는 대안으로 HTTP 서버를 이용하여 미디어 파일이 다운되는 동안 미디어를 재생하는 기법이다. 이 방법의 장점은 소프트웨어가 간단하다는 것이다. 그러나 HTTP는 파일 단위의 다운로드하는 프로토콜[11, 12]이기 때문에 실시간 방송의 요건을 만족하기가 쉽지 않다.

본 논문에서 개발하는 실시간 라디오 방송 시스템은 소규모 방송 시스템을 전제로 HTTP를 이용한 점진적 다운로드 기법을 사용한다. 이 방법의 단점인 실시간 방송의 어려움을 극복하기 위하여 서버 측에서 실시간으로 방송되는 원음을 짧은 시간단위로 잘라서 저장하고, 클라이언트 측에서는 짧은 시간단위로 잘라진 파일을 수신하여 연속적으로 재생하는 것이다. 이 경우 발생하는 가장 큰 문제는 서버 측에서 실시간으로 방송되는 원음을 잘라서 파일로 저장할 때 파일과 파일 사이에 원음의 잃음이 없어야 하고, 클라이언트 측에서는 잘라진 파일들의 재생과 재생사이에 끊어짐이 나타나지 않고 자연스럽게 재생되어야 하는 것이다. 본 논문에서는 이 같은 문제를 해결하는 소규모 실시간 라디오 방송 시스템 설계 및 구현에 대한 연구이다. 서버로는 아파치 HTTP 2.0 웹서버[11]가 운영되는 개인용 PC를 사용하였으며, 클라이언트로는 최근 많이 사용되는 안드로이드[13] 기반 스마트폰을 사용하였다.

2장은 개발하는 시스템의 구조 및 설계 내용을 설명하며, 3장에서는 다양한 구현 내용 중 중요한 부분을 설명한다. 4장에서는 개발된 시스템의 성능을 분석하였으며, 5장에서 결론을 논한다.

II. 실시간 라디오 방송 시스템의 설계

1. 실시간 라디오 방송 시스템의 구조

본 논문에서의 실시간 라디오 방송 시스템은 전통적인 클라이언트-서버 구조로 설계되었다. 서버 시스템에서는 방송될 음원 데이터를 지속적으로 생성하며, 클라이언트 시스템에서는 서버에게 지속적으로 데이터 송신을 요청하며 수신된 데이터를 재생한다. 개발된 실시간 라디오 방송 시스템의 구조는 그림 1과 같다. 서버 시스템은 HTTP를 지원하는 웹서버와 음원 데이터를 생성하는 녹음 시스템으로 구성된다. 웹서버는 클라이언트 시스템으로부터 요청된 데이터 파일을 지속적으로 송신하는 기능을 하며 녹음 시스템은 실시간으로 입력되는 원음 데이터를 캡쳐하고 압축하여 파일에 저장하는 기능을 한다. 클라이언트 시스템은 안드로이드가 탑재된 스마트폰이다. 클라이언트 시스템은 지속적으로 서버 시스템에 저장된 원음 파일의 송신을 요청하며, 송신되는 파일을 수신하여 이를 재생하는 기능을 한다.

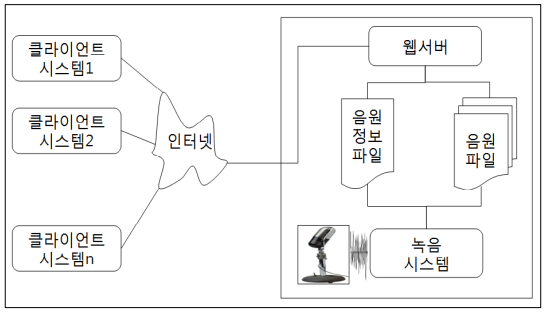

그림 1. 방송 시스템의 구조
Fig 1. Architecture of broadcasting system

2. 음원 파일 송수신 프로토콜

서버와 클라이언트 사이의 송수신 프로토콜로 HTTP를 사용하므로 실시간 방송 시스템으로 한계가 있다. 이러한 한계를 해결하는 하나의 방법은 녹음 시스템에서 생성되는 음원 파일을 짧은 시간 단위로 분할하는 것이다. 예를 들어 그림 1의 녹음 시스템에서 그림 2에서와 같이 음원을 t초 단위로 잘라서 음원 데이터를 캡쳐하여 음원

파일을 생성하고, 바로 클라이언트 시스템으로 전송한다면 클라이언트 시스템에서는 t초 단위로 구성된 음원 파일을 지속적으로 수신하여 재생할 수 있다. 이런 이상적인 환경에서 클라이언트 시스템은 실제 방송되는 음원보다 t초 후부터는 지속적으로 재생할 수 있으므로 t가 매우 작다면 실시간적 요건을 만족한다고 할 수 있다.

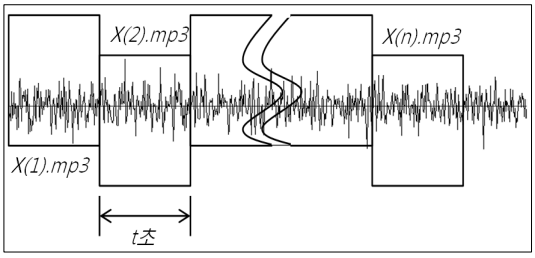

그림 2. 음원 데이터 캡쳐 방법
Fig 2. A capturing method of audio data

본 논문에서 개발한 시스템은 실시간 라디오 방송을 지원하기 위하여 그림 2와 같이 음원 데이터를 캡쳐하고 송신한다. t초 간격으로 음원 데이터를 잘라서 파일로 저장하고 이 파일을 클라이언트 측에 지속적으로 송신한다. 이러한 프로토콜에서 클라이언트와 서버의 통신을 줄이기 위하여 서버 측의 녹음 시스템은 그림 1에서와 같이 음원정보를 저장하는 하나의 파일과 일정한 시간 간격으로 저장되는 음원 파일을 유지한다. 음원정보 파일에는 클라이언트의 효율적인 재생을 위해 음원파일의 평균 크기, 타입 및 가장 최근에 생성된 음원 파일에 대한 음원파일 ID를 저장한다. 음원파일 ID는 녹음 시스템이 음원을 캡쳐하여 파일로 저장할 때 이 파일에 대한 ID이다. 녹음 시스템은 음원 데이터를 저장할 때 파일 이름을 순차적으로 정하여 저장한다. 예를 들어 현재의 저장 파일 이름이 $X(32).mp3$이면 그 다음에 저장되는 파일 이름을 $X(33).mp3$로 정하는 것이다. 이때 33이라는 ID를 음원정보 파일에 적어 두는데 이것이 음원 파일 ID이다. 클라이언트 시스템은 그림 3과 같이 서버 측으로부터 음원정보 파일을 가장 먼저 수신하여, 가장 최근의 음원 파일 ID를 추출함으로써 음원파일 이름을 생성할 수 있다. 이후 별도의 음원정보 파일 수신 없이 그림 3에서처럼 ID를 하나씩 증가함으로써 음원파일을 지속적으로 요청한다.

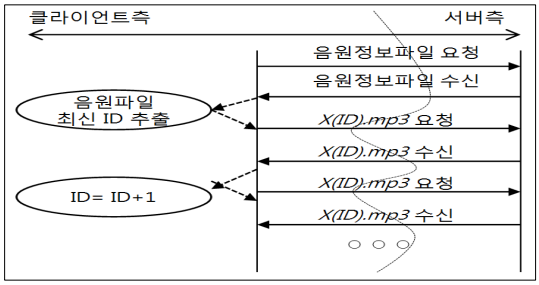

그림 3. 데이터 송수신 프로토콜
Fig 3. Protocol of transferring data

HTTP를 사용한 실시간 라디오 방송을 지원하기 위하여 그림 2와 같은 방법을 사용하는 경우 그림 4와 같은 두 가지 문제가 발생할 수 있다. 첫 번째 문제는 실시간으로 입력되는 음원 데이터를 캡쳐하여 저장하는 경우에 캡쳐와 저장 사이의 음원 데이터를 잃을 수 있다는 것이다. 이러한 현상은 그림 4-(a)에 원으로 표시된 부분과 같다. 두 번째 문제는 음원 파일들을 연속적으로 재생하는 경우 파일과 파일 사이의 오버헤드 때문에 오디오의 끊어짐 현상이 발생한다. 이러한 현상은 그림 4-(b)에 원으로 표시된 부분과 같다. 본 논문에서는 서버 시스템과 클라이언트 시스템의 설계 시 이러한 문제들을 해결하는데 초점을 맞추어 설계하였다.

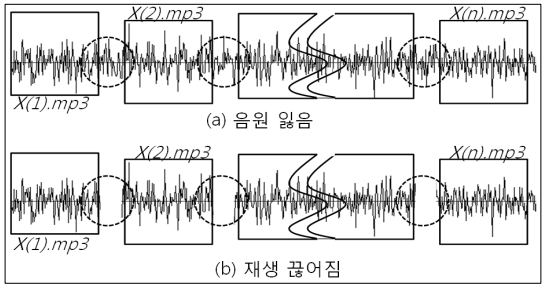

그림 4. 실시간 라디오 방송에서 문제점
Fig 4. Problems of real-time broadcasting system

3. 서버 시스템 설계

서버 시스템은 그림 1과 같이 크게 웹서버와 녹음 시스템으로 구성된다. 웹서버는 단순히 파일을 송신만하면 되므로 웹서버를 위한 어떠한 프로그램도 필요하지 않다(향후 간단한 채팅 등이 도입되는 경우 서버 측 CGI 프로그램이 필요할 것이다). 녹음 시스템은 일정한 시간 간격으로 음원 데이터를 캡쳐하고, 이 데이터를 압축하고, 최종적으로 파일에 저장한다. 또한 음원파일이 생성될 때

마다 음원정보 파일에 파일 ID를 갱신한다. 이러한 모든 작업은 실시간으로 이루어져야하고 그림 4-(a)에서 나타난 음원 데이터의 잃음 현상도 극복하여야 한다.

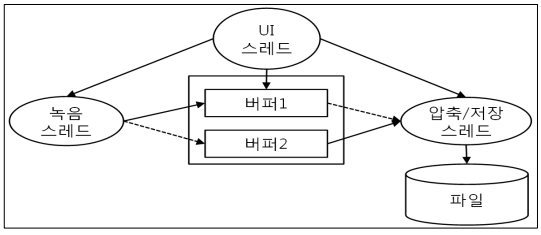

그림 5. 녹음 시스템의 구조
Fig 5. Architecture of capturing system

본 논문에서는 이러한 기능의 녹음 시스템을 그림 5와 같이 세 개의 스레드로 분리하여 설계하였다. UI 스레드는 사용자 인터페이스를 담당하며 캡쳐 스레드와 압축/저장 스레드를 제어한다. 캡쳐 스레드는 컴퓨터에 설치된 마이크, 라인인 등의 하드웨어로부터 지속적으로 입력되는 원음을 캡쳐하여 PCM 형태의 데이터를 생성한다. 생성된 PCM 데이터는 버퍼를 통하여 압축/저장 스레드에 전달된다. 압축/저장 스레드는 UI 스레드로부터 지정된 데이터 형태로 음원 데이터를 압축하여 저장한다. 저장할 때 저장 파일 이름은 순차적으로 증가되는 이름으로 정하여지며, 이 이름과 관련된 ID를 음원정보 파일에 저장한다.

4. 클라이언트 시스템 설계

클라이언트 시스템의 구조는 그림 6과 같다. UI 스레드는 사용자 인터페이스를 담당하며 사용자로부터 명령을 받아 재생 스레드와 수신 스레드를 제어한다. 수신 스레드는 서버 측으로 부터 그림 3의 프로토콜에 따라 음원정보 파일과 음원 파일을 수신하여 저장하는 기능을 한다. 재생 스레드와 수신 스레드 사이의 데이터 전달은 그림 6에서 보이는 음원큐 파일을 통해서이다. 이것은 파일로 구성된 일종의 큐이다. 재생 스레드는 안드로이드에서 제공하는 미디어플레이어[13]를 사용한다. 미디어플레이어를 사용한 음원 재생에서 발생하는 가장 큰 문제점은 다수의 음원 파일을 순차적으로 재생할 때 그림 4-(b)에서와 같이 각 파일과 파일 사이 재생 시 오디오의 끊어짐 현상이 나타난다는 것이다.

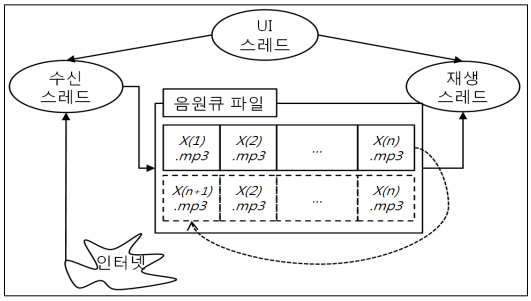

그림 6. 클라이언트 시스템의 구조
Fig 6. Architecture of client system

이러한 문제를 해결하기 위하여 본 논문에서는 그림 6에서와 같이 음원큐 파일을 하나의 파일로 구성하고 그 하나의 파일에 수신 스레드는 지속적으로 수신된 음원 파일을 순차적으로 저장하기만 하고, 재생 스레드는 지속적으로 음원큐 파일로부터 데이터를 읽어 들여 이들을 재생하는 것이다. 이 경우 이러한 끊어짐 현상을 최소화할 수 있고 실제로 구현된 시스템을 통한 실험에서는 끊어짐 현상을 발견할 수 없었다. 이 방법에 나타나는 새로운 문제는 음원큐 파일이 지속적으로 성장한다는 것이다. 이를 제한하기 위하여 그림 6에서와 같이 사전에 미리 n 개의 음원 파일 이후에는 다시 파일의 맨 처음으로 이동하여 저장하도록 하였다. 이 경우에도 $X(n).mp3$를 재생한 후 $X(n+1).mp3$를 재생할 때 오디오의 작은 끊어짐 현상은 피할 수 없다.

III. 실시간 라디오 방송 시스템의 구현

1. 서버 시스템 구현

서버의 녹음 시스템에서 캡쳐 스레드는 윈도우즈에서 제공하는 winmm 라이브러리[14]를 사용하여 구현하였다. winmm 시스템에서 오디오 캡쳐 구조는 실시간으로 입력되는 음원을 캡쳐하여 지정된 버퍼에 저장하고 버퍼에 데이터가 모두 채워지면 콜백 함수를 호출한다. 콜백 함수를 실행 중에도 음원의 캡쳐는 지속되는데 이때 데이터의 잃음을 방지하기 위하여 그림 7에서와 같이 다수의 버퍼를 등록하였다.

콜백 함수 *CallBackWhenFullBuffer*를 통하여 UI 스레드에게 캡쳐된 데이터는 보내지고, UI 스레드는 이 데이터를 버퍼에 데이터를 저장한다. 사용자가 지정한 시간만큼 데이터가 버퍼에 쌓이면 압축/저장 스레드에게

데이터를 저장하도록 한다. UI 스레드는 압축/저장 스레드에게 데이터 저장을 명령함과 동시에 그림 5와 같이 이중 버퍼링을 위하여 새로운 버퍼를 캡쳐 스레드를 위하여 준비한다. 압축/저장 스레드는 버퍼에 저장된 데이터를 압축하고 이를 파일에 저장한다. 또한 파일 이름에 대한 정보를 음원정보 파일에 갱신한다. 압축/저장 스레드에서 압축은 wav와 $mp3$을 제공하도록 구현하였다. $mp3$ 압축을 위하여 공개용 소프트웨어인 *bladeenc.dll*[15]을 사용하였다.

```
void doStart(){
    ...
    waveInOpen(...) // 초기화 및 콜백함수 등록

    // 데이터의 잃음을 방지를 위한 다수의 버퍼 등록
    for(int i=0;i<maxBuffer;i++){
        waveInPrepareHeader(...);
        waveInAddBuffer(...);
    }
    waveInStart();
}

CALLBACK CallBackWhenFullBuffer(){ // 콜백 함수
    waveInUnPrepareHeader(...)
    PostMessage(...); // UI 스레드에 전달

    // 사용된 버퍼 재등록
    waveInPrepareHeader(...);
    waveInAddBuffer(...);
}
```

그림 7. 캡쳐 스레드 동작 의사코드
Fig 7. Pseudo code for operation of capturing thread

2. 클라이언트 시스템 구현

클라이언트 시스템의 수신 스레드는 안드로이드 SDK에서 제공하는 *HttpURLConnection*, *InputStream* 클래스[13]를 이용하여 구현하였다. 재생 스레드는 안드로이드에서 제공하는 미디어플레이어 객체를 사용하였다. 수신 스레드와 재생 스레드는 그림 8과 같이 동작한다. 안드로이드의 *RandomAccessFile*로 만들어진 음원큐 파일에 대하여 수신 스레드는 재생 스레드에 앞서 수신된 데이터를 음원 파일에 저장하게 된다.

그림 8과 같은 구현에서 중요한 문제는 네트워크의 속도가 느려서 수신 스레드의 저장 속도가 재생 스레드의 재생 속도보다 느린 경우이다. 이러한 경우 그림 8에서 표시된 r이 점차적으로 줄어들게 되고 최악의 경우 수신 스레드보다 재생 스레드가 앞서 가는 경우가 발생할 수도 있다. 따라서 이 경우 r의 값이 일정 이상 작아지면 재생을 중지해야 하는 제어가 필요하다.

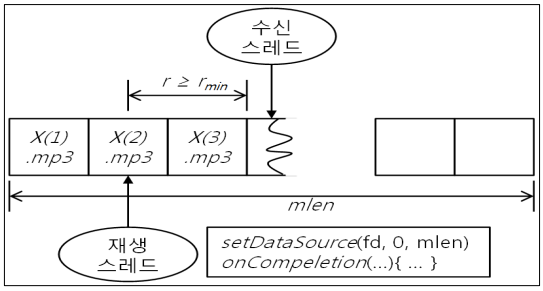

그림 8. 재생 스레드 및 수신 스레드의 동작
Fig 8. Operation of playing and receiving thread

```
void doWork(RandomAccessFile rf, int mlen){
    ...
    player= new MediaPlayer(); // 객체 생성

    player.setOnCompletionListener(...); // 콜백함수 등록
    player.setDataSource(rf.getFD(), 0, mlen);
    player.prepare();
    player.start();          // 재생시작

    while(true){  // 미디어플레이어 제어
        Sleep(100);
        if(GetRemainingTime() < r_min){
            player.pause();
        }else if(GetRemainingTime() > r_min){
            if(player.isPlaying())    player.start();
        }
    }
}
// 콜백 함수
void     onCompletion(MediaPlayer player){
    player.seekTo(0);      // 파일의 처음으로 이동
    player.start();        // 재시작
}
```

그림 9. 재생 스레드 동작 의사코드
Fig 9. Pseudo code for operation of playing thread

미디어플레이어를 제어하는 재생 스레드의 동작은 그림 9에 나타나 있다. 먼저 미디어플레이어 객체를 생성하고 콜백 함수 *onCompletion*을 등록한다. *setDataSource* (fd, 0, mlen)을 설정함으로써 음원큐 파일을 모두 재생할 때 *onCompletion* 함수가 호출 되도록 연결한다. *onCompletion* 함수에서는 *seekTo*(0) 함수를 호출하여

처음부터 다시 재생하도록 함으로써 $X(n+1).mp3$를 재생하도록 한다. 미디어플레이어가 재생하는 동안 재생 스레드는 매 100msec 마다 그림 8에서의 r을 계산하고 이것이 $r_{\min}$보다 작은 경우 미디어플레이어를 중지시킨다. 지속적으로 이러한 체크를 실행하면서 재생 스레드가 충분히 데이터를 수신($r \geq r_{\min}$)하여 저장한 경우에는 다시 미디어플레이어가 재생을 하도록 명령한다. 그림 9에서 *GetRemainingTime* 함수는 r을 계산하는 함수로 미디어플레이어의 *GetCurrentPosition* 함수를 이용하여 현재까지 재생한 시간을 계산하고 수신 스레드가 읽어들어 $X(i).mp3$의 개수를 계산하여 현재까지 수신된 음원 데이터의 시간을 계산하여 차이를 계산한다.

그림 10은 개발된 녹음 시스템과 클라이언트 시스템의 화면이다. 녹음 시스템은 샘플링율로 32K 및 44.1K을 선택할 수 있도록 하였으며, 음원 파일의 크기로 2초, 5초 및 10초를 선택할 수 있도록 하였다. 또한 저장 파일 포맷으로 $mp3$와 wav을 선택할 수 있도록 하였다. 클라이언트 시스템은 방송/중지, 볼륨 조정 등 일반적인 라디오 기능을 제공도록 구현하였다.

그림 10. 구현된 녹음 시스템 및 클라이언트 시스템
Fig 10. Implemented capturing and client system

IV. 성능 분석

본 논문에서 설계 및 구현한 실시간 라디오 방송 시스템은 실시간성을 얼마나 만족하느냐가 중요하다. 이러한 실시간성은 다음 세 가지 기본적인 환경이 제공되어야 한다. 첫째 녹음 시스템은 실시간으로 캡쳐하고 압축/저장할 수 있어야 한다. 두 번째로 스마트폰에서 음원 파일을 실시간으로 재생할 수 있어야 한다. 마지막으로 네트워크의 대역폭이 음원 파일($mp3$)을 실시간으로 송수신하기에 충분하여야 하는 것이다. 첫 번째 조건은 그림 4-(a)의 문제 해결 조건이다. 최근 PC, 스마트폰의 성능 향상, 3~4G의 무선 네트워크 속도를 고려하면 상기 조건은 충분히 만족된다.

상기와 같은 기본적인 조건 하에서도 본 논문에서 개발한 시스템의 실시간적 요건으로 방송의 지연이 일정하여야 한다. 그림 2에서 각 $X(i).mp3$의 캡쳐 시간이 t초라고 하고, 이를 압축/저장 스레드에 의하여 압축/저장하는 시간을 α초라 하자. 또한 이 파일을 클라이언트 시스템에 전송되는 시간을 β초라 하자.

정리1. $\alpha \leq t$이고 $\beta \leq t$이면 개발된 방송 시스템은 지연 시간이 일정할 수 있다.

증명: 대부분의 현대 컴퓨터 시스템은 캡쳐, 압축 및 데이터 통신은 서로 다른 데이터에 대해서는 동시에 실행할 수 있다. 캡쳐 스레드가 $X(i).mp3$을 캡쳐하고 있다고 하자. 동일한 시간에 압축/저장 스레드는 $X(i-1).mp3$을 압축/저장할 것이다. 마찬가지로 이 시간 $X(i-2).mp3$을 포함하여 이전에 압축/저장된 파일은 전송될 수 있다. 즉, $X(i).mp3$의 캡쳐, $X(i-1).mp3$의 압축/저장 및 $X(i-k).mp3$의 송신은 동시에 수행될 수 있다. 여기서 $k \geq 2$ 이다. 따라서 이 3가지 연산중 가장 긴 시간이 캡쳐 시간 t이라면 3가지 연산은 특별한 지연 없이 반복적으로 실행될 수 있다.

정리1에서 $\alpha > t$이면 캡쳐되는 데이터가 버퍼에 축척되게 되므로 그림 4-(a)처럼 원음의 잃음이 발생할 것이고, $\beta > t$이면 저장된 파일이 클라이언트에 제때 제공되지 못하므로 그림 4-(b)에서와 같이 재생의 끊어짐이 발생할 것이다.

정리2. $\alpha \leq t$이고 $\beta \leq t$이면 개발된 방송 시스템은 적어도 $3t$만큼 지연시간을 유지하면서 방송할 수 있다.

증명: 일관성의 잃음 없이 α=t이고 β=t라고 하자. 그리고 $X(i).mp3$의 첫 음이 캡쳐된 시간을 t_1이라

고 하자. 이 경우 $t_1 + t + \alpha + \beta \leq t_1 + 3t$이므로 첫 음은 $3t$초 후에 방송될 수 있다.

정리2는 음원파일이 수신되자마자 재생된다는 가정이다. 그러나 개발된 방송시스템은 그림 8과 같이 수신 스레드와 재생 스레드 간 음원 파일을 액세스할 때 일정 시간 간격(r)이 필요하다. $\alpha \leq t$이고 $\beta \leq t$가 충분히 만족된다면 그림 8에서 $r_{\min}$은 t이면 충분하다. 그러나 모바일 환경에서 평균적인 대역폭은 $\beta \ll t$를 만족할지라도 장소에 따라서는 일시적으로 네트워크 속도가 크게 저하되는 경우가 흔히 발생한다. 이 경우 $r_{\min}$이 작은 값을 가지는 경우 재생의 잦은 끊김 현상이 발생할 것이다. 이러한 현상을 피하기 위하여 $r_{\min}$을 좀 더 크게 할 필요가 있다.

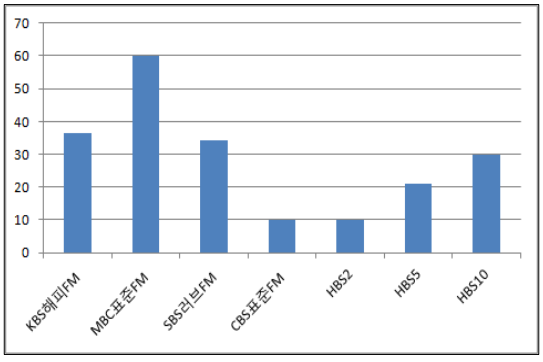

그림 11. 다양한 시스템의 지연시간 비교
Fig 11. Comparison of delay time of several systems

그림 11은 공중파 라디오 방송에서 제공하는 인터넷 방송의 지연시간과 개발된 시스템의 지연 시간을 측정하여 비교한 것이다. *HBS2*, *HBS5* 및 *HBS10*은 개발 시스템에서 음원큐 파일의 크기는 100초로 고정한 상태에서 그림 2에서 t를 각각 2초, 5초, 10초로 설정한 경우이다. 따라서 *HBS2*가 *HBS5*, *HBS10*보다 지연 시간이 적은 것은 정리2에 따라 당연하다. 그러나 지연 시간이 정리2의 $3t$보다 크게 나타나는 이유는 그림 8에서의 $r_{\min}$을 적어도 6초 이상 되도록 설정하였기 때문이다. $r_{\min}$이 작은 경우 잦은 재생 중단이 발생하기 때문이다. *CBS*가 *HBS2*와 거의 비슷한 지연을 나타낸다. *KBS*, *MBC*, *SBS*와는 큰 차이가 있다.

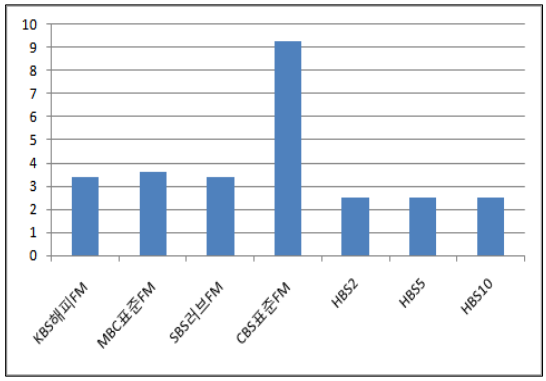

그림 12. 첫 음이 재생되는 시간
Fig 12. Time to play the first sound

그림 12는 방송을 요청한 후 첫 음이 나오기까지의 시간을 측정하여 비교한 것이다. *KBS*, *MBC*, *SBS*는 약 3초 정도의 지연이 있었으며, *CBS*는 약 9초 정도의 지연이 있었다. 반면 본 논문에서 개발한 시스템은 거의 3초 이내에 첫 음이 재생되었는데 이것은 $r_{\min}$만큼 이전 파일부터 읽어왔기 때문이다. 예를 들어, $r_{\min}$이 6초이고 t가 2초인 경우 그림 3에서 읽어온 ID_x에 대하여 ID_y=ID_x-3(6 ÷ 2)로 계산하여 ID_y에 해당하는 파일부터 송신 요청하기 때문이다. 이것은 본 시스템은 최신의 파일(ID_x에 해당하는 파일)을 수신하여도 6초까지의 데이터가 수신될 때까지 기다린 후 재생하기 때문에 6초 동안 기다릴 필요 없이 6초전의 데이터를 바로 수신하여 재생하는 것이다. 이 경우 3β 시간이면 재생이 시작될 수 있는데, 대부분의 환경에서 $\beta \ll t$이기 때문에 그림 12에서와 같이 3초 이내에 첫 음을 방송하는 결과를 얻을 수 있었다.

V. 결론

본 논문에서 개발하는 실시간 라디오 방송 시스템은 소규모 방송 시스템을 전제로 HTTP를 이용한 점진적 다운로드 기법을 사용하였다. 이 방법의 단점인 실시간 방송을 극복하기 위하여 서버 측에서 실시간으로 방송되는 원음을 짧은 시간단위로 잘라서 저장하고, 클라이언트 측에서는 짧은 시간단위로 잘라진 파일을 지속적으로 수신하여 연속적으로 재생하는 것이다. 이 경우 발생하는 가장 큰 문제는 서버 측에서 실시간으로 방송되는 원

음을 잘라서 파일로 저장할 때 파일과 파일 사이에 원음의 잃음이 없어야 하고, 클라이언트 측에서는 잘라진 파일의 재생과 재생사이에 끊어짐이 나타나지 않고 자연스럽게 재생되어야 하는 것이다. 이러한 문제들을 서버에서는 캡쳐 스레드와 압축/저장 스레드로 분리하고, 이중 버퍼링을 사용하여 해결하였으며, 클라이언트에서는 하나의 파일큐에 다수의 음원 파일을 저장하여 해결하였다.

개발된 시스템은 기존의 공중파 라디오 방송에서 제공하는 인터넷 라디오 방송과 비교할 때 지연 시간이나 첫 음의 방송 시간 면에서 우수함을 실험적으로 알 수 있었다.

참 고 문 헌

[1] http://www.kbs.co.kr/radio/kong/

[2] http://www.imbc.com/broad/radio/minimbc/setup_pc/index.html

[3] http://gorealra.sbs.co.kr/g3/web/introduce.jsp

[4] http://rainbow.cbs.co.kr/Rainbow/

[5] Jongdeok Kim, Younggil Kim, "A Study of Internet Radio Platform based on MIPS Core", Journal of The Korea Institute of Information and Communication Engineering Vol. 12, No. 8, 2008, pp.1370-1376

[6] Jongtaek Oh, A Study on the Data Frame Length and Repetitive Transmission for IP Local Broadcasting, Journal of the Institute of Webcasting, Internet and Telecommunication, Vol. 11, No. 3, 2011, pp. 123-127.

[7] In-Lyong Ko, "Network implementation for convergences of mobile communication and broadcasting", Journal of Korean Institute of Information Technology, Vol.2 No.2 2004, pp. 121-126.

[8] Min-Young Sung, "Design and Implementation of Dual-Mode Cordless Phone and walkie-Talky System: A Software Radio Approach", Journal of the Korea Academia-Industrial cooperation Society, v.9, no.3, 2008, pp.674-680.

[9] Jin-Young Lee, "Development of an integrated amplifier for MATV/Satellite Broadcasting", Journal of the Korea Academia-Industrial cooperation Society, v.11, no.12, 2010, pp.5007-5014.

[10] H. Schulzrinne, A. Rao, R. Lanphier: Real Time Streaming Protocol (RTSP), http://ds.internic.net/internet-drafts/ draft-ietf-mmusic-rtsp-03.txt

[11] http://httpd.apache.org/

[12] Zeki Yetgin, Gamze Seckin, "Progressive Download for 3G Wireless Multicasting", International Journal of Hybrid Information Technology, Vol. 1, No. 2, April, 2008

[13] http://developer.android.com

[14] http://msdn.microsoft.com

[15] http://www.free-codecs.com/download/bladeenc.htm

저자 소개

이 재 문(정회원)

- 1994년 2월 : 한성대학교 멀티미디어 공학과 교수
- 1992년 4월 ~ 1994년 2월 : 한국전기통신공사 선임연구원
- 1988년 2월 ~ 1992년 2월 : 한국과학기술원 박사

<관심분야 : 게임프로그래밍, 게임인공지능, 기계학습, 데이터베이스>

※ 이 논문은 한성대학교 교내연구비에 의하여 지원되었음.