碩士學位論文

안드로이드 플랫폼 상에서 H.264 SVC(Scalable Video Coding) Player의 설계 및 구현

A Design and Implementation of H.264 SVC(Scalable Video Coding) Player for the Android Platform

2010年

漢城大學校 大學院

컴퓨터 工 學 科

컴퓨터 工學 專攻

鄭 愿 太

碩 士 學 位 論 文
指導敎授 黃琦太

안드로이드 플랫폼 상에서 H.264 SVC(Scalable Video Coding) Player의 설계 및 구현

A Design and Implementation of H.264 SVC(Scalable Video Coding) Player for the Android Platform

2010年 6月 日

漢城大學校 大學院

컴퓨터 工 學 科

컴퓨터 工學 專攻

鄭 愿 太

碩 士 學 位 論 文
指導敎授 黃琦太

안드로이드 플랫폼 상에서 H.264 SVC(Scalable Video Coding) Player의 설계 및 구현

A Design and Implementation of H.264 SVC(Scalable Video Coding) Player for the Android Platform

위 論文을 컴퓨터工學 碩士學位論文으로 제출함

2010年 6月 日

漢城大學校 大學院
컴퓨터 工 學 科
컴퓨터 工學 專攻
鄭 愿 太

鄭愿太의 工學 碩士學位論文을 認准함

2010年 6月 日

審査委員長 강 희 중 印

審 査 委 員 황 기 태 印

審 査 委 員 김 남 윤 印

목 차

【 표 목 차 】

【 그 림 목 차 】

제 1 장 서 론

제 1 절 연구 동기

네트워크 전송 기술의 발달 및 컴퓨터 처리 능력의 향상으로 인하여 VOD, 인터넷 방송, IP TV, DMB등 스트리밍 시스템의 요구 분야 및 응용분야가 증대하고 있고 많은 연구 들이 진행되어 왔다.

최근 다양한 사용자 단말환경 및 네트워크 환경으로 인하여 스트리밍 시스템 사용자가 확장됨에 따라 이에 대한 이슈들이 발생하게 되었고(우문섭, 2007) 실험 환경을 위하여 시뮬레이션에 관한 이슈도 발생하게 되었다(김혜선, 2007).

안드로이드는 구글이 주도하여 세계 각국의 이동통신 관련 회사 연합체인 OHA에서 개발한 오픈 플랫폼이다. 2007년 안드로이드 발표 이후 여러 휴대폰뿐만 아니라 임베디드 장비에도 안드로이드가 탑재되고 있다.

우리나라에는 최근에 들어서야 안드로이드가 탑재된 모바일 단말기가 발매되고 있으며 아이폰과 더불어 크게 각광받고 있다.

모바일 단말기기의 처리능력은 일반 PC에 비하여 상당히 떨어질뿐 아니라 무선 네트워크를 사용하기 때문에 일반 PC에서처럼 원활한 스트리밍 데이터를 처리할 수 있는 능력이 부족하다. 이러한 각기 다른 사용자의 환경을 위하여 스트리밍 서버에서는 다른 버전의 데이터를 유지해야만 하고 저장공간 낭비라는 단점을 초래하게 된다.

이러한 단점을 해결하기 위하여 H.264 AVC(Advanced Video Coding)의 확장 형태인 H.264 SVC(Scalable Video Coding)가 개발되게 되었다.

H.264 SVC는 부호화된 영상을 여러 가지 계층으로 유지를 하여 사용자의 환경에 따라 계층을 선택적으로 전송할수 있는 압축형식이다. 계층별로 전송이 가능하기 때문에 스트리밍 서버에서는 사용자의 환경에 맞춰진 다른 버전의 데이터를 유지할 필요가 없어서 저장공간의 낭비를 방지 할수 있게 된다(http://ip.hhi.de/imagecom_G1/savce/index.htm, Heiko Schwarz

외 2명, 2007 : 1103-1120).

현재 모바일 단말기에서 H.264 SVC 재생을 위한 Player가 존재하지 않고 안드로이드에서 미디어 재생을 위하여 다음 표와 같이 여러 가지 미디어 타입을 지원하지만 H.264 SVC를 위한 디코더를 내장하고 있지는 않기 때문에 본 논문에서는 H.264 SVC 스트림 데이터를 수신하여 재생할 수 있는 Player를 안드로이드 환경에서 개발하였다.

표 1.1 안드로이드에서 지원하는 미디어 포맷 종류

(http://developer.android.com)

Type	Format	Encoder	Decoder	File Type(s) Supported
Audio	AAC LC/LTP		X	3GPP (.3gp) and MPEG-4 (.mp4, .m4a). No support for raw AAC (.aac)
	HE-AACv1 (AAC+)		X	
	HE-AACv2 (enhanced AAC+)		X	
	AMR-NB	X	X	3GPP (.3gp)
	AMR-WB		X	3GPP (.3gp)
	MP3		X	MP3 (.mp3)
	MIDI		X	Type 0 and 1 (.mid, .xmf, .mxmf). Also RTTTL/RTX (.rtttl, .rtx), OTA (.ota), and iMelody (.imy)
	Ogg Vorbis		X	Ogg (.ogg)
	PCM/WAVE		X	WAVE (.wav)
Image	JPEG	X	X	JPEG (.jpg)
	GIF		X	GIF (.gif)
	PNG		X	PNG (.png)
	BMP		X	BMP (.bmp)
Video	H.263	X	X	3GPP (.3gp) and MPEG-4 (.mp4)
	H.264 AVC		X	3GPP (.3gp) and MPEG-4 (.mp4)
	MPEG-4 SP		X	3GPP (.3gp)

제 2 절 연구 목표

본 논문에서는 안드로이드에서 H.264 SVC Player를 설계 및 구현하고 이를 가지고 발생할 수 있는 문제점 발견 및 해결, 스트리밍 가능성을 증명하는게 목표를 두고 있다. 세부적인 내용은 다음과 같다.

- H.264 SVC Player 설계 및 구현

안드로이드에서 H.264 SVC Decoder를 사용가능하도록 포팅하고 이를 가지고 영상을 재생할 수 있는 Player를 설계 및 구현한다.

- H.264 SVC Streaming 영상 재생시 문제점 분석 및 해결

안드로이드가 탑재된 모바일 단말기에서 H.264 SVC 스트리밍 영상 재생시에 발생 할수도 있는 여러 가지 문제점을 발견하고 이를 해결할 수 있는 방법을 찾는다.

- 모바일 단말기에서 스트리밍 가능성 증명

모바일 단말기에서 H.264 SVC 스트리밍을 문제없이 재생하여 실시간 스트리밍이 가능하다는 것을 증명한다.

제 3 절 논문 구성

본 논문의 구성은 다음과 같다.

- 제 2장 연구배경

H.264 SVC와 안드로이드 및 스트리밍 시스템에 관련된 내용에 대하여 기술한다.

- 제 3장 H.264 SVC 스트리밍 시스템 모델링

H.264 SVC를 이용한 스트리밍 시스템이 필요로 구성, 동작원리에 대하여 기술한다.

- 제 4장 안드로이드상에서 H.264 SVC Player 구현

안드로이드가 탑재된 모바일 단말기에서 동작하는 H.264 SVC Player의 구조 및 구현한 내용에 대하여 기술하고 성능에 대하여 기술한다.

- 제 5장 Player 동작 확인

Player의 테스트를 위한 모델링과 동작 확인에 대하여 기술한다.

- 제 6장 결론 및 향후연구

본 논문에서 설계 및 구현한 Player에 대한 결론을 내리며 향후 연구에 대하여 기술한다.

제 2 장 연구 배경

제 1 절 H.264 SVC

1.1 개요

H.264 SVC는 H.264 AVC의 확장으로써 ITU-T(International Telecommunication Union)의 VCEG(Video Coding Experts Group)와 ISO(International Organization for Standardization)/IEC(International Electrotechnical Commission)의 연합 그룹인 JVT(Joint Video Team)에서 개발하여 표준화된 영상 압축 코덱이다.

H.264 SVC를 이용하여 부호화하게 되면 다양한 해상도, 프레임율로 복호화 할 수 있도록 한 개의 기본 계층과 여러개의 향상 계층으로 비트스트림을 구성할 수 있다. 이것이 가능하도록하는 것은 SVC에서 제공하는 3가지 확장성때문이며, 시간적 확장성(Temporal Scalability), 공간적 확장성(Spatial Scalability), 양질적 확장성(Quality Scalability)으로 구성된다.

시간적 확장성은 비트스트림의 프레임율(fps)을 표현한 것이고 공간적 확장성은 비트스트림의 사이즈를 나타낸 것이며 양질적 확장성은 화질을 나타낸다(http://ip.hhi.de/imagecom_G1/savce/index.htm, Heiko Schwarz외 2명, 2007 : 1103-1120).

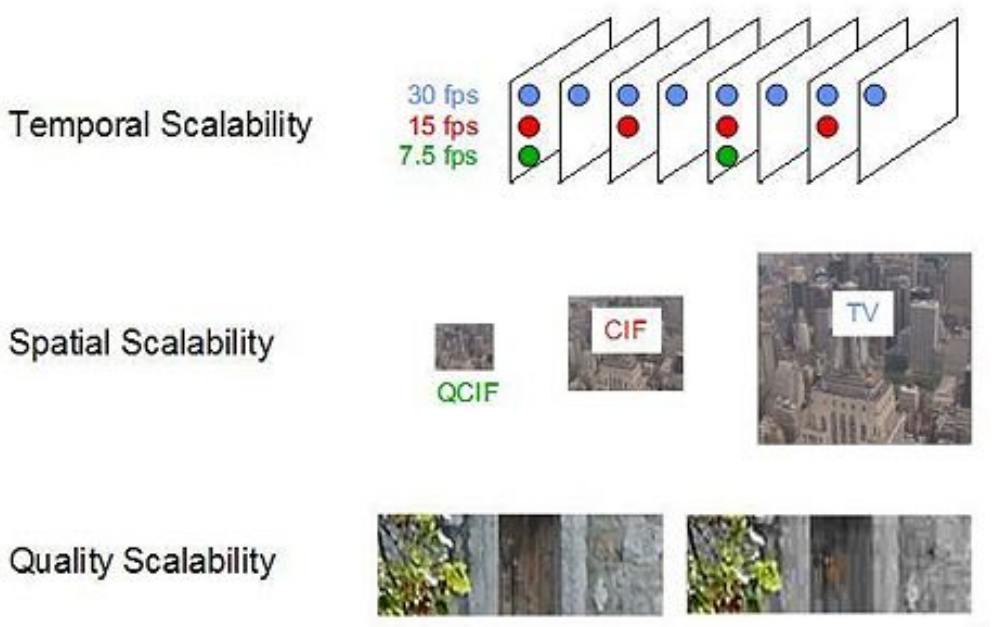

그림 2.1 H.264 SVC 확장성의 종류

(http://ip.hhi.de/imagecom_G1/savce/index.htm)

1.2 H.264 SVC 비트스트림 구조

H.264 AVC의 구조와 마찬가지로 VCL(Video Coding Layer)와 NAL(Network Abstraction Layer)로 구분된다.

VCL은 H.264 SVC로 부호화된 데이터를 나타내며 VCL데이터는 NAL Unit을 통하여 외부와 통신할 수 있게 된다. H.264 SVC로 부호화된 모든 데이터는 NAL Unit으로 구성된다(S. Wenger 외 3명, 2010).

NAL Unit Header는 4바이트로 구성되며 내용은 아래와 같다.

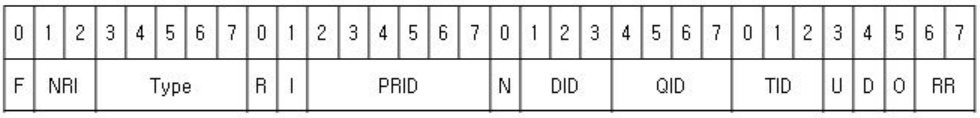

그림 2.2 NAL Unit Header

표 2.1 NAL Unit Header 필드 설명

헤더 필드	사이즈(bits)	설 명
F	1	• forbidden_zero_bit • 문법에 어긋나면 1, 아니면 0
NRI	2	• nal_ref_idc • 00일 경우 예측을 위해 참조 픽처를 사용하지 않음 • 00보다 클 경우 예측을 위해 참조 픽처 사용
Type	5	• nal_unit_type • 표 2.2 참조
R	1	• reserved_one_bit • 미래를 위해 예약된 비트, 무조건 1
I	1	• idr_flag • 1일 경우 IDR 픽처를 초기화 • 0일 경우 IDR 픽처를 초기화 하지 않음
PRID	6	• dependency_id • NAL Unit의 우선순위 표시
N	1	• no_inter_layer_pred_flag • Coded Slice NAL Unit이 출력될 때 내부 예측에 Coded Slice를 사용할것인지를 표시(1이면 사용, 0이면 사용안함)
DID	3	• dependency_id • 표현 계층에서 의존 레벨을 나타냄
QID	4	• quality_id • MGS 표현 계층의 질적 레벨을 나타냄
TID	3	• temporal_id • 해당하는 계층의 시간적 레벨을 나타냄
U	1	• use_ref_base_pic_flag • 1일 경우 내부 예측에 참조 기본 픽처만 사용
D	1	• discardable_flag • 1일 경우 현재 NAL Unit보다 큰 dependency_id의 값을 가지는 NAL Unit을 복호화하지 않음
O	1	• output_flag • 복호화된 픽처가 출력 프로세스에 영향을 줄것인지 표시
RR	2	• reserved_three_2bits • 미래를 위해 예약된 비트, 무조건 11

표 2.2 NAL Unit Type

식별자	설 명
0	UNSPECIFIED_0
1	CODED_SLICE
2	CODED_SLICE_DATAPART_A
3	CODED_SLICE_DATAPART_B
4	CODED_SLICE_DATAPART_C
5	CODED_SLICE_IDR
6	SEI
7	SPS
8	PPS
9	ACCESS_UNIT_DELIMITER
10	END_OF_SEQUENCE
11	END_OF_STREAM
12	FILLER_DATA
13	SPS_EXTENSION
14	PREFIX
15	SUBSET_SPS
16	RESERVED_16
17	RESERVED_17
18	RESERVED_18
19	AUX_CODED_SLICE
20	CODED_SLICE_SCALABLE
21	RESERVED_21
22	RESERVED_22
23	RESERVED_23

제 2 절 JSVM Reference Software

JSVM(Joint Scalable Video Model) Reference Software(http://ip.hhi.de/imagecom_G1/savce/downloads/SVC-Reference-Software.htm)는 ISO/IEC의 MPEG(Moving Pictures Experts Group)과 ITU-T의 VCEG(Video Coding Experts Group)의 연합인 JVT(Joint Video Team)에서 SVC표준화 과정에서 개발된 소프트웨어이다. H.264 SVC를 위한 연구는 계속 진행중이며 JSVM Reference Software 역시 연구 진행에 따라서 계속 갱신된다. C++로 작성되어져 있고 이에 대한 모든 소스코드가 공개되어 있다.

이번절에서는 최신버전의 JSVM Reference Software를 구하는 방법과 이에 포함된 라이브러리를 설명한다.

2.1 최신 버전의 JSVM 구하는 방법

최신 버전의 JSVM을 제공하기 위하여 Rheinisch-Westfälische Technische Hochschule (RWTH) Aachen에서 CVS 서버를 운영하고 있다. CVS서버는 WinCVS 또는 다른 어떤 CVS 클라이언트를 사용하여 접근할 수 있다. CVS 서버에 접속하기 위하여 다음표에 나온 파라미터를 이용하면 된다.

표 2.3 CVS 접근 파라미터

authentication:	pserver
host address:	garcon.ient.rwth-aachen.de
path:	/cvs/jvt
user name:	jvtuser
password:	jvt.Amd.2
module name:	jsvm or jsvm_red

다음은 JSVM 소프트웨어 접근을 위한 CVS 클라이언트 명령이다.

표 2.2 JSVM 소프트웨어 접근을 위한 CVS 클라이언트 명령

```
cvs -d   :pserver:jvtuser:jvt.Amd.2@garcon.ient.rwth-aachen.de:/cvs/jvt
login
cvs -d    :pserver:jvtuser@garcon.ient.rwth-aachen.de:/cvs/jvt checkout
jsvm
```

2.2 JSVM 폴더 구조

다음 표는 CVS 클라이언트를 이용하여 JSVM 소프트웨어를 다운로드 받았을 때 생성된 폴더들의 구조이다.

표 2.3 JSVM 소프트웨어 폴더 구조

폴더 이름	내 용
bin	소프트웨어 빌드 후에 생성되는 바이너리들이 존재
lib	소프트웨어 빌드 후에 생성되는 라이브러리들이 존재
JSVM	JSVM 소프트웨어의 소프코드와 프로젝트 파일들이 존재
JSVM/H264Extension/build	마이크로소프트 비주얼 스튜디오에서 컴파일을 위한 workspace 파일들과 리눅스에서 컴파일을 위한 makefile이 존재
JSVM/H264Extension/data	Encoder의 환경설정 예제 파일들이 존재.
JSVM/H264Extension/include	JSVM에서 제공하는 라이브러리들에 필요한 헤더파일들이 존재.
JSVM/H264Extension/src	JSVM에서 제공하는 라이브러리들과 테스트 프로젝트들의 소스 파일과 헤어파일이 존재.
MVC-Configs	다시점 부호화를 위한 환경설정 예제 파일이 존재하고 다시점 부호화를 위한 간단한 설명도 존재.

2.2 JSVM 라이브러리

다음은 JSVM에 포함된 라이브러리 목록이다. 라이브러리 파일의 이름은 컴파일시 release/debug 모드에 따라 그 이름이 결정된다. 라이브러리 이름의 맨 끝이 d로 끝나게 되면 debug모드의 라이브러리 파일임을 나타낸다.

표 2.4 JSVM Reference Software에서 제공하는 라이브러리 목록

라이브러리 이름	설 명
H264AVCCommonLibStatic	인코더와 디코더에서 사용하는 클래스들을 제공한다. 예를 들면 매크로 블록 데이터 구조, 이미지 데이터 저장과 접근을 위한 버퍼와 디블록킹을 위한 알고리즘이 있다.
H264AVCEncoderLibStatic	인코더에만 사용되는 클래스들을 제공한다. 예를 들면 움직임예측, 모드결정, 엔트로피 부호화를 위한 클래스가 있다.
H264AVCDecoderLibStatic	디코더에만 사용되는 클래스들을 제공한다. 예를 들면 엔트로피 복호화와 비트스트림 파싱을 위한 클래스가 있다.
AvcRewriterLibStatic	SVC 비트스트림을 AVC 비트스트림을 바꿔주는 기능을 수행한다. H.264AVCDecoderLibStatic과 소스 파일을 공유하면 컴파일시 SHARP_AVC_REWRITE_OUTPUT 옵션이 설정된 경우에만 컴파일된다.
H264AVCVideoIoLibStatic	YUV형태의 raw 비디오 데이터를 읽고 쓰기 위한 클래스들과 NAL 유닛을 읽고 쓰는 클래스들을 제공한다.

제 3 절 안드로이드 플랫폼

3.1 개요

세계 각국의 이동통신 관련 회사 연합체인 OHA(Open Handset Alliance)가 개발하여 2007년 11월에 공개하였는데, 실질적으로는 세계적 검색엔진 업체인 구글(Google)이 주도하였으므로 '구글 안드로이드'라고도 한다.

안드로이드는 리눅스2.6 커널을 기반으로 강력한 운영체제와 포괄적 라이브러리 세트, 풍부한 멀티미디어 사용자 인터페이스, 폰 애플리케이션 등을 제공한다. 휴대폰뿐 아니라 다양한 정보 가전 기기에 적용할 수 있는 연동성도 갖추고 있다.

안드로이드가 기존의 휴대폰 운영체제인 Windows Mobile이나 Symbian과 차별화되는 것은 완전 개방형 플랫폼이라는 점이다. 소스 코드를 모두 공개함으로써 누구라도 이를 이용하여 소프트웨어와 기기를 만들어 판매할 수 있도록 하였다. 개발자들은 이를 확장, 대체 또는 재사용하여 사용자들에게 풍부하고 통합된 모바일 서비스를 제공할 수 있게 된 것이다.

안드로이드를 탑재한 휴대폰 단말기를 안드로이드폰이라고 하며, 이 플랫폼에서 응용할 수 있는 애플리케이션을 거래하는 온라인 공간을 안드로이드 마켓이라고 한다.

개발 언어는 자바형식의 자체적인 API를 지원하고 자바 API의 일부분을 지원한다. C와 C++로 개발되어진 외부 라이브러리를 제약적으로 사용이 가능하다.

3.2 안드로이드 구조

다음 그림은 안드로이드의 중요한 구성요소를 보여준다 (http://developer.android.com).

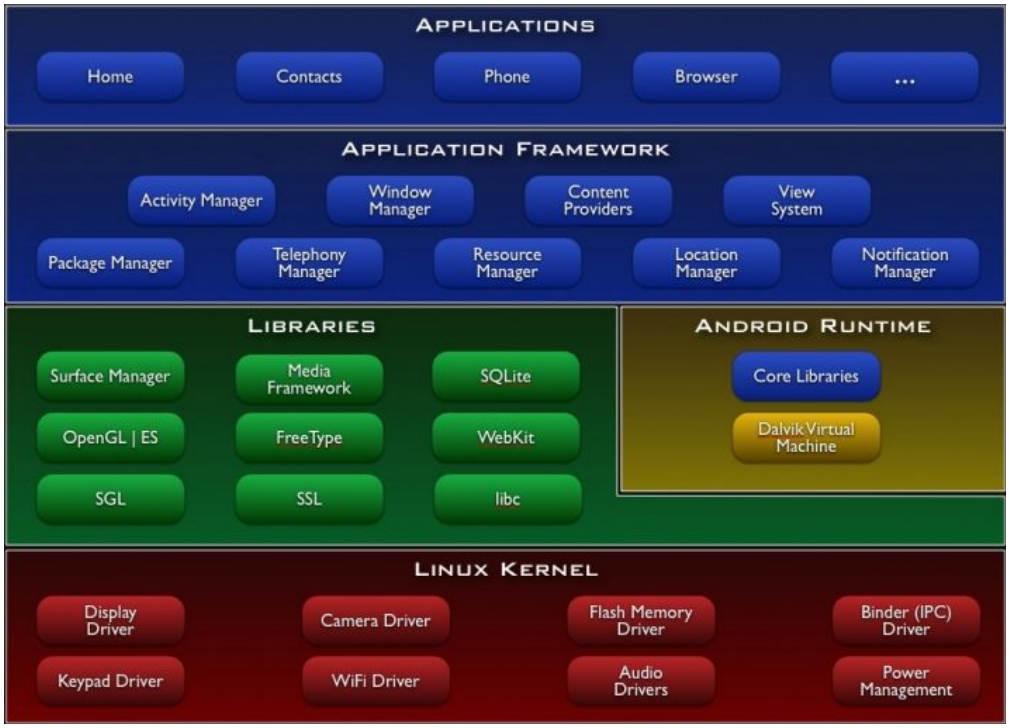

그림 2.3 안드로이드 구조도

3.2.1 Applications

안드로이드는 email 클라이언트, SMS 프로그램, 달력, 지도, 웹 브라우저등을 포함하는 핵심 프로그램들을 기본적으로 포함하고 있다. 모든 프로그램은 자바 언어로 만들어져있다.

3.2.2 Application Framework

개방형 개발 플랫폼을 제공함으로써, 안드로이드는 개발자들이 풍부하고 혁신적인 프로그램을 개발할 수 있도록 한다. 개발자들은 무료로 하드웨어의 이점을 취하고, 위치 정보에 접근하고, 백그라운드 서비스를 수행하고, 상태바에 알림 기능을 추가하는 등 많은 일을 할 수 있다.

개발자들은 핵심 프로그램에서 사용된 것과 동일한 프레임워크 API에 완전하게 접근할 수 있다. 프로그램 구조는 구성요소를 간단하게 재사용할 수 있도록 설계되었는데, 어떤 프로그램이라도 그 기능을 배포하고, 다른 어떤 프로그램이라도 그 기

능을 사용할 수 있다(프레임워크에 의해 강제적으로 보안 제약사항의 적용을 받는다). 같은 방식으로 사용자는 구성요소를 교체할 수 있다.

3.2.3 Libraries

안드로이드에는 다수의 C 또는 C++로 개발된 라이브러리가 포함되어 있으며, 안드로이드 시스템의 다양한 요소요소에 사용되고 있다. 이 기능들은 Application Framework를 통하여 개발자들에게 공개되어 있다.

3.2.4 Android Runtime

Android Runtime 계층에서는 달빅 VM(Dalvik Virtual Machine)과 코어 자바 라이브러리를 포함한다.

달빅은 모바일 단말기용으로 최적화된 VM이다. 모든 안드로이드용 프로그램은 달빅에 의하여 실행된다.

3.2.5 Linux Kernel

안드로이드는 보안, 메모리관리, 프로시스 관리, 네트워크 스택, 드라이버 같은 핵심 시스템 서비스에 있어서 Linux 버전 2.6에 의존하고 있다.

3.3 NDK

안드로이드에서는 기본적으로 자바언어를 사용하지만 C나 C++로 개발된 외부 라이브러리를 가져다 쓸수 있도록 지원하기 위하여 NDK를 제공한다.

NDK가 제공하는 것은 다음과 같다

- tools과 빌드 도구 (c, c++ 소스에서 native code 라이브러리를 제작하는데 사용)
- 어플리케이션 패키지 파일(.apks)들에 대응하는 native 라이브러리를 내장 하는 방법. 이 패키지 파일(.apks)이 안드로이드 장치에 배포된다.
- navtive 시스템의 헤더파일들과 라이브러리들.
- 예제와 설명서

제 4 절 스트리밍 시스템

스트리밍이란 네트워크를 통하여 전송되는 음성 또는 영상 데이터를 끊임없고 지속적으로 처리할 수 있는 기술을 의미한다(http://100.naver.com/100.nhn?docid=769519). 스트리밍 기술은 인터넷의 성장과 함께 더욱더 중요해지고 있는데, 그 이유는 대부분의 사용자들이 대용량 멀티미디어 파일들을 즉시 다운로드할 만큼 빠른 접속회선을 가지고 있지 못하기 때문이다.
데이터의 재생 방식에 따라서 VOD(Video On Demand)방식과 Live방식으로 나눌 수 있다.

VOD방식은 미리 생성해 둔 파일을 서버에 저장해 두고 사용자가 요청시에 전송해 주는 방식이고 Live방식은 실시간으로 인코딩된 데이터를 전송하는 방식이다.

스트리밍 서버와 클라이언트 사이에 세션을 설정하고 정보를 교환하기 위하여 RTSP(Real Time Streaming Protocol) 프로토콜이 표준화 되었고, RTSP 프로토콜은 데이터를 전송할 때 RTP(Real-time Transport Protocol)을 사용한다.

4.1 RTSP

4.1.1 개요

RTSP(Real Time Streaming Protocol, 실시간 스트리밍 프로토콜)은 IETF가 1998년에 개발한 통신 규약이고 RFC 2326(http://www.ietf.org/rfc/rfc2326.txt)에 정의되어 있다. RTSP는 스트리밍 시스템에 사용되며, 스트리밍 서버와 클라이언트 사이의 세션 연결 및 정보 교환을 할 때 쓰인다.

RTSP를 사용하여 실제 미디어 스트리밍 데이터를 전송하지는 않고 대부분의 RTSP 서버는 UDP 또는 TCP를 이용한 RTP 규약을 사용해서 전송 계층으로 음성 또는 영상 데이터를 전송한다.

4.1.2 RTSP 패킷 구조

RTSP 요청 패킷은 Method, URL, 버전 헤더필드로 구성되고 응답 패킷은 버전, 응답코드 및 헤더필드로 구성된다.

가. RTSP URL

요청하는 컨텐츠 자원의 위치를 지정하는데 사용된다.

나. RTSP 버전

스트리밍 서버 및 클라이언트가 사용하는 버전을 명시함으로써 버전간의 충돌을 막기 위하여 사용된다.

다. RTSP 헤더필드

RTSP 각 메시지 내에서 사용하는 요소들을 정의한다. 아래 표는 RTSP 메시지에서 사용되는 주요 헤더와 그에 대한 설명이다.

표 2.5 RTSP 주요 헤더 필드

헤 더	설 명
Server	스트리밍 서버 소프트웨어 정보
User-Agent	스트리밍 클라이언트 소프트웨어 정보
Session	클라이언트의 세션 ID
CSeq	요청 메시지와 응답 메시지의 동기를 맞추기 위한 Sequence Number
Transoprt	컨텐츠 전송 방식 및 포트번호
Range	컨텐츠의 재생 구간

라. RTSP 응답코드

요청 패킷에 대한 처리 결과를 담고 있다. 아래 표는 응답코드의 유형이다.

표 2.6 RTSP 응답코드

응답 코드	설 명
1XX - Informational	요청을 수신하였고 처리가 진행중
2XX - Success	수신 및 처리 완료
3XX - Redirection	요청한 처리를 완료하기 위해서는 더 많은 정보 필요
4XX - Client Error	요청 패킷의 문법이 잘못 되었거나 요청을 수행할수 없음
5XX - Server Error	요청 패킷의 문제는 없지만 서버에서 수행할 수 없음

4.1.3 RTSP Method

RTSP의 Method는 처리해야할 작업을 지정하는 것이다. RTSP의 주요 Method는 다음 표와 같다.

표 2.7 RTSP Method 목록

Method 이름	전달 방향
OPTION	C->S, S->C(Optional)
DESCRIBE	C->S
SETUP	C->S
PLAY	C->S
PAUSE	C->S
TEARDOWN	C->S

가. OPTION

지원하는 Method의 리스트를 요청한다.

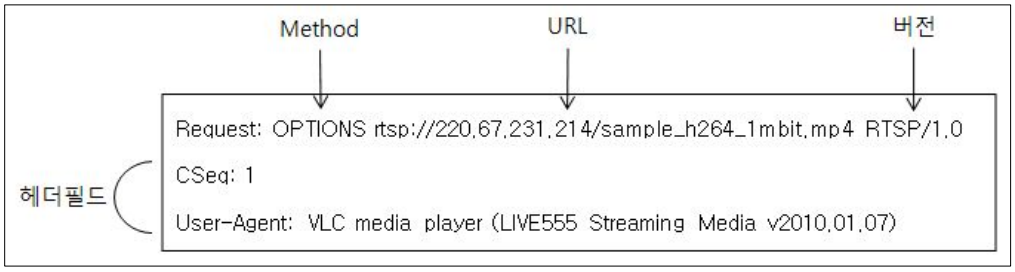

그림 2.4 OPTION 요청 패킷

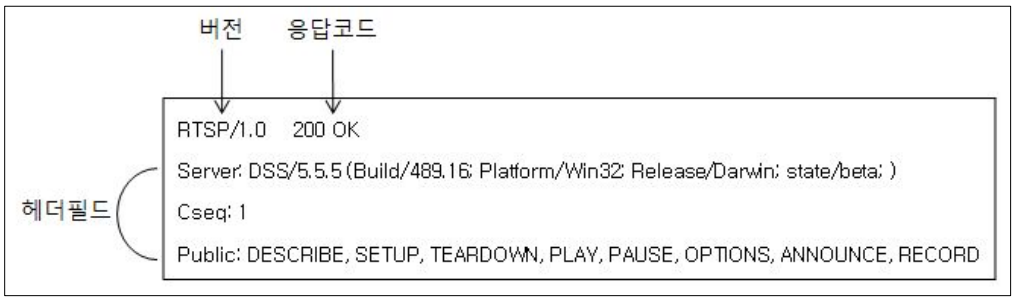

그림 2.5 OPTION 응답 패킷

나. DESCRIBE

서버에게 원하는 컨텐츠에 관한 설명을 요청한다.

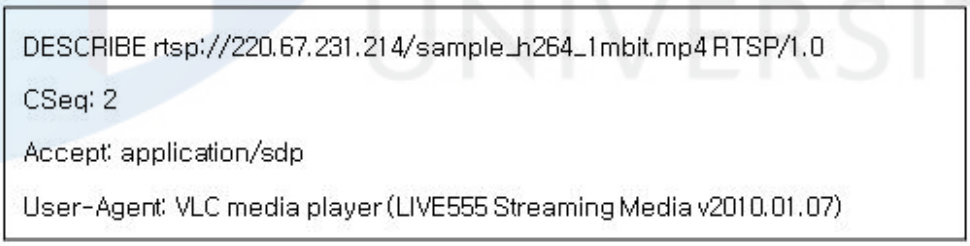

그림 2.6 DESCRIBE 요청 패킷

RTSP/1.0 200 OK

Server: DSS/5.5.5 (Build/489.16; Platform/Win32; Release/Darwin; state/beta;)

Cseq: 2

Last-Modified: Fri, 25 Feb 2005 02:50:40 GMT

Cache-Control: must-revalidate

Content-length: 683

Date: Wed, 23 Jun 2010 06:35:50 GMT

Expires: Wed, 23 Jun 2010 06:35:50 GMT

Content-Type: application/sdp

x-Accept-Retransmit: our-retransmit

x-Accept-Dynamic-Rate: 1

Content-Base: rtsp://220.67.231.214/sample_h264_1mbit.mp4/

profile-level-id=15;mode=AAC-hbr;sizelength=13;indexlength=3;indexdeltaleng

th=3;config=1190

a=mpeg4-esid:101

그림 2.7 DESCRIBE 응답 패킷

```
v=0
o=StreamingServer 3486263747 1109299840000 IN IP4 220.67.231.214
s=₩sample_h264_1mbit.mp4
u=http:///
.
.
.
m=audio 0 RTP/AVP 97
b=AS:127
a=rtpmap:97 mpeg4-generic/48000/2
a=control:trackID=4
a=fmtp:97
```

그림 2.8 DESCRIBE 응답 패킷에 포함된 SDP

다. SETUP

서버와 클라이언트 사이에 컨텐츠 전송 방식 및 포트 번호를 전달한다.

요청 패킷에는 클라이언트의 RTP 수신 포트 번호와 RTCP 포트번호가 순서대로 포함되고 응답 패킷에는 클라이언트의 RTP 수신 포트 번호, RTCP 포트번호, 서버의 RTP 송신 포트번호, RTCP 포트번호가 순서대로 포함된다.

```
SETUP rtsp://220.67.231.214/sample_h264_1mbit.mp4/trackID=3 RTSP/1.0
CSeq: 3
Transport: RTP/AVP;unicast;client_port=9154-9155
User-Agent: VLC media player (LIVE555 Streaming Media v2010.01.07)
```

그림 2.9 SETUP 요청 패킷

```
RTSP/1.0 200 OK
Server: DSS/5.5.5 (Build/489.16; Platform/Win32; Release/Darwin; state/beta; )
Cseq: 3
Last-Modified: Fri, 25 Feb 2005 02:50:40 GMT
Cache-Control: must-revalidate
Session: 94553705039273
Date: Wed, 23 Jun 2010 06:35:50 GMT
Expires: Wed, 23 Jun 2010 06:35:50 GMT
Transport:
RTP/AVP;unicast;source=220.67.231.214;client_port=9154-9155;server_port=6
970-6971;ssrc=000018BE
```

그림 2.10 SETUP 응답 패킷

라. PLAY

서버에게 컨텐츠 전송 시작을 요청한다.

```
PLAY rtsp://220.67.231.214/sample_h264_1mbit.mp4/ RTSP/1.0
CSeq: 7
Session: 67718749385127
User-Agent: VLC media player (LIVE555 Streaming Media v2010.01.07)
```

그림 2.11 PLAY 요청 패킷

```
RTSP/1.0 200 OK
Server: DSS/5.5.5 (Build/489.16; Platform/Win32; Release/Darwin; state/beta; )
Cseq: 7
Session: 67718749385127
Range: npt=3.76667-70.00000
RTP-Info:
url=rtsp://220.67.231.214/sample_h264_1mbit.mp4/trackID=3;seq=26932;rtptim
e=767609,url=rtsp://220.67.231.214/sample_h264_1mbit.mp4/trackID=4;seq=2
5892;rtptime=418611
```

그림 2.12 PLAY 응답 패킷

마. PAUSE

서버에게 컨텐츠 전송 중지를 요청한다.

```
PAUSE rtsp://220.67.231.214/sample_h264_1mbit.mp4/ RTSP/1.0
CSeq: 6
Session: 67718749385127
User-Agent: VLC media player (LIVE555 Streaming Media v2010.01.07)
```

그림 2.13 PAUSE 요청 패킷

```
RTSP/1.0 200 OK
Server: DSS/5.5.5 (Build/489.16; Platform/Win32; Release/Darwin; state/beta; )
Cseq: 6
Session: 67718749385127
```

그림 2.14 PAUSE 응답 패킷

바. TEARDOWN

```
TEARDOWN rtsp://220.67.231.214/sample_h264_1mbit.mp4/ RTSP/1.0
CSeq: 14
Session: 48786533534996
User-Agent: VLC media player (LIVE555 Streaming Media v2010.01.07)
```

그림 2.15 TEARDOWN 요청 패킷

```
RTSP/1.0 200 OK
Server: DSS/5.5.5 (Build/489.16; Platform/Win32; Release/Darwin; state/beta; )
Cseq: 14
Session: 48786533534996
Connection: Close
```

그림 2.16 TEARDOWN 응답 패킷

4.1.5 RTSP 메시지 흐름 및 RTP 데이터 전송

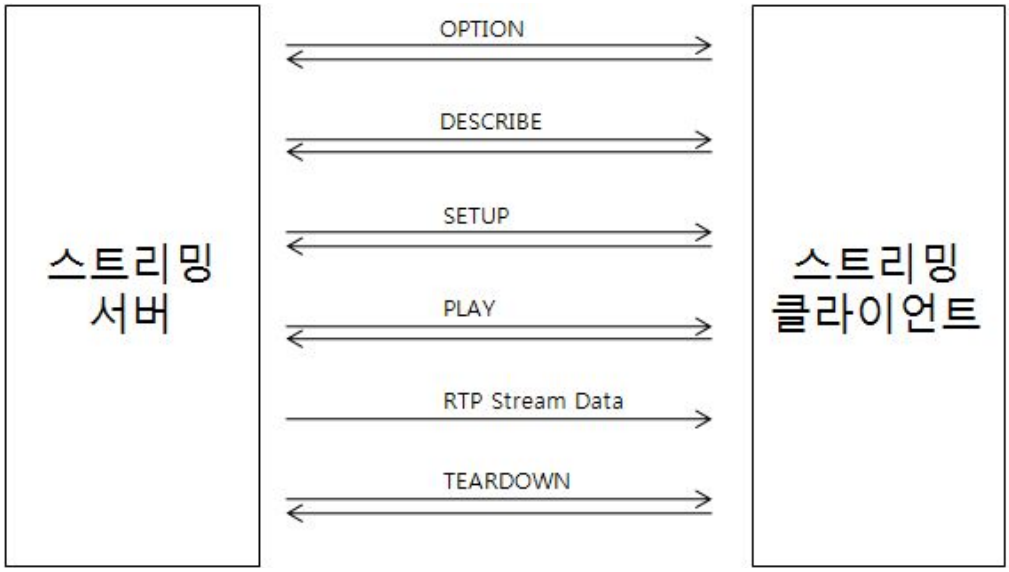

그림 2.17 RTSP 메시지 흐름 및 RTP 데이터 전송

4.1.6 SDP(Session Description Protocol)

SDP는 스트리밍 미디어의 초기화와 설명을 위한 포맷이다. 이 규격은 IETF의 RFC 4566(http://tools.ietf.org/html/rfc4566)로 규정되어 있다.

SDP는 세션 공지, 세션 초대, 그리고 그 밖의 멀티미디어 세션 초기화를 위한 폼들을 목적으로 멀티미디어 세션들의 기술을 위해 작성되었다. SDP는 미디어 폼의 콘텐츠 그 자체를 위해서 제공된 것은 아니지만, 양 끝단 간에 미디어 타입과 포맷에 대해 협상할 수 있는 수단을 제공한다. 이로 인해서 SDP는 새롭게 추가되는 미디어 타입과 포맷을 지원할 수 있으며, 앞으로 나올 기술에 대한 호환성을 시스템적으로 지원할 수 있다.

SDP는 Session Announcement Protocol (SAP)의 한 부분으로 시작되었지만, RTP, RTSP, SIP 와 멀티캐스트 세션을 기술하기 위하여 사용 가능하다.

아래 그림은 DESCRIBE Method의 응답 패킷에 포함되어 전송되는 SDP의 예이다.

```
v=0
o=StreamingServer 3486263747 1109299840000  IN IP4 220.67.231.214
s=₩sample_h264_1mbit.mp4
u=http:///
e=admin@
c=IN IP4 0.0.0.0
b=AS:2097279
t=0 0
a=control:*
a=isma-compliance:2,2.0,2
a=range:npt=0- 70.00000
m=video 0 RTP/AVP 96
b=AS:2097151
a=rtpmap:96 H264/90000
a=control:trackID=3
a=cliprect:0,0,480,380
a=framesize:96 380-480
a=fmtp:96
packetization-mode=1;profile-level-id=4D401E;sprop-parameter-sets=J01AHqkYMB73oA==,KM4C+IA=
a=mpeg4-esid:201
m=audio 0 RTP/AVP 97
b=AS:127
a=rtpmap:97 mpeg4-generic/48000/2
a=control:trackID=4
a=fmtp:97
```

그림 2.18 SDP 예제

4.2 RTP(Real-time Transport Protocol)

4.2.1 개요

RTP는 IETF에서 표준화된 오디오와 비디오와 같은 실시간 데이터를 전송하기 위한 인터넷 프로토콜이며 이 규격은 RFC 3350(http://tools.ietf.org/html/rfc3550)으로 정의되어 있다. RTP 그 자체가 데이터의 실시간 전송을 보장하지는 않는다. RTP는 일반적으로, UDP 프로토콜을 이용하고 특정 상황에서 TCP 프로토콜을 이용하기도 한다.

4.2.2 RTP 패킷 구조

RTP 패킷은 12바이트의 헤더와 가변적인 크기의 페이로드로 나누어지며 헤더의 구조는 다음 그림과 같다.

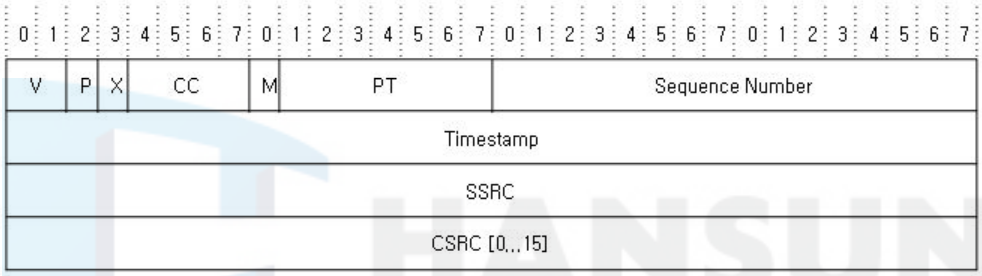

그림 2.19 RTP 패킷 헤더 구조

다음 표는 RTP 패킷 헤더의 각 부분에 대한 설명이다.

표 2.8 RTP 패킷 헤더

필드	크기(bit)	설명
V(Version)	2	• RTP 버전을 나타낸다
P(Padding)	1	• 페이로드의 일부가 아닌 추가적인 바이트가 있는지의 여부. • 패딩의 마지막 바이트에는 무시해야 하는 패딩 바이트의 길이에 대한 정보.
X(Extension)	1	• 고정 RTP 헤더(12bytes) 이후에 하나의 확장 헤더가 있는지에 대한 정보.
CC(CSRC Count)	4	• 고정 RTP 헤더 이후에 있는 CSRC의 개수.
M(Marker)	1	• 패킷 스트림 내에서 프레임의 경계와 같은 중요한 이벤트를 표시하는데 사용.
PT(Payload Type)	7	• RTP 페이로드의 형식. • 수신하는 측에서는 모르는 형식일 경우 무시.
Sequence Number	16	• RTP 데이터 패킷을 보낼 때 마다 하나씩 증가시키는 순서 번호. • 수신측에서는 이 값을 이용하여 패킷 손실을 감지하고, 패킷 순서 복구에 사용. • 순서 번호 초기 값은 보안상의 이유로 무작위하게 생성.
Timestamp	32	• RTP 데이터 패킷의 샘플링 순간을 반영. • 초기값은 보안상의 이유로 무작위로 결정. • 데이터 동기화 및 지터 계산에 사용.
SSRC(Synchronization Source)	32	• 동기화 소스에 대한 식별자. • 무작위한 값을 사용하며, 같은 RTP 세션 내에서는 유일한 값.
CSRC(Contributing Source)	32	• 해당 RTP 패킷의 페이로드를 구성하는데 기여한 SSRC 의 리스트. • 최대 갯수는 15개.

RTP 페이로드는 RTP 헤더 다음에 전송되는 부호화된 영상 또는 음성

데이터이다. 페이로드의 구조는 부호화된 데이터의 형식에 따라서 다를 수도 있다.

다음 그림은 Wireshark(http://www.wireshark.org/)를 이용하여 캡처한 실제로 전송되는 RTP 패킷의 내용이다.

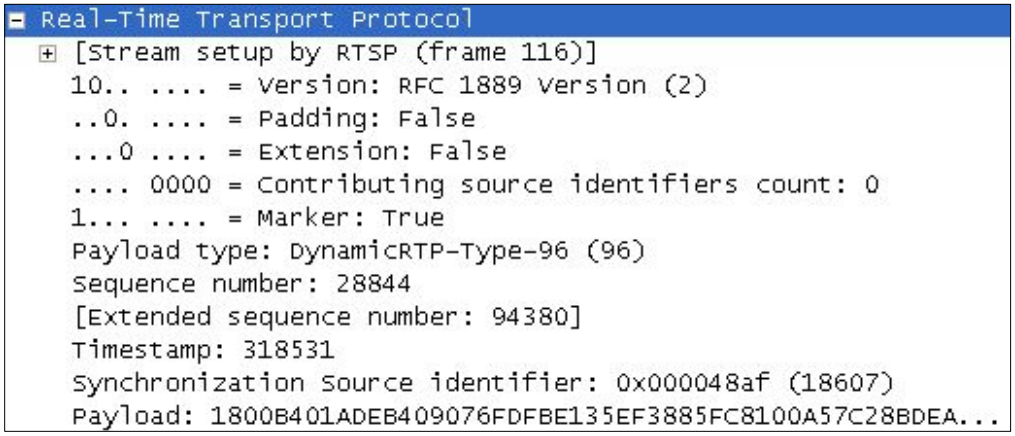

그림 2.20 RTP 패킷의 내용

제 3 장 H.264 SVC 스트리밍 시스템 모델링

제 1 절 시스템 구성

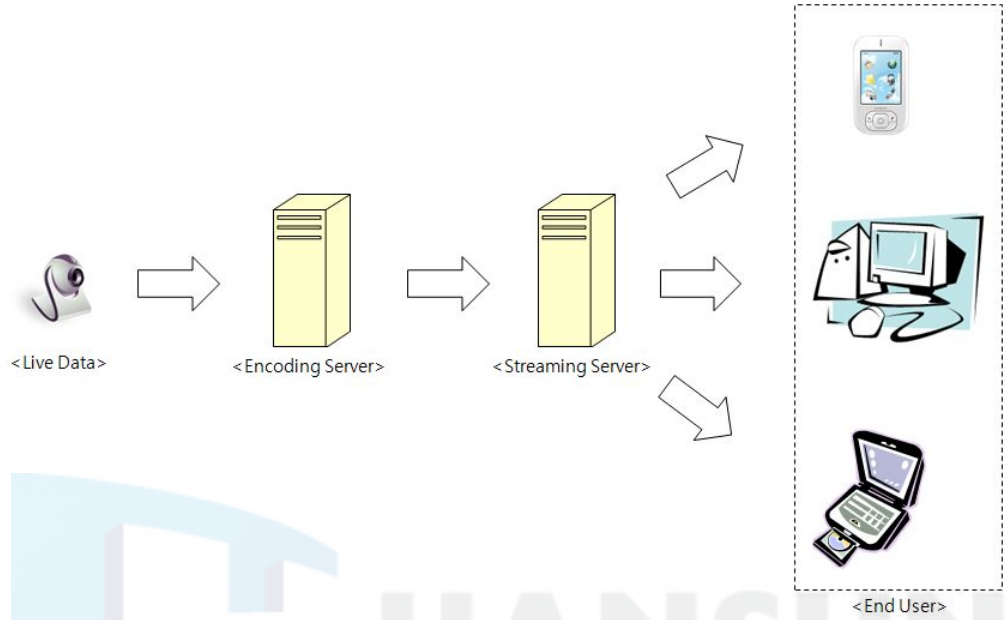

그림 3.1 스트리밍 시스템 구성

• Encoding Server

라이브 비디오 소스를 H.264, MPEG4등과 같은 형식을 이용하여 부호화한 후 Streaming Server로 전송하는 서버

• Streaming Server

인코딩 서버로부터 수신받은 데이터를 사용자들에게 분배하는 서버

• End User

스트리밍 서버로부터 수신한 데이터를 재생하는 사용자 컴퓨터 또는 모바일 단말기

제 2 절 동작 원리

2.1 스트리밍

인코딩 서버로부터 부호화된 데이터가 스트리밍 서버를 통하여 End User까지 데이터가 전송되는 과정이다.

인코딩 서버는 라이브 데이터를 인코딩하여 RTP 패킷으로 만든 후 스트리밍 서버에 전달한다. 스트리밍 서버는 수신한 RTP 패킷을 버퍼에 저장하고 있다가 RTSP 세션이 맺어진 End User에게 수신한 RTP 패킷을 전송한다.

2.2 End User의 스트림 데이터 요청

End User는 스트리밍 서버와 RTSP 세션을 맺음으로써 연결이 된다. 라이브 데이터를 수신하기 위하여 스트리밍 서버에게 SDP파일을 요청한다.

스트리밍 서버는 End User가 SDP 파일을 요청하였을때부터 인코딩 서버가 송신하는 RTP 패킷을 수신한다. 인코딩 서버로부터 수신한 RTP 패킷은 버퍼링이 완료된 후에 End User로 송신한다.

제 3 절 H.264 SVC Encoder

3.1 개요

본 논문에서는 RAW 데이터를 H.264 SVC 영상으로 부호화하기 위하여 JSVM 소프트웨어에서 제공하는 인코더를 사용하였다.

JSVM 소프트웨어에서 제공하는 H.264 Encoder는 환경설정파일에 따라서 H.264 AVC 또는 SVC로 부호화가 가능하다 http://ip.hhi.de/imagecom_G1/savce/downloads/SVC-Reference-Software.htm).

3.1.1 H.264 AVC 모드 환결설정 파일

다음 표는 H.264 AVC모드로 부호화하는 환경설정파일의 예이다.

표 3.1 H.264 AVC를 위한 환경설정 파일의 예

AVCMode	1
InputFile	akiyo_qcif.yuv
OutputFile	stream.264
ReconFile	rec.yuv
SourceWidth	176
SourceHeight	144
FrameRate	30.0
FramesToBeEncoded	300
SymbolMode	1
DPBSize	16
NumRefFrames	16

- AVCMode

H.264영상을 AVC모드로 부호화할 것인지 SVC모드로 부호화할것인지

를 나타내는 플래그 값이다.

1일 경우 H.264 AVC모드로 부호화되고 0일 경우 H.264 SVC모드로 부호화된다.

• InputFile

부호화될 입력영상의 파일명을 나타낸다.

• OutputFile

부호화되어 저장될 파일명을 나타낸다.

• ReconFile

부호화된 파일을 복원했을 때 얻어지는 복원된 영상에 대한 파일명을 나타낸다.

• SourceWidth

입력 비디오 시퀀스의 가로 해상도를 나타낸다.

• SourceHeight

입력 비디오 시퀀스의 세로 해상도를 나타낸다.

• FrameRate

입력 비디오 시퀀스의 프렘임율(Hz)을 나타낸다.

• FramesToBeEncoded

입력 비디오 시퀀스 중에서 부호화할 프레임 수를 나타낸다.

• SymbolMode

사용할 엔트로피 부호화 방법을 나타낸다. 0으로 되어 있으면 컨텐스트 기반의 가변길이 부호화(VLC, variable length codes)가 사용되고, 1로 설정되어 있으면 컨텐스트 기반의 이진 산술부호화(CABAC, context-based binary arithmetic coding)가 사용된다.

• DPBSize

DPB(Decoded Picture Buffer)의 최소 크기를 나타낸다.

3.1.2 H.264 SVC 모드 환결설정 파일

가. 기본 환경설정 파일

H.264 SVC로 부호화할 경우에는 환경설정 파일에 AVCMode 변수를

사용하는 것은 선택사항이다. 즉 사용하지 않아도 된다.

다음 표는 H.264 SVC 모드로 부호화하기 위한 메인 환경설정파일의 예이다.

표 3.2 H.264 SVC를 위한 환경설정 파일의 예

OutputFile	test.264
FrameRate	30
MaxDelay	1200.0
FramesToBeEncoded	30
GOPSize	16
IntraPeriod	16
NumberReferenceFrames	1
BaseLayerMode	1
NumLayers	4
LayerCfg	layer0.cfg
LayerCfg	layer1.cfg
LayerCfg	layer2.cfg
LayerCfg	layer3.cfg

- OutputFile

  부호화되어 저장될 파일명을 나타낸다.

- FrameRate

  입력 비디오 시퀀스의 프렘임율(Hz)을 나타낸다.

- MaxDelay

  부호화 과정에서 생기는 구조적인 시간 지연에 대한 최대 허용값을 ms 단위로 나타낸 것이다.

- FramesToBeEncoded

  입력 비디오 시퀀스 중에서 부호화할 프레임 수를 나타낸다.

- GOPSize

일반적으로 P Picture 사이의 거리라고 생각하면 된다.

• IntraPeriod

주기적인 Intra Picture 삽입주기를 나타내며 -1인 경우 첫 프레임에만 Intra Picture로 부호화한다.

• NumberReferenceFrames

참조 프레임의 개수를 나타낸다.

• BaseLayerMode

기본계층 부호화 모드를 나타내며 3가지 경우로 나누어진다.

0, 1 - SEI 메시지가 없는 AVC

2 - 시간적 확장성을 갖기 위해 SEI 메시지를 갖는 AVC

• NumLayers

공간 및 화질 계위적 계층을 개수를 나타낸다.

• LayerCfg

메인 환경설정파일 이외에 계위적 계층들을 부호화하기 위해 사용되는 계층별 환경설정파일을 나타낸다.

나. 계위적 계층의 환경설정 파일

메인 환경 설정 파일에 명시되어 각 계위 계층별로 부호화 조건을 나타내기 위한 환경설정파일이다.

아래 표는 계위 계층의 환경설정 파일의 예이다.

표 3.3 H.264 SVC의 계위 계층을 위한 환경설정 파일의 예

SourceWidth	176
SourceHeight	144
FrameRateIn	30
FrameRateOut	30
InputFile	foreman_qcif.yuv
ReconFile	rec_layer0.yuv
SymbolMode	1

• SourceWidth

입력 비디오 시퀀스의 가로 해상도를 나타낸다.

• SourceHeight

입력 비디오 시퀀스의 세로 해상도를 나타낸다.

• FrameRateIn

입력 비디오 시퀀스의 프렘임율(Hz)을 나타낸다.

• FrameRateOut

부호화될 비디오 시퀀스의 프렘임율(Hz)을 나타내고 FrameRateIn보다 클 수 없다.

• InputFile

부호화될 입력영상의 파일명을 나타낸다.

• ReconFile

부호화되어 복원했을 때 얻어지는 복원된 영상에 대한 파일명을 나타낸다.

• SymbolMode

사용할 엔트로피 부호화 방법을 나타낸다. 0으로 되어 있으면 컨텐스트 기반의 가변길이 부호화(VLC, variable length codes)가 사용되고, 1로 설정되어 있으면 컨텐스트 기반의 이진 산술부호화(CABAC, context-based binary arithmetic coding)가 사용된다.

제 4 절 H.264 SVC Extractor

JSVM 소프트웨어에서는 SVC 비트스트림으로부터 원하는 계층의 비트스트림 형태로 추출하는 프로젝트를 제공하고 BitStreamExtractorStatic라는 이름의 프로젝트로 제공된다 (http://ip.hhi.de/imagecom_G1/savce/downloads/SVC-Reference-Software.htm).

4.1 BitstreamExtractor 사용법

표 3.4 BitstreamExtractorStatic 사용법

```
BitStreamExtractorStatic [-pt trace]   <in>
           [<out> [[-e] [-ql |   -qlord]] | [-sl] | [[-l] [-t] [-f]]
           | [-b] | [-et] ]

options:
-pt trace  -> 주어진 스트림으로부터 Trace 파일 추출
-sl SL     -> SL(layer id)에 종속되는 계층들에 대한 파일 추출
-l L       -> dependency_id  <= L 인 모든 계층 추출
-t T       -> temporal_level <= T 인 모든 계층 추출
-f F       -> quality_level  <= F 인 모든 계층 추출
-b B       -> target bitrate = B 인 계층 추출
-e AxB@C:D -> 다음과 같은 계층을 추출
              A: frame width [luma samples]
              B: frame height [luma samples]
              C: frame rate [Hz]
              D: bit rate [kbit/s]
```

인자로 아무것도 주지 않고 파일명만 줬을 경우에는 해당하는 파일이 저장하고 있는 계층에 대한 설명이 다음 그림과 같이 화면에 출력된다.

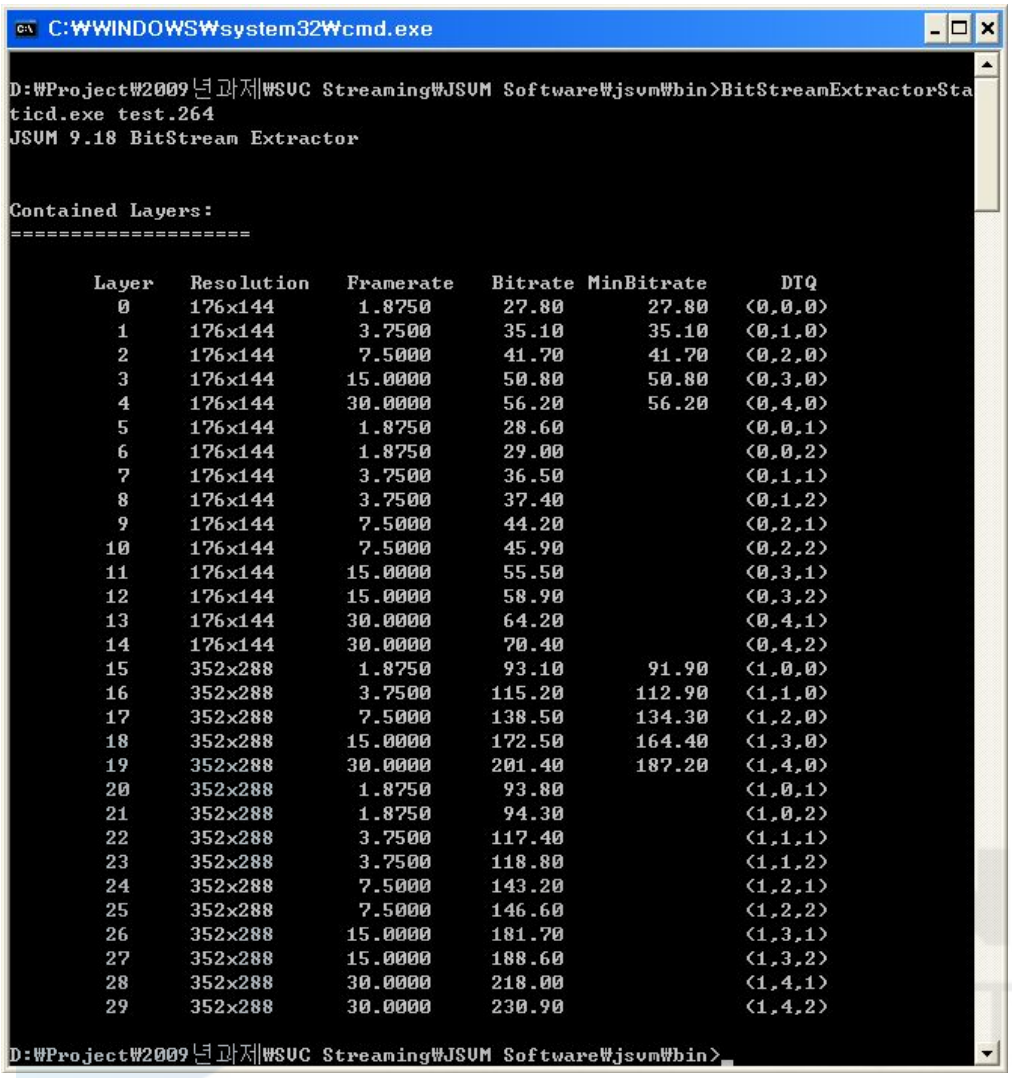

그림 3.2 BitstreamExtractorStatic을 이용하여 비트스트림의 정보를 출력한 모습

제 4 장 안드로이드상에서 H.264 SVC Player 구현

제 1 절 Player 구조

본 논문에서 정의한 H.264 SVC Player의 구조는 다음 그림과 같다.

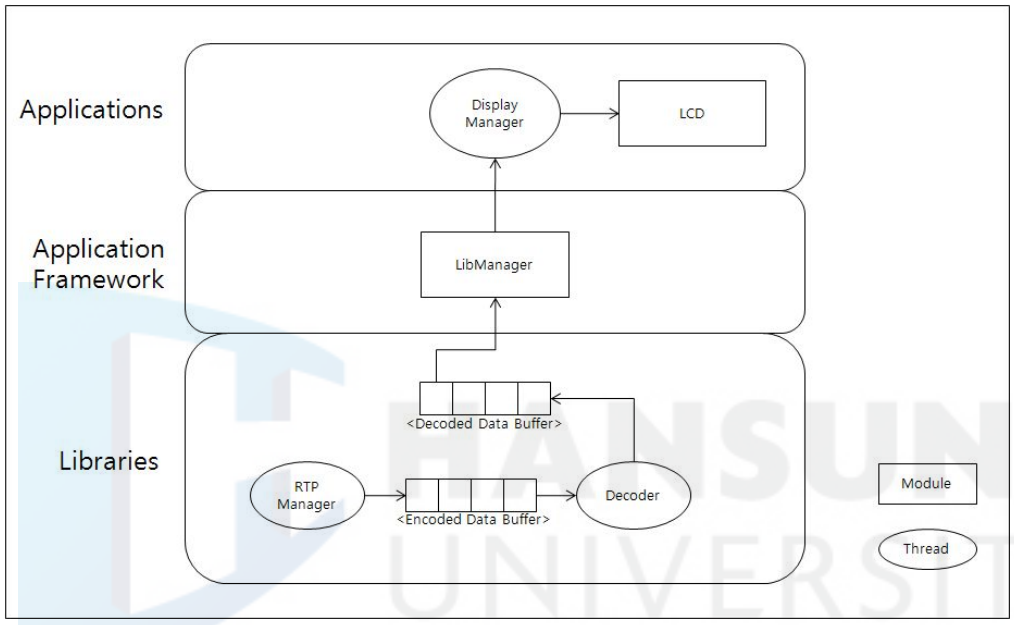

그림 4.1 H.264 SVC Player 구조도

Player의 구조는 Libraries, Application Framework, Apllications 3개의 계층으로 구분할 수 있고 이 계층들은 모두 안드로이드의 기본 계층에 포함된다.

Libraries 계층에는 Player가 동작하는데 필요한 C/C++로 개발되어진 라이브러리가 위치하고 Application Framework 계층에는 라이브러리의 기능을 사용하기 위한 JNI 함수가 위치한다. Applications 계층에서는 Application Framework 계층에서 제공하는 함수를 호출하여 Player를 동

작시킨다.

Player 구현에 사용한 클래스의 목록은 아래 표와 같다.

표 4.1 H.264 SVC Player 구현에 사용한 클래스 목록

클래스 이름	설 명
LibManager	Libraries 계층에 있는 라이브러리들의 기능을 사용할 수 있도록 API제공
DisplayManager	YUV 데이터를 변환과정을 거쳐 Bitmap으로 변환 후 화면에 출력
SStreamBuffer	Encoding/Decoding 된 데이터 저장
RTPManager	RTP 패킷 수신
Decoder	H.264 SVC로 부호화된 데이터를 복호화하여 YUV Data 생성

Player의 동작 구조 중에서 버퍼를 중심으로 본 구조도는 다음 그림과 같다.

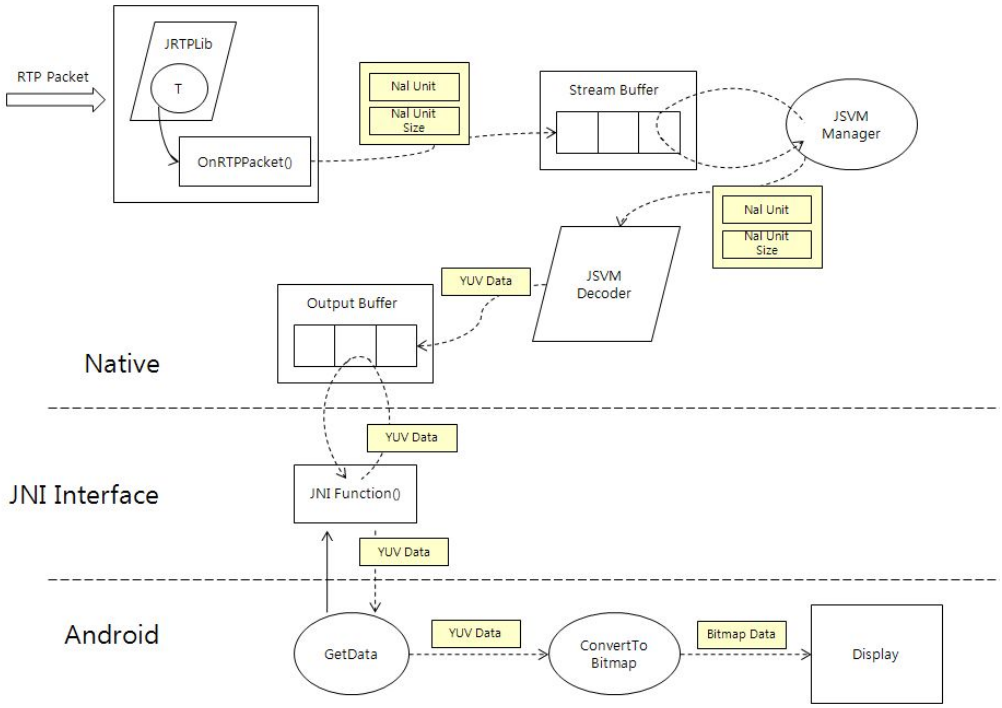

그림 4.2 Player에서의 데이터 전달

Player의 동작에 수행되는 모든 쓰레드는 수행중인 작업이 끝나면 wait 상태가 되고 수행할 작업이 있을 때 notify된다.

RTPManager을 통하여 수신된 RTP 패킷은 Encoded Data Buffer에 저장된다. Encoded Data Buffer에 데이터가 들어가게 되면 Decoder가 wait 상태에서 깨어나 데이터를 꺼내간 후 복호화를 시작한다. 복호화의 결과로 나온 YUV 데이터는 Decoded Data Buffer에 저장된다. Decoded Data Buffer에 값이 들어가면 Display Manager가 wait상태에서 깨어나 데이터를 가져간후 변환 과정을 거쳐 YUV 데이터를 Bitmap 데이터로 변환시키게 되고 이를 화면에 출력하게 된다.

제 2 절 라이브러리 포팅

본 논문에서 구현한 Player에서는 안드로이드에서 제공하지 않는 기능을 사용하기 위하여 C/C++로 개발되어진 라이브러리를 안드로이드에서 사용가능하도록 포팅하여 사용하였다. 포팅한 라이브러리는 stlport, JRTPLib, JSVM Decoder이다.

외부 라이브러리를 안드로이드용으로 포팅하기 위하여 안드로이드에서 제공하는 NDK를 사용하였다.

2.1 stlport

NDK에서는 기본적으로 STL을 지원하지 않는다. 하지만 외부 라이브러리가 STL을 사용하였다면 STL을 지원하도록 하던지 아니면 STL에서 제공하는 클래스와 동일한 기능을 수행하는 클래스를 구현하여야 한다. STL과 동일한 기능의 클래스를 개발하는 것은 효율적이지 못하다고 판단하여 STL 기능을 수행할 수 있도록 하는 라이브러리중 stlport(http://www.stlport.org)를 사용하였다.

안드로이드에서 stlport를 사용하기 위해서는 안드로이드 환경에 맞도록 포팅을 하여 사용하여야 하지만 이미 포팅된 소스(http://www.anddev.org/viewtopic.php?p=29939)가 존재하여 이를 사용하였다.

2.2 JRTPLib

JRTPLib(http://research.edm.uhasselt.be/~jori/page/index.php?n=CS.Jrtplib)은 C++로 개발된 RTP 라이브러리이다.

안드로이드 API중에서 RTP를 지원하는 API가 없기에 이를 사용하였다.

2.2.1 NDK 사용을 위한 디렉토리 구성

JRTPLib은 jthread와 jrtplib으로 구성된다. 이 두 개의 라이브러리를 한

번에 빌드하도록 구성하였다.

jthread 디렉토리 안에 여러개의 폴더와 파일로 구성되어져 있지만 NDK에서 컴파일 할때는 소스파일과 헤더 파일만이 필요하다. 소스 파일과 헤더 파일은 “/src″에 있다.

“{NDK_HOME}/apps/RTPClient/project/jni/jthread”를 만들고 이곳에 jthread의 “/src″의 내용을 복사한다. ″/src″에 있는 파일과 디렉토리를 보면 헤더파일들과 ″pthread″ 디렉토리가 필요하고 ”win32″디렉토리는 필요없으니 삭제한다.

jrtplib 디렉토리 안에 여러개의 폴더와 파일로 구성되어져 있지만 NDK에서 컴파일 할때는 소스파일과 헤더 파일만이 필요하다. 소스 파일과 헤더 파일은 “/src″에 있다.

“{NDK_HOME}/apps/RTPClient/project/jni/jrtplib”를 만들고 이곳에 jrtplib의 “/src″의 내용을 복사한다.

jrtplib은 stl을 사용하였으므로 안드로이드에서 stl을 사용할수 있도록 하기위하여 stlport를 사용한다.

그림 4.3은 설명처럼 파일과 디렉토리를 구성한 모습이다.

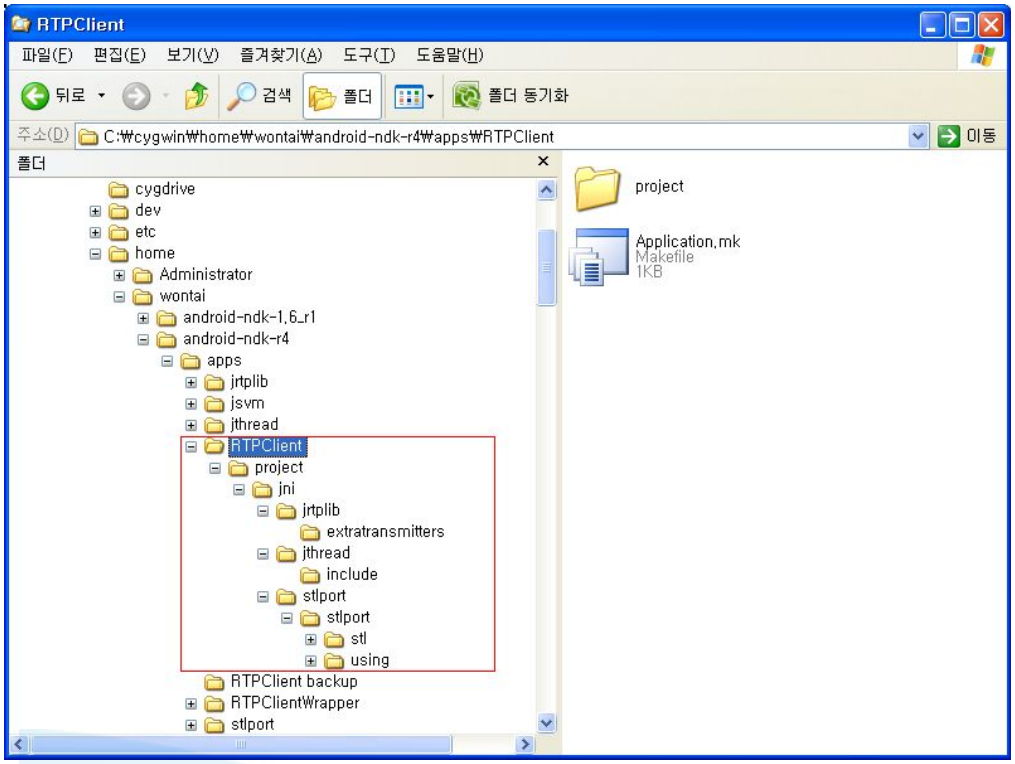

그림 4.3 NDK사용을 위한 디렉토리 구성

2.2.2 ".mk"파일 생성 및 구성

NDK를 사용하기 위하여 "Application.mk"와 "Android.mk"파일을 생성한다.

표 4.2 Application.mk

```
APP_PROJECT_PATH := $(call my-dir)/project
APP_MODULES      := \
        jthread \
        jrtplib
```

표 4.3 jthread의 Android.mk

```
LOCAL_PATH := $(call my-dir)
include $(CLEAR_VARS)
LOCAL_MODULE := jthread
LOCAL_CPP_EXTENSION := .cpp

LOCAL_CFLAGS += -I$(LOCAL_PATH)/include

LOCAL_SRC_FILES := \
        jmutex.cpp \
        jthread.cpp

LOCAL_LDLIBS += -llog

include $(BUILD_STATIC_LIBRARY)
```

표 4.4 jrtplib의 Android.mk

```
LOCAL_PATH := $(call my-dir)
include $(CLEAR_VARS)
LOCAL_MODULE := jrtplib
LOCAL_CPP_EXTENSION := .cpp

JTHREAD_PATH := /home/wontai/android-ndk-r4/apps/RTPClient/project/jni/jthread
OUTPUT_PATH := /home/wontai/android-ndk-r4/out/apps/RTPClient/armeabi
STLPORT_PATH := /home/wontai/android-ndk-r4/apps/RTPClient/project/jni/stlport

LOCAL_CFLAGS += -DMSYS_LINUX -I$(STLPORT_PATH)/stlport \
            -I$(JTHREAD_PATH)/include -DLINUX \
            -D_NEW_ -D__SGI_STL_INTERNAL_PAIR_H -DANDROID -DOS_ANDROID \
           -DRTP_SUPPORT_THREAD
LOCAL_SRC_FILES := \
            rtcpapppacket.cpp rtcpbyepacket.cpp rtcpcompoundpacket.cpp \
            rtcpcompoundpacketbuilder.cpp rtcppacket.cpp rtcppacketbuilder.cpp \
            rtcprrpacket.cpp rtcpscheduler.cpp rtcpsdesinfo.cpp \
            rtcpsdespacket.cpp rtcpsrpacket.cpp rtpcollisionlist.cpp \
            rtpdebug.cpp rtperrors.cpp rtpinternalsourcedata.cpp \
            rtpipv4address.cpp rtpipv6address.cpp rtplibraryversion.cpp \
            rtppacket.cpp rtppacketbuilder.cpp rtppollthread.cpp \
            rtprandom.cpp rtpsession.cpp rtpsessionparams.cpp \
            rtpsessionsources.cpp rtpsourcedata.cpp rtpsources.cpp \
            rtptimeutilities.cpp rtpudpv4transmitter.cpp extratransmitters/rtpfaketransmitter.cpp \
            SStreamData.cpp Content.cpp

LOCAL_LDLIBS += -ldl -llog -L$(OUTPUT_PATH) -ljthread -L$(STLPORT_PATH) -lstlport
LOCAL_STATIC_LIBRARIES := jthread stlport

include $(BUILD_STATIC_LIBRARY)
```

2.2.3 소스 파일 수정

안드로이드에서 사용하는 C/C++ 컴파일러는 C/C++에서 제공하는 모든 기능을 지원하지는 않는다.

가. jthread.cpp

표 4.5 JThread::Kill()원본 소스

```
int JThread::Kill()
{
	runningmutex.Lock();
	if (!running)
	{
		runningmutex.Unlock();
		return ERR_JTHREAD_NOTRUNNING;
	}
	pthread_cancel(threadid);

	running = false;
	runningmutex.Unlock();
	return 0;
}
```

표 4.6 JThread::Kill() 수정된 소스

```
int JThread::Kill()
{
	runningmutex.Lock();
	if (!running)
	{
		runningmutex.Unlock();
		return ERR_JTHREAD_NOTRUNNING;
	}
	//pthread_cancel(threadid);

	int status = 0;

	pthread_exit((void*)NULL);
	pthread_join(threadid, (void **)status);

	running = false;
	runningmutex.Unlock();
	return 0;
}
```

나. rtprandom.h

```
58      /** Returns a random thirty-two bit value. */
59      uint32_t GetRandom32();
60
61      /** Returns a random number between $0.0$ and $1.0$. */
62      double GetRandomDouble();
63  private:
64  #if defined(RTP_SUPPORT_GNUDRAND)
65      struct drand48_data drandbuffer;
66  #elif defined(RTP_SUPPORT_RANDR)
67      unsigned int state;
68  #endif // RTP_SUPPORT_GNUDRAND
69  };
```

그림 4.4 rtprandom.h 원본

```
58      /** Returns a random thirty-two bit value. */
59      uint32_t GetRandom32();
60
61      /** Returns a random number between $0.0$ and $1.0$. */
62      double GetRandomDouble();
63  private:
64  /*#if defined(RTP_SUPPORT_GNUDRAND)
65      struct drand48_data drandbuffer;
66  #elif defined(RTP_SUPPORT_RANDR)
67      unsigned int state;
68  #endif // RTP_SUPPORT_GNUDRAND*/
69  };
```

그림 4.5 rtprandom.h 수정

다. rtprandom.cpp

RTPRandom::Random()의 파일의 내용을 실제로는 주석처리를 해 두었다.

소스중 RTP_SUPPORT_GNUDRAND 또는 RTP_SUPPORT_RANDR 로 정의가 되어져 있다면 실행하는 소스부분이 있는데 이 부분을 모두 주석처리 하였고 rand()를 사용하도록 하였다.

표 4.7 RTPRandom::RTPRAndom() 수정

```
RTPRandom::RTPRandom()
{
        uint32_t x;
        x = (uint32_t)getpid();
        x += (uint32_t)time(0);
        x -= (uint32_t)clock();

        x ^= (uint32_t)((uint8_t *)this - (uint8_t *)0);
        srand((unsigned int)x);

#ifdef RTP_SUPPORT_RANDS
        } // !support_rand_s
#endif
}
```

라. rtpudpv4transmitter.cpp

"RTP_SUPPORT_IFADDRS"이 선언되어 있을 때 실행되는 RTPUDPv4Transmitter::GetLocalIPList_Interfaces()에서 문제가 발생하였다.

"RTP_SUPPORT_IFADDRS"이 선언되었을 때 실행되는 GetLocalIPList_Interfaces()을 주석처리 하였다.

표 4.8 rtpudpv4transmitter.cpp 원본

```
int RTPUDPv4Transmitter::PollSocket(bool rtp)
{
        RTPSOCKLENTYPE fromlen;
        int recvlen;
        char packetbuffer[RTPUDPV4TRANS_MAXPACKSIZE];
#if (defined(WIN32) || defined(_WIN32_WCE))
        SOCKET sock;
        unsigned long len;
```

표 4.9 rtpudpv4transmitter.cpp 수정

```
int RTPUDPv4Transmitter::PollSocket(bool rtp)
{
            //RTPSOCKLENTYPE fromlen;
          socklen_t fromlen;
            int recvlen;
            char packetbuffer[RTPUDPV4TRANS_MAXPACKSIZE];
#if (defined(WIN32) || defined(_WIN32_WCE))
            SOCKET sock;
            unsigned long len;
```

마. rtpsession.cpp

Create()에서 ip version 6에 대하여 정의되어져 있을 때 실행되는 부분에 관련된 모든 라인을 주석처리 하였다.

getlogin_r()에 대하여 정의된 것이 없기 때문에 CreateCNAME()에서 이 부분에 관련된 부분을 모두 주석처리 하였다.

2.2.4 Build

"{NDK_HOME}"으로 이동한후 make를 하면 컴파일이 된다. 순서는 아래와 같다.

표 4.10 RTPClient build 순서

```
cd {NDK_HOME}
make APP=RTPClient
```

컴파일 성공시에는 아래 그림 4.6과 같이 아무런 에러 메시지 없이 라이브러리가 생성된다.

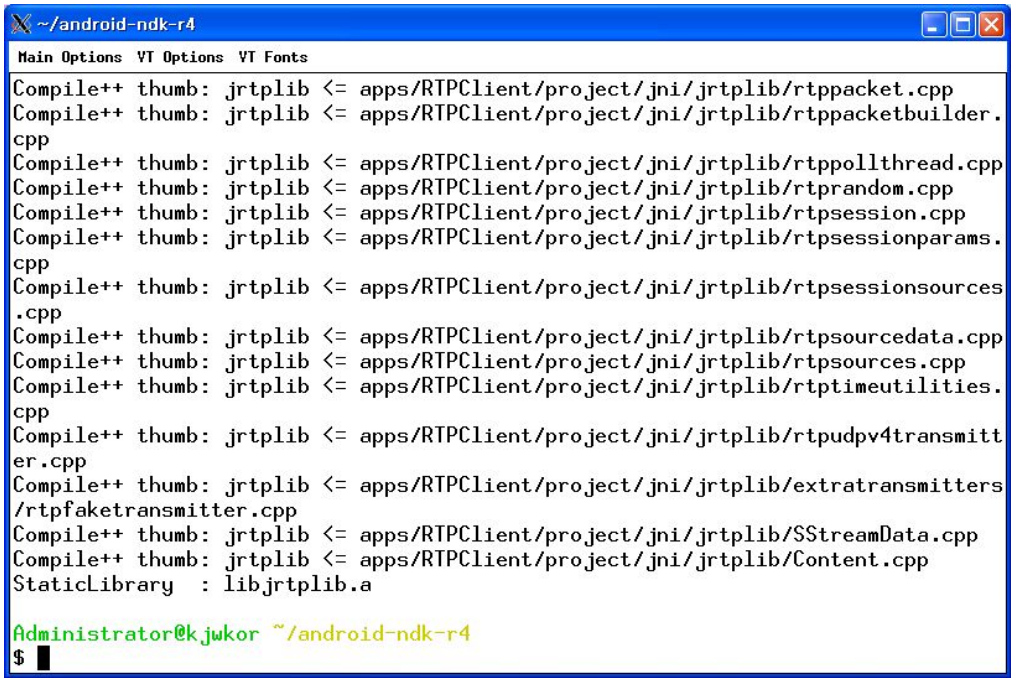

그림 4.6 RTPClient(jthread, jrtplib) Build 성공

2.3 JSVM Decoder

JSVM 소프트웨어 중에서 Decoder부분만을 쓰려면 H264AVCCommonLib, H264AVCVideoIoLib, H264AVCDecoderLib 3개의 라이브러리가 필요하다. 여기서는 이 3개의 라이브러리를 포함하고 콘솔상에서 실행도 가능하도록 포팅하였다.

2.3.1 NDK 사용을 위한 디렉토리 구성

JSVM 소프트웨어는 그림 4.7과 같이 여러개의 디렉토리로 구성된다. 이 중에서 소스가 있는 디렉토리는 "{JSVM_ROOT}\JSVM\H264Extension\src"이다. 이중 필요한 라이브러리는 H264AVCCommonLib, H264AVCVideoIoLib, H264AVCDecoderLib 이 3개이고 콘솔 실행용 어플리케이션을 만들기위한 소스는 "{JSVM_ROOT}\JSVM\H264Extension\src\test\H264AVCDecoderLibTest"에 존재한다.

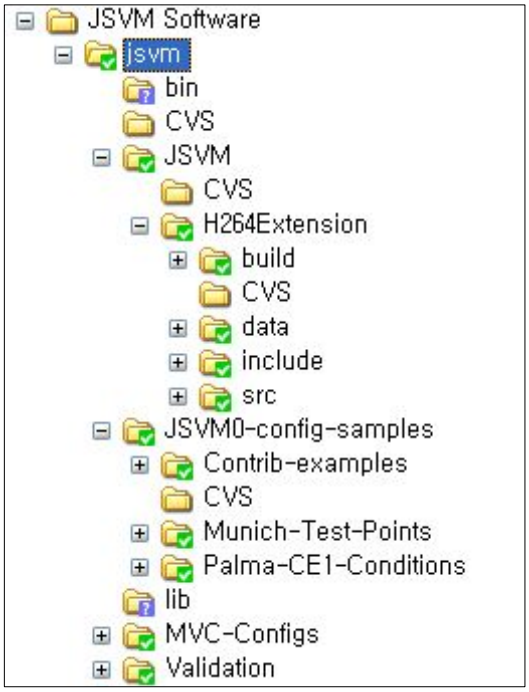

그림 4.7 JSVM 소프트웨어 디렉토리 구조

필요한 파일들을 복사하여 그림 4.8과 같은 구조로 만들었다.

JSVM 소프트웨어는 stl을 사용하였으므로 안드로이드에서 stl을 사용할 수 있도록 하기위하여 stlport를 사용한다.

그림 4.8은 설명처럼 파일과 디렉토리를 구성한 모습이다.

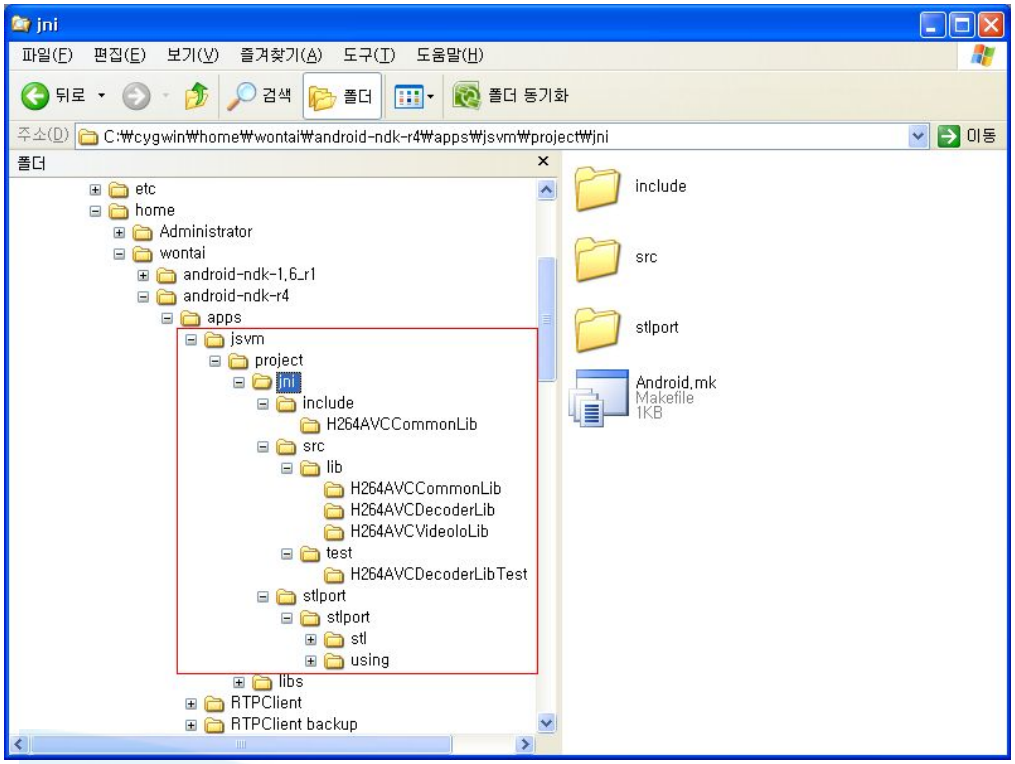

그림 4.8 NDK 사용을 위한 JSVM Decoder 디렉토리 구조

2.3.2 ".mk"파일 생성 및 구성

NDK를 사용하기 위하여 "Application.mk"와 "Android.mk"파일을 생성한다. 테스트용 콘솔 어플리케이션까지 생성하기 위해서는 APP_MODULES에 "H264AVCDecoderLibTest"를 추가해 주면 된다.

표 4.11 JSVM Decoder의 Application.mk

```
APP_PROJECT_PATH := $(call my-dir)/project
APP_CFLAGS += -I$(APP_PROJECT_PATH)/jni/include
APP_MODULES      := \
        H264AVCCommonLib \
        H264AVCVideoIoLib \
        H264AVCDecoderLib

APP_BUILD_SCRIPT := $(APP_PROJECT_PATH)/jni/src/Android.mk
```

표 4.12 H264AVCCommonLib의 Android.mk

```
LOCAL_PATH := $(call my-dir)
include $(CLEAR_VARS)
LOCAL_MODULE := H264AVCCommonLib
LOCAL_CPP_EXTENSION := .cpp

STLPORT_PATH := /home/wontai/android-ndk-r4/apps/RTPClient/project/jni/stlport

LOCAL_CFLAGS += -DMSYS_LINUX \
        -I$(STLPORT_PATH)/stlport \
        -DLINUX \
        -D_NEW_ \
        -D__SGI_STL_INTERNAL_PAIR_H \
        -DANDROID \
        -DOS_ANDROID
LOCAL_SRC_FILES := \
        CabacContextModel.cpp CabacContextModel2DBuffer.cpp DownConvert.cpp \
        Frame.cpp H264AVCCommonLib.cpp IntraPrediction.cpp \
        LoopFilter.cpp MbData.cpp MbDataAccess.cpp \
        MbDataCtrl.cpp MbDataStruct.cpp MbMvData.cpp \
        MbTransformCoeffs.cpp MotionCompensation.cpp MotionVectorCalculation.cpp \
        Mv.cpp ParameterSetMng.cpp PictureParameterSet.cpp \
        PocCalculator.cpp Quantizer.cpp QuarterPelFilter.cpp \
        ReconstructionBypass.cpp ResizeParameters.cpp SampleWeighting.cpp \
        SequenceParameterSet.cpp ScalingMatrix.cpp Sei.cpp \
        SliceHeader.cpp SliceHeaderBase.cpp Tables.cpp \
        TraceFile.cpp Transform.cpp YuvBufferCtrl.cpp \
        YuvMbBuffer.cpp YuvPicBuffer.cpp CFMO.cpp \
        Vui.cpp Hrd.cpp

LOCAL_LDLIBS += -ldl -llog -L$(STLPORT_PATH) -lstlport

include $(BUILD_STATIC_LIBRARY)
```

표 4.13 H264AVCVideoIoLib의 Android.mk

```
LOCAL_PATH := $(call my-dir)
include $(CLEAR_VARS)
LOCAL_MODULE := H264AVCVideoIoLib
LOCAL_CPP_EXTENSION := .cpp

STLPORT_PATH := /home/wontai/android-ndk-r4/apps/jsvm/project/jni/stlport
JRTPLIB_PATH := /home/wontai/android-ndk-r4/apps/RTPClient/project/jni/jrtplib

LOCAL_CFLAGS += -I$(STLPORT_PATH)/stlport \
            -I$(JRTPLIB_PATH)  -DLINUX  -D_NEW_ \
            -D__SGI_STL_INTERNAL_PAIR_H \
            -DANDROID -DOS_ANDROID  -DMSYS_LINUX \
             -D_LARGEFILE64_SOURCE -D_FILE_OFFSET_BITS=64 -DMSYS_UNIX_LARGEFILE

LOCAL_SRC_FILES := H264AVCVideoIoLib.cpp LargeFile.cpp \
            ReadBitstreamFile.cpp ReadYuvFile.cpp \
            WriteBitstreamToFile.cpp WriteYuvToFile.cpp \
            StreamData.cpp Content.cpp SStreamBuffer.cpp

LOCAL_LDLIBS += -ldl -llog -L$(STLPORT_PATH) -lstlport
LOCAL_STATIC_LIBRARIES := H264AVCCommonLib

include $(BUILD_STATIC_LIBRARY)
```

표 4.14 H264AVCDecoderLib의 Android.mk

```
LOCAL_PATH := $(call my-dir)
include $(CLEAR_VARS)
LOCAL_MODULE := H264AVCDecoderLib
LOCAL_CPP_EXTENSION := .cpp

STLPORT_PATH := /home/wontai/android-ndk-r4/apps/RTPClient/project/jni/stlport

LOCAL_CFLAGS += -DMSYS_LINUX -I$(STLPORT_PATH)/stlport \
        -DLINUX -D_NEW_ -D__SGI_STL_INTERNAL_PAIR_H \
        -DANDROID -DOS_ANDROID

LOCAL_SRC_FILES := BitReadBuffer.cpp CabacReader.cpp \
        CabaDecoder.cpp ControlMngH264AVCDecoder.cpp \
        CreaterH264AVCDecoder.cpp GOPDecoder.cpp \
        H264AVCDecoder.cpp H264AVCDecoderLib.cpp \
        MbDecoder.cpp MbParser.cpp NalUnitParser.cpp \
        SliceDecoder.cpp SliceReader.cpp UvlcReader.cpp

LOCAL_LDLIBS += -ldl -llog -L$(STLPORT_PATH) -lstlport
LOCAL_STATIC_LIBRARIES := H264AVCCommonLib

include $(BUILD_STATIC_LIBRARY)
```

2.3.3 Build

"{NDK_HOME}"으로 이동한후 make를 하면 컴파일이 된다. 순서는 아래와 같다.

표 4.15 JSVM Decoder build 순서

```
cd {NDK_HOME}
make APP=jsvm
```

컴파일 성공시에는 아래 그림 4.9과 같이 아무런 에러 메시지 없이 라이브러리가 생성된다.

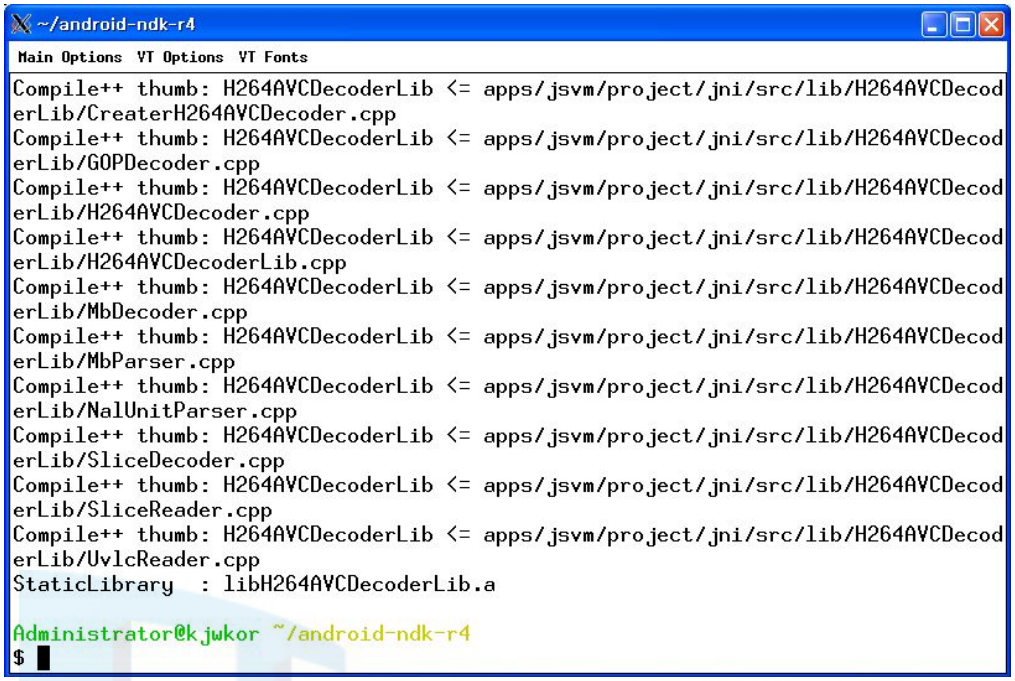

그림 4.9 JSVM Decoder Build 성공

제 3 절 Player 구현

표 4.16 각 클래스 및 라이브러리의 개발 환경 및 개발 언어

계 층	클래스 및 라이브러리 이름	개발 환경 및 언어	기 타
Libraries	JRTPLib	C++ Cygwin 1.7.5 Android SDK 2.1 NDK 1.6_r1	RTP 패킷 수신을 위한 라이브러리
	JSVM Decoder		H.264 SVC 데이터를 복호화하기 위한 라이브러리
Application Framework	LibManager	JDK 1.6 C++ Eclipse 3.5.2 Android SDK 2.1 NDK 1.6_r1	
Applications	Display Manager	Android SDK 2.1	

위의 표는 Player의 구현에 사용된 클래스 및 라이브러리의 개발 환경 및 언어에 대한 설명이다. Libraries계층에서 RTP를 수신하기 위해 JRTPLib이라고 하는 라이브러리를 사용하였고 디코더 라이브러리로는 JSVM 소프트웨어에서 제공하는 Decoder를 사용하였다.

제 5 장 Player 동작 확인

제 1 절 테스트 모델

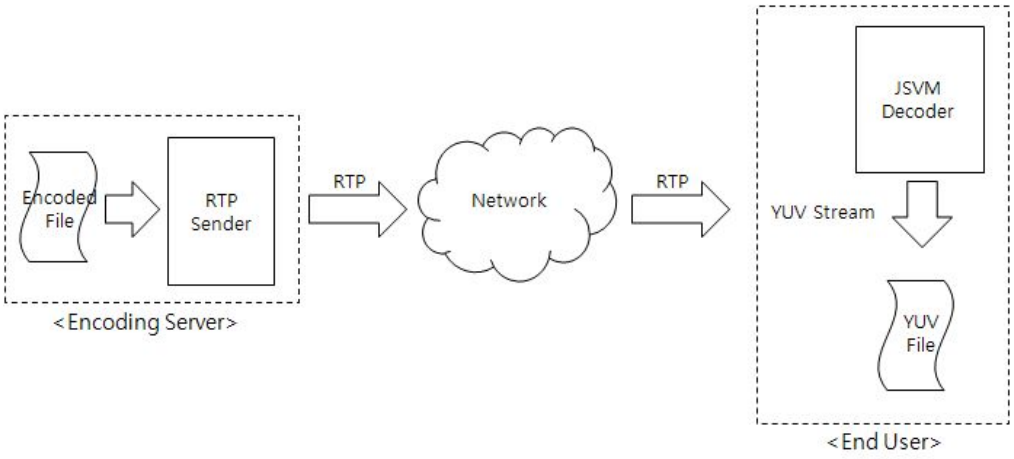

그림 5.1 테스트를 위한 시스템 구성

본 논문에서 제안한 Player가 올바르게 동작을 하는지의 테스트를 위하여 위의 그림과 같이 테스트 시스템을 구성하였다.

JSVM 인코더를 사용하여 300 frames, 재생시간이 12초인 영상을 부호화할 때 소요되는 시간이 10분이상인데 이를 이용하면 실시간으로 부호화하여 전송하는데에 부적합하다고 판단하였다.

그래서 미리 부호화된 파일을 일정 시간 간격으로 RTP 패킷으로 만들어 계속 전송하여 최대한 라이브 스트리밍에 가깝에 구성하였다.

End User에서는 RTP 패킷을 수신하여 복호화 한 후 파일로 저장하였다.

제 2 절 동작 확인

이번 절에서는 테스트 시스템을 이용하여 Player가 제대로 동작하는 모습을 나타낸다. RTP 수신 모듈의 동작확인과 디코더 모듈의 동작확인을 하였다.

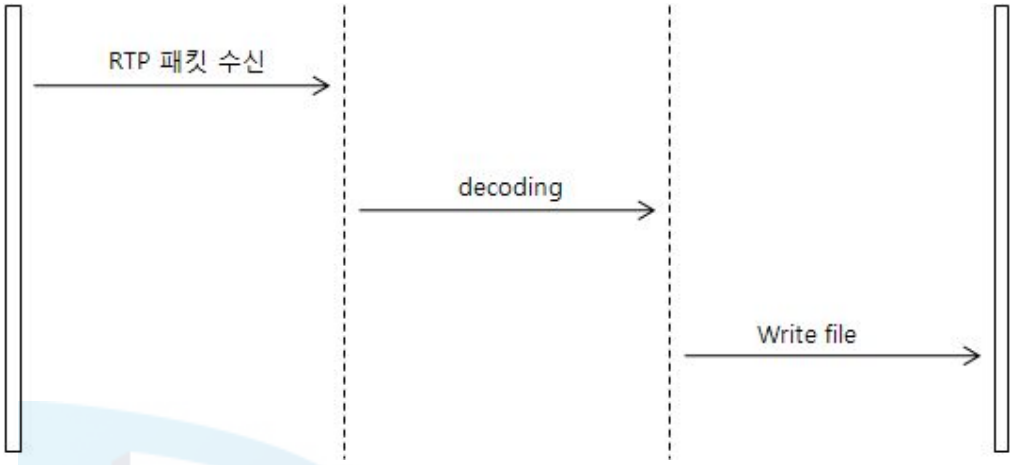

그림 5.2 Player 수행 시간

그림 5.2는 Player의 수행 시간을 나타낸 그림이다. RTP 패킷을 수신하여 복호화한후 파일로 쓰는것까지를 Player 수행의 한 사이클로 볼 수 있다.

2.1 RTP 패킷 수신

다음 그림 5.3은 RTP 패킷 수신 시간을 나타낸 그래프이다. 패킷 수신 시간이란 그림 5.2의 "RTP 패킷 수신"에 해당하는 부분으로써 하나의 RTP 패킷이 수신 된 후 다음 RTP 패킷이 수신될때까지의 시간을 나타낸다. RTP 패킷 수신횟수란 RTP 패킷을 수신받은 횟수(카운트)를 뜻한다.

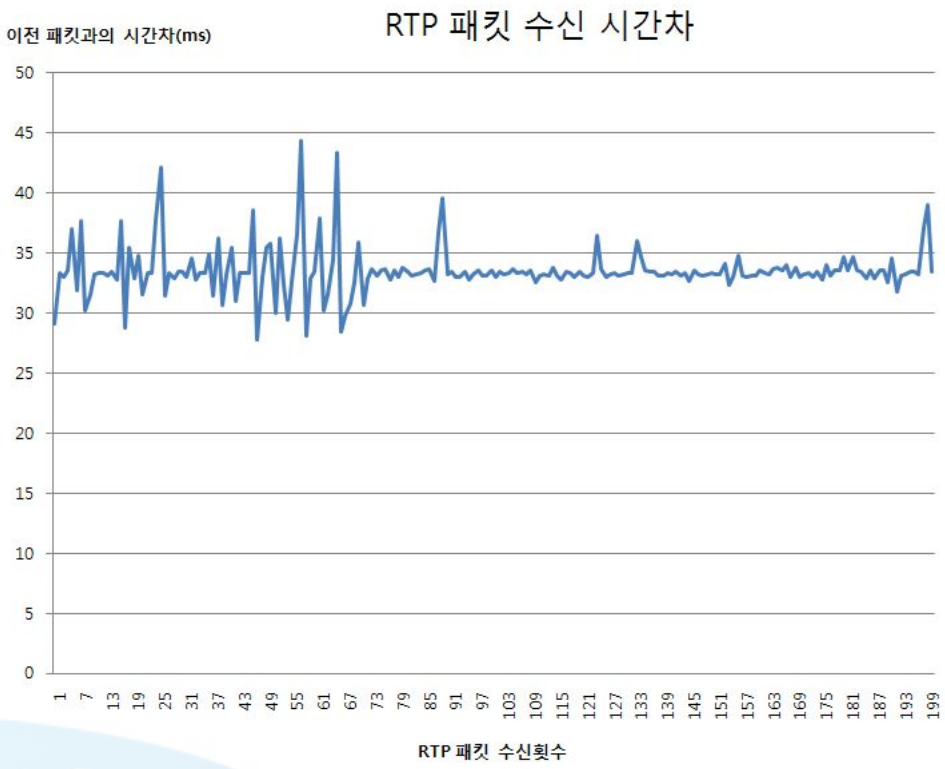

그림 5.3 RTP 패킷 수신 시간차

2.2 디코더 동작 시간

그림 5.4는 그림 5.2에서 "decoding"에 해당하는 부분으로써 디코더의 수행 시간을 그래프로 나타내었다.

디코딩 시간이란 디코딩을 수행하는 함수 실행 전의 시간과 실행 후의 시간을 측정하여 그 시간차를 밀리초(millisecond)단위로 나타낸 것이다.

디코딩 횟수란 디코딩을 수행하는 함수의 실행 횟수이다.

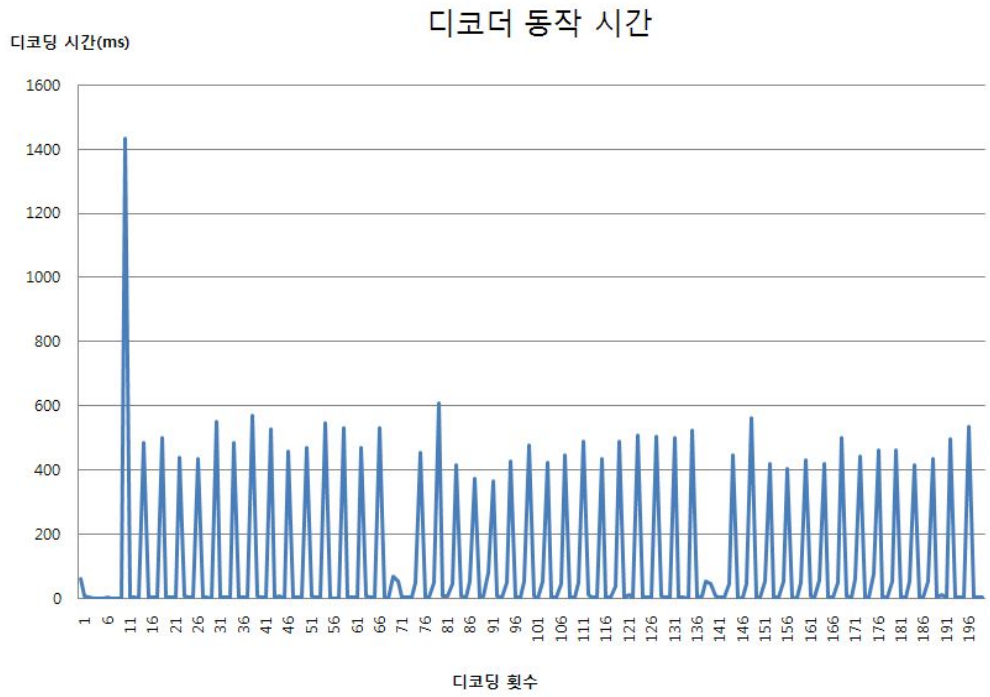

그림 5.4 디코더 동작 시간

RTP 패킷 수신과 디코더 수행의 동작을 확인 하였고 복호화 결과는 파일로 저장하여 VLC Player(http://www.videolan.org)를 사용하여 재생해 보았다.

아래 그림 5.4는 부호화를 하기 위해 사용하였던 원본 YUV 파일(왼쪽)과 부호화를 한후 본 논문에서 개발한 Player에서 복호화를 하여 파일로 저장한 YUV 파일(오른쪽)의 재생모습이다.

육안으로 보기에도 화질상의 차이는 발생하였으며 원본 YUV 파일의 화질이 좋다는 것을 확인 할 수 있었다.

<원본 YUV>

<수행 후 YUV>

그림 5.4 재생 모습

제 6 장 결론 및 향후연구

제 1 절 결론

본 논문에서는 사용자 환경에 따른 스트리밍 서버의 저장공간 낭비 및 인코딩 시스템 낭비를 방지하기 위하여 H.264 SVC를 사용하였고 현재 모바일 단말기에서 H.264 SVC 스트림을 재생할 수 있는 Player가 없기 때문에 H.264 SVC 스트림 데이터를 실시간으로 복호화하는 Player를 설계 및 구현하였다.

모바일 단말기 환경으로 안드로이드 환경을 사용하였고 안드로이드에서 H.264 SVC Player 개발을 위하여 외부 라이브러리들을 NDK를 이용하여 안드로이드 환경에 맞는 라이브러리를 생성하고 JNI를 통해 호출하여 그 기능을 사용하도록 하였다.

본 논문에서 설계한 Player의 동작 테스트를 위하여 테스트 시스템 모델을 구성하여 Player의 동작을 확인하였다.

인코딩 서버부분에서 JSVM 인코더를 사용할 경우 부호화 시간이 너무 오래 소요됨을 발견하고 실시간에 부적합하다고 판단하여 최대한 라이브 스트리밍에 가깝도록 테스트 환경을 구성하였고 테스트 환경을 통하여 구현된 Player가 제대로 동작함을 알 수 있었다.

제 2 절 향후 연구

현재 개발된 Player는 향후 연구 과제들이 존재하며 세부적인 사항은 다음과 같다.

- JSVM Decoder의 결과로 생성된 YUV 데이터를 화면에 출력할 수 있도록 구현한다.
- Darwin Streaming Server와의 연결을 위하여 RTSP 모듈을 구현한다.
- 실행 중인 H.264 SVC Player에서 스트리밍 서비스를 동적으로 중지, 재생, 종료 할 수 있도록 사용자 인터페이스를 추가한다.
- 인코딩 서버에 Extractor를 추가하고 동적으로 계층을 추출하여 전송하도록 구현하여 여러 가지 환경에서 동시에 스트리밍 서비스를 받을 수 있도록 구현한다.

【참고문헌】

1. 국내문헌

김혜선, P2P 스트리ALD 시스템 성능 평가를 위한 NS2 기반 시뮬레이터 개발, 한성대학교 일반대학원, 석사학위논문, 2007

우문섭, P2P를 이용한 H.264 인터넷 방송 시스템 설계 및 구현, 한성대학교 일반대학원, 석사학위논문, 2007

2. 국외문헌

Heiko Schwarz, Detlev Marpe, Member, IEEE, and Thomas Wiegand, 2007. Overview of the Scalable Video Coding Extension of the H.264/AVC Standard. IEEE TRANSACTIONS ON CIRCUITS AND SYSTEMS FOR VIDEO TECHNOLOGY, VOL. 17, NO. 9, SEPTEMBER. 1003-1120

3. 기타

스트리밍, http://100.naver.com/100.nhn?docid=769519

Android Developer, http://developer.android.com

JRTPLib, http://research.edm.uhasselt.be/~jori/page/index.php?n=CS.Jrtplib

JSVM Reference Software,
http://ip.hhi.de/imagecom_G1/savce/downloads/SVC-Reference-Software.htm

RFC 2326, http://www.ietf.org/rfc/rfc2326.txt

RFC 3550, http://tools.ietf.org/html/rfc3550

RFC 4566, http://tools.ietf.org/html/rfc4566

STLport, http://www.stlport.org

SVC Introduction, http://ip.hhi.de/imagecom_G1/savce/index.htm

Wireshark, http://www.wireshark.org

VLC Player, http://www.videolan.org

ABSTRACT

A Design and Implementation of H.264 SVC(Scalable Video Coding) Player for the Android Platform

Jung, Won Tai
Major in Computer Engineering
Dept. of Computer Engineering
Graduate School, Hansung University

The current streaming systems keep various versions of files in response to users' environments and encode them in real time to send, so the storage space or encoding system is being wasted. To solve this, H.264 SVC codec is suggested and standardized.

The H.264 SVC coding system is the compressive technique that can satisfy diversity of a regenerative device according to different memories and CPU processing speed from variability of network bandwidth and users' environments by compressively coding video as basic layers and several layers.

As player that can decrypt H.264 SVC stream in the current mobile environment in real time doesn't exist, this study designed H.264 SVC player and implemented it in the mobile platform Android.

To check the operation of player, the study tested the operation by

constructing the test system.