제 5 강좌 : RTP(Realtime Transport Protocol)을 이용하자

지난 강좌에서는 라이브 미디어데이터의 캡쳐 작업을 위한 캡쳐버퍼 및 렌더러 버퍼의 크기를 실험적으로 직접 설정하는 방법과 함께 비표준적인 데이터소스를 생성하여 JMF 에 등록하고 재생을 하는 방법에 관하여 알아보았다. 또한 마지막부분에서는 간략하게나마 JMF 에서 기본적으로 제공하는 JMStudio 프로그램을 이용하여 RTP 프로토콜을 이용한 영상 음성 데이터를 송수신하는 테스트를 해보았다. 지난 강좌들을 통하여 독자들은 이제 로컬 컴퓨터에서의 라이브 데이터획득과 처리 및 관련 이벤트에 관한 사항들을 전부 살펴보게 되었다. 이제는 JMF 2.x 버전에서 가장 큰 주목을 받고 있는 라이브 미디어 데이터의 RTP 를 이용한 실시간 전송 및 수신기능에 대하여 알아보자. 이를 위하여 실시간 영상 음성 데이터의 송수신을 위한 RTP 의 개념과 내부구조에 관하여 알아보고 JMF 에서의 구현부분 및 각 이벤트 처리의 방법에 관하여 살펴보기로 하자. 강좌에 관한 질문이나 관련된 프로그램은 아래의 홈페이지를 참조하기 바란다.

이재훈 전임연구원
삼성테크윈 정밀기기연구소 (Samsung Techwin)
kingseft@samsung.co.kr
http://myhome.naver.com/kingseft

실시간 미디어 데이터의 스트리밍

실시간 미디어 방송 또는 화상회의 정보를 인터넷이나 인트라넷에서 구동하기 위해서는 Real time 으로 미디어 데이터를 전송하고자 하는 요구가 증가하고 있다. 이러한 네트워크 밴드 대역의 요구를 충족하기 위해서 Real Time Transport Protocol(RTP) 이 소개되었고, 네트워크를 통한 미디어 데이터의 송신 및 수신을 위해 JMF 에서는 RTP 를 구현하는 패키지를 기본적으로 제공하고 있다.

먼저 RTP 라는 새로운 프로토콜이 나온 이유를 살펴보자. 실시간 전송환경에서는 높은 대역의 BandWidth 를 필요로 하며, 데이터를 수신할때는 전송중의 전송지연에 대한 보상을 위한 처리보다는 손실 데이타에 대한 보상이 구현상 더 쉬운편이기 때문에 이런 측면에서 멀티미디어 데이터의 전송시에 요구되는 환경은 파일 전송같은 완전한 데이타의 전송이 요구되는 무손실 통신 환경과는 다른 측면이 있음을 알 수 있다. 결과적으로 이러한 파일같은 정적인 데이타를 보내는 프로토콜은 스트리밍 데이타를 전송하기 위한 프로토콜로는 적합하지 않다는 것이다. 우리가 일반적으로 사용하는 통신 프로토콜의 문제점과 이를 해결하기 위한 RTP 의 기능을 비교하여 보자.

HTTP, FTP 프로토콜의 문제점

HTTP, FTP 등의 일반적인 연결지향형 프로토콜은 일반적인 인터넷 프로토콜로서 송신 및 수신시에 전송 데이터의 확고한 안정성 보장에 중점을 둔 프로토콜이므로, 속도에 중점을 두어 설계된 프로토콜이 아니다. 즉, HTTP, FTP 등은 전송되는 목적지에 전송하는 모든 데이타 패킷의 완전한 전송을 보장한다. 이러한 프로토콜들은 TCP layer 를 이용하는데, 멀티미디어 전송의 관점에서 TCP 전송은 3 가지의 주요한 단점을 지닌다.

1. Processing overhead
2. Network Transmission Delay
3. Lack of multimedia functionalities

위와같은 단점으로 인해 화상회의 , 인터넷 브로드캐스팅, 멀티캐스팅과 같은 실시간 전송 프로토콜은 UDP layer 에서 설계된다. UDP 전송은 전송을 보장하거나 수신을 위한 데이터의 안정성을 보장하는 프로토콜이 아니다. TCP 와는 달리 UDP 는 연결에 대한 정보를 갖지 않는다. 예를들어 서버측에서 데이타 패킷을 클라이언트에 전송한다고 하더라도, 이렇게 전송되는 패킷이 실제로 클라이언트에게 안전하게 전송되었는지에 대한 확신을 할 수 없다. 실제로 UDP 전송이 이상하게 여겨질 수 있으나, 데이터 전송의 손실에 대한 복구는 수신단의 부담으로 남겨두고, 네트워크 오버헤드나 애플리케이션 구동에 대한 적은 제약때문에 실시간 멀티미디어 데이타 전송에 UDP 가 많이 이용된다는 것이다.

HTTP 나 FTP 는 TCP 기반의 프로토콜로서 TCP 프로토콜은 안정적인 데이타전송과, low bandwidth, high network error 환경에서 적합한 프로토콜이다. 만약 송신 및 수신단에서 데이타 손실이 발생하는 경우 이를 송신측으로 재요청을 통하여 데이터의 재전송 및 재수신이 이루어지게 된다. 그러나 이러한 데이타 전송에 대한 무손실 보장은 결국 네트워크에 대한 오버헤드로 작용하게 된다. 그래서 스트리밍 데이타의 전송에서는 TCP 와는 다른 프로토콜을 이용하게 되었고, 일반적으로 추천받는 프로토콜이 UDP 이다. UDP 는 Unreliable 프로토콜로서 수신단 측에서의 데이타 수신에 대한 보장을 하지 않으며, 또한 전송되는 데이타 패킷이 올바른 순서대로 수신단에서 받는다는 보장이 없다. 이러한 UDP 환경에서는 수신단측에서 결국, 데이타 손실에 대한 보상처리를 해주어야 하고, 패킷의 복제 기능, 패킷의 재정렬 기능을 구현해주어야 하는 책임이있다. TCP 와 마찬가지로 UDP 역시 일반적인 Transport Layer 의 통신 프로토콜이며, 오디오나 비디오와 같은 스트리밍 데이타 전송에 대한 인터넷 표준으로 RTP 를 이용하게 되었고, RTP 는 IEEE RFC 1889 로 규정되어 있다.

RTP 는 Unicast 와 Multicast 양측에 모두 사용가능하며, Unicast 서비스 모드에서는 각각의

데이터 카피본이 전송 데이터의 소스 위치에서 목적지로 전송이 된다. 멀티캐스트 서비스 모드에서는 데이타가 소스에서만 보내지고, 네트워크 자체가 다중 목적지로의 전송에 대한 책임을 지는 구조를 구현한다. 멀티캐스팅이 비디오 오디오데이터를 발생시키는 화상 회의와같은 응용 애플리케이션에 보다 더 효율적으로 적용될 수 있으며 멀티캐스팅을 지원하는 IP 프로토콜을 이용하게 된다. RTP 는 전송된 데이타의 구별능력과 패킷의 순서에 대한 결정, 다중 미디어에 대한 효율적인 동기화 기능을 수행하지만 RTP 는 송신단에서 보내지는 패킷의 순서대로 수신단에서 패킷을 수신한다는 보장이 없다. 또한 송신한 모든 데이타 패킷이 전부 다 수신단에 도달한다는 보장도 없으므로 수신단의 패킷 시퀀스를 다시 만들고, 패킷 헤더에서 제공되는 정보를 가지고 손실된 데이타 패킷을 검출하는 모든 부분이 바로 수신단의 책임이된다. 주의할것은 RTP 는 주기적인 전송의 보장이나, 서비스 quality 에 관한 보장이 없기때문에, 데이터 전송의 품질을 보증하기 위해 RTCP 를 이용하게된다. RTCP 는 또한 RTP 전송에 대한 control 기능과 identification 메커니즘을 제공한다. 이제는 RTP 프로토콜을 이해하기 위해 필요한 용어들을 정리하여 보자.

RTP (Realtime Tranport Protocol)

일반적으로 엄격한 타이밍 요구조건을 갖는 TCP 기반의 HTTP, FTP 와는 달리, RTP 는 주로 실시간 미디어 스트림에 대한 전송과 수신을 목적으로 설계되었으며, 모든 RTP 버퍼들은 timestamp 를 가지고 있어서, timestamp 와 실제 전역적인 동기화된 클럭(예를들어서 JMF Player 의 TimeBase)사이의 매핑을 수행한다. timestamp 의 역할은 다양한 데이터 소스로부터 제공되는 미디어들을 통합하는 기능을 지닌다. 이때 전송되는 각각의 패킷은 Sequence Number 와 함께 timestamp 를 갖게되는데, SequenceNumber 는 각 패킷마다 서로 다른 고유의 번호로서, 수신단에서 패킷 손실에 대한 검출과 복구에 이용된다. timestamp 는 동일한 번호를 지님으로서 특정한 동일시간에 함께 복호화 되야함을 의미한다. RTP 역시 UDP 기반의 프로토콜로서 수신단에서는 수신되어지는 데이타 패킷이 일정하게 알맞은 순서로 전송되었는지 알수 없다. 그러므로 수신단에서는 패킷의 sequence Number 를 이용하여 패킷을 재정렬해야한다.

Session 과 Participants

RTP 통신의 주체는 바로 Session 과 Participants 라고 할수있다. 이제 이 두가지 용어에 관하여 살펴보자. Session 은 둘이상의 Participants 간의 데이타 교환을 의미한다. Participants 란 데이타 교환에 참여하는 각각의 송/수신단을 의미한다. 예를 들어서 A 와 B 가 서로 RTP 를 이용해 통신을 한다면, 서로를 연결하는 하나의 Session 이 만들어지고 이러한 환경에서 Participants 는 2 가된다. 만약 새로운 참여자인 C 가 A 와 B 의 통신에 참여하여 3 자간 통신이 이루어 진다면, Participants 는 3 이되지만, Session 은 그대로

1 개가 된다. RTP 는 실시간 멀티미디어 전송을 목적으로 만들어졌기때문에, 최소의 오버헤드 및 프리젠테이션을 위한 데이타 품질에 대한 모니터링 능력이 없다. 또한 Session 으로부터 Participant 간의 추가/삭제에 대한 모니터링 기능역시 없다.

이러한 단점을 극복하기 위해서 RTCP 프로토콜이 개발되었고, RTCP 는 프리젠테이션의 데이타 품질과 함께 RTP session 에 대한 모니터링 기능을 제공한다. HTTP 나 FTP 같은경우에는 MIME 정보(예를들어서 audio/x-wav)등과 같이 단일한 MIME 정보에 의해 그 데이타의 종류가 구별된다. MIME 타입은 비디오, 텍스트, 또는 서로 다른 오디오 포맷과 같은 MIME 정보를 갖는 스트림 전송을 제약한다. 이와는 대조적으로 RTP 에서는 스트림내에서 임의의 위치에서 다양한 미디어 타입이 서로 공존할 수 있기 때문에, RTP 역시 각각의 데이터를 구분하기 위한 식별자로서 Payload type(PT)을 이용한다. 실제로 이러한 payload type 번호는 RTP 규격으로 지정되어 있으며, 사용자에 의해 임의로 바뀌어질 수 없다. RTP 의 다양한 미디어 타입에 대한 단일 스트림내에서의 공유기능은 특히 네트워크의 전송대역에 따라 전송량을 달리해야하는 실제상황에서 그 기능을 크게 발휘할 수 있다. 예를들어서 네트워크의 트래픽에 대한 부담이 없는경우 실시간 비디오와 음성을 보내고, 트래픽에 대한 부담이 클 경우 오디오는 PCM 으로 압축을 수행하고 비디오는 H.263 과 같이 높은 압축율을 나타내는 다른 압축 타입으로 미디어 데이타를 전송할 수 있다.

RTSP

RTSP 는 RTP 보다 상위 단계의 프로토콜로서 멀티미디어 스트림에대한 Command/ Control 기능을 제공한다. RTSP 는 애플리케이션으로 하여금 서버로부터 데이타를 요청하게 하거나, 멀티미디어 회의에 참여하게 할수있다. UDP 와 같이 RTSP 도 비연결지향 프로토콜이며, 각각의 스트림은 세션 ID 에 의해 서로 구별이된다. 그러나 주의할 것은 RTSP 의 control 요청은 연결이 보장된 프로토콜 즉, TCP 를 통해 전송된다는 점이다. 실제로 control request 는 멀티미디어 데이타를 포함하지 않기 때문에, 안정된 연결지향의 프로토콜을 이용한다는 것에 주의해야한다. 다음으로는 RTSP 가 JMF 에서 어떻게 구현되는지를 살펴보자

JMF 와 RTSP 의 관계

RTSP 의 control request 는 이미 JMF 에서 서로 유사한 의미의 메쏘들로 정해져있다.

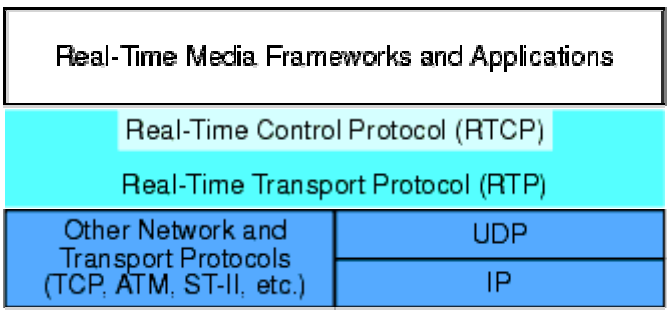
RTSP Control	JMF Counterpart	Explains

Request		
describe	DataSource.getContentType()	retrieves a description of the media stream
setup	Manager.createPlayer()	informs the server of the transport protocols for the media streams
play	start()	request tells the server to start sending data
pause	stop()	stop the stream
tear down	close()	stop the stream delivery and releases all resources associated with the stream

[표 1] RTSP 와 JMF 의 관계

여기서 주의할 점이 있다. RTSP 는 HTTP 와 연동해서 사용할 수 있도록 설계되었으나 , 실제로는 주요한 차이점이 있다. RTSP 는 미디어 스트림의 전송을 위해서 별도의 프로토콜을 제공하지만, HTTP 의 경우 control request 와 미디어 데이타의 전송을 위해서 단지 TCP 만을 이용한다는 점이다. RTSP 가 별도의 구별되는 프로토콜을 이용함으로서 미디어 스트림이 가장 우수한 효율의 방식으로 전송됨을 알수있다. 일반적으로 RTSP 에서 미디어 전송을 위해서는 RTP 를 이용하지만, 다른 프로토콜도 특정 상황에 맞게 서로 바뀌어 이용될 수 있다.

RTP 구조 살펴보기



[그림 1] RTP 의 구조

RTP는 실시간 데이터에 대하여 양 종단간의 네트워크 전송 서비스를 제공하며, UDP 상위단에서 이용된다. 또한 RTP는 네트워크에서 유니캐스트 전송방식과 멀티캐스트 전송방식 서비스 모두에 이용이 가능하다. 유니캐스트 네트워크 서비스를 통하여 각각의 데이터 복사본들이 소스로부터 각 수신경로로 전송이 가능해진다. 또한 멀티캐스트 네트워크 서비스를 통하여 데이터는 단지 소스의 위치로부터 전송이 되어지며, 다중 수신지역에 대한 데이터의 전송에 대한 책임은 네트워크 그 자체가 지게된다. 표준적인 IP 프로토콜에서 이러한 멀티캐스팅을 지원하기 때문에 멀티캐스팅 전송방식은 비디오 화상회의 시스템 같은 다양한 데이터가 요구되는 환경에서 보다 더 효율적으로 이용될 수 있다.

RTP 서비스

RTP는 사용자로 하여금 전송되어지는 데이터의 타입정보를 구분하게 하며, 어떠한 순서로 데이터 패킷이 표현되어지는지를 결정하며, 각각의 서로 다른 미디어 데이터 소스로부터의 데이터 동기화 방법을 제공하게 된다. 이미 앞서 언급한바 있지만, 여기서 주의할점은 사실상, RTP 데이터 패킷은 전송되어지는 순서대로 수신단에서 수신되는것을 보장하지 않는다는 점이다. 또한 전송되는 모든 데이터가 전혀 도달하지 못하는것에 대한 처리에 대하여 전혀 보장을 하지도 않는다. 그러므로 송신측의 데이터 패킷 순서를 재 구축하는 것은 전적으로 수신측의 수신자에게 달려있으며 패킷의 헤더에서 제공되는 정보를 이용하여 손실패킷을 검출하는 것 또한 전적으로 수신단에서의 책임이 된다. RTP 가 주기적인 전송에 대한 어떠한 메커니즘이나 기타 다른 서비스 보장적인 방법을 제공하지 못하는 반면에, 데이터 전송의 퀄리티에 대한 모니터링의 수단을 제공하는 것이 바로 RTCP 이다. 만약 특별한 애플리케이션을 위하여 퀄리티가 특별히 요구된다면, 연결지향형 서비스를 제공하는 리소스 보존 프로토콜상에서 RTP가 이용되어질 수 있다.

RTP 구조

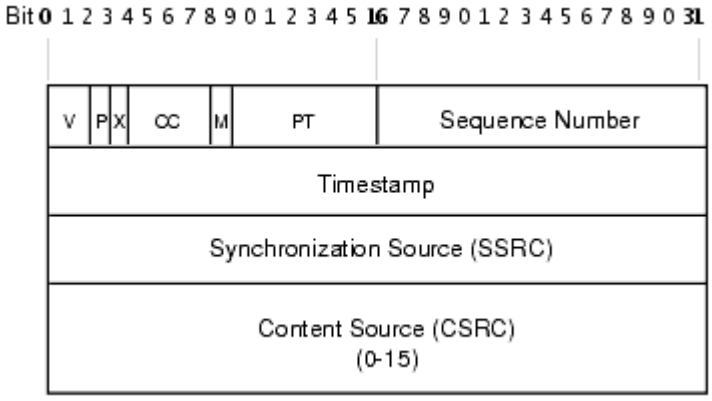
직접적인 RTP 구조를 살펴보기 이전에 먼저 앞에서 언급한바 있는 Session과 Participant 의 개념에 대하여 다시 한번 확인하고 넘어가자. RTP Session이라는 것은 RTP를 통하여 연결되는 일련의 애플리케이션들의 집합들을 연결하는 연결점의 역할을 한다고 정의하였다. 이러한 Session들은 네트워크의 어드레스와 이용포트의 쌍으로 각각 다른 session들과 구분되어 지게된다. 여기서 주의할점은

session 연결시에는 내부적으로 2 개의 포트를 이용하게 되는데 하나의 포트는 미디어 데이터의 실제전송을 위한 포트와 또하나의 포트는 RTCP 컨트롤 데이터를 위해 사용되어지는 포트이다. Participant 라는 것은 단일머신, 호스트 혹은 세션에 참여하고 있는 사용자를 의미한다. 하나의 세션내에서의 participant 는 receiver, sender, receiver-sender 의 3 가지 종류로 구분이 된다. receiver 라는 것은 수동적인 데이터 수신만을 하는 participant 를 의미하며, sender 는 실제적인 데이터 전송역할을 수행하는 participant, sender-receiver 는 데이터의 송수신을 모두하는 participant 를 의미한다. 또하나 RTP 에서 주의할 점은 전송하고자하는 각각의 미디어 타입은 하나의 세션에서 동시에 전송되어 질 수 없다는 점이다.

즉, 각각의 미디어 타입은 서로 다른 세션을 통하여 전송되어져야 한다는 것이다. 실제로 RTP 를 처음 접하는 많은 독자들이 구현상 오류를 범하는 부분이기도 하다. 하나의 예를 들어보자. 만약 네트워크 회의 시스템에서 영상과 음성을 동시에 이용한다면, 하나의 세션은 오디오 데이터를 전송하기 위해 사용하고, 또 다른 하나의 세션을 이용하여 비디오 데이터를 전송해야 한다. 이렇게 각각의 미디어 타입의 전송을 위한 세션을 분리 함으로서 세션에 참여하는 각각의 participant 들은 네트워크 상의 모든 데이터를 수신하는 것이 아니라 원하는 타입의 미디어 데이터만 수신하는 것이 가능하게 된다. 이러한 기능을 통하여 네트워크의 대역폭이 크지 않은 경우에는 영상과 음성 데이터중 음성부분만 수신할 수 있고, 네트워크의 대역폭이 커지면 영상과 음성 및 기타 다른 미디어 타입의 데이터들도 수신할 수 있도록 선택이 가능하게 된다.

Data Packets

이젠 데이터 패킷에 관하여 알아보도록 하자. 하나의 세션에서 전송되어지는 미디어 데이터는 일련의 패킷들의 집합으로 나뉘어 전송되어진다. 이때 미디어 데이터가 생성되어지는 특정한 소스로부터 발생되는 일련의 데이터 패킷의 집합을 RTP Stream 이라고 한다. 이 부분에서 살펴보아야 할 것이 바로 하나의 스트림에서 각각의 RTP 데이터 패킷은 구조화된 헤더부분과 실제적인 데이터를 포함하는 2 부분으로 분리가 된다는 점이다. 아래의 그림을 살펴보자.



[그림 2] RTP Packet 헤더구조

그림에서 보는바와 같이 RTP 데이터 패킷은 약간 복잡한 형태를 띄고 있다.
간략하게 나마 그 의미를 살펴보도록 하자.

The RTP version number (V): 2 bits로 할당되어 있으며 현재의 RTP 스펙으로 규정되어 있는 값은 2이다.

Padding (P): 1 bit로 할당되어 있으며, 만약 이 padding 비트가 1이 되면, 실제 payload의 일부분이 아닌 하나혹은 그 이상의 바이트 정보들이 패킷의 마지막부분에 추가되어져 있다는 것을 표시한다. 패킷내에서의 마지막 바이트 값은 실제 padding으로 추가된 바이트수를 표시하게되며, 이러한 padding 정보는 미디어 데이터의 암호화나 Digital Water Marking 방법을 위하여 사용되어 질 수 있다.

Extension (X): 1 bit로 할당되어 있으며, 만약 이 비트가 1이되면, 고정 크기의 헤더 정보 다음에 하나의 헤더 확장정보가 추가된다. 이러한 헤더확장 메커니즘을 통하여 RTP 헤더에 특정한 정보들을 추가할수 있게된다.

CSRC Count (CC): 4 bit로 할당되어 있으며, CSRC의 갯수는 뒤따라오는 고정헤더를 구분하게 된다. 만약 CSRC값이 0이라면, 동기화 소스가 바로 이 payload의 소스가 된다.

Marker (M): 1 bit로 할당되며 이러한 marker 비트는 특정 미디어 프로파일에 의해 정의가 된다.

Payload Type (PT): 7 bit로 할당이 되며, 실제 payload 포맷을 서술하는 미디어 프로파일 테이블의 인덱스 번호를 나타낸다. 오디오나 비디오 미디어들에 대한 payload 값들은 RFC 1890으로 명시되어 있다.

Sequence Number: 16 bits 로 구성이 되며, 이 Sequence Number 는 일련의 나열된 패킷에서 현재의 payload 가 어떤 위치에 있는가를 나타내는 유일한 값이다. 이러한 packet Number 는 각 패킷이 전달될때마다 하나씩 증가하게 된다.

Timestamp: 32 bit 로 구성이 되며, paylaod 에서 첫번째 바이트의 샘플링 인스턴스를 나타낸다. 예를들어서 동일한 비디어 프레임정보에서 동일한 시간에 생성된 몇개의 연속적인 패킷들이라면 이러한 패킷들은 동일한 timestamp 값을 가질수도 있다.

SSRC: 32 bit 로 구성되며, 동기화 소스를 구분한다. 만약 CSRC 값이 0 이라면 , 이 payload 가 전체 동기화 소스가 된다. 만약 CSRC 값이 0 이 아니라면 SSRC 는 mixer 를 나타내게된다.

CSRC: 32 bit 로 구성되며, 이 payload 를 위한 contributing 소스를 구분하게된다. 실제 contributing 소스의 갯수는 CSRC 값에 의해 결정이 되며 16 개 소스까지로 구성되어질 수 있다.

위와 같은 값들중 실제로 독자들이 관심있게 살펴봐야 할 부분은 바로 Payload Type 이다. 이 값을 통하여 현재 세션에서 이용되어지는 패킷이 오디오 데이터 인지 비디오 데이터인지의 구분과 더불어 특정 압축 포맷으로 압축이 된 데이터인지를 나타내어준다. 각 오디오 비디오 압축 방식별로 이 payload type 의 값이 이미 규정이 되어있기 때문에 좀더 관심있는 독자들은 RTP 규약을 참고하기 바란다.

Control Packets

자. 이제는 하나의 문제를 풀어보도록 하자. 데이터 통신 과정에서 데이터가 잘 전송이 되는지 세션에서 누가 참여하고 누가 세션을 이탈했는지, 송수신은 과연 잘 되고 있는지 등등 이러한 정보는 어떻게 알려지게될까. 이러한 궁금증에 대한 해답이 바로 컨트롤 패킷이다. 다음으로 알아볼 부분은 컨트롤 패킷 부분이다. 실제 하나의 세션을 통하여 전달되어지는 패킷은 위에서 살펴본 바와같이 데이터 패킷뿐만이 아니라 컨트롤 데이터, 즉 RTCP 패킷도 세션내의 모든 participant 들에게 주기적으로 전달이 되어진다는 것이다. 이러한 RTCP 컨트롤 패킷에 대하여 알아보자. JMF 에서도 이러한 RTCP 정보에 접근하는 API 들을 제공하여 준다. RTCP 패킷은 세션 participant 와 하나의 데이터 포트를 통하여 전송되어지는 미디어의 소스에 관한 정보, 이제까지 전송되어진 데이터에 대한 통계정보에 관한 정보들을 가지고 있다. 이미 눈치챈 독자들도 있겠지만, JMStudio 를 구동하여 RTP 통신을 하다보면 현재 몇 바이트가 전송이 되었고, 몇바이트가 전송중 손실이 되었으며, 세션에 참여한 participant 들이 누구이고, 어떤 미디어 타입의 정보가 현재 전송및

수신되고 있는지를 확인할 수 있다. RTCP 패킷들은 다음과 같이 다양한 타입으로 구성되어 있다.

- Sender Report
- Receiver Report
- Source Description
- Bye
- Application-specific

RTCP 패킷의 또 하나의 특징으로서, RTCP 패킷은 중첩가능한 패킷이며, 적어도 report 패킷과 소스정보 패킷을 포함하는 복합 패킷으로서 전송이 된다. 이러한 report 패킷을 통하여 수신측의 report, 송신측의 report 정보를 취할 수 있으며, 현재 수신되는 데이터의 원 소스가 어디인지를 소스정보 패킷을 통하여 구별해 낼 수 있게된다. 하나의 세션내에서 참여한 모든 participant 들은 RTCP 패킷을 보내게된다. 바로 최근에 데이터 패킷을 보낸 하나의 participant 가 연속해서 sender report 를 보낸다. 이러한 SR(Sender Report) 정보는 전체 패킷의 갯수와 바이트 수 및 서로 다른 세션으로부터 미디어 스트림의 동기화에 이용되기위한 정보를 포함하게된다. 세션에 참여한 participant 들은 또한 participant 들이 어떠한 데이터 패킷을 수신하였는지를 알려주기위한 모든 소스를 위한 RR(Receiver Report)를 전달하게 된다. 이러한 RR 정보는 전체 손실된 패킷의 갯수와 수신된 최대 시퀀스 번호와 송신측과 수신측사이의 round-trip 시간지연을 측정하기위해 이용되는 timestamp 정보를 포함하게된다. 하나의 복합 RTCP 패킷의 첫번째 패킷은 바로 report 패킷이다. 심지어 어떠한 데이터도 수신이나 송신이 되지 않더라도 수신측은 아무런 정보도 없는 RR 정보를 보내게된다. 모든 복합 RTCP 패킷들은 반드시 SDES(Source Descrip tion) 요소들을 포함하는 소스 정보를 포함해야 한다. 이러한 SDES 정보는 각 미디어 소스들의 발생위치를 구분해주는 CNAME(Cannonical Name)을 포함하게된다. 또한 미디어 데이터가 발생되는 소스의 이름, E-mail 주소, 전화번호, 지리적위치정보, 애플리케이션 이름, 현재 소스의 정보를 표시하는 메시지등의 추가정보를 포함할 수도 있다. 만약 소스가 더이상 active 상태가 아니라면, RTCP Bye 패킷을 전송하게된다. BYE 패킷정보는 미디어 데이터를 발생시키는 소스가 참여한 세션에서 종료하는 원인에 대한 정보를 포함하게된다. RTCP APP 패킷은 RTP 컨트롤 포트를 통하여 커스텀 정보를 전송하고 애플리케이션의 정의를 위한 메커니즘을 제공하게된다.

RTP Applications

RTP 애플리케이션은 일반적으로 네트워크를 통하여 데이터의 수신이 가능한 RTP Client 와 네트워크를 통하여 데이터 송신이 가능한 RTP Server 로 구분이 되어진다. 하나의 예로서,

영상회의 시스템 애플리케이션의 경우 영상및 음성 데이터의 캡쳐 및 전송을 동시에 하고 또한 네트워크를 통하여 데이터를 수신하기 한다. 이러한 경우에는 RTP Server 의 기능과 RTP Client 의 기능을 모두 수행하는 복합 애플리케이션이 된다. 이제 RTP 를 통하여 네트워크를 통한 송수신의 사례들을 살펴보자.

(1) Receiving Media Streams From the Network
화상회의 애플리케이션 RTP session 으로부터 media stream 을 받아서 그것을 콘솔로 보낼 필요가 있다.
전화 응답 기계 애플리케이션은 RTP session 으로부터 media stream 을 받아서 그것을 파일에 저장할 필요가 있다.
대화나 회의를 기록하는 애플리케이션은 RTP session 으로부터 media stream 을 받아서 그것을 콘솔로 보내고 파일에 저장할 필요가 있다.

(2) Transmitting Media Streams Across the Network
RTP Server 애플리케이션들은 캡쳐되고 저장된 media streams 을 전송한다.
Media streams 은 multiple media formats 으로 인코드되고, 하나 이상의 RTP sessions 에 보내진다.

JMF 에서의 RTP 구현
이제까지 JMF 에서 지원하는 RTP 를 이용하기 위하여 많은 부분을 RTP 구조에 할애하였다. 독자들은 특히 RTP 헤더구조의 payload type 정보와 RTCP 컨트롤 패킷정보에 대하여 좀더 세심히 살펴보아야 할것이다. 자. 이제까지 살펴본 RTP 의 기본 개념을 정리하면서 이젠 JMF 에서 과연 어떻게 RTP 스펙을 구현하고 있으며, 프로그램 구현 측면에서는 어떤 면을 고려해야 할런지를 생각해보자.

JMF 에서는 RTP 와 관련된 패키지들은 다음과 같이 정의되어 있다.
javax.media.rtp
javax.media.rtp.event
javax.media.rtp.rtp

위와 같은 패키지들에 속해있는 수많은 메쏘드들을 이용하여 JMF 에서 미디어 데이터에 대한 재생이나 송수신기능을 구현할 수 있으며 JMF plug-in 기능을 통하여 추가적인 RTP 포맷이나 동적 payload 의 할당도 가능하다. 먼저 JMF 에서 제공하는 RTP 관련 API 지원사항을 알아보자.

JMF RTP API 를 통하여 다음과 같은 기능을 수행한다.

- RTP sessions 을 생성하고 세션열기 수행
- 세션으로 들어오는 모든 스트림을 수신하고, 스트림의 데이터소스를 이용하여 JMF Player 또는 Processor를 생성하고 재생
- 수신되는 스트림의 통계정보를 획득하고, RTCP 정보를 통한 세션의 모니터링 수행
- 데이터 소스로부터 RTP 스트림을 전송
- Send 스트림의 통계정보 획득 및 RTCP를 이용한 모니터링 방법제공
- RTPSessionManager 를 이용한 동적 Payload의 이용 및 유저 Plug - In 기능.

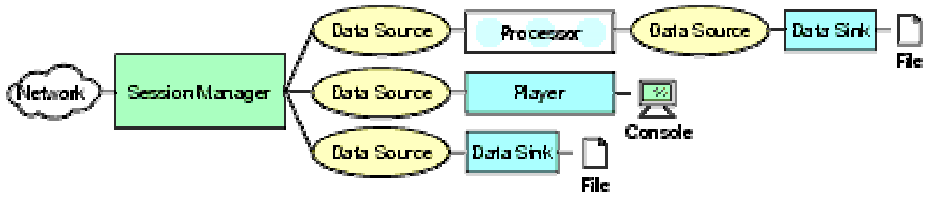
JMF 2.1 에서 추가된 사항은 다음과 같다.

- JMF player를 이용한 RTP 스트림의 표현.
- JMF 프로세서와 데이터 싱크를 이용한 RTP 스트림의 전송.
- 동적 RTP Payload 정보의 표현과 전송.
- 한 세션내에서의 동적 Payload 타입의 스위칭 기능.
- RTCP를 통한 세션의 통계정보와 모니터링 기능 제공.
- 전송과 재생을 위한 Unicast, multicast,broadcast 세션 기능제공 .
- 네트워크 프로토콜에 독립적인 전송기능과 재생기능 수행

자. 이제 2가지의 시나리오를 생각해보자. 모든 통신 과정이 그렇듯 일단
송신기능을 위한 JMF 의 구현 부분과 수신기능을 위한 JMF 의 구현 부분이다.

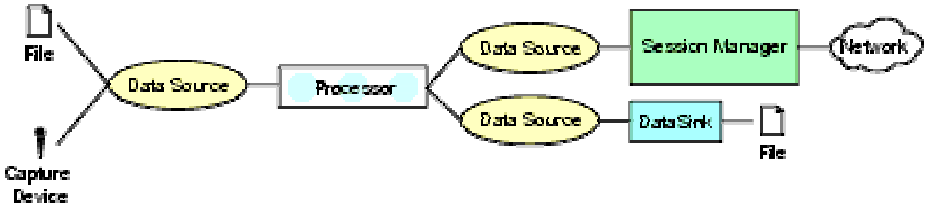
시나리오 첫번째 - 네트워크를 통한 RTP 데이타의 수신
첫번째 과정은 네트워크를 통한 RTP 데이터의 수신과정이다. JMF 에서는 모든 RTP
통신을 주관하는 기능을 제공해주는데 그것이 바로 SessionManager 이다.
네트워크를 통해 전달되는 RTP 데이터는 일단 Session Manager 에 의해 세션별로
분리가 되어서 각각의 별도의 데이타 소스들로 분리가 된다. 만약 수신단에 저장을
하기전에 수신단에서 알맞은 프로세싱을 하고 그 후에 저장을 하고자 한다면
데이타 소스를 프로세서로 넘기고 프로세서의 출력을 다시 데이타 소스로 만들고,
그것을 데이타 싱크로 넘긴후 데이타 싱크에서 저장을 하게 된다. 또한
수신단에서 수신받은 데이터를 재생 하고자 하는 경우에는 Session Manager 에서
나오는 데이타 소스를 Player 에게 넘겨주면 우리가 화면에서 데이터를 볼수있다.
또 다른 가능성은 Session Manager 를 통해서 나온 데이타 소스를 아무런 가공도

하지 않고 단순히 저장하기 위해 바로 DataSink 로 넘겨서 저장하는 방법이다.



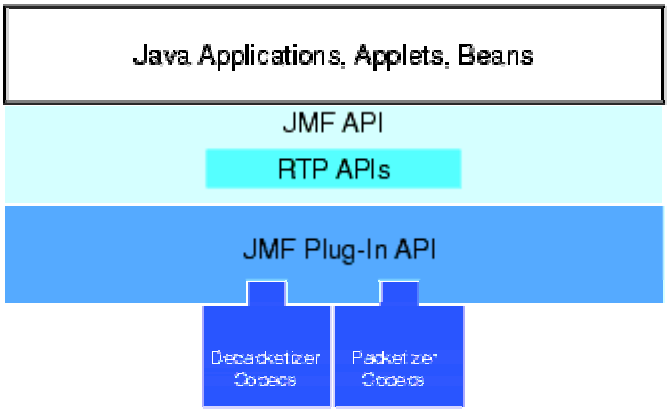
[그림 3] JMF 에서의 RTP 를 통한 데이터 수신

시나리오 두번째 - 네트워크로 RTP 데이타 송신
이제 데이터를 송신하는 과정을 살펴보자. 먼저 데이타 송신을 위해 일단 보낼 데이터를 획득해야한다. 이런 데이타 획득을 위해서 하드디스크에 저장된 파일이나 캡쳐카드를 통한 오디오/ 비디오 미디어 데이타도 가능합니다. 이러한 데이타들은 DataSource 로 보내어 지고, 프로세싱을 위해서 Processor 생성을 위한 입력단이 된다. Processor 의 출력물로 다시 DataSource 형태로 출력물이 나오면 그것을 Session Manager 를 통해 네트워크로 전송하거나, 혹은 DataSink 를 이용해서 하드디스크에 저장할 수 있다.



[그림 4] JMF 에서 RTP 를 이용한 데이터 송신

JMF 의 구조와 RTP
자. 이제 [그림 5]를 살펴보자. JMF API 내부에 실제적인 RTP API 가 존재하게된다. 그러므로 독자들은 직접적으로 하위레벨인 통신의 흐름에 관하여 전해 고려할 필요가 없으며, 단지 송수신을 주관하는 세션 메니져를 생성하고, 이렇게 생성된 세션 메니져에게 알맞은 정보를 제공하는 것으로 통신을 구현할 수 있다.



[그림 5] JMF 와 RTP API 구조

자. 그렇다면 Session Manager 란 과연 무엇일까. 그림에서도 살펴본바와같이 세션 메니져는 송신측, 수신측 모두에서 사용이 되었다. 세션 메니져란 RTP Session 들간의 데이터 통신 통제 역할을 하고, session 에 참가한 모든 participants 에 대한 추적과 관리, 전송되어지는 스트림에 대한 관리기능을 수행한다. 즉 JMF 의 RTP 통신을 전체적으로 담당하는 종합적인 역할을 한다.

또한 세션의 상태정보를 관리하고, RTCP Control channel 을 취급하며, Sender, Receiver 를 위한 RTCP 를 지원한다. 무엇보다도 중요한 Session Manager 의 기능은 바로 우리가 작성한 애플리케이션을 초기화한 후, 세션에 참여를 시작하고, 애플리케이션에 의해서 생성된 각각의 스트림을 제거하고, 전체 세션을 닫는 기능등을 수행한다는 것이다. 결국 JMF 에서의 통신 프로그램 작성은 세션을 생성하고, 세션에 입력될 데이터를 캡쳐하여 가공하여 세션의 입력으로 이용하고 세션에 참여한 Participant 들에게 데이터를 전송하거나 수신하며 이러한 기능을 수행하는 중에 발생하는 각종 이벤트에 대응하는 과정이라고 말할 수 있다.

자. 이제 독자들이 할일은 앞에서 살펴본 RTP 스펙이 JMF 를 통하여 어떻게 구현되었고 , 우리는 이러한 정보를 어떻게 이용할 수 있는가이다. 먼저 Session 이 수립된후에 획득할 수 있는 세션의 통계정보인 Session Statistics 에 관하여 살펴보자. 여러분들은 이미 JMStudio 를 통하여 RTP 통신을 테스트할때 아마도 Statistics 라는 것을 보았을 것이다. 이 Statistics 를 선택을 하면 무엇인가 데이타가 변화하고 있는 것을 본 경험이 있을 것이다. 이렇듯이 Session Manager 에서는 세션 내부에서 주고 받는 RTP, RTCP 패킷에 대한 통계 정보를 관리한다. 이러한 통계 정보치는 각각의 스트림에 대한 해당 정보이며, 크게 2 개로 구분이 된다.

GlobalReceptionStat : 특정 세션에 대한 전역 수신 통계치
GlobalTransmissionStat : 특정 세션에 대한 전역 송신 통계치

또한 위의 전역적인 통계치 이외에도 각각의 Participants들을 위한 통계치를 제공하기도 한다.

ReceptionStat : 특정 개별 participant에 대한 소스 수신 통계치
TransmissionStat : 특정 개별 participant에 대한 소스 전송 통계치

자. 그럼 위와 같은 통계 정보를 어떻게 얻을 수 있을까. 아래의 프로그램을 통하여 간략하게나마 각 통계치를 얻는 방법에 관하여 알아보자. 아직까지는 아래에 나열된 모든 부분을 전부 확인할 필요는 없다. 이 부분에서는 단지 이러한 통계정보를 이용할 수 있다는 정도만 확인해 두기 바란다. 그러나 한 가지 주의할점은 전역 통계 정보는 세션의 세션 메니져로부터 얻어오는 것이고, 각각 participant의 통계정보는 각 participant가 제공하는 데이터 스트림으로부터 얻어온다는 점이다.

[리스트 1] RTP 통계 정보 구하기 - 전역 통계 정보 얻기

```
public void run(){
    while(true){
    // 전역 정보 구하기
    GlobalReceptionStats stats = mymgr.getGlobalReceptionStats();
    pktsrecdtf.setText( (new Integer(stats.getPacketsRecd())).toString());
    bytesrecdtf.setText((new Integer(stats.getBytesRecd())).toString());
    badrtppktstf.setText((new Integer(stats.getBadRTPkts())).toString());
    localcolltf.setText((new Integer(stats.getLocalColls())).toString());
    remotecolltf.setText((new Integer(stats.getRemoteColls())).toString());
    pktsloopedtf.setText((new Integer(stats.getPacketsLooped())).toString());
    transfailedtf.setText((new Integer(stats.getTransmitFailed())).toString());
    rtcprectfd.setText((new Integer(stats.getRTCPRecd())).toString());
    srrecdtf.setText((new Integer(stats.getSRRecd())).toString());
    badrtcppktstf.setText((new Integer(stats.getBadRTCPPkts())).toString());
    unknowntf.setText((new Integer(stats.getUnknownTypes())).toString());
    malrrtf.setText((new Integer(stats.getMalformedRR())).toString());
    malsdestf.setText((new Integer(stats.getMalformedSDES())).toString());
```

```
    malbyetf.setText((new Integer(stats.getMalformedBye())).toString());
    malsrtf.setText((new Integer(stats.getMalformedSR())).toString());
    if (isShowing() && alwaysOnTop.getState())
        show();
    try{
        Thread.sleep(1000);
    }catch (InterruptedException e){}
  }
}
```

[리스트 2] RTP 통계 정보 구하기 - 개별 Participant 통계 정보 얻기

```
ReceptionStats stats = stream.getSourceReceptionStats();
pdulost.setText( (new Integer(stats.getPDUlost())).toString());
pduproc.setText((new Integer(stats.getPDUProcessed())).toString());
pdumisord.setText((new Integer(stats.getPDUMisOrd())).toString());
pduinvalid.setText((new Integer(stats.getPDUInvalid())).toString());
pdudup.setText((new Integer(stats.getPDUDuplicate())).toString());
```

이번에는 세션에 참여하는 participant 에 대하여 알아보자. Session Manager 는 Session 에 참여하는 모든 Participants 에 대해 그 관리를 수행한다. 각각의 Participants 들은 Participant 인터페이스를 구현한 클래스의 인스턴스로서 표현이 되며 Session Manager 는 Session 내에서 이제껏 본적이 없거나 마지막으로 사용된후 타임 아웃된 source description(SDES) 정보와 canonical name(CNAME)을 갖는 RTCP 패킷이 전달될때마다 새로운 Participant 를 만들어 낸다. 이렇게 생성된 Participant 는 데이타를 전송만하는 passive, 혹은 하나 이상의 RTP 데이타 스트림을 전송하는 active 로 구분되어진다.

통신과정을 스트림의 흐름으로 이해할 때 이러한 스트림은 송신을 위한 스트림과 수신을 위한 스트림으로 구분이된다. Session Manager 는 세션내의 각각의 RTP 데이타 패킷에 대한 RTPStream 객체를 갖고 있으며, 이러한 RTPStream 에는 두 종류가 있다.

ReceiveStream : 원격 participant 로부터 수신되어진 스트림의 표현
SendStream : 네트워크를 통해서 전송할 스트림의 표현으로서 Processor, 또는 입력 DataSource 로부터 생성된다.

주의할점은, ReceiveStream 은 Session Manager 가 새로운 RTP data 를 검출해낼 때마다 자동적으로 생성이 되지만, SendStream 은 Session Manager 의 createSendStream 을

호출해서 생성한다는 것이다.

RTP Events

RTP 와 관련된 이벤트는 javax.media.rtp.event 패키지에 정의되어 있다. 이벤트의 등록과 처리는 예전에 우리가 애플릿이나 애플리케이션으로 미디어 플레이어를 만들고 이벤트 처리를 했던것과 거의 유사하다. RTP 관련 이벤트 처리는 우선 RTP 리스너를 구현해야 하고, Session Manager 를 이용해서 RTP 리스너를 등록하는 과정을 거쳐야한다. 자. 먼저 JMF 에서 지원하는 RTP 관련 이벤트들에 대하여 살펴보도록 하자.

SessionListener

먼저 통신을 직접 실행하는 세션에 관하여 세션의 상태 변화를 감지하는 리스너이다. 이 리스너는 새로운 Participant 가 세션에 추가되는 상황이 발생될 때 이용할 수 있다. 이 Session Listener 에는 두가지의 이벤트가 있다.

- NewParticipantEvent : 세션에 새로운 participant 가 참여하는 것을 알려줌
- LocalCollisionEvent : participant 의 동기화 소스가 이미 사용중임을 알려줌

이 이벤트의 구현 부분은 다음과 같다. 아래의 부분에서는 Session 에서 발생하는 이벤트중에서 NewParticipant 이벤트를 감지하고 이 이벤트가 로컬에서 구동되는 자기 자신의 Participant 인지 혹은 원격지에서 참여하는 Participant 의 정보인지를 구별하여 준다.

[리스트 3] SessionEvent 이용하기

```
public void update( SessionEvent event){
        Participant part = null;
        ReceiveStream stream = null;
        if (event instanceof NewParticipantEvent){
            part = ((NewParticipantEvent)event).getParticipant();
            if (part == null)
                return;
            if (part instanceof LocalParticipant)
                // Local 컴퓨터에서 발생한 정보처리 부분
            if (part instanceof RemoteParticipant)
                // 원격지 컴퓨터에서 발생한 정보처리 부분
            return;
        }
```

```
    }
```

SendStreamListener

SendStreamListener 는 세션 매니져를 통하여 수신측으로 전송될 전송될 RTP stream 의 상태 변화 감지를 위한 리스너이다. 즉, 데이터를 송신하기 위한 준비와 송신시작, 중단 및 포맷변화에 대한 상태 변화를 감지한다. 구체적인 기능은 다음과 같다.

- local participant 에 의해 새로운 send stream 이 생성될때
- send stream 의 생성에 이용된 DataSource 의 전송이 시작/중지 되었을때
- send stream 의 포맷과 payload 변화되었을때
- 총 5 가지의 이벤트로 구성되어져 있다.
- NewSendStreamEvent
- ActiveSendStreamEvent
- InactiveSendStreamEvent
- LocalPayloadChangeEvent
- StreamClosedEvent

아래에 이 이벤트의 구현 사례를 살펴보자. 아래의 리스트에서는 송신측에서 수신측으로 데이터를 전달하고자 할 때 발생하는 이벤트중 새로운 스트림 발생에 대한 이벤트 및 전송하려고 하는 스트림의 중단을 알려주는 이벤트 처리를 구현하였다.

[리스트 4] SendStreamEvent 의 이벤트 처리 부분

```
public void update ( SendStreamEvent evt ) {
        if ( evt instanceof NewSendStreamEvent ) {
            streamSend = ((NewSendStreamEvent)evt).getSendStream();
            this.setVisible ( true );
        }
        else if ( evt instanceof StreamClosedEvent ) {
            closeWindow ();
        }
    }
```

ReceiveStreamListener :

ReceiveStreamListener 는 세션 매니져를 통하여 수신될 수신될 RTP stream 의 상태변화 감지하기 위한 리스너이다. 이 리스너의 주요 기능은 다음과 같다.

- 새로운 receive stream 이 생성되었을때
- 데이타 전송의 시작/중단시에
- 데이타 전송이 Time out 되는경우
- 이전에 제대로 처리되지 못한 Orphaned ReceiveStream 이 participant 와 연결되었을때
- RTCP APP 패킷이 수신되었을때
- receive 스트림의 포맷이나 payload 변경시
- 총 7 개의 관련 이벤트로 구성되어 진다.
- NewReceiveStreamEvent
- ActiveReceiveStreamEvent
- InactiveReceiveStreamEvent
- TimeoutEvent
- RemotePayloadChangeEvent
- StreamMappedEvent
- ApplicationEvent

위의 이벤트들에 대한 구현 예를 들어보자. 아래의 구현 부분에서는 새롭게 수신측에 스트림이 감지되었을때의 이벤트 처리 부분을 통하여 수신되는 스트림과 이를 보내준 송신측의 Participant 정보를 얻어오는 부분을 구현하였다. 또한 시간경과에 대한 이벤트 처리를 보여준다.

[리스트 5] ReceiveStream 이벤트의 처리부분

```
public void update( ReceiveStreamEvent event){
        Participant sender = null;
        ReceiveStream stream =  null;
        if (event instanceof NewReceiveStreamEvent){
            stream = ((NewReceiveStreamEvent)event).getReceiveStream();
            if (stream == null)
                return;
            sender = stream.getParticipant();
            if (sender != mypartc)
                return;
            updaterecvlist(ADD, stream);
        }
        if (event instanceof TimeoutEvent){
            stream =  ((TimeoutEvent)event).getReceiveStream();
            if (stream == null)
```

```
                return;
            sender = stream.getParticipant();
            if (sender != mypartc)
                return;
            updaterecvlist(DELETE,stream);
        }
        if (event instanceof ByeEvent){
            stream = ((ByeEvent)event).getReceiveStream();
            if (stream == null)
                return;
            sender = stream.getParticipant();
            if (sender != mypartc)
                return;
            updaterecvlist(DELETE,stream);
        }
    }
```

RemoteListener : 원격 participant 로부터 수신된 이벤트 혹은 RTP Control 을 처리하는 부분이다.

Message 에 대한 감지

- 3 개의 이벤트로 구성되어 진다.
- ReceiverReportEvent
- SenderReportEvent
- RemoteCollisionEvent

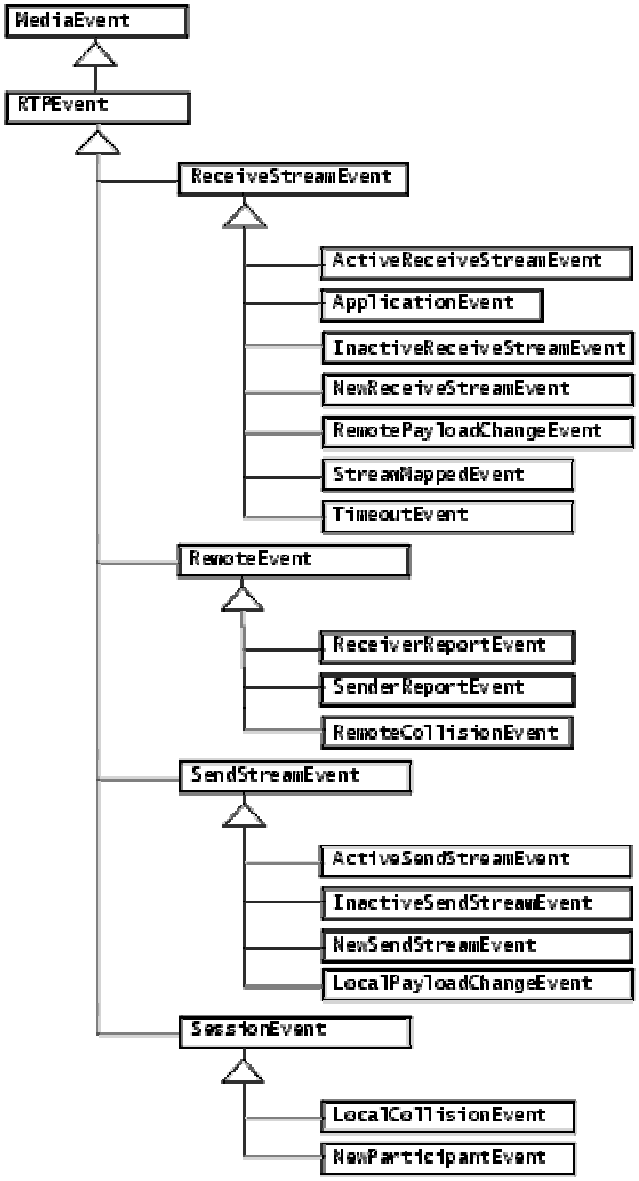
리모트 이벤트에 대한 처리는 아래와 같으며, 수신 및 송신단의 Report Event 처리를 구현하였다.

[리스트 6] 원격이벤트의 처리 부분

```
public void update( RemoteEvent event)
{
        Participant part = null;
        if (event instanceof ReceiverReportEvent){
            part =
          ((ReceiverReportEvent)event).getReport().getParticipant();
```

```
            if ((part != null) &&
            (part.getStreams().size() == 0)) {
                updateplist(activelistgui,DELETE,part);
                updateplist(passivelistgui,ADD,part);
            }
        }
        if (event instanceof SenderReportEvent){
            part = ((SenderReportEvent)event).getReport().getParticipant();
            if ((part != null) &&  (part.getStreams().size() > 0)){
                updateplist(passivelistgui,DELETE,part);
                updateplist(activelistgui,ADD,part);
            }
        }
    }
```

이상으로 JMF 에서 구현되어진 RTP 관련 이벤트들을 살펴보았다. 실제로 모든 이벤트 처리를 전부 독자들이 구현할 필요는 없으며, 여러분의 개발 애플리케이션의 요구에 맞도록 이벤트 처리를 해주어야 하며, 좀더 자세한 이벤트 처리 방식이나 각 해당 이벤트에 대한 상세한 설명은 Document 를 참조해주기 바란다.



[그림 6] RTP Event 의 구조

RTP Data Format

그렇다면 RTP 포맷에서 지원되는 영상 및 음성의 데이터 압축 방식을 JMF 에서는 어떤 형식으로 표현하며 이를 구현하고 있을까? 모든 RTP-specific data 는 RTP- specific format encoding 을 이용하는데, AudioForamt, VideoForamt 클래에 정의되어 있다. 예를들어 오디오 압축 포맷인 GSM 은 AudioFormat.GSM_RTP 와 같이 정의되어 있고, Jpeg 영상을 RTP 로 전송하기 위한 포맷인 VideoForamt.JPEG_RTP 와 같이 정의가 되어 있다. JMF RTP 패키지에서는 AudioFormat 클래스에서 살펴보면 표준 RTP-specific encoding 방식으로 아래와 같이 4 가지를 정의해 놓고 있다.

```
public static final String ULAW_RTP         = "JAUDIO_G711_ULAW/rtp";
public static final String DVI_RTP                = "dvi/rtp";
public static final String G723_RTP           = "g723/trp":
public static final String GSM_RTP           = "gsm/trp":
```

JMF RTP 패키지에서의 VideoFormat 에서 살펴보면 3 가지의 RTP 포맷을 정의해놓고 있다.

```
public static final String JPEG_RTP       = "jpeg/rtp":
public static final String H261_RTP         ="h261/trp";
public static final String H263_RTP         ="h263/rtp":
```

RTP 수신 방법

수신되는 RTP Stream 에 대한 표현방법은 Player 를 통해서 이루어 진다. RTP Session 으로부터 single stream 을 수신받고, 이를 presentation 하기 위해서 먼저, MediaLocator 를 이용해서 session 정보를 기술해 주어야 Player 를 생성할 수 있다. RTP session 에 대한 MediaLocator 표현은 다음과 같은 방식을 사용한다.

rtp://address:port[:ssrc]/content-type/[ttl]

이렇게 하면 player 가 생성되고 session 내의 첫번째 스트림에 connect 기능이 수행된다. 주의 할점은 MediaLocator 를 이용할 경우 단지 하나의 스트림만 표현이 가능하다는 것이다. 만약에 독자들이 특정 세션에 다중 스트림에 대한 처리를 원한다면 MediaLocator 를 이용하는 대신에 Session Manager 를 이용해야만 한다. Session Manager 를 이용해서 stream 이 session 에 추가될때마다 이벤트를 받아서 각각의 새로운 스트림에 더해서 player 를 생성해 주어야 한다. 또한 Session Manager 를 이용하면, session 에 대해서 직접적으로 모니터링과 컨트롤링을 할 수 있다.

RTP 송신 방법

RTP stream 데이타의 전송을 위해서는 Processor 를 이용한다. 아래에 데이타 전송을 위한 live capture source 로부터의 send stream 의 생성과정이 나타나 있다.

1. session 에 대해서 create, init, start 를 수행한다.
2. capture dataSource 를 통해서 processor 를 생성해야 한다.
3. processor 의 output 을 RTP-specific format 으로 변경해 주어야 한다.

이 부분에서 우리가 전송하고자 하는 데이타의 포맷을 위해서는 적절한 RTP packetizer CODEC 가 있는지 확인해야 한다.
4. processor 로부터 output DataSource 를 획득한다.
5. Session Manager 에서 createSendStream 을 호출하고, DataSource 로 인자를 넘겨야 한다.
6. SendStream 의 start, stop 을 통해 전송을 제어한다.

RTP 프로그래밍시의 주의점

이미 눈치가 빠른 독자들은 JMF RTP API 문서를 살펴보면서 이상하다는 생각을 하였을 것이다. RTP 통신을 하기위해서는 RTPSessionManager 를 이용하라고 하였는데, 실제 문서를 살펴보면 RTPSessionManager 는 인터페이스의 형태를 띄고 있다. 실제적인 SessionManager 의 구현은 바로 com.sun.media.rtp.RTPSessionMgr 에서 이 인터페이스를 구현하고 있으므로 독자들은 이 클래스를 생성하고 초기화함으로서 RTPSessionMager 의 각 기능들을 구현하게 된다.

또하나의 주의점을 살펴보자. 독자들은 RTP 수신프로그램을 통하여 유지되는 Buffer 를 제어할 수가 있다. 이러한 수신버퍼에는 Length 와 Minimum Threshold 정보를 나타내는 2 가지의 파라미터를 포함한다. Length 는 버퍼의 실제적인 길이를 나타낸다. 만약 수신되는 데이터가 이 버퍼의 크기를 초과하는 경우 Buffer queue 의 앞단에 놓이는 데이터들은 버려지게된다. Minimum threshold 는 RTP 데이터소스상에서 어떠한 데이터도 포워딩되기 이전의 Jitter Buffer 의 역할을 수행한다. 즉, 버퍼내의 데이터가 minimum threshold 보다 적은 경우에만 데이터가 포워딩되지 않으며, JMF 에서는 이러한 두개의 파라미터들을 밀리세컨드 단위로 명시하고 있다. JMF 의 RTP 구현에서는 Session Manager 가 디폴트 값을 설정해두고 있다. 오디오 버퍼의 경우 기본 250ms, 최대 1000ms 를 가지고 있으며 오디오 minimum threshold 의 경우 기본 125ms, 최대 500ms 를 가지고 있다. 비디오 버퍼의 길이는 기본적으로 135ms 를 가지는데 이는 15 프레임 스트림에서 약 2 프레임 정도를 타나내며, 비디오 minimum threshold 의 경우 기본 0ms, 최대 0ms 로 정의를 해두고 있다. 오디오의 경우 이러한 값은 입력되는 오디오 스트림의 포맷에 프레임 또는 오디오 샘플의 특정 값으로 변환이 되어지며, 비디오의 경우 입력되는 비디오 프레임 레이트는 수신측에서 알수가 없으므로, 이러한 경우 수신단은 버퍼와 threshold 값에 대응되는 완전한 비디오 프레임으로 15 프레임을 기본값으로 설정하게된다. 예를 들어서 만약 비디오 프레임 3 개정도를 버퍼가 갖고 있게한다면, 15 프레임 스트림에서 실제 스트림의 레이트에 상관없이 버퍼의 길이를 200ms 로 지정해주어야 한다. 이러한 부분은 JMStudio 에서 Open RTP Session 메뉴를 선택할 때 버퍼 컨트롤 옵션으로 제공되어 진다. 다른 방법으로는 독자들이 개발하는 프로그램내에서도 BufferControl 컴포넌트를 통하여 위와 같은 기능을 구현할수 있다는 점이다. getControls ("javax.media.control.BufferContr") 를

호출함으로서 세션 메니져를 통하여 버퍼 컨트롤을 얻을 수 있으며
BufferControl.getControlComponent() 를 이용하여 관련 컴포넌트를 획득할수 있다.

맺으면서

이번 강좌에서는 JMF 의 핵심기능으로서 실시간 통신부분을 담당하는 RTP 프로토콜에 관하여 소개하였으며, JMF 의 RTP 관련 패키지들과 이를 구현한 이벤트 및 해당 이벤트 처리에 관하여 논의하여 보았다. 또한 RTP 를 이용한 데이터의 송신과정과 수신 과정을 도식적으로 살펴보았으며, 세션 메니져와 RTP 관련 API 의 적용 범위에 관해서도 언급을 하였다. 다음 달에는 이렇게 살펴본 RTP 관련 이벤트와 API 를 이용하여 실제적인 영상 데이터와 음성 데이터의 송수신 방법들에 관하여 논의해보고 네트워크상의 RTP 스트림들을 모니터링 하는 방법에 관하여 알아본다. 본 연재에 관하여 문의사항은 kingseft@samsung.co.kr 로 하여주기 바란다.