컴퓨터 통신
(네트워크 프로그래밍)

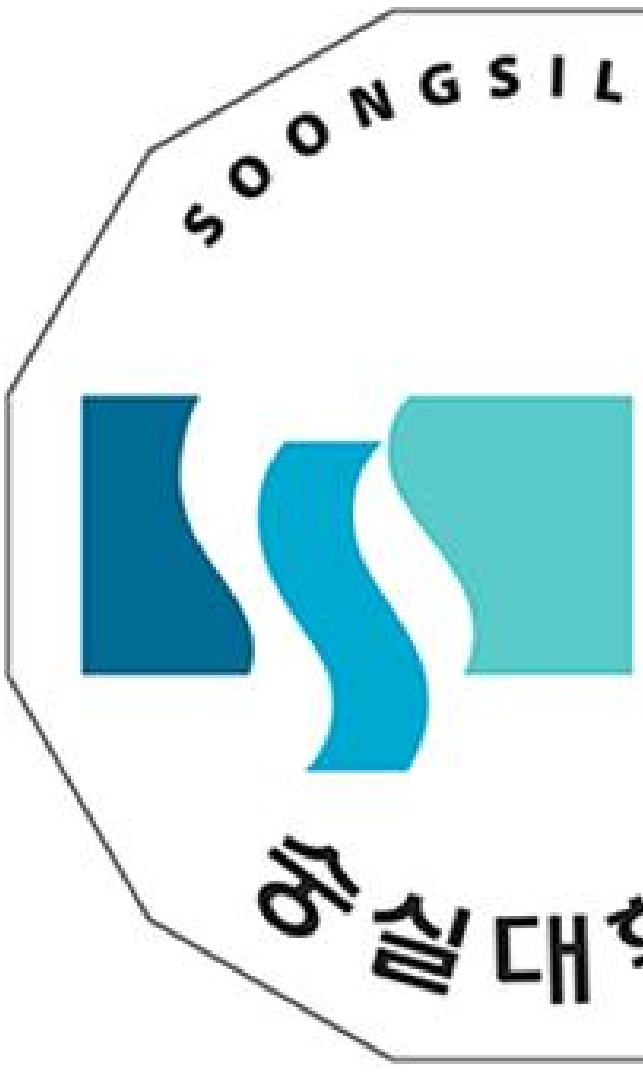
SSU Tube

-스트리밍 서버-

- 팀　　명 : Ssugle
- 이　　름 : 계 희 준
- 전　　공 : 컴퓨터학부
- 학　　번 : 20142300
- 제 출 일 : 2019.06.08

The First University in Korea
Soongsil University

목차



1. 개요

본 프로젝트에서는 TCP 프로토콜을 활용한 UCC 공유 프로그램을 개발한다. 서버의 부하를 줄이기 위해 두 개의 서버로 운용하고 각 서버와 통신하는 두 개의 데이터베이스를 운용한다. 클라이언트는 기능에 따라 알맞은 서버와 통신한다.

본인은 본 프로젝트에서 영상 업로드 및 영상 스트리밍 데이터 전송 기능을 수행하기 위해 스트리밍 서버를 개발한다.

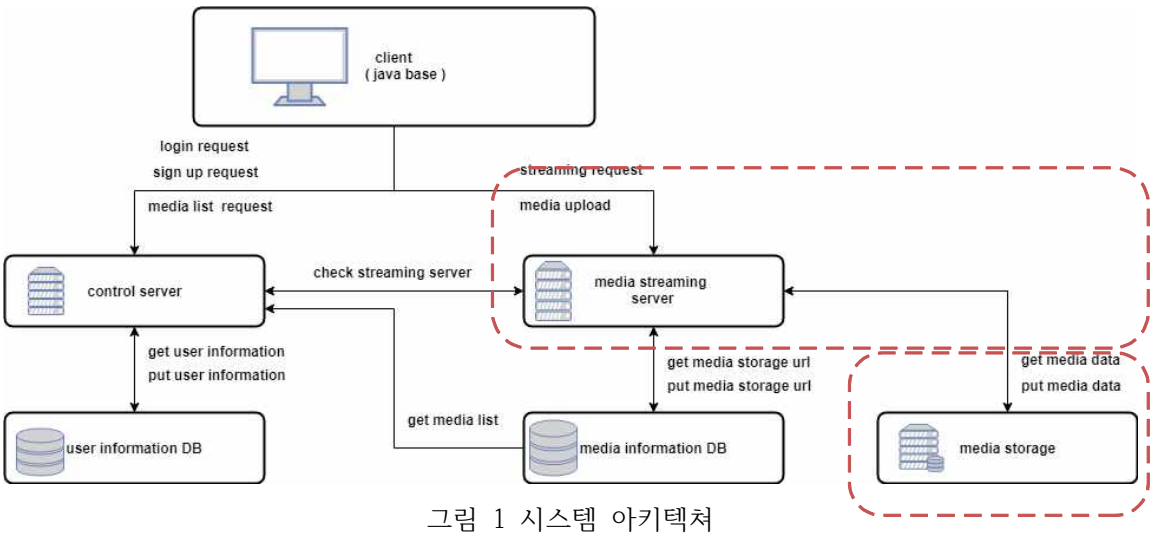


그림 1 시스템 아키텍쳐

2. 개발 역할

2-1 시스템 구성도

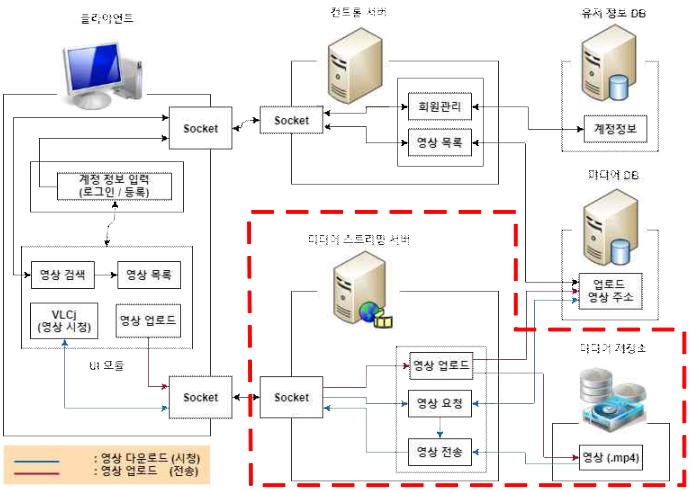


그림 2 전체 시스템 구성도

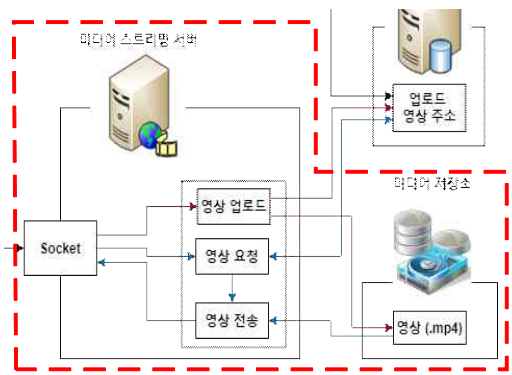


그림 3 스트리밍 서버

스트리밍 서버는 여러 클라이언트를 동시에 처리하기 위해 멀티 쓰레드를 사용한다. 또한 각 쓰레드는 클라이언트의 요청에 따라 영상 데이터 수신 (업로드) 및 영상 데이터 송신 (스트리밍) 의 기능을 수행한다.

● 다중 클라이언트

① 새로운 클라이언트가 연결 요청을 하면 새로운 쓰레드를 만든다.
② 새로운 쓰레드에서 클라이언트의 메시지를 기다린다.
③ 클라이언트의 메시지에 따라 영상 데이터를 수신하거나 송신한다.

● 영상 데이터 수신 (업로드)

① 클라이언트로부터 수신 받을 영상의 이름과 크기, id 정보가 담긴 메시지를 받는다.

flag (4)	Title	id	size
1 bytes	60 bytes	10 bytes	4 bytes

② 영상 파일을 서버의 media storage에 만든다.
③ 클라이언트가 보내는 영상의 데이터를 영상 파일에 쓴다. 이때 미리 전달받은 크기만큼 받는다.
④ 올린 사람, 영상의 이름, 저장된 url주소를 유일한 key값과 함께 DB에 저장 하도록 요청한다.

● 영상 데이터 송신 (스트리밍)

① 클라이언트로부터 사용자가 선택한 동영상의 key값이 포함된 메세지를 받는다.

flag (5)	key
1 bytes	4 bytes

② DB에게 해당 key값을 같는 동영상의 url 주소를 요청한다.
③ DB로부터 받은 url주소를 이용하여 동영상 파일의 크기를 확인한다.
④ 동영상 파일의 크기를 클라이언트에게 보낸다.

key
4 bytes

⑤ 동영상 파일의 내용을 특정 크기만큼 읽어 클라이언트에게 전송한다. 내용을 전부 전송할 때 까지 반복한다.

Video Data
1024 bytes

2-2 자료조사

스트리밍이란 주소 음악이나 동영상 등의 멀티미디어 파일을 전송하고 재생하는 방법 중의 하나이다. 동영상과 같은 멀티미디어 파일은 크기가 크기 때문에 파일을 내려 받은 후 재생한다면 시간이 오래 걸릴 수 있다. 이를 해결하기 위해 서버에서는 멀티미디어 파일 데이터를 여러 패킷으로 나누어 전송하고 클라이언트에서는 전송을 받으며 동시에 재생을 한다. 이와 같은 방법으로 멀티미디어 송수신을 제공하는 것을 스트리밍 서비스라고 한다.

스트리밍 서비스의 종류는 크게 on-demand streaming 방식과 Live Streaming 방식으로 나눌 수 있다. on-demand streaming 방식은 서버에 멀티미디어 데이터를 저장해 놓았다가 사용자가 멀티미디어 데이터를 요철 할 경우 멀티미디어 데이터를 전송한다. 이 방식은 흔히 VoD (Video on Demand, 주문형 비디오) 서비스에 사용되는 방식이다. Live Streaming 방식은 한 사용자가 서버로 전송하는 영상 데이터를 서버에 저장하지 않고 실시간으로 다수의 시청자에게 제공한다. 이 방식은 실시간 생중계에 많이 사용되는 방식이다.

본 프로젝트는 사용자가 동영상을 서버로 업로드 하면 다른 사용자가 원할 때 동영상을 볼 수 있는 환경을 제공하는 것이 목적이다. 따라서 on-demand streaming 방식으로 구현한다. 이와 같은 스트리밍 서비스를 구현하기 위해 progressive download 방식과 RTSP에 대해 조사를 하였다.

● RTSP

RTSP는 Real Time Streaming Protocol 의 약자로 응용 계층에서 사용되는 프로토콜이다. RTSP는 스트리밍 시스템에 사용되며 미디어 서버를 원격으로 제어하는 기능을 한다. 이를 위해 PLAY, PAUSE, TURNDOWN 등의 명령어를 제공한다. RTSP에서 실제로 데이터를 전송하는 과정은 RTP를 이용한다. RTP는 음성이나 동영상 등의 데이터 스트림을 실시간으로 전달할 때 사용하는 데이터 전송 프로토콜이다. 또한 RSTP는 서버의 상태를 계속해서 유지하는 stateful 프로토콜이며 세션을 유지하기 위해 지속적인 TCP 연결을 요구한다. 이 방식은 이미 시청한 데이터는 버리고 사용자가 재생을 원하는 프레임만 전송받는다.

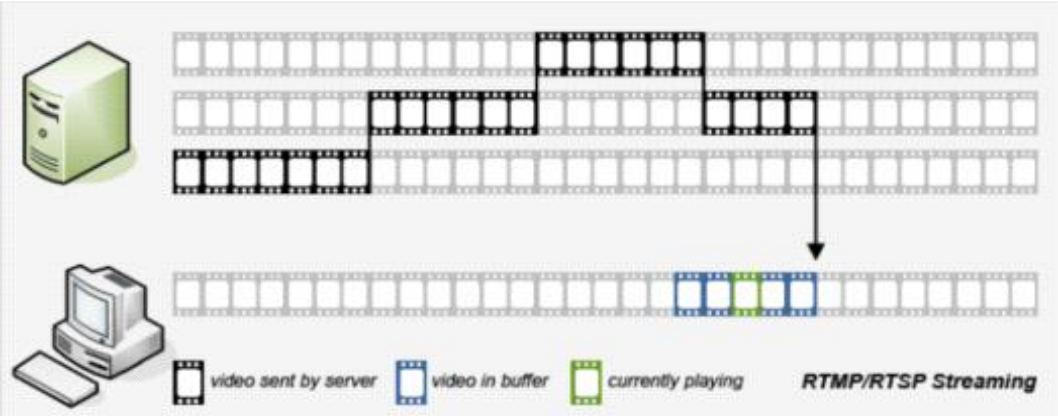


그림 4 RTSP / RTMP 의 미디어 재생 방식

(출처 : https://www.jwplayer.com/blog/what-is-video-streaming/)

본 프로젝트에서 RTSP를 이용하여 서버를 구현하기 위해 『C언어로 구현하는 RTSP VOD 스트림 서버. 정재훈 저』 서적을 참고하였다. 위 서적에서는 RTSP의 구현을 위해 동영상의 헤더를 분석하였다. 클라이언트에서 이 헤더 정보를 참조하며 전송받은 동영상 데이터를 재생함으로써 RTSP를 구현한다. 하지만 위 서적에서 제시한 방식대로 구현하기 위해서는 사용자가 업로드 할 동영상의 헤더 구성 정보를 알아야한다. 또한 라이브러리를 제공하지 않으며 RTSP를 구현하기 위해 RTP, RTCP 등의 프로토콜 역시 구현해야 하므로 복잡하고 긴 코드로 구현해야 한다.

● progressive download

progressive download 방식은 미디어 파일을 서버에서 클라이언트로 전송 할 때 클라이언트에서 데이터를 전부 받을 때 까지 기다렸다가 재생하는 다운로드 방식이 아닌 데이터를 받으면서 파일을 재생 하는 방식이다. 즉 서버에 저장된 파일을 내려 받으면서 동시에 파일을 재생한다. 이 방식은 세계적인 스트리밍 사이트인 youtube에서 사용한 방식이며 광범위하게 사용되고 있다.

progressive download 방식은 RTSP 방식과 달리 전송된 미디어 데이터가 클라이언트의 저장소에 저장

된다. 동영상 파일의 헤더 정보와 같이 파일을 재생 할 수 있는 일부 데이터가 전송이 되면 클라이언트에서는 미디어 파일을 재생한다. 이때 미디어 파일을 재생하며 계속해서 동영상 데이터를 서버로부터 전송받는다. 따라서 동영상 데이터를 전부 받기 위해 기다리는 시간을 절약 할 수 있다.

progressive download 방식은 RTSP에 비해 구현하기 쉽다는 장점이 있다. 하지만 클라이언트에 동영상 파일이 저장되는 방식이므로 보안 측면에서 문제가 발생 할 수 있다. 또 유료 비디오 서비스의 경우 이 방식은 적절하지 않다. 게다가 미디어 데이터가 클라이언트에게 순서대로 도달해야 스트리밍 기능을 보장할 수 있다. 이와 같은 특성 때문에 동영상의 중간 부분을 보기 위해서는 앞선 데이터를 모두 전송받아야 한다. 또한 데이터의 일부만 재생하기를 원해도 전체 데이터를 다운 받아야 하는 경우가 생길 수 있다는 단점이 존재한다.

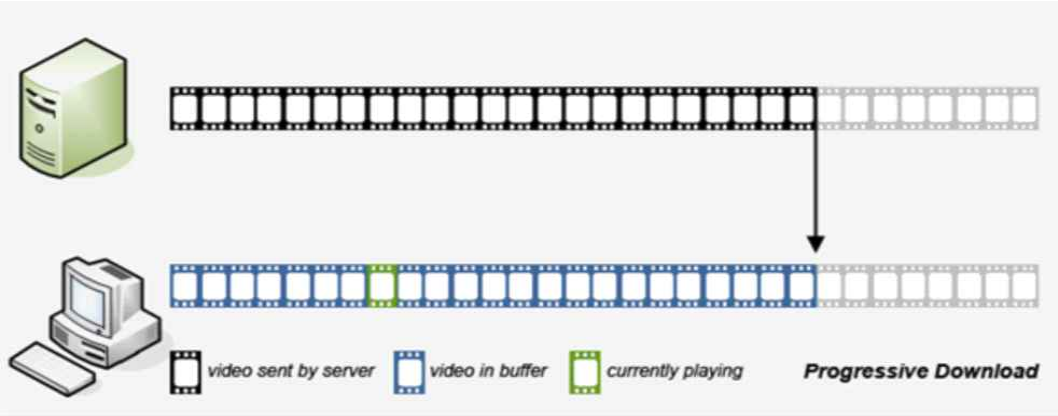


그림 5 Progressive download 의 미디어 재생 방식

(출처 : https://www.jwplayer.com/blog/what-is-video-streaming/)

progressive download 방식의 단점은 영상의 크기가 클수록 명확하게 드러난다.

● 구현 내용

본 프로젝트는 숭실대 학생들의 UCC 공유를 목적으로 두고 있다. UCC는 동영상이지만 공유를 목적으로 한다. 또한 progressive download 방식의 단점에 큰 영향을 받지 않을 정도의 작고 짧은 영상이다. 따라서 본 프로젝트는 progressive download 방식을 활용하여 개발한다.

```
client_sock = accept(server_sock, (struct sockaddr *)&client_addr,&client_size);
write(client_sock, send_header, sizeof(send_header));
printf("client accpeted[]\n");
lseek(fd, 0, SEEK_SET);
while(read(fd, send_msg, MAX_MSG_LEGNTH) > 0){
    write(client_sock,send_msg,strlen(send_msg));
    memset(send_msg, 0, sizeof(send_msg));
}
```

그림 6 파일의 데이터를 전송하는 서버의 코드 일부

위의 코드는 서버에서 파일의 내용을 읽어 클라이언트로 전송하는 코드이다. 동영상 파일도 파일의 일종이기 때문에 위와 같이 파일을 전송하는 코드로 동영상 데이터를 전송 할 수 있다. send_header에는 전

송할 동영상 데이터의 이름과 파일의 크기가 포함된다. 클라이언트에서는 이 메시지를 참조하여 동영상의 정보를 알 수 있다. 클라이언트에게 파일의 데이터를 전송할 때에는 1024 바이트의 크기로 전송한다. 만약 읽은 내용이 1024보다 작은 경우 읽은 만큼만 클라이언트로 전송한다. 전송할 때의 패킷의 크기는 전송받을 경우보다 작게 설정한다. 이는 write 버퍼에 쓸 곳이 없어 생기는 오버헤드의 발생을 최소화하기 위함이다.

```
client_socket = new Socket(IP, PORT);
InputStream stream = client_socket.getInputStream();
DataInputStream dstream = new DataInputStream(stream);
 ...
fstream = new FileOutputStream(filename);

recv_filesize = 0;
while(recv_filesize <= filesize) {
    if((readsize = dstream.read(contentsBuffer))<0)
        continue;
    fstream.write(contentsBuffer, 0, readsize);
    recv_filesize += readsize;
    System.out.println("read : "+readsize+" recv "+recv_filesize);
    for(int i = 0; i < contentsBuffer.length; i++) {
        contentsBuffer[i] = 0;
    }
}
```

그림 7 파일의 데이터를 전송받아 저장하는 클라이언트의 코드 일부

위의 코드는 서버로부터 동영상 데이터를 전송 받아 파일에 저장하는 클라이언트의 코드이다. 동영상 데이터를 모두 수신하기 전 미리 받은 크기 정보를 참조하여 소켓으로부터 데이터를 읽는다. 클라이언트에서 받은 데이터를 읽은 때에는 충분한 크기의 버퍼를 이용하여 전송 받는다. 이는 read 버퍼가 꽉 차서 전송받을 수 없어 생기는 오버헤드의 발생을 최소화하기 위함이다.

위의 두 코드를 이용하여 동영상 스트리밍이 가능한지 테스트를 해 보았다. VLC Player를 이용하여 테스트 하였다. 테스트를 수행 해 본 결과 영상을 송신 중에 동영상 재생이 가능하였고 이는 곳 동영상 스트리밍이 가능 하다는 것을 의미한다. 또한 영상이 재생 중에 전송을 멈추면 동영상이 전송 받은 부분 까지만 재생되는 것도 확인할 수 있었다.

본 프로젝트에 위 두 코드를 적용하여 스트리밍 서버를 구현하기 위해서 첫 번째로 서버에 동영상을 업로드 할 수 있는 기능이 필요하다. 서버에 동영상을 업로드 할 수 있는 기능은 위의 코드를 활용하여 구현할 수 있다.

두 번째로 DB를 이용하여 동영상 데이터를 관리하는 과정이 추가적으로 필요하다. 클라이언트가 서버로 동영상을 요청할 때 동영상의 primary key 값을 전달한다. 이 key 값은 클라이언트가 동영상 재생을 요청하기 전 이미 리스트 형태로 받은 데이터로 DB에서 동영상들을 구분해 주는 값이다. 스트리밍 서버에서는 전달받은 key 값을 이용하여 DB에서 해당 동영상이 저장된 URL 주소를 반환받는다. 반환 받은 URL을 이용하여 실제 저장된 영상 데이터에 접근하여 클라이언트로 전송 할 수 있다.

```
char* getMediaUrl(int _key);
```

리턴값 : key에 해당하는 동영상이 저장된 저장소의 URL 주소

이를 구현하기 위해 먼저 동영상 업로드 시 서버의 저장소에 저장된 URL과 id, 동영상 이름, key를 DB에 저장해야 한다. 동영상 이름과 id는 클라이언트로부터 전송 받고 URL은 스트리밍 서버에서 결정한다. key값은 DB에 저장할 때 결정한다.

```
int putMediaData(char *_URL, char *_id, char *_mediaName);
```

리턴값 : DB에 저장이 성공했으면 1을 리턴한다. 실패하면 -1을 리턴한다.

사례 조사와 코드를 바탕으로 본 프로젝트에서 본인은 우분투 환경에서 클라이언트가 보내는 동영상 데이터를 저장하고 클라이언트가 요청한 동영상 데이터를 progressive download 방식으로 전송하여 스트리밍과 업로드가 가능하도록 스트리밍 서버 구축을 담당한다. 이 때 동영상이 저장된 주소를 DB에 저장함으로써 동영상 관리가 수월하도록 구현한다.

ref

스트리밍 (위키백과) :
https://ko.wikipedia.org/wiki/%EC%8A%A4%ED%8A%B8%EB%A6%AC%EB%B0%8D
progressive download (위키백과) :
https://ko.wikipedia.org/wiki/%ED%94%84%EB%A1%9C%EA%B7%B8%EB%A0%88%EC%8B%9C%EB%B8%8C_%EB%8B%A4%EC%9A%B4%EB%A1%9C%EB%93%9C
RTSP (위키백과) :
https://ko.wikipedia.org/wiki/%EC%8B%A4%EC%8B%9C%EA%B0%84_%EC%8A%A4%ED%8A%B8%EB%A6%AC%EB%B0%8D_%ED%94%84%EB%A1%9C%ED%86%A0%EC%BD%9C
정재훈 저. 『C언어로 구현하는 RTSP VOD 스트림 서버』 북랩. 2017.
스트리밍 용어 및 개념: https://blog.kollus.com/?p=136
https://www.jwplayer.com/blog/what-is-video-streaming/
https://linuxism.ustd.ip.or.kr/1267