[2013-2 Project 1]

HEVC를 지원하는 스트리밍 서버 개발

E조

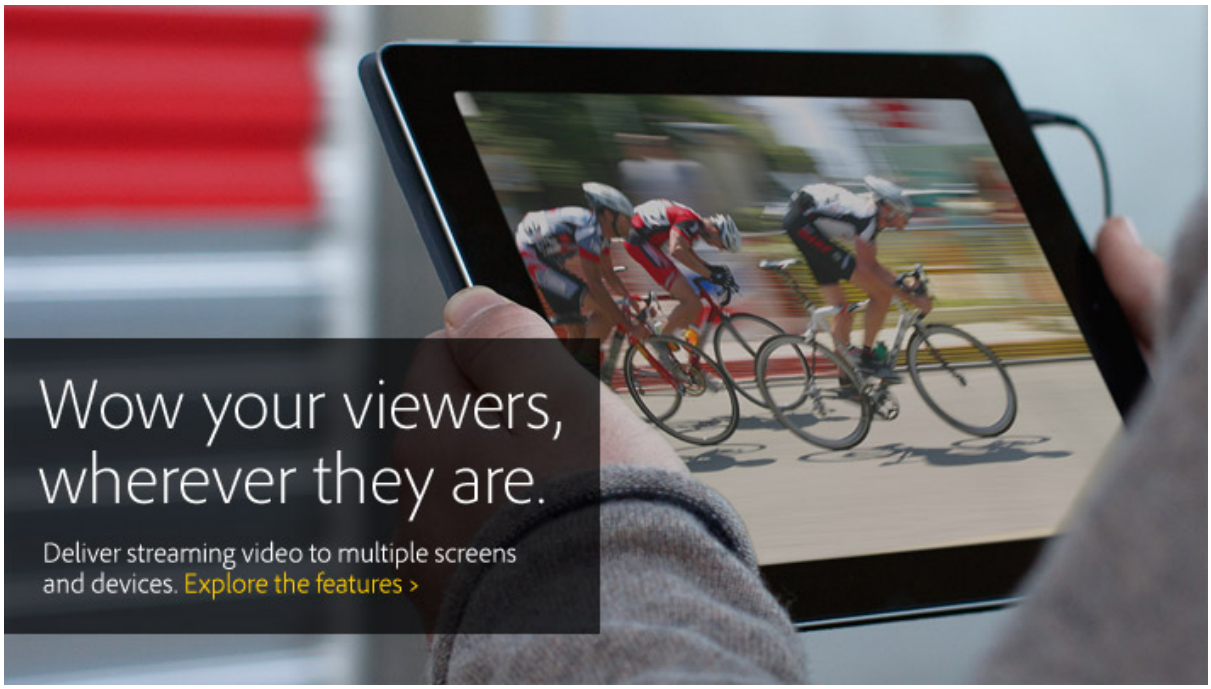
김원용　박종찬　정창주

엄현상 교수님

황인수 팀장님(캐스트이즈 미디어솔루션개발 2팀)

Table Contents

1. Abstraction



[그림 1]. Adobe Media Server Family

위의 그림은 Adobe사가 제공하는 Media Server Family라는 미디어 스트리밍 서비스의 한 광고 그림이다. Adobe사 뿐만 아니라 우리가 익히 들어 알고 있는 YouTube를 비롯하여 요즘은 미디어 스트리밍이라는 용어는 컴퓨터 공학도라면 몰라서는 안되는 용어가 되어 있다. 미디어 스트리밍 서비스는 다음과 같은 이유로 개발되기 시작하였다. 동영상 재생 기술은 하드웨어와 소프트웨어 산업이 발전함에 따라 함께 급속도로 발전하게 되었고 사람들은 동일한 환경에서 더 높은 화질의 동영상을 감상하게 될 수 있었다. 하지만 더 높은 화질의 수요는 현재 네트워크 망의 용량을 넘어서기 시작하자, 컴퓨터나 단말기 등에 동영상을 직접 다운받아서 재생하는 것이 아닌 직접 감상할 수 있는 서비스에 대한 수요가 증가하게 된다. 이를 ***스트리밍 서비스(Streaming Service)***라고 하는데 인터넷에서 사운드나 비디오 파일을 실시간으로 전송, 재생할 수 있게 하는 서비스를 말한다. 특히 사운드나 비디오 파일을 하나가 아닌 여러 개의 패킷(Packet)스트림으로 나누어 물이 흐르는 것처럼(Streaming) 전송하는 것을 뜻한다.

스트리밍 서비스에서는 더 높은 화질의 동영상을 동일한 환경의 네트워크에서 전송하기 위해 동영상 압축 방법에 대한 관심이 증가하게 된다. 기존에 존재하던 MPEG-1, MPEG-2 등으로는 이러한 수요를 감당하기에 벅찬 감이 있었고, 이러한 배경에서 등장한 것이 H.264이다. H.264는 MPEG-4의 파트10에 해당하는 것으로서 기존보다 우수한 압축률을 가질 뿐 아니라 네트워크 상에서 전달되기에 적합한 파일 구조로 설계되어 있다. 2003년 H.264가 표준으로 등장한 이후 TV 화질도 계속해서 발전하여

결국 HD 화질을 넘어서는 UHD(울트라 HD), 4K 등의 초고화질 동영상이 등장함에 따라 좀 더 높은 수준의 압축률을 보이는 파일 규약이 필요하게 되었고, 이에 따라 *HEVC(H.265)*가 등장하게 된다.

2. Introduction

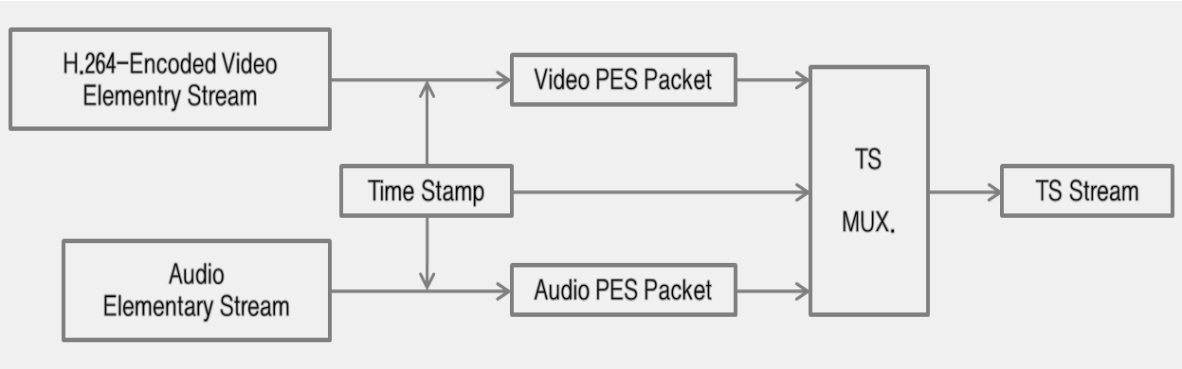
우리 E조는 이러한 H.264와 HEVC를 지원하는 ***다중 접속 스트리밍 서버***를 구현해 보도록 한다. 위에서 언급한 H.264와 HEVC는 영상(Video) 압축 표준으로서. 우리가 실제로 보는 동영상은 실제로 이런 Video Stream과 Audio Stream이 합쳐져 있는 Elementary Stream(MPEG2-ES)의 형태이고, 네트워크 망에로의 전송을 위한 Transport Sream(MPEG2-TS)의 형태가 있다. 본 프로젝트에서는 1시간 이상의 Transport Stream 파일(TS 파일)을 스트리밍 방식으로 전송하는 서버를 구현하는 것을 목적으로 한다. 최대 10개의 클라이언트에게 동시에 전송이 가능한 다중 접속 서버이어야 하고 클라이언트는 RTSP 프로토콜을 통해 서버에게 요청을 보내게 된다. 동영상을 재생할 수 있는 것뿐만 아니라 클라이언트가 동영상의 임의 지점을 접근할 수 있도록(Random Access) 한다.

3. Background Study

A. Related Technique

A.1. Transport Stream

*Transport Stream*이란 Video Elementary Stream과 Audio Elementary Stream들이 동기화를 위한 *Time stamp*가 찍혀 있는 가변적인 *PES(Packetized Elementary Stream)*이 *188 byte*라는 고정된 길이로 된 형태의 Stream이다.



[그림 2]. Transport Stream(TS) 의 구조

우리는 이러한 Transport Stream인 .ts 파일을 인풋으로서 받게 된다. 따라서 임의

접근까지 고려한다면 우리는 반드시 RTSP로 받은 영상 플레이 시점에 해당하는 Time Stamp를 Transport Stream에서 찾아야 할 것이다.

A.2. H.264

그러나 우리가 재생하고자 하는 지점에서 Video Elementary Stream과 Audio Elementary Stream을 재생하게 되면 올바르지 않은 값들이 출력하게 된다. 왜냐면 H.264부터는 압축률을 대량으로 높이기 위해 영상을 구성하는 모든 사진을 각각 압축하는 것이 아니라 한 장의 사진을 독립적으로 Decoding 할 수 있도록 압축한 뒤 그 뒤의 사진들은 오차값 또는 예상값(Predicted)을 Encode하기 때문이다. 여기서 독립적으로 Decode 가능한 Picture를 Key Picture라고 하고, 이러한 Key Picture는 MPEG 계열에서는 *I-frame*, 특히 H.264에서는 *IDR frame*이라고 일컫는다. 또한 예상값, 또는 오차값만을 Encode 했기 때문에 다른 Frame에의 참조를 통해서만 Decode할 수 있는 Frame에는 *B Frame*과 *P Frame*이 있다. 즉 우리가 재생하고자 하는 특정 지점을 찾았다고 하더라도 그 위치의 Frame이 그 Frame만으로 Decode할 수 없는 B Frame 또는 P Frame이라면 재생이 불가능하게 된다. H.264의 경우 임의 재생을 위해 IDR Frame이 0.5 - 2초 단위로 삽입되어 있기 때문에, 이 경우 앞에 있는 IDR Frame으로 이동해 Decode를 시작하는 것이 옳은 접근이 된다.

A.3. H.265(HEVC)

H.264에서 HEVC(H.265)로 발전하면서 압축률에 큰 증가가 있었다. 그러한 압축률의 증가를 위해 파일 구조도 많은 변화가 있었는데, 이 전까지 하나 뿐이던 Key Picture(IDR Picture)가 3개로 증가하고(이를 *RAP - Random Access Picture* 라고 한다) Non-RAP가 5개로 늘어났다. 이는 IDR Picture의 Coding Efficiency 때문이다. IDR Picture는 Key Picture로서의 기능은 충실히 수행하지만 IDR Picture에 접근하기 전에는 항상 DPB(Decoding Picture Buffer)를 비우기 때문에 일련의 Picture들이 순차적으로 Encode되지 않았을 경우 여러 번 돌아가는 과정이 필요하기 때문이다. 이렇듯 IDR Picture의 사용은 약 6%의 Coding Loss를 야기한다고 한다. 새로 만들어진 RAP중의 하나인 CRA(Clean Random Access) Picture는 이러한 Decoding의 비효율성을 극복하기 위해 생겨났다. IDR Picture보다 늦게 Display해야 할 B, P Picture들이 IDR보다 앞에 Encoding이 되어있는 경우, 이러한 IDR Picture 대신에 CRA Picture를 사용하여 앞의 B, P Picture들이 DPB를 비우지 않고 Decoding 하는 것을 허락하여 효율적인 재생이 가능하도록 한다. 이 경우 Picture들 간의 재생 순서가 H.264에 비해 복잡해졌기 때문에, 그에 맞추어 알고리즘도 수정해야 한다.

A.4. Streaming Server

Media Streaming Server

본 프로젝트에서 구현할 스트리밍 서버는 클라이언트에서 *RTSP(Real-Time Streaming Protocol)* 프로토콜로 요청을 받으면 RTP 패킷을 서버에서 클라이언트로 전송한다. *RTP(Real-time Transport Protocol)*는 음성이나 동영상 등의 데이터 스트림을 실시간을 전달할 때 사용하는 데이터 전송 프로토콜이다. 클라이언트는 RTP 패킷에 담긴 시간정보를 분석하여 데이터를 재생한다. 하지만 RTP는 기본적으로 UDP 전송 계층 프로토콜을 사용하는데, 이는 네트워크 전송 시 패킷 손실과 지연이 발생할 수 있음을 의미한다. 따라서 자원 예약 등을 위한 별도의 프로토콜과 병행해야 하는데 이 때 *RTCP(RTP Control Protocol)*이 함께 사용된다.

따라서 서버는 RTCP 프로토콜을 통해서 클라이언트에게 보내는 RTP의 양과 정보를 조정해야 하며 이러한 이슈들을 처리할 수 있도록 스트리밍 서버를 구현하여야 한다.

B. Development Environment

RTSP 서버는 리눅스 환경에서 개발하도록 하며, 본 프로젝트에서는 리눅스 배포판 중 서버용 OS로 적합한 CentOS 6.4 환경에서 개발을 진행하였다. C 언어로 구현하였으며, GIT 를 통하여 프로젝트 형상 관리를 함으로써 팀원 간의 협업이 순조롭게 이루어질 수 있도록 개발 환경을 구축하였다. 클라이언트는 vlc player를 이용한 user로 가정하였다.

4. Goal

본 프로젝트에서 구현할 주요 내용은 다음과 같다.

- *RTSP 프로토콜을 이용한 스트리밍 전송*
- *최대 10개의 클라이언트에 동시에 전송 가능*
- *영상 스트림이 H.264 또는 HEVC로 압축된 1시간 이상의 TS 파일 입력*
- *일반적인 재생(Play) 뿐 아니라 특정한 시간에 대한 재생(Seek) 가능*

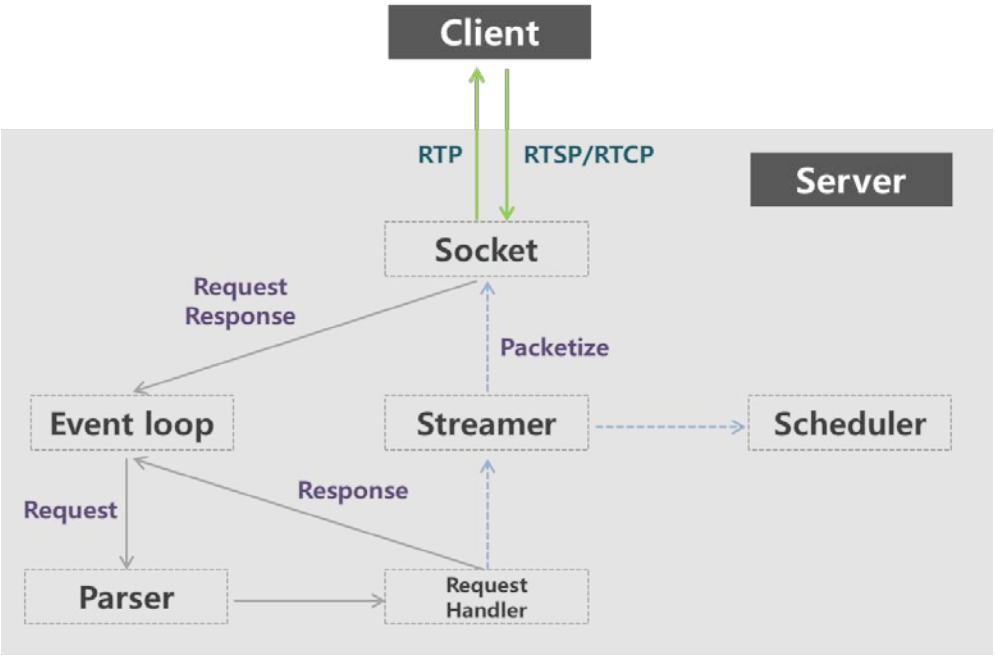
5. Approach

먼저 본 프로젝트의 목표를 달성하기 위하여, 첫 2주는 우리가 필요한 것들이 무엇인지 알아야 했고, 개발환경 또한 모든 조원과 동일하게 할 필요가 있었고, 정해진 기간

이 약 3개월인만큼 구체적인 일정표 역시 필요하였다. 개발기간 중 정기적으로 회사로 방문하여 담당자와의 피드백도 역시 필요하였다. 2주간의 결과로 1주 단위로 3개월어치의 목표량을 설정하고 각 조원에게 목표량을 배분함에 따라 일정을 진행하였다. 첫 1달은 각 조원이 RTP/RTCP/RTSP의 프로토콜, H.264 / HEVC의 구조, 실제 구현된 미디어 스트리밍 서버의 구조와 동작원리 등에 대한 공부를 각자 맡아서 공부해서 배경지식을 갖추는 것부터 하여 개발준비를 한다. 그리고 틈틈이 회사 담당자 분과 교수님간의 면담을 통해 부족한 것을 파악하고 도움을 받아 개발을 진행한다.

6. Project Architecture

A. Architecture Diagram



[그림 3] Architecture Diagram

B. Architecture Description

서버는 RTSP 요청을 받을 수 있는 RTSP 소켓 하나를 생성하여 클라이언트로부터 연결을 기다린다. 연결이 수락되면 서버는 즉시 클라이언트와 송신할 수 있는 RTSP 소켓 하나를 생성하여, Event loop을 통해서 RTSP 명령을 파싱하고, 스트리밍하는 역할을 한다. RTSP 서버 소켓은 다시 다른 클라이언트의 연결 요청을 기다리게 된다.

Event loop은 RTSP 명령어를 분석하여 클라이언트가 요청하는 동영상 파일, 재생하고 싶은 시각(재생이면 0일 것이고, 임의부분에의 재생이면 임의의 시간대가 될 것이

다.)을 얻은 후 Streamer를 통해서 해당하는 동영상 파일을 바이트 스트림으로 읽고 해당 시각으로 접근한다. 동영상은 네트워크 망을 통하여 전송하기 때문에 Elementary Stream으로 된 동영상을 Transport Stream으로 전환하여야 할 것이지만, 우리는 TS파일을 가지고 있다고 가정하였기에 우리는 TS파일을 RTP를 이용하여 전송하게 될 것이다. 하지만 단지 전송만 하는 것이 중요한 것이 아니라 일정한 ***전송속도***를 유지하는 것 또한 중요하다. 이를 위해 우리는 인코딩을 *CBR*이라고 가정하여 인코딩율에 맞춘 전송소도로 TS파일을 전송한다.

7. Implementation Spec

A. Input / Output Interface

[그림 4] Input & Output Description

B. Modules

B.1. Socket

먼저 클라이언트와 기본적인 소켓 통신이 가능하도록 TCP/IP 소켓과 관련된 모듈을 구현한다. 클라이언트에서 서버로 보내는 RTSP, RTCP 프로토콜 패킷은 TCP 소켓을 통해 전송받도록 하고, 서버에서 클라이언트로 보내는 RTP 패킷은 UDP 패킷을 통해 전송하도록 하였다.

B.2. Eventloop, Parser, Request Handler

소켓에서 읽은 RTSP 프로토콜은 문자열 형식으로 소켓으로부터 전송 받는데 이를 서버가 적절하게 처리할 수 있도록 ***파싱하는 작업을 하게 된다.*** 어떠한 RTSP 요청(Directive)인지에 따라서 파싱해야 될 정보가 다른데, 먼저 어떠한 요청인지를 파악한 후 이에 따른 필요 정보를 적절하게 파싱하도록 한다.

```
void parse_rtsp()
void handle_option()
void handle_describe()
void handle_setup()
void handle_play(){
void handle_teardown(){
void handle_pause()
```

[그림 5] Parser 구현 부분

이 때, 실질적인 파싱 과정이 일어나는 parse_rtsp( ) 함수에서는 RTSP 패킷에 함께 담기는 명령어, 호스트, 파일명에 대한 정보를 파싱한다.

B.3. Streamer, Scheduler

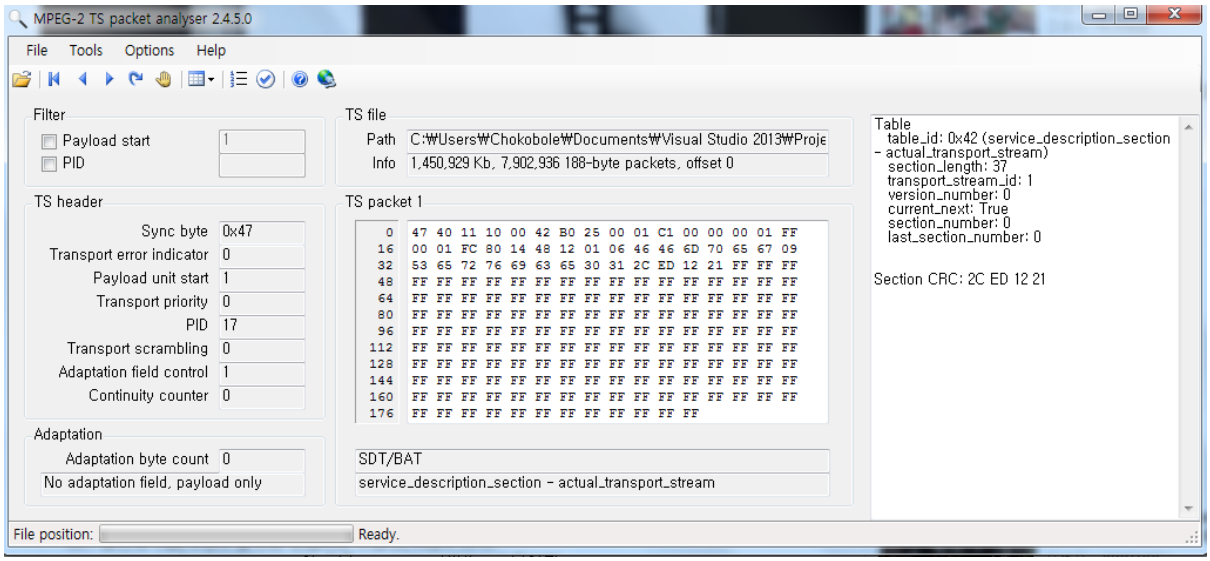
Streamer에서는 실질적으로 영상 데이터 정보가 담긴 ***RTP 패킷을 만드는 실질적인 역할을 한다.*** 먼저 RTP 헤더를 만든 뒤에 헤더와 동영상 데이터 부분을 함께 붙여서 보낸다. RTP 헤더는 다음과 같다.

```
typedef struct RTPPacketHeader{
          /* byte 0 */
        uint8_t csrc_len:4;
        uint8_t extension:1;
        uint8_t padding:1;
        uint8_t version:2;
          /* byte 1 */
        uint8_t payload:7;
        uint8_t marker:1;
          /* bytes 2, 3 */
        uint16_t seq_no;
          /* bytes 4-7 */
        uint32_t timestamp;
          /* bytes 8-11 */
```

```
        uint32_t ssrc;
} RTP_HDR;
```

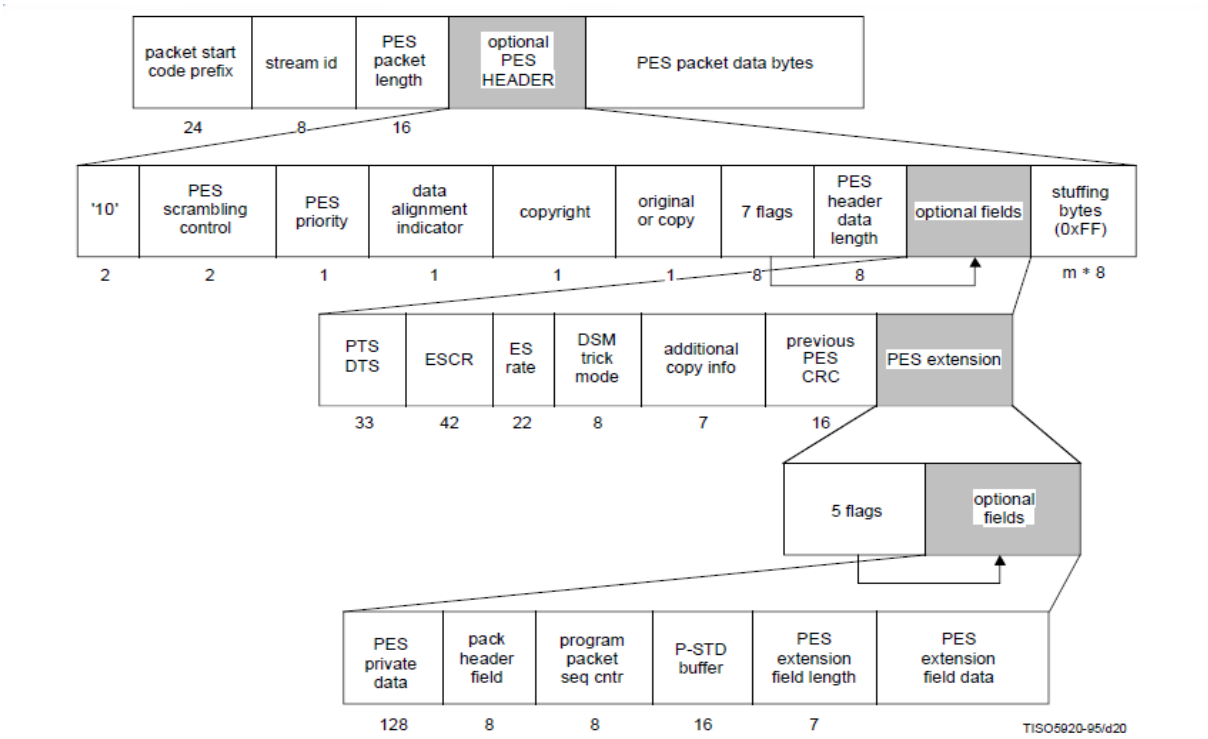
[그림 6] RTP Header 부분

Streamer에서 PLAY 요청에 따라 패킷을 전송할 때 유의해야 할 점은 RTP패킷의 데이터 부분인 동영상이다. TS파일은 다음과 같다.



[그림 7] TS 패킷 Analysis

TS 패킷은 *4 바이트의 고정된 길이의 헤더*와 *가변 길이의 Adaptation field*를 가진다. 원래 TS 패킷은 방송용으로도 쓰이기 때문에 하나의 스트림안에는 여러 프로그램의 스트림이 들어있지만, 우리는 하나의 동영상을 보내기 때문에 전혀 이것을 고려하지 않아도 된다. 그렇기 때문에 이러한 여러 프로그램의 *PID(Packet ID)*를 관리하는 PAT(Program Association Table)이 없다. 우리는 해당하는 비디오 스트림과 오디오 스트림의 PID를 찾아서 전송하기만 하면 된다.



[그림 8] TS 패킷 헤더 구조

PAUSE 연산의 경우에는 좀 더 복잡한 알고리즘이 요구된다. 현재 스트리밍 서버가 작동하는 방식은 기본적으로 RTSP 소켓을 제어하는 단일 프로세스에서 PLAY 요청을 받을 때 PLAY를 하는 Child 프로세스를 fork()를 이용하여 독립적으로 일을 수행하도록 만드는 방식이다. 따라서 PAUSE 요청이 들어오면 Child 프로세스에서 패킷을 보내는 ***연산을 중지하도록 만들어야 한다.*** 여기에는 크게 두 가지 방법이 있다.

첫 번째는 Process kill를 이용하는 방식이다. PLAY시에 fork()를 하면서 자식 프로세스의 pid를 받을 수 있으므로, 이를 가지고 있다가 RTSP로 PAUSE Protocol이 들어오면 Process Kill를 통해 PLAY를 멈추게 한다.

```
void pauseStreamer(Streamer* streamer){
    kill(streamer->playpid, SIGQUIT);
}
```

[그림 9] pauseStreamer 구현 일부 부분

그러나 이 방법은 PAUSE시 필연적으로 패킷 손상이 일어나게 되는 단점이 있었다. RTSP 소켓을 제어하는 Parent Process에서 Kill하는 순간과 Child Process가 Kill되는 순간에는 시간차가 있어 패킷 손상이 필연적으로 있게 되기 때문이다.

따라서 우리는 두 번째 방법인 파이프를 통해서 구성하였다. PLAY를 하는 Child Process가 Parent Process로부터 PAUSE명령이 날아왔다는 것을 인지한 순간부터 패킷을 보내지 않고, 다음 PLAYBACK 시에는 그 패킷부터 바로 보낼 수 있도록 하였다. 즉 클라이언트로부터 RTSP PAUSE Protocol을 받으면 Parent Process는 파이프에다 PauseSet을 한다. Child Process에서는 동적으로 계속 파이프에서 PauseSet 변수를 확인하며, PauseSet이 되었음을 안 순간 멈추고 PauseSet이 0으로 바뀔 때 까지 기다린다. PLAYBACK 요청이 클라이언트에서 Parent Process로 날아오면, 다시 Parent Process는 PauseSet을 0으로 만들고, Child Process는 패킷 전송을 재개한다.

```
void playStream(Streamer* streamer){
 while(1){
   비트율 조절에 따른 패킷 전송;
   Read(pauseSetPipe[0], streamer->pipebuffer, BUFF_SIZE)
   If(strcmp(streamer->pipebuffer, "1")==0){    // PauseSet된 경우
      while(1){
        read(pauseSetPipe)[0], streamer->pipebuffer, BUFF_SIZE)
        if(strcmp(streamer->pipebuffer, "0"==0){   //PauseSet==0
            break;
        }
      }
   }
}
```

[그림 10] playStream 구현 일부 부분

```
void handle_play(Streamer* streamer){

 S->C Protocol 응답;
 Read(streamer->pauseSetPipe[0], streamer->pipebuffer, BUFF_SIZE);
 If(strcmp(streamer->pipebuffer, "1")==0){        // pauseSet인 경우, 즉 Pause되었다
  PLAYBACK 되는 경우이므로, pauseSet = 0으로 만들면 된다.
  Sprint(streamer->pipebuffer, "0");
  Write(streamer->pauseSetPipe[1], streamer->pipebuffer, strlen(stremer->pipebuffer);
   // pauseSet을 0으로 만든다
   Break;
  }
  Else{       // PauseSet이 0인 경우이므로, 처음 Play시에만 접근된다.
  Pid = fork()
```

```
  If(pid==-1){perror("fork error"₩n"); exit(0);}
  Else if(pid==0){          // Child process
    close(rtsp_sock);
    playStream(streamer);
  }
  Else{     // Parent Process
    Streamer->playPid= pid;
  }
}
```

[그림 11] handle_play 구현 일부 부분

Child Process에서 매 While문 마다 파이프를 read해야 하므로, 파이프를 NONBLOCK 옵션을 주어 만들어 주어야 한다.

```
void initStreamer(Streamer* streamer){
  if(-1==pipe(streamer->pauseSetPipe){ perror("pipe error₩n"); exit(0); }
  fcntl(pauseSetPipe[0], F_SETFL, O_NONCLOCK);
}
```

[그림 12] initStreamer 구현 일부 부분

8. Solution

A. Implementation details - play stream

하지만 여기서 동영상을 보낼 때 주의할 점은 ***Bit Rate를 고려***해야 한다는 점이다. 동영상을 인코딩할 때 어떠한 Bit Rate 방식으로 할 지 결정할 수 있는데, *VBR(Variable Bit Rate)*와 *CBR(Constant Bit Rate)* 두 가지 방식이 있다. VBR 방식은 단위 시간 당 출력하는 양이 가변적이다. 따라서 영상의 전 구간에서 Bit rate를 다르게 할당할 수 있다. 그렇기 때문에 많은 데이터가 요구되는 구간에는 높은 Bit rate를 할당할 수 있고 그렇지 않은 구간에는 낮은 Bit rate를 할당할 수 있다. 따라서 Bit rate 할당이 효율적이지만, 그렇기 때문에 더욱 복잡한 연산이 필요하게 된다. 반면에 CBR은 출력 데이터 량이 전 구간에서 일정함을 의미한다. 고정된 Bit rate 이기 때문에 파일의 크기를 예측하거나 데이터를 전송하기는 VBR보다 용이하지만 그만큼 비효율적으로 Bit rate가 할당되어 있기 때문에, 데이터 량이 많이 필요한 구간 때문에 Bit rate를 높인다면 다른 구간에서도 불필요하게 Bit rate가 증가하게 되는 것이고, 반면에 Bit rate를 줄인다면 파일의 크기는 작아지겠지만 전체적인 데이터의 품질이 저하되는 단점이 존재한다.

이 프로젝트에서는 CBR로 인코딩 된 영상파일을 재생하는 것으로 한정한다. VBR의 경우에는 앞서 언급한 것과 같이 매번 Bit rate를 고려해야 하기 때문에 더 복잡한 알고리즘이 필요하게 된다. RTSP 스트리밍 서버의 본질적인 구현에 집중하기 위하여 CBR로 인코딩된 영상파일을 재생하는 것으로 하며, 본 서버에서 스트리밍할 동영상 파일 샘플들은 ***FFmpeg을 통해 모두 4K Byte의 CBR로 인코딩하였다***.

FFmpeg을 통해 인코딩한 옵션은 다음과 같다. 함께 TS 포맷의 파일로 인코딩 될 수 있도록 하였다. 파싱된 데이터를 통해 Streamer에 어떠한 수행을 하도록 전달할지 실질적인 분기점이 되는 Request Handler를 구현한다.

```
ffmpeg -i sample.mp4 -an -tune zerolatency
       -x264opts bitrate=4000:vbv-maxrate=4000:vbv-bufsize=166
        -vcodec libx264 -f mpegts -muxrate 4000K -y sample.ts
```

[그림 13] ffmpeg 인코딩 명령어

따라서 RTP 헤더를 만드는 함수를 따로 만들어서 패킷 전송 요청시마다 호출할 수 있도록 하였다. 헤더 다음에는 함께 보내야 할 동영상 데이터를 붙여서 전송하는데, 이 때 Bit rate를 고려해서 현재 패킷을 바로 보내야 할지 아니면 조금 기다려야 할지를 결정해야 한다. 즉 처음 시간부터 현재 시간까지를 계산하고 여기에 Bit rate를 곱하면 지금까지 보낸 파일의 크기를 알 수 있다. 즉 동영상 재생 시간에 따른 클라이언트에 보낸 파일의 예상 크기가 서버 내에서 실제로 보낸 파일의 크기와 동일해야 하는 것이다. 그래서 현재까지 보낸 데이터의 양을 계산한다.

$$\text{Bit Count} = (\text{현재 시간} - \text{처음 시간}) \times \text{Bit rate}$$

서버에서는 CBR로 인코딩 된 4000K byte의 일정한 Bit rate의 동영상 파일들만 있기 때문에 위의 식을 통해서 현재의 Bit Count를 구할 수 있다. 매번 패킷을 보내기 전에 Bit Count보다 더 큰 크기의 패킷들을 보냈다면 Sleep 하도록 하고, Bit Count보다 작아질 경우에는 정상적으로 패킷을 보낼 수 있도록 수행한다.

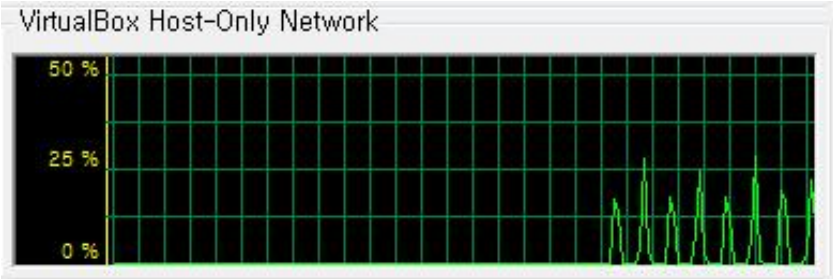
```
void playStream(STREAMER* streamer)
{
        …
```

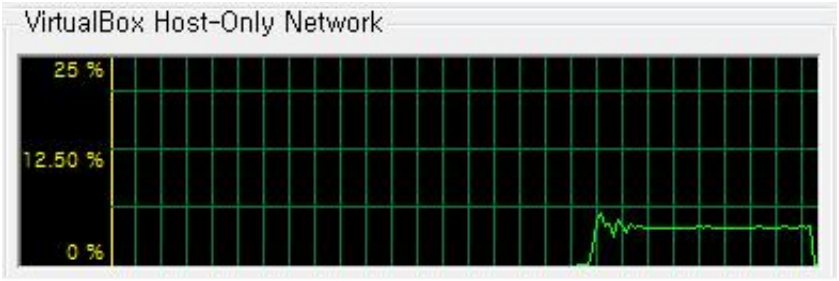
```
    while(1)
    {
            time(&currentTime);
            while( bitCount > (currentTime-startTime_t)*(4*1024*1024 + 400000))
            {
                usleep(100000);
                time(&currentTime);
            }
             …
            buildRTPHeader(streamer, rtppacket, &rtp_pkt);
            if(!streamer->transportMode) // 0x0 : TCP
            {
                printf("first\n");
                int errorno = write(…);
                printf("tcp called : %d error no : %d\n", t, errorno);
            }
            else
            {
                int errorno = sendto(…);
                bitCount = bitCount + 4608
            }
        …
        }

        printf("\n\n\n\n While Finished \n\n\n\n");
    }
}
```

[그림 14] playStream 구현 일부 부분

위는 Bit rate를 고려하여 구현한 playStream 함수이다. 여기에서 패킷을 전송할 때마다 4,608을 더해준 이유는 헤더가 12 bytes, 데이터를 188 bytes 씩 세 번 붙이고 Bit로 단위를 환산하기 위해서 8을 곱해준 값이다. 즉 $(12 + 188 \times 3) \times 8 = 4{,}608$ 이라는 값이 산출된다. (TS 파일은 한 단위가 188 bytes 씩 구분되어 있다)

B. Implementation issues - play stream





[그림 15] 네트워크 이용률 (위 VBR / 아래 CBR)

실제로 클라이언트의 OS에서의 네트워크 이용률을 확인해본 결과, Bit rate를 고려하지 않은 경우에는 네트워크 이용률에 큰 변화가 생기게 되고 영상도 너무 빠르게 재생되거나 너무 늦게 재생되고 또한 영상이 깨지는 현상이 자주 발견된다. 반면에 Bit rate를 고려하게 되면 네트워크 이용률도 안정적이게 되며 영상이 재생되는 것도 부드럽게 정상적인 속도를 유지하게 된다.

9. Results

A. Experiments

A.1.OPTION

```
0000  08 00 27 da 27 10 08 00  27 00 50 bd 08 00 45 00   ..'.'... '.P...E.
0010  00 ab 17 b9 40 00 80 06  f0 db c0 a8 38 01 c0 a8   ....@... ....8...
0020  38 66 23 7b 0b bd 5c 84  41 dd 2b e9 61 16 50 18   8f#{..\. A.+.a.P.
0030  40 29 d2 61 00 00 4f 50  54 49 4f 4e 53 20 72 74   @).a..OP TIONS rt
0040  73 70 3a 2f 2f 31 39 32  2e 31 36 38 2e 35 36 2e   sp://192 .168.56.
0050  31 30 32 3a 33 30 30 35  2f 73 61 6d 70 6c 65 32   102:3005 /sample2
0060  2e 74 73 20 52 54 53 50  2f 31 2e 30 0d 0a 43 53   .ts RTSP /1.0..CS
0070  65 71 3a 20 32 0d 0a 55  73 65 72 2d 41 67 65 6e   eq: 2..U ser-Agen
0080  74 3a 20 4c 69 62 56 4c  43 2f 32 2e 30 2e 30 20   t: LibVL C/2.0.0
0090  28 4c 49 56 45 35 35 35  20 53 74 72 65 61 6d 69   (LIVE555  Streami
00a0  6e 67 20 4d 65 64 69 61  20 76 32 30 31 31 2e 31   ng Media  v2011.1
00b0  32 2e 32 33 29 0d 0a 0d  0a                       2.23)... .
```

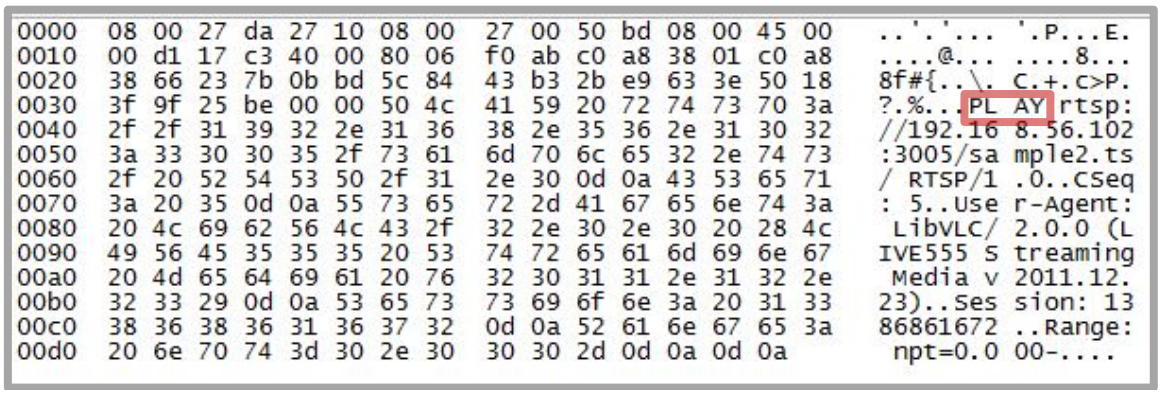
A.2.DESCRIBE

```
0000  08 00 27 da 27 10 08 00  27 00 50 bd 08 00 45 00   ..'.'... '.P...E.
0010  00 c5 17 ba 40 00 80 06  f0 c0 c0 a8 38 01 c0 a8   ....@... ....8...
0020  38 66 23 7b 0b bd 5c 84  42 60 2b e9 61 63 50 18   8f#{..\. B`+.acP.
0030  40 15 c2 b0 00 00 44 45  53 43 52 49 42 45 20 72   @.....DE SCRIBE r
0040  74 73 70 3a 2f 2f 31 39  32 2e 31 36 38 2e 35 36   tsp://19 2.168.56
0050  2e 31 30 32 3a 33 30 30  35 2f 73 61 6d 70 6c 65   .102:300 5/sample
0060  32 2e 74 73 20 52 54 53  50 2f 31 2e 30 0d 0a 43   2.ts RTS P/1.0..C
0070  53 65 71 3a 20 33 0d 0a  55 73 65 72 2d 41 67 65   Seq: 3.. User-Age
0080  6e 74 3a 20 4c 69 62 56  4c 43 2f 32 2e 30 2e 30   nt: Libv LC/2.0.0
0090  20 28 4c 49 56 45 35 35  35 20 53 74 72 65 61 6d    (LIVE55 5 Stream
00a0  69 6e 67 20 4d 65 64 69  61 20 76 32 30 31 31 2e   ing Medi a v2011.
00b0  31 32 2e 32 33 29 0d 0a  41 63 63 65 70 74 3a 20   12.23).. Accept:
00c0  61 70 70 6c 69 63 61 74  69 6f 6e 2f 73 64 70 0d   applicat ion/sdp.
00d0  0a 0d 0a                                          ...
```

A.3.SETUP

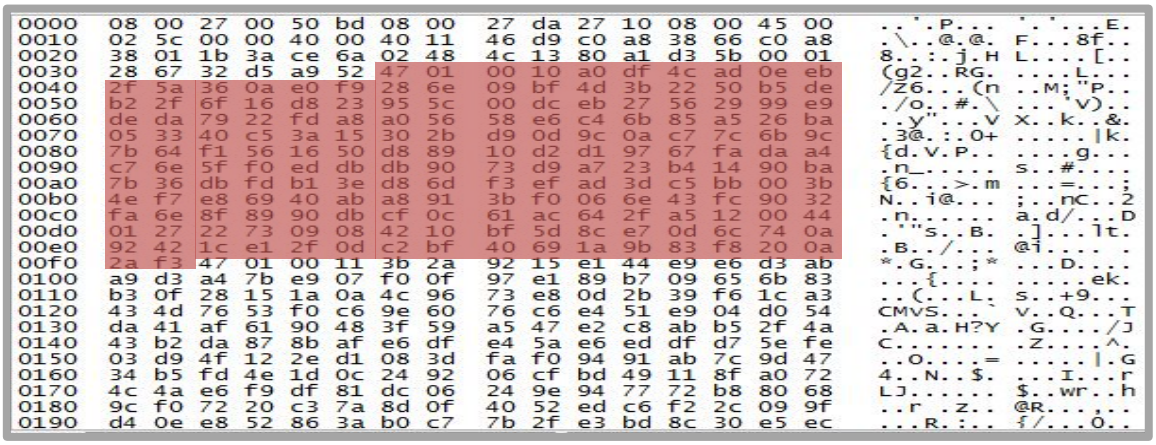
```
0000  08 00 27 da 27 10 08 00  27 00 50 bd 08 00 45 00   ..'.'... '.P...E.
0010  00 de 17 be 40 00 80 06  f0 a3 c0 a8 38 01 c0 a8   ....@... ....8...
0020  38 66 23 7b 0b bd 5c 84  42 fd 2b e9 62 73 50 18   8f#{..\. B.+.bsP.
0030  3f d1 7e e3 00 00 53 45  54 55 50 20 72 74 73 70   ?.~...SE TUP rtsp
0040  3a 2f 2f 31 39 32 2e 31  36 38 2e 35 36 2e 31 30   ://192.1 68.56.10
0050  32 3a 33 30 30 35 2f 73  61 6d 70 6c 65 32 2e 74   2:3005/s ample2.t
0060  73 2f 20 52 54 53 50 2f  31 2e 30 0d 0a 43 53 65   s/ RTSP/ 1.0..CSe
0070  71 3a 20 34 0d 0a 55 73  65 72 2d 41 67 65 6e 74   q: 4..Us er-Agent
0080  3a 20 4c 69 62 56 4c 43  2f 32 2e 30 2e 30 20 28   : LibVLC /2.0.0 (
0090  4c 49 56 45 35 35 35 20  53 74 72 65 61 6d 69 6e   LIVE555  Streamin
00a0  67 20 4d 65 64 69 61 20  76 32 30 31 31 2e 31 32   g Media  v2011.12
00b0  2e 32 33 29 0d 0a 54 72  61 6e 73 70 6f 72 74 3a   .23)..Tr ansport:
00c0  20 52 54 50 2f 41 56 50  3b 75 6e 69 63 61 73 74    RTP/AVP ;unicast
00d0  3b 63 6c 69 65 6e 74 5f  70 6f 72 74 3d 35 33 39   ;client_ port=539
00e0  31 34 2d 35 33 39 31 35  0d 0a 0d 0a               14-53915 ....
```

A.4.PLAY



```
0000  08 00 27 da 27 10 08 00  27 00 50 bd 08 00 45 00   ..'.'... '.P...E.
0010  00 d1 17 c3 40 00 80 06  f0 ab c0 a8 38 01 c0 a8   ....@... ....8...
0020  38 66 23 7b 0b bd 5c 84  43 b3 2b e9 63 3e 50 18   8f#{..\. C.+.c>P.
0030  3f 9f 25 be 00 00 50 4c  41 59 20 72 74 73 70 3a   ?.%...PL AY rtsp:
0040  2f 2f 31 39 32 2e 31 36  38 2e 35 36 2e 31 30 32   //192.16 8.56.102
0050  3a 33 30 30 35 2f 73 61  6d 70 6c 65 32 2e 74 73   :3005/sa mple2.ts
0060  2f 20 52 54 53 50 2f 31  2e 30 0d 0a 43 53 65 71   / RTSP/1 .0..CSeq
0070  3a 20 35 0d 0a 55 73 65  72 2d 41 67 65 6e 74 3a   : 5..Use r-Agent:
0080  20 4c 69 62 56 4c 43 2f  32 2e 30 2e 30 20 28 4c    LibVLC/ 2.0.0 (L
0090  49 56 45 35 35 35 20 53  74 72 65 61 6d 69 6e 67   IVE555 S treaming
00a0  20 4d 65 64 69 61 20 76  32 30 31 31 2e 31 32 2e    Media v 2011.12.
00b0  32 33 29 0d 0a 53 65 73  73 69 6f 6e 3a 20 31 33   23)..Ses sion: 13
00c0  38 36 38 36 31 36 37 32  0d 0a 52 61 6e 67 65 3a   86861672 ..Range:
00d0  20 6e 70 74 3d 30 2e 30  30 30 2d 0d 0a 0d 0a       npt=0.0 00-....
```

A.5.DATA



```
0000  08 00 27 00 50 bd 08 00  27 da 27 10 08 00 45 00   ..'.P... '.'...E.
0010  02 5c 00 00 40 00 40 11  46 d9 c0 a8 38 66 c0 a8   .\..@.@. F...8f..
0020  38 01 1b 3a ce 6a 02 48  4c 13 80 a1 d3 5b 00 01   8..:.j.H L....[..
0030  28 67 32 d5 a9 52 47 01  00 10 a0 df 4c ad 0e eb   (g2..RG. ....L...
0040  2f 5a 36 0a e0 f9 28 6e  09 bf 4d 3b 22 50 b5 de   /Z6...(n ..M;"P..
0050  b2 2f 6f 16 d8 23 95 5c  00 dc eb 27 56 29 99 e9   ./o..#.\ ...'V)..
0060  de da 79 22 fd a8 a0 56  58 e6 c4 6b 85 a5 26 ba   ..y"...V X..k..&.
0070  05 33 40 c5 3a 15 30 2b  d9 0d 9c 0a c7 7c 6b 9c   .3@.:.0+ .....|k.
0080  7b 64 f1 56 16 50 d8 89  10 d2 d1 97 67 fa da a4   {d.V.P.. ....g...
0090  c7 6e 5f f0 ed db db 90  73 d9 a7 23 b4 14 90 ba   .n_..... s..#....
00a0  7b 36 db fd b1 3e d8 6d  f3 ef ad 3d c5 bb 00 3b   {6...>.m ...=...;
00b0  4e f7 e8 69 40 ab a8 91  3b f0 06 6e 43 fc 90 32   N..i@... ;..nC..2
00c0  fa 6e 8f 89 90 db cf 0c  61 ac 64 2f a5 12 00 44   .n...... a.d/...D
00d0  01 27 22 73 09 08 42 10  bf 5d 8c e7 0d 6c 74 0a   .'"s..B. .]...lt.
00e0  92 42 1c e1 2f 0d c2 bf  40 69 1a 9b 83 f8 20 0a   .B../... @i.... .
00f0  2a f3 47 01 00 11 3b 2a  92 15 e1 44 e9 e6 d3 ab   *.G...;* ...D....
0100  a9 d3 a4 7b e9 07 f0 0f  97 e1 89 b7 09 65 6b 83   ...{.... .....ek.
0110  b3 0f 28 15 1a 0a 4c 96  73 e8 0d 2b 39 f6 1c a3   ..(...L. s..+9...
0120  43 4d 76 53 f0 c6 9e 60  76 c6 e4 51 e9 04 d0 54   CMvS...` v..Q...T
0130  da 41 af 61 90 48 3f 59  a5 47 e2 c8 ab b5 2f 4a   .A.a.H?Y .G..../J
0140  43 b2 da 87 8b af e6 df  e4 5a e6 ed df d7 5e fe   C....... .Z....^.
0150  03 d9 4f 12 2e d1 08 3d  fa f0 94 91 ab 7c 9d 47   ..O....= .....|.G
0160  34 b5 fd 4e 1d 0c 24 92  06 cf bd 49 11 8f a0 72   4..N..$. ...I...r
0170  4c 4a e6 f9 df 81 dc 06  24 9e 94 77 72 b8 80 68   LJ...... $..wr..h
0180  9c f0 72 20 c3 7a 8d 0f  40 52 ed c6 f2 2c 09 9f   ..r .z.. @R.....
0190  d4 0e e8 52 86 3a b0 c7  7b 2f e3 bd 8c 30 e5 ec   ...R.:.. {/...0..
```

B. Result analysis and Discussion

A.1 ~ A.4 는 rtsp를 통해서 client에서 server로 명령 요청 온 패킷을 캡쳐한 부분이다. 각 rtsp명령은 RFC2326 문서에 정의한 대로 명령어 Method | Request-URI | RTSP-Version 으로 구성된 Request Line와 여러 Request-header로 구성된 Request Header Fields로 구성됐다. 캡쳐된 사진을 보면 명령이 잘 들어 왔는 것을 확인할 수 있고, 서버는 해당되는 method에 따라 다르게 응답해준다. 그리고 A.5는 rtp를 통해서 시제 ts패킷 파일을 보내고 있는 부분이다. 0x47이 sync바이트로 ts패킷의 맨 첫 1바이트에 해당한다. 빨간색 음영으로 칠해진 사각형의 처음 역시 0x47로 시작된 것을 알 수 있고, 사각형 바로 뒤 역시 0x47로 시작한 것임을 알 수 있다.

10.Role

김 원 용 (조장)

- 전반적인 프로젝트 관리 및 일정 조정
- 설계 및 디자인 총괄
- Parser 및 Request Handler 구현
- 소켓 프로그래밍 관련 주요 이슈 담당

박 종 찬

- 개발 환경 구축 및 GitHub 초기 연동 작업
- 테스트 셋 총괄
- Event Handler 구현
- Linux 프로그래밍 관련 주요 이슈 담당

정 창 주

- RTP 패킷화 구현(Streamer, Scheduler)
- RTSP, RTCP 주요 기능 총괄
- HEVC 타이밍 이슈 해결
- MPEG, HEVC 주요 이슈 담당

11. Conclusion

본 프로젝트는 미디어 스트리밍 서버의 핵심이라고 할 수 있는 서버 아키텍쳐 설계와 패킷 전송률 조절에 그 중점을 두고 진행하였다. 따라서 중요 이슈에 대한 Abstract Solution 들은 토론 및 연구를 통해 초기 한 달 만에 얻는 것이 가능하였으나, 오픈 소스를 참조하지 않았기 때문에 네트워크, 프로세스, 파일 입출력 및 동영상 파일 시스템의 체계에 지식을 이용하여 구현해 가는 데서 많은 시행착오가 있었다. 그러나 오픈 소스를 참조하지 않더라도, 미리 구현되어 있는 스트리밍 서버에서 WireShark 등을 이용하여 패킷 이동 및 프로토콜의 Implementation 등을 미리 분석하여 진행하였다면 더 빠른 프로젝트 완성이 가능하였을 것이라는 추측이 든다. 매주 진행된 프로젝트 모임에서 각종 이슈에 대한 스터디 및 프로그래밍을 진행하였으며, 이 과정에서 회사측 담당자와 프로젝트 담당 교수님이신 엄현상 교수님의 적절한 비전 제시가 많은 도움이 되었다.

[Appendix] User manual

실행환경은 Linux 의 배포판인 CentOS 6.4 를 기준으로 한다.

1. Server Source Code Compile

먼저 서버를 구동하기 위해서 소스파일을 컴파일을 수행해야 한다. 전체 파일의 구성은 다음과 같다.

```
prj_threeguys/
    rtsp/
        main.c
        eventloop.c,   eventloop.h
        parser.c,   parse.h
        streamer.c,   streamer.h
        rtp.h
        config.cfg
        Makefile
media/
```

1) prj_threeguys
   서버에 필요한 디렉토리이이다. rtsp 폴더를 하나 가지고 있다.

2) rtsp
   서버 컴파일 소스 파일이 담겨있다.

3) media
   스트리밍 될 동영상들이 들어 있는 폴더이다. 보고서에서 밝힌대로 CBR 방식으로 인코딩하며, 본 서버는 4,000K Byte의 Bit rate의 동영상에 최적화되어 있다. 동영상을 인코딩하기 위하여 자주쓰는 프로그램인 FFmpeg 에서 인코딩하기 위한 명령어는 다음과 같다.

```
ffmpeg -i sample.mp4 -an -tune zerolatency
       -x264opts bitrate=4000:vbv-maxrate=4000:vbv-bufsize=166
       -vcodec libx264 -f mpegts -muxrate 4000K -y sample.ts
```

Makefile이 있기 때문에 아래 명령어로 해당 디렉토리 내에서 손쉽게 자동으로 컴파일을 수행할 수 있다.

```
make
```

2. Run Server

컴파일이 완료되면 rtsp 의 실행파일이 생성되며 이 파일을 통해 서버를 실행한다. 명령어는 다음과 같다.

```
./rtsp
```

서버를 정상적으로 실행하게 되면 터미널에서 다음과 같은 서버 실행 결과 메시지를 볼 수 있다.

```
[ThreeGuys@localhost rtsp]$ ./rtsp
============================
ThreeGuys RTSP Server Start
RTSP port : 3005.
Default Media Path : ./media/.
============================
```

[서버 실행 화면]

여기서 포트번호는 config.cfg 파일에서 지정을 할 수 있는데 config.cfg 파일의 내용은 다음과 같다.

```
File  Edit  View  Search  Terminal  Help
port=3005
media_path=./media/
~
~
~
```

[config.cfg]

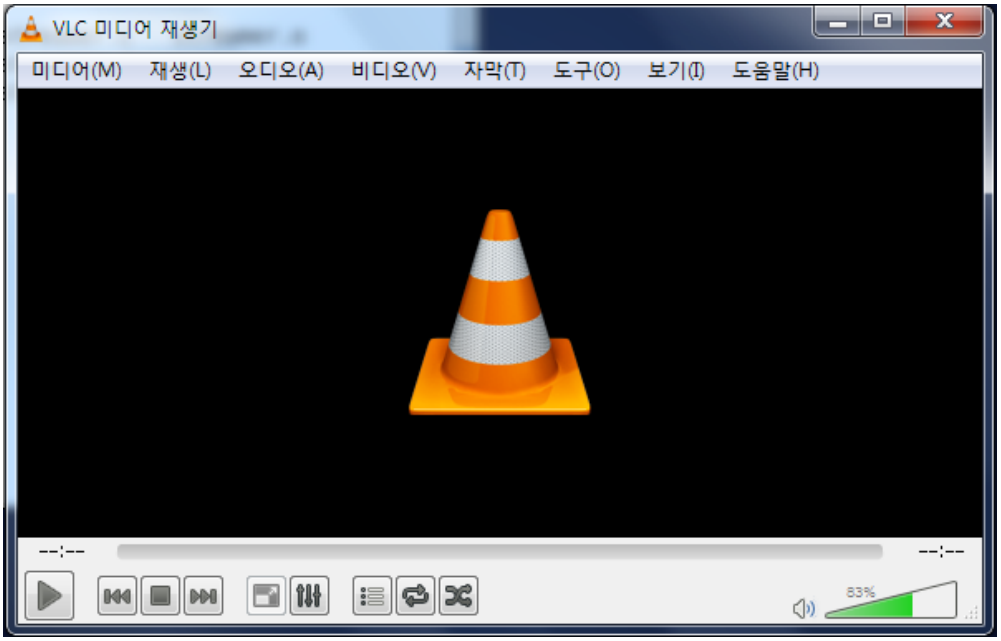
이 파일을 통해 포트번호와 미디어 파일들의 경로를 지정할 수 있지만, 현재는

미디어 경로를 이 파일을 통해 인식하지 못하며, 위의 프로젝트 디렉토리 중 media 폴더 내에 동영상 파일들이 위치해야 한다.

3. Request From Client

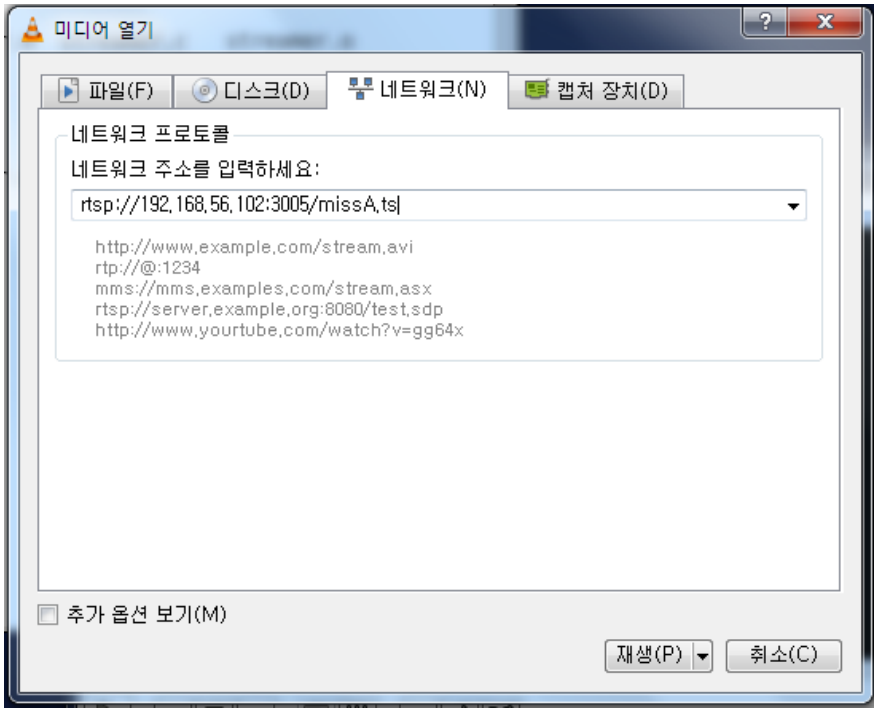
서버가 실행되면 클라이언트에서 요청을 보내본다. 클라이언트의 종류는 특별히 관계 없으나 미디어 스트리밍, RTSP 프로토콜이 지원되는 미디어 전용 플레이어를 권장한다.
윈도우 미디어 플레이어는 미디어 스트리밍을 지원하지만 본 프로젝트 서버 뿐만 아니라 다른 상용 서버에도 정상적으로 접근하지 못하는 경우가 많은 것을 확인하였으며, 오픈소스 무료 미디어 플레이어인 VLC 플레이어에서는 본 프로젝트 서버 뿐만 아니라, 다른 RTSP 서버에도 정상적으로 접근할 수 있음을 확인하였다.

따라서 본 프로젝트에서도 원활한 RTSP 스트리밍 서비스를 위해서 VLC 플레이어 사용을 권장하며 여기에서도 VLC 플레이어로 클라이언트 요청을 보내는 과정을 설명한다.



[VLC Media Player]

메뉴의 '미디어 > 네트워크 스트림 열기' 화면의 네트워크 탭에서 RTSP 스트리밍 요청을 보낼 수 있다.



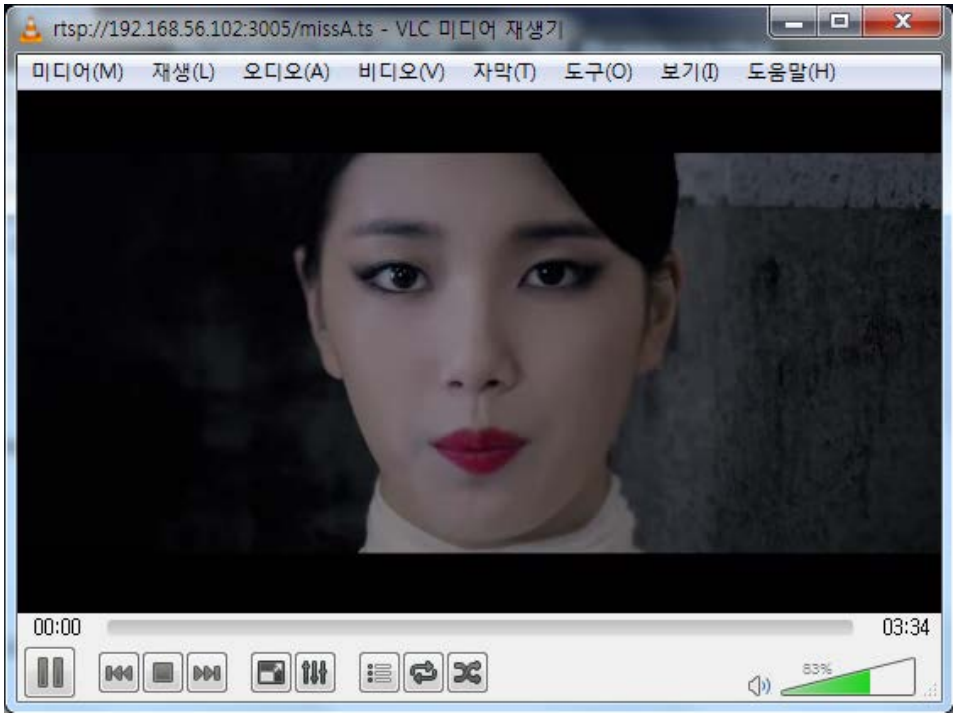
[VLC 플레이어에서 RTSP 요청을 보내는 화면]

위와 같이 RTSP 프로토콜을 통해서 서버의 주소와 파일의 이름을 지정하여 함께 보내게 되면, 정상적으로 RTSP요청이 서버로 전달된 경우, 서버에서 다음과 같이 서버에서 로그 메시지를 확인할 수 있다.

```
ThreeGuys@localhost:~/prj/prj_threeguys/rtsp
File  Edit  View  Search  Terminal  Help
SETUP rtsp://192.168.56.102:3005/missA.ts/ RTSP/1.0
CSeq: 4
User-Agent: LibVLC/2.1.1 (LIVE555 Streaming Media v2012.12.18)
Transport: RTP/AVP;unicast;client_port=55406-55407


setup
-------------S -> C-------------
RTSP/1.0 200 OK
CSeq:  4
Date: Tue Dec 17 2013 02:08:58 GMT
Transport: RTP/AVP;unicast;destination=192.168.0.6;source=192.168.0.6;client_p
t=55406-55407;server_port=6970-6971
Session: 1387246138


-------------C -> S-------------
PLAY rtsp://192.168.56.102:3005/missA.ts/ RTSP/1.0
CSeq: 5
User-Agent: LibVLC/2.1.1 (LIVE555 Streaming Media v2012.12.18)
Session: 1387246138
Range: npt=0.000-


play
-------------S -> C-------------
RTSP/1.0 200 OK
CSeq:  5
Date: Tue Dec 17 2013 02:08:58 GMT
Range: npt=0.000-214.000
Session: 1387246138
RTP-Info: url=rtsp://192.168.56.102:3005
```

[정상적으로 RTSP 요청이 수행된 경우 서버에서 로그 출력]

OPTION, DESCRIBE, SETUP 의 과정을 거치고 PLAY 요청이 들어온 경우 서버는 해당 요청에 응답하게 되고 정상적으로 동영상이 수행됨을 확인할 수 있다.



[클라이언트에서 동영상이 PLAY되는 화면]