# 멀티미디어 스트리밍 프레임워크에서 전송 관리자의 설계 및 구현

정찬균, 임익진, 이승룡
경희대학교 전자계산공학과
e-mail:{cgjeong,limga,sylee}@oslab.kyunghee.ac.kr

# The Design and Implementation of Transport Manager in Multimedia Streaming Frameworks

Changyun Jeong, Eakjin Lim, Sungyoung Lee
Dept. of Computer Engineering, Kyung-Hee University

요약

본 논문에서는 통합 스트리밍 프레임워크 (ISSA)[1][2]의 주요 모듈인 전송 관리자의 구현에 대한 개발 경험을 소개한다. 전송 관리자는 ISSA에서 지원하는 다양한 형식의 멀티미디어 데이터를 네트워크의 전송에 알맞게 분해(packetization), 조립(depacketization)하는 기능과 패킷화된 미디어 데이터를 실시간으로 전달하는 역할을 수행한다. 이러한 기능은 RTP/RTCP[3][4] 프로토콜을 이용하여 구현되었으며, 코드 레벨에서의 이기종 플랫폼간 호환성과 네트워크의 투명성을 보장하도록 설계 되었다.

## 1. 서론

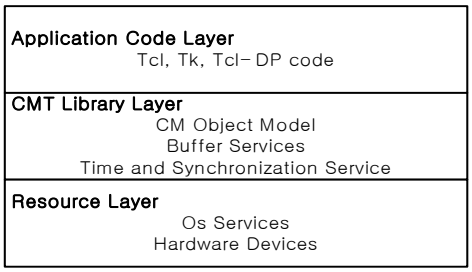
전송 관리자는 ISSA에서 다양한 형식의 멀티미디어 데이터를 알맞게 분해 및 조립하는 기능과, 패킷화된 미디어 데이터를 다양한 네트워크 환경에서 실시간으로 전달하는 역할을 수행한다. 즉, 전송관리자는 실제 데이터를 송수신하는 데 있어 최소한의 오버헤드를 가지고 효율적으로 동작하며, RTP/UDP 와 TCP 프로토콜을 지원함으로써 응용서비스에 매우 유연한 프로토콜 선택이 가능하다. 그리고, 객체지향적으로 설계되어 다른 미디어 전송 프로토콜을 첨가할 수 있어 확장이 용이하며, 컴포넌트 방식으로 다른 응용 프로그램에서 쉽게 사용할 수 있는 특징을 가지고 있다. 그리고 네트워크 하위단을 감싼 네트워크 인터페이스를 사용하여 네트워크에 대한 독립성을 보장하며, Microsoft Windows NT환경과 UNIX 환경에서의 멀티플랫폼간 코드 레벨 호환성을 고려하여 구현되었다.

† 본 논문은 정보통신부 국제공동연구과제 (과제번호:IJRP-9803-6)에 의해 지원받았음.

## 2. 관련 연구

본 장에서는 기존의 스트리밍 시스템에 대한 관련 연구와 전송 관리자에 대한 기존의 연구에 대해 살펴보도록 하겠다.

CMT는 미국 버클리 대학에서 만든 동영상과 같은 연속적인 미디어를 위한 개발 툴킷으로서, 스트리밍 응용프로그램의 신속한 개발을 위한 프로그래밍 환경을 제공한다[5].




[그림 1] CMT 시스템 구조

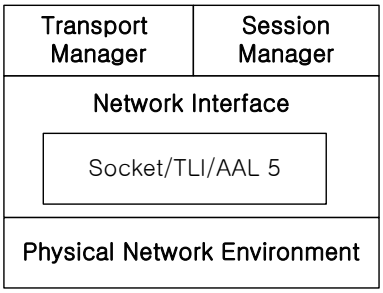
CMT 전체 시스템 구조는 [그림 1]과 같이 크게 Application Code Layer, CMT Library Layer,

Resource Layer의 세 부분으로 나누어진다. 최상위 레벨의 Application Code Layer는 Tcl/Tk 등을 사용하여 응용프로그램 개발자에게 프로그래밍 환경을 제공하고, 중간 레벨의 CMT Library Layer는 연속적인 미디어 객체를 위한 Tcl 인터프리터와 다양한 서비스를 제공하며, 최하위 레벨의 Resource Layer는 CPU, 네트워크 인터페이스, 메모리, 입출력 장치 등의 하드웨어와 운영체제 자원을 표현한다. 그러나 CMT는 내부 구조가 너무 평면적이며 일관적인 객체 지향 프로그래밍 모델을 따르지 않으므로, 시스템을 확장하기 어렵고 내부 구조가 매우 복잡한 형태를 가진다.

KISS(Communication Infrastructure for Streaming Services)는 독일의 GMD에서 개발한 스트리밍 서비스를 위한 통신 인프라구조로서 네트워크 서비스 응용의 투명한 통합과 실시간 컨텐츠 스트림의 이기종 네트워크 적응을 지원한다[6]. KISS는 데이터속도와 컨텐츠 적응을 위한 SA(Service Applications)와 단말 시스템 사이의 중도적 역할을 수행하는 NAP(Network Access Points)를 사용하여 네트워크 환경의 투명성을 제공한다. 그러나 KISS는 아직 PCM과 MPEG 오디오 형식의 미디어만을 지원하도록 구현되었으며, NAP를 통한 성능 오버헤드를 줄이는 것이 관건이다.

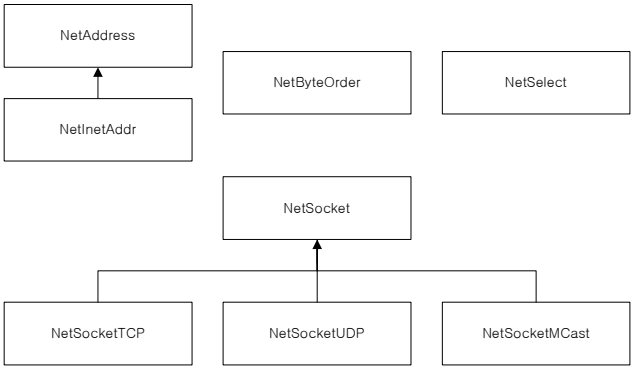
## 3. 네트워크 인터페이스 (Network Interface)

ISSA 프레임워크는 다양한 네트워크 환경에 대해 투명성을 제공하기 위해 네트워크 인터페이스를 통해 네트워크 송수신 기능을 수행한다[그림 2]. 네트워크 인터페이스는 네트워크 환경에 독립적으로 실행될 수 있고 다양한 네트워크 환경을 지원하는 일종의 네트워크 wrapper 인터페이스로서, 현재 윈도우 환경의 WinSock 및 유닉스 환경의 버클리 소켓에 대한 인터페이스를 제공하며, 향후 TLI(Transport Layer Interface), ATM(Asynchronous Transfer Mode) 등의 다양한 네트워크 프로그래밍 인터페이스를 지원할 수 있는 유연한 구조를 지니고 있다. 또한, 네트워크 인터페이스를 통해 유니캐스트와 멀티캐스트를 손쉽게 구현할 수 있으며, ISSA 프레임워크의 전송관리자나 세션관리자와 같이 네트워크 사용이 많은 다른 모듈에서 이용되어 프로토콜의 독립성을 지원 해줄뿐 만 아니라 소스 코드 레벨에서 이기종간 플랫폼의 호환성도 보장해 준다.




[그림 2] 네트워크 투명성 제공을 위한 네트워크 인터페이스

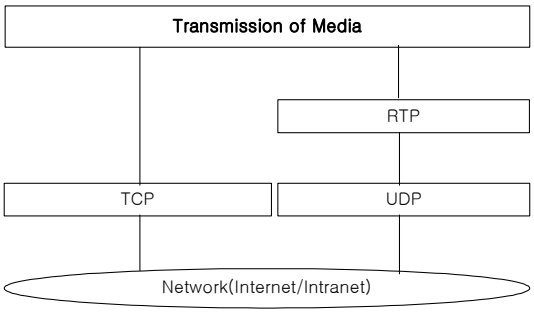
네트워크 인터페이스는 [그림 3]와 같이 구현되었는데, 네트워크 주소를 나타내기 위한 NetAddress 객체와 인터넷 주소를 처리하기 위한 NetInetAddr 객체, 네트워크 바이트의 Byte Alignment를 위한 NetByteOrder 객체, 윈도우와 유닉스를 동일한 이벤트 방식으로 처리하기 위한 NetSelect 객체가 있다. 그리고, NetSocket 객체는 WinSock, 버클리 소켓에서 일관되지 않은 인터페이스를 통합하기 위해 구현되었으며, 여기서 파생된 NetSocketTCP 객체와 NetSocketUDP 객체는 TCP 및 UDP 연결을 NetSocket 객체 함수를 이용하여 구현되었고, NetSocketMCast는 멀티캐스트를 용이하게 처리하기 위해 Time-To-Live 및 그룹 관리 기능이 추가되었다. 이러한 클래스의 구분은 기본적으로 지원되는 버클리 소켓의 인터페이스를 객체 지향적으로 재설계하면서 나뉘어 졌으며 버클리 소켓을 이용한 기존의 클래스 라이브러리를 참고하여 라이브러리 내에서 생소한 용어들을 철저히 배제시켰다.




[그림 3] 네트워크 인터페이스의 클래스 다이어그램

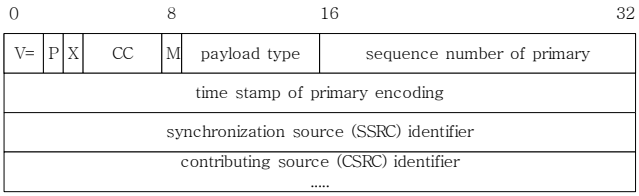
## 4. 전송관리자의 설계

전송관리자는 전술한바와 같이 ISSA에서 다양한 형식의 멀티미디어 데이터를 알맞게 분해, 조립하는 기능과, 패킷화된 미디어 데이터를 다양한 네트워크 환경에서 실시간 전달하는 역할을 수행한다[1][2]. 즉 전송관리자는 다양한 네트워크 환경에서 여러 종류의 멀티미디어 데이터를 스트리밍 하기 위한 유연하고 확장성 있는 전송 환경을 제공한다. [그림 4]은 전송관리자의 전송 프로토콜 스택으로서 전송관리자에서는 RTP[3, 4], TCP기반의 다양한 미디어 전송 프로토콜과 IP-유니캐스트 및 IP-멀티캐스트 동작환경을 모두 지원하며, ISSA의 세션관리자에서 제공하는 세션 제어 프로토콜에 관계없이 유연하게 사용할 수 있다.




[그림 4] 전송관리자의 전송프로토콜 스택

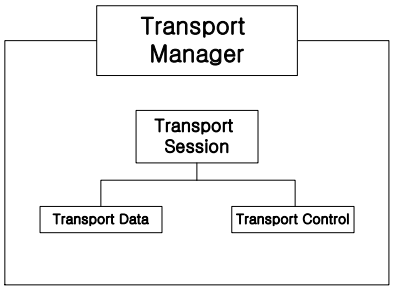
RTP는 멀티미디어 데이터의 실시간 전송을 지원하는 프로토콜로서, 비디오와 같이 실시간 특성을 가진 멀티미디어 데이터의 종단간 실시간 전달 서비스를 제공한다. 이러한 실시간 전달 서비스는 TCP가 가지고 있는 오버헤드를 효율적으로 극복할 수 있으며, RTP 프로토콜은 패이로드 타입, 소스식별자, 순서 번호, 타임 스탬프, 그리고 제어정보 모니터링을 위한 RTCP등을 이용해 실시간 전달 기능을 지원한다. 일반적으로 RTP는 UDP위에서 동작하나, ATM 등과 같은 기타 다른 전송 환경 위에서도 동작할 수 있게 설계되었다. 또한 RTP를 이용하면, MBone(Multicast Backbone)과 같이 IP-멀티캐스트를 지원하는 네트워크에서 멀티캐스트 전송 기능을 사용하여 하나의 패킷을 특정 그룹의 다중 사용자에게 한번에 모두 전송하는 것이 가능하다. 하지만 RTP 프로토콜 자체로는 QoS를 보장할 수 없으며, 다만 다른 제어 기법 등을 사용하여 QoS를 보장하는 것은 가능하다. [그림 5]은 RTP 데이터 패킷의 고정 헤더 형식으로서 RTP로 전달되는 데이터를 식별해주는 필드들을 포함한다.




[그림 5] RTP 패킷 고정 헤더의 형식

한편 RTCP 프로토콜은 하나의 RTP 세션에서 RTP와 같이 사용되며, QoS 정보를 모니터링하고, 진행중인 세션에 참가한 사용자들의 정보를 전달하는 것을 목적으로 한다. 즉 RTP는 전송기능만을 가지며 RTCP가 세션에 대한 제어 기능을 수행한다. 하지만 RTCP는 최소한의 세션 제어 기능만을 가지고 있다.

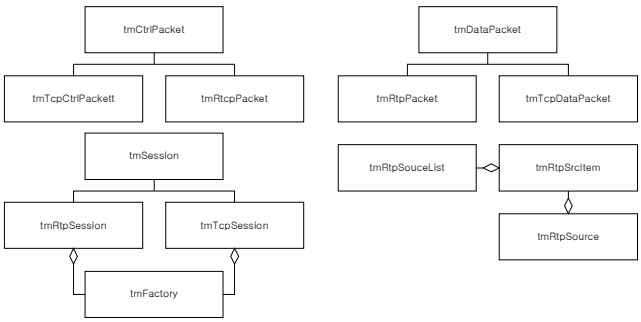
전송관리자는 다음 [그림 6]과 같이 논리적으로 크게 전송 세션, 전송 데이터 그리고 전송 제어의 세 부분으로 구성된다. 전송 세션 부분은 전송 세션의 상태, 송수신 통계 등의 전송 세션에 관한 모든 정보를 저장하고, 전송 데이터는 데이터 패킷의 송수신 기능을 수행하며, 마지막으로 전송 제어는 제어 패킷의 송수신 기능을 수행한다.




[그림 6] 전송관리자의 구성

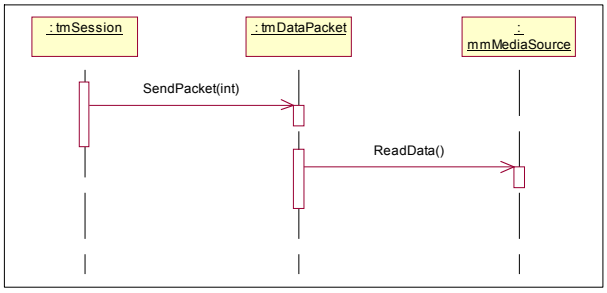
전송관리자는 위의 세 가지 구성요소를 RTP, TCP 각 전송 프로토콜마다 가지고 있고, 이에 대한 UML 클래스 다이어그램은 [그림 7]와 같으며, 여기에서 전송관리자의 핵심 클래스들은 크게 tmFactory, tmSession, tmDataPacket, tmCtrlPacket 으로 구성되어 있다. tmFactory 클래스는 적절한 미디어 전송 프로토콜(RTP 혹은 TCP)에 해당하는 세션 객체인 tmSession을 생성하는 CreateSession() 오퍼레이션과 데이터 패킷 전송을 위한 tmDataPacket 객체를 생성하는 CreateDataPacket() 오퍼레이션, 그리고 제어 패킷 전송을 위한 tmCtrlPacket 객체를 생성하는 CreateCtrlPacket() 오퍼레이션을 제공한다. tmSession 객체는 현재 생성된 전송 세션에 대한 상태 혹은 통계 정보 등과 같은 모든 정보를 저장하며, 데이터 패킷과 제어 패킷을 보내기 위한 객체인 m_pDataPacket 객체

와 m_pCtrlPacket 객체를 저장하고 있다. 이 때 tmSession, tmDataPacket, tmCtrlPacket 클래스는 추상 클래스로서 실제 기능은 RTP 프로토콜의 경우 자식 클래스인 tmRtpSession, tmRtpPacket, tmRtcpPacket 에서 수행하며, TCP 프로토콜의 경우 tmTcpSession, tmTcpDataPacket, tmTcpCtrlPacket 에서 수행한다. 따라서 새로운 미디어의 전송 프로토콜의 추가시에는 tmSession, tmDataPacket, tmCtrlPacket을 상속하는 새로운 자식 클래스들을 구현하여 주면 쉽게 새로운 전송 프로토콜을 구현할 수 있다.



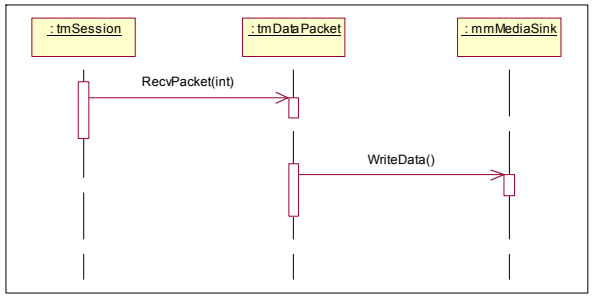

[그림 7] 전송관리자의 UML 클래스 다이어그램

다음 [그림 8]은 데이터 패킷을 송신할 때의 UML 시퀀스 다이어그램으로서, tmSession 객체에서 tmDataPacket 객체의 SendPacket() 오퍼레이션을 호출하면 tmDataPacket 객체는 mmMediaSource의 ReadData() 오퍼레이션을 통해 실제 전송할 미디어 스트림 데이터를 얻어와 이를 패킷에 실어 전송한다.



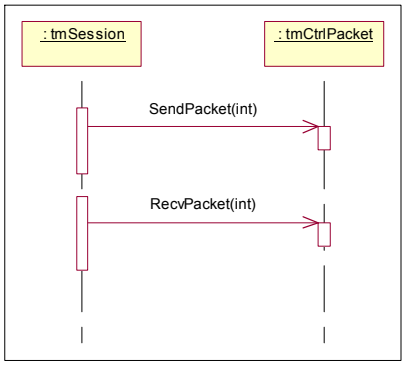

[그림 8] 데이터 패킷 송신시의 UML 시퀀스 다이어그램

[그림 9]은 데이터 패킷 수신시의 UML 시퀀스 다이어그램으로서, 데이터 패킷이 네트워크를 통해 전송되면 tmSessioin 객체에서 tmDataPacket 객체의 RecvPacket() 오퍼레이션을 호출하고, tmDataPacket 객체는 패킷을 읽어 디코딩한 후 미디어 스트림 데이터를 mmMediaSink 객체의 WriteData() 오퍼레이션을 통해 mmMediaSink 객체로 넘겨준다.




[그림 9] 데이터 패킷 수신시의 UML 시퀀스 다이어그램

[그림 10]는 제어 패킷의 송수신에 관한 UML 시퀀스 다이어그램으로서, tmSession 객체에서는 제어 패킷 송신 때는 tmCtrlPacket 객체의 SendPacket() 오퍼레이션을 호출하며 수신 때는 RecvPacket() 오퍼레이션을 호출한다.




[그림 10] 제어 패킷 송수신시의 UML 시퀀스 다이어그램

한편 자체적인 TCP를 이용한 전송프로토콜의 경우 RTP와 같이 미디어 데이터를 효율적으로 전달하기 위해 [그림 11]과 같은 패킷 구조를 가지도록 설계하였다. [그림 11]에서 4 바이트 크기의 media time은 RTP 패킷 헤더의 timestamp와 같이 현재 전달되는 데이터의 타임값을 표현하는 역할을 하며 그 다음 4 바이트의 media data length는 패킷에 실려있는 실제 미디어 데이터의 크기를 나타낸다.

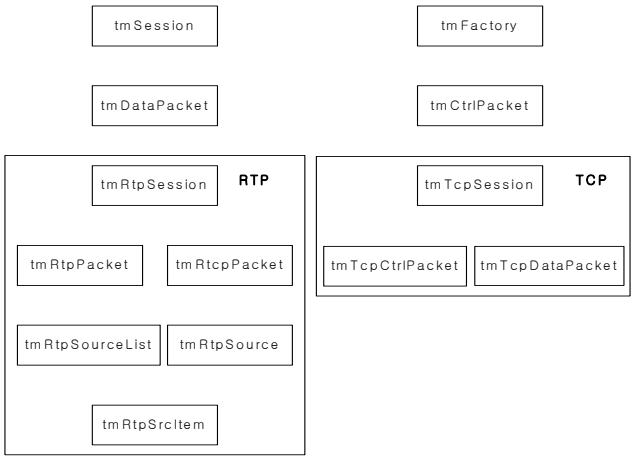
| media time (4 byte) |
| --- |
| media data length (4 byte) |
| media data<br>... |

[그림 11] TCP 데이터 패킷의 형식

## 5. 전송관리자의 구현

전송관리자의 구현은 윈도우와 유닉스 환경에서 모두 동작할 수 있도록 ANSI C++ 언어로 구현하였으며, 플랫폼 의존적인 코드는 다양한 플랫폼으로 이식 가능한 쓰레드, 타이머, 네트워크 인터페이스 라이브러리를 만들어 이를 사용함으로써 쉽게 구현이 가능하게 하였다. 네트워크 인터페이스는 윈도우 환경의 Winsock과 유닉스 환경의 버클리 소켓을 지원하여 하나의 일관된 인터페이스로 네트워크에 대한 접근할 수 있도록 해준다. 또한 타이머의 경우 패킷을 주기적으로 전송하기 위해 필요한 기법으로서, 미디어 데이터 패킷이나 혹은 제어 패킷이 서버 측으로부터 클라이언트 측으로 Push 방식을 이용하여 전송될 때 사용된다. [그림 12]는 구현된 전송관리자의 전체 모듈 구성도로서 총 13개의 모듈로 구성되며, RTP와 관련 있는 6개의 모듈과 TCP와 관련된 3개의 모듈, 그리고 상위 레벨의 추상적인 모듈 4개로 구분된다.

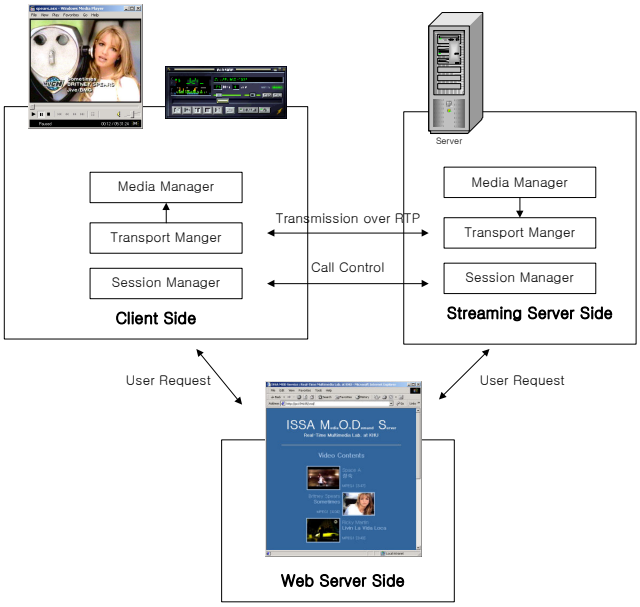
전송 프로토콜을 가동시키는 프로그래밍 인터페이스는 앞장에서 설계된 대로 매우 단순하고 유연하게 구현되었으며, 데이터를 보내는 서버 프로그램의 경우 InitSender()로 세션을 초기화시키고, 클라이언트의 경우는 InitReceiver()로 초기화시키며, 그 후 Run()을 통해 세션을 동작시킬 수 있다. 이 때 서버는 미디어 데이터를 얻어올 미디어 소스, 클라이언트는 전달된 데이터를 표현할 미디어 싱크의 객체를 넘겨받는다. 전송 세션이 동작중일 때 Pause()를 통해 잠시 중단시키거나 Deinit()를 통해 완전히 정지시킬 수 있다. 따라서 전송관리자는 응용 프로그램에서 사용할 전송 프로토콜에 따라 해당 세션을 생성하고 위와 같은 인터페이스를 통해 전송을 쉽게 제어할 수 있도록 구현되었다.




[그림 12] 전송관리자의 전체 모듈 구성도

## 6. 구현사례

본 전송 관리자를 이용하여 다양한 스트리밍 서비스를 구현할 수 있는데, 다음 [그림 13]은 스트리밍서버와 웹 서버, 클라이언트로는 Windows MeidaPlaeyer와 WinAmp를 이용한 구현의 한 사례를 보여주고 있다. 스트리밍 서버는 Windows NT와 Sun Solaris 2.X 환경에서 동작할 수 있으며, 클라이언트 프로그램은 Microsoft Windows 환경에서 동작할 수 있게 구현된, ISSA의 Meida 관리자의 응용 Wrapper인 DirectShow Source Filter와 WinAmp Plug-in을 사용하였다.




[그림 13] 전송 관리자를 이용한 스트리밍 서비스의 구현 사례

Web Server를 사용하여 User의 요청이나 응용단에서의 직접적인 Media 요청을 할 경우 Session 관리자에서 RTSP[7]를 사용하여 Call Control을 한후 실제적인 전송은 전송 관리자가 RTP로 데이터를 보내면서 시작된다. 이때 서버측의 전송 될 미디어의 소스는 미디어 관리자로부터 받게 되며 이 소스를 전송 관리자가 패켓화를 한 후 전송을 하게 되는 것이다. 클라이언트측은 받은 패킷을 해석한후 이를 미디어 관리자에 전해 주며 미디어 관리자는 이를 사용자에게 보여준다. 이때 전송 관리자는 RTCP를 사용하여 서버에게 모니터링된 정보, 즉 잃어버린 패킷수라든가 받은 패킷 수등의 정보를 보내주게 된다. 스트리밍 종료시 전송관리자는 RTCP로 BYE 패킷을 보냄으로 종료가 된다.

## 7. 결론

MPEG과 같은 멀티미디어 데이터를 실시간으로 스트리밍하기 위해서는 RTP 프로토콜과 같이 패킷에 미디어의 표현시간, 미디어의 형식, 순서번호 등과 같은 기능을 제공하는 강건한 전송 프로토콜이 요구된다. 따라서 본 연구에서는 이러한 미디어 데이터의 전송 기능을 객체 지향적인 모델로 제공하는 전송관리자를 설계하고 구현하였다.

구현된 전송관리자는 실제 데이터를 송수신하는 데 있어 최소한의 오버헤드를 가지고 효율적으로 동작하였으며, RTP/UDP 와 TCP 프로토콜을 지원함으로써 응용에게 매우 유연한 프로토콜 선택을 가능케 하였다. 또한 객체 지향적으로 설계되어 다른 미디어 전송 프로토콜을 첨가하는 등의 확장이 용이하며, 컴포넌트 방식으로 다른 응용 프로그램에서 쉽게 사용할 수가 있다. 하지만 RTP/UDP의 경우 MPEG과 같이 고대역폭을 요구하는 데이터를 전송할 경우 패킷 손실이 자주 일어날 수 있으므로, 이런 문제를 해결하기 위한 손실된 패킷의 재전송 기법이 요구되며, 패킷 전송 레벨에서의 QoS 지원을 위한 패킷 스케줄러의 필요성이 제기된다.

## 참고문헌


[1] 정찬균, 이승룡,""통합 스트리밍 프레임워크의 설계", 제 11회 한국 정보처리학회 춘계 학술발표 논문집, pp. 319-322, 1999년 4월.

[2] 정찬균, 김형일, 홍영래, 임익진, 이승재, 이승룡, 정병수, “분산 스트리밍 시스템 설계”, 제 4회 한국멀티미디어 학회 추계 학술발표 논문집, pp. 338-343, 1999년 11월.

[3] H. Schulzrinne, "RTP: A Transport Protocol for Real-Time Applications", IETF RFC 1889, Jan. 1996

[4] H. Schulzrinne, "RTP Profile for Audio and Video Conferences with Minimal Control", IETF RFC 1890, Jan. 1996

[5] K. Mayer-Patel and L.A. Rowe, "Design and Performance of the Berkeley Continuous Media Toolkit", Multimedia Computing and Networking 1997, pp 194-206, 1997.

[6] K. Jonas, M. Kretschmer, and J. Mödeker, "Get a KISS - Communication Infrastructure for Streaming Services in a Heterogeneous Environment", In Proc. of ACM Multimedia '98, Bristol, UK, pp. 401-410, 1998.

[7] H. Schulzrinne, "RTSP: Real-Time Streaming Protocol", IETF RFC 1890, Apr. 1998