내장형 시스템의 원활한 멀티미디어 서비스 지원을 위한 커널 수준의 RTP

(A Kernel-level RTP for Efficient Support of Multimedia Service on Embedded Systems)

선 동 국 † 김 태 웅 †† 김 성 조 †††
(Dong Guk Sun) (Tae Woong Kim) (Sung Jo Kim)

요 약 RTP는 VoD, AoD, VoIP와 같은 멀티미디어 서비스를 위해 실시간 데이타를 전송하기에 적합하도록 설계되어 있어, RTSP, H.323, SIP 등 실시간 데이타 전송 프로토콜로서 사용되고 있다. 내장형 시스템에서도 원활한 멀티미디어 서비스를 위해 내장형 시스템용 RTP 프로토콜 스택이 필요하지만 현재까지 이러한 RTP 프로토콜 스택은 전무한 상태이다. 본 논문은 내장형 시스템에 적합하도록 RTP 프로토콜 스택을 커널 수준에서 지원하는 embeddedRTP에 대하여 설명한다. EmbeddedRTP는 UDP 모듈에 포함되도록 설계되었기 때문에, TCP/IP 서비스를 필요로 하는 애플리케이션은 기존과 같이 동작하고, RTP 프로토콜 스택을 사용하는 애플리케이션은 embeddedRTP API를 이용하여 RTP 서비스를 받을 수 있다. EmbeddedRTP는 RTP 패킷이 도착했을 때, 패킷의 포트 번호와 세션 정보를 이용하여 세션별 패킷 버퍼에 저장한다. 애플리케이션과 embeddedRTP간의 통신은 시스템 콜과 시그널 메커니즘을 통해 이루어지며, embeddedRTP API를 제공함으로써 애플리케이션 개발이 용이하도록 했다. PDA 상에서 멀티미디어 스트리밍 서비스를 이용하여 embeddedRTP의 성능을 분석한 결과, 비교대상인 UCL RTP에 비해 코드 사이즈는 약 58%정도 감소된 반면, 패킷 처리속도는 약 7.8배 빨라졌다.

키워드 : 내장형 시스템, 실시간 서비스, 멀티미디어 서비스, RTP

Abstract Since the RTP is suitable for real-time data transmission in multimedia services like VoD, AoD, and VoIP, it has been adopted as a real-time transport protocol by RTSP, H.323, and SIP. Even though the RTP protocol stack for embedded systems has been in great need for efficient support of multimedia services, such a stack has not been developed yet. In this paper, we explain *embeddedRTP* which supports the RTP protocol stack at the kernel level so that it is suitable for embedded systems. Since *embeddedRTP* is designed to reside in the UDP module, existing applications which rely on TCP/IP services can proceed the same as before, while applications which rely on the RTP protocol stack can request RTP services through *embeddedRTP* API. *EmbeddedRTP* stores transmitted RTP packets into per session packet buffer, using the packet's port number and multimedia session information. Communications between applications and *embeddedRTP* is performed through system calls and signal mechanisms. Additionally, *embeddedRTP* API makes it possible to develop applications more conveniently. Our performance test shows that packet-processing speed of *embeddedRTP* is about 7.8 times faster than that of UCL RTP for multimedia streaming services on PDA in spite that its object code size is reduced about by 58% with respect to UCL RTP's.

Key words : Embedded System, Real-time Service, Multimedia service, RTP

· 본 연구는 정보통신부 ITRC 프로그램의 지원을 받아 중앙대학교 HNRC(Hone Network Research Center)에서 수행되었음
† 학생회원 : 중앙대학교 컴퓨터공학과
dgsun@konan.cse.cau.ac.kr
†† 비 회 원 : 텔리언 연구원
twkim@tellion.com
††† 종신회원 : 중앙대학교 컴퓨터공학과 교수
sjkim@cau.ac.kr
논문접수 : 2004년 3월 4일
심사완료 : 2004년 8월 12일

1. 서 론

현재의 초고속 인터넷에서는 텍스트를 비롯하여 고화질의 그림, 음성, 동영상 등의 멀티미디어 데이타에 대한 서비스 요구가 날로 커지고 있다. 멀티미디어 서비스를 제공 받을 수 있는 환경도 기존의 데스크톱 PC에서 비-PC까지 확대되고 있다.

비-PC 기기 환경, 즉 내장형 시스템에서의 멀티미디

어 서비스는 크게 VoD(Video On Demand)와 AoD (Audio On Demand) 같은 주문형 멀티미디어 서비스와 화상회의 시스템, VoIP(Voice over IP)와 같은 인터넷 전화 서비스로 구분될 수 있다. 이들 서비스를 지원하는 대표적인 프로토콜로는 H.323[1], SIP[2], RTSP [3] 등이 있는데, 이들 프로토콜들은 RTP(Realtime Transport Protocol)[4]를 멀티미디어 데이타 전송 프로토콜로 채택하고 있다. 따라서 인터넷 정보 가전과 같은 내장형 시스템에서 멀티미디어 서비스를 이용하기 위해서는 RTP를 지원해야 한다. 내장형 시스템은 일반 데스크톱 PC 및 멀티미디어 서버 등에 비해 상대적으로 CPU의 처리속도가 느리고 부동 소숫점 처리장치(FPU:Floating Point Unit)가 없어 컴퓨팅 파워가 낮다. 또한, 실제 메모리의 크기가 수백 KByte에서 수십 MByte 정도로 작고, 별도의 저장장치가 없어 사용가능한 메모리에 제약이 많다. 내장형 시스템의 이런 특성 때문에 내장형 시스템에 적합한 RTP를 지원하면 자원 활용 효율성과 처리 속도가 높아져 멀티미디어 서비스 처리 성능을 전반적으로 향상시킬 수 있다. 그러나 최근 연구[5-14]는 주로 RTP를 이용하는 애플리케이션의 성능 개선과 애플리케이션을 위한 라이브러리 개발에 머무르고 있어 내장형 시스템에 적합한 RTP 프로토콜 스택 및 구현 기법에 대한 연구는 미흡한 실정이며, 특히 본 논문에서 제안하는 커널 수준에서의 구현 사례는 전무한 실정이다.

본 논문은 내장형 시스템에서 원활한 멀티미디어 서비스를 지원하기 위한 커널 수준의 RTP 프로토콜 스택인 embeddedRTP의 설계 및 구현에 대하여 설명한다. RTP 프로토콜 스택을 내장형 시스템에 적합하도록 설계·구현하는 이유는 작은 코드 사이즈와 빠른 패킷 처리속도 및 높은 자원 활용성을 보장할 수 있기 때문이다. 또한, 프로토콜 스택을 커널 수준에서 제공함으로써 애플리케이션이 RTP 프로토콜 스택을 중복 구현하여 발생할 수 있는 자원 낭비 가능성을 제거할 수 있다.

본 논문의 구성은 다음과 같다. 제2절에서는 기존 RTP의 문제점 및 관련 국내외 연구동향에 대하여 기술한다. 제3절에서는 embeddedRTP의 설계 및 구현에 대해 기술하고, 제4절에서는 구현된 프로토콜의 실행 결과 및 성능을 분석한다. 마지막으로 제5절에서는 본 논문의 결론과 향후 연구과제에 대해서 기술한다.

2. RTP의 문제점과 연구 및 개발 동향

2.1 기존 RTP의 문제점

RTP는 IETF AVT WG(Internet Engineering Task Force Audio/Video Transport Working Group)에서 실시간 데이타 전송을 위해 설계한 인터넷 표준 프로토콜로서 RFC 1889[4]에 명시되어 있다. RTP는 전송할 데이타의 페이로드(payload) 타입의 지정을 통해 코덱(혹은 데이타의 포맷)에 적합한 전송을 가능하게 함으로써 애플리케이션 계층에서 QoS(Quality of Service)를 지원할 수 있다. RTCP(RTP Control Protocol)는 전송만을 담당하는 RTP를 보완하는 프로토콜로서 전송 품질에 대한 정보를 서버에 전달하고 멀티미디어 세션에 관련된 처리를 하는 등 RTP를 이용한 전송에서 요구되는 최소한의 제어정보를 제공한다. RTP와 RTCP는 하위의 전송 및 네트워크 계층에 무관하게 설계되었으나 일반적으로 IP 망 기반에서는 IP 패킷의 재조립(reassembly)과 검사합(checksum)의 기능을 활용하기 위해 UDP를 하위 전송 프로토콜로 사용한다. RTP와 RTCP를 묶어 RTP로 통칭하거나 RTP/RTCP로 표기하기도 한다.

RTP는 인터넷을 이용하기 위한 필수 프로토콜도 아니고, 현재 내장형 시스템에 적합한 RTP 프로토콜 스택 또한 개발되지 않은 상황이기 때문에, 내장형 시스템용 네트워크 모듈에는 대부분 RTP가 포함되어 있지 않다. 따라서, RTP를 이용하는 애플리케이션이 내장형 시스템에서 사용되어야 할 경우, 프로토콜 스택의 구성은 그림 1과 같이 RTP가 애플리케이션에 직접 포함될 수 밖에 없다. 이 경우 내장형 시스템이 H.323, SIP, RTSP 등 여러 멀티미디어 서비스를 지원하여야한다면 RTP 프로토콜 스택은 시스템에 중복으로 포함될 수밖에 없어 메모리 자원이 충분하지 않은 내장형 시스템에 이러한 방법은 적합하지 않다.

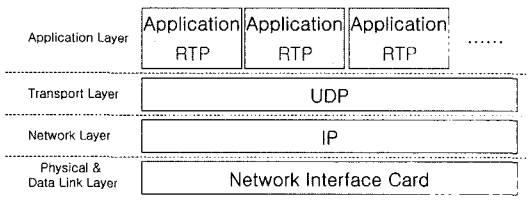

그림 1 RTP가 애플리케이션에 포함된 경우의 프로토콜 스택 구성

또 다른 방법으로는 RTP가 애플리케이션 계층의 라이브러리로써 제공되는 것이다. 이 경우 시스템에서 사용하는 프로토콜 스택의 구성은 그림 2와 같다. 이와 같이 애플리케이션이 시스템에서 제공하는 라이브러리를 이용함으로써 RTP의 중복 문제가 해결된다. 그러나 UDP에서 RTP로 데이타를 전송할 때, 혹은 RTP 모듈에서 애플리케이션으로 데이타를 전송할 때, 메모리 복사, 문맥 교환(context switching) 등의 오버헤드가 발생할 수 있다. 또한, 다양한 애플리케이션이 하나의 라

이브러리를 공유하여야하므로 이 방법은 라이브러리 호환성 문제를 유발할 수 있다.

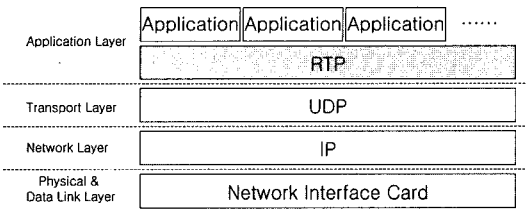

그림 2 RTP가 시스템 라이브러리로 제공될 때의 프로토콜 스택

RTP가 그림 3과 같이 커널 수준에서 구현된다면 프로토콜 스택의 중복 구현이나 문맥 교환 오버헤드가 제거될 수 있다. 또한, 개발자들은 프로그램 내에 RTP 프로토콜 스택을 포함시킬 필요가 없어 애플리케이션의 코드 사이즈를 줄일 수 있다. RTP 패킷인지 여부는 UDP 패킷에 포함되어 있는 포트 번호 정보를 이용하여 검사하여야 하기 때문에, 그림 3에서 모든 UDP 패킷에 대한 RTP 패킷 여부 검사는 오버헤드로 작용한다. 하지만, UDP를 사용하는 애플리케이션 계층 중에서 멀티미디어 통신에 이용되지 않는 TFTP, DHCP, ECHO 등은 비교적 네트워크 트래픽이 많지 않고, 높은 성능을 요구하지 않는 것을 고려한다면 이들 프로토콜의 성능에는 그다지 지장을 주지 않을 것으로 예상된다.

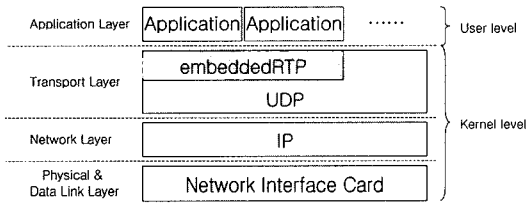

그림 3 EmbeddedRTP를 이용할 때의 프로토콜 스택 구성

2.2 연구 및 개발 동향

국내 연구 기관에서는 RTP를 더욱 효율적으로 활용하기 위해 특정 클라이언트 또는 서버에 적합한 API를 개발하거나 RTP가 QoS를 지원할 수 있도록 하기 위한 노력이 활발하다. 첫 번째 예[5]로는 RTP를 이용하는 전송의 특성을 고려한 파일 I/O를 제공함으로써 멀티미디어 서버의 성능을 향상시키고자 하는 SRTPIO (Special RTP I/O) 모듈이 구현되었다. 또 다른 예로서 RTP 스택 자체 개발 보다는 RTCP로부터 QoS 파라미터를 추출하여 애플리케이션에 대한 QoS를 동적으로 제공하는 방안이 연구[6]되었다. 국내 기업의 경우에는 자사의 제품에 사용할 RTP 라이브러리를 개발하는 사례가 대부분이다. 온타임텍(onTimetek)사에서는 핸드폰 등의 무선 모바일 기기에서의 동영상 서비스를 위해 RTSP, RTP를 비롯한 멀티미디어 지원 프로토콜을 개발하였다[7]. 팜캐스트(palmcast)사에서는 RTSP 기반의 멀티미디어 스트리밍 서버인 PalmCast RTSP Media Server를 개발하였다[8].

국외 동향을 살펴보면 기업, 대학 연구기관, 오픈 프로젝트팀 등에서 자신의 연구 목적을 위해 부가적으로 별도의 RTP 라이브러리나 RTP를 포함하고 있는 멀티미디어 라이브러리를 개발한 사례가 많다. 그중 대표적인 예로는 벨연구소의 RTPlib[9], UCL(University College London)의 common multimedia library[10] 등이 있다. 라이브러리 개발외에도 [11]의 경우에는 화상회의 시스템인 VIC(VIdeo Conferencing Tool)를 개발하면서 본 연구 결과와 유사하게 UDP와 RTP을 합쳐놓은 듯한 형태를 지니는 RTIP(Real-Time Internet Protocol)를 구현하였다. 그러나 이 프로토콜은 독립적으로 동작할 수 없고 U.C.Berkeley의 Tenet 프로토콜 슈트와 함께 사용되어야 하며, 인터넷 표준을 따르지 않기 때문에 호환성이 떨어지는 문제점을 가지고 있다.

국내외적으로 그림 2와 같이 하나의 라이브러리 형태로 RTP를 지원하면서 애플리케이션과 하드웨어 호환성 문제를 해결하기 위해서 자바를 이용하여 RTP를 구현한 몇몇 연구 및 개발 사례[12-14]도 있다. 특히 [14]의 경우에는 J2ME(Java 2 Platform, Micro Edition)[15] 등 임베디드 자바 기술이 발전하면서 실행 속도 및 메모리 관리 등 자바가 가지고 있는 기본적인 문제점에도 불구하고 핸드폰 등으로 그 사용범위 확대되고 있다.

3. EmbeddedRTP 설계 및 구현

본 절에서는 embeddedRTP의 설계 및 구현에 관련된 중요한 이슈들을 기술한다. 본 논문에서는 설계된 embeddedRTP를 우선 리눅스 상에서 구현하여 동작 여부를 검증한 후, 이를 내장형 시스템으로 이식하는 방식으로 개발하였다.

3.1 전체 구조 설계

EmbeddedRTP는 패킷의 처리속도 증대와 메모리의 효율적인 사용, 그리고 RTP 프로토콜 스택의 중복 구현 방지를 위해 2절의 그림 3에서와 같이 RTP 프로토콜을 UDP 안에 내장시킨다. 애플리케이션이 TCP나 UDP를 사용할 때는 기존과 마찬가지로 BSD 소켓 계층(Socket Layer)을 이용한다. 그러나 애플리케이션이 RTP를 사용할 때는 BSD 소켓 계층 대신 바로 embeddedRTP를 이용하고, embeddedRTP는 내부적으로 UDP를 이용한다.

EmbeddedRTP는 그림 4와 같이 embeddedRTP API, 멀티미디어 세션 확인 모듈, RTP 패킷 수신 모듈, RTP 패킷 처리 모듈, RTCP 패킷 수신 모듈, RTCP 패킷 송신 모듈, RTCP 패킷 처리 모듈로 구성되는데, 각 모듈들의 기능을 간략히 설명하면 다음과 같다.

- embeddedRTP API : embeddedRTP 관련 함수 호출을 위한 시스템 콜의 캡슐화.
- 멀티미디어 세션 확인 모듈: RTP/RTCP 패킷 여부를 확인한 다음, RTP 또는 RTCP 패킷 수신 모듈을 호출.
- RTP 패킷 수신 모듈 : RTP 패킷의 유효성 검사를 한 후, 패킷을 RTP 패킷 처리 모듈로 전달.
- RTP 패킷 처리 모듈 : RTP 패킷을 패킷 버퍼내의 적절한 위치에 삽입하고, 세션별 통계정보를 갱신한 다음, RTP 패킷 수신 이벤트를 애플리케이션 전달.
- RTCP 패킷 수신 모듈: RTCP 패킷의 유효성을 검사한 후, 패킷을 RTCP 패킷 처리 모듈로 전달.
- RTCP 패킷 처리 모듈: RTCP 메시지를 추출하고 처리한 후, RTCP 메시지 수신 이벤트를 애플리케이션 전달.
- RTCP 패킷 송신 모듈 : RTCP 패킷 송신 주기가 되면 세션별 통계정보를 바탕으로 RTCP 패킷을 생성하여 전송.

본 논문에서 구현한 embeddedRTP는 내장형 시스템을 대상으로 하고 있으므로 멀티미디어 서버가 갖추어야 할 변환기, 혼합기 등의 기능을 제공하지 않는다.

3.1.1 통신 메커니즘

EmbeddedRTP에서는 RTP 프로토콜이 커널 수준에서 구현되기 때문에, 애플리케이션 계층의 RTP와 커널의 UDP 모듈이 통신을 하던 구조를 커널 수준에서 embeddedRTP와 UDP 모듈이 통신하는 구조로 변경해야 한다. 본 절에서는 이러한 통신 메커니즘에 대하여 기술한다.

가. 애플리케이션에서 embeddedRTP 호출

애플리케이션에서 embeddedRTP의 호출은 시스템 콜(system call)을 사용하였는데, 이는 리눅스 시스템에서 애플리케이션이 커널 모듈을 사용할 수 있는 유일한 방법이다. 현재는 RTP 프로토콜 스택이 리눅스 커널에 포함되어 있지 않기 때문에 embeddedRTP 관련 시스템 콜을 커널에 신규로 등록해야 애플리케이션은 커널 모듈 호출 및 커널과의 데이타 교환이 가능하다. EmbeddedRTP 관련 시스템 콜을 등록하기 위해 수정이 필요한 리눅스 커널의 소스 파일은 다음과 같다.

- LINUXSRCROOT/arch/i386/kernel/entry.S : 임베디드 리눅스에서는 새로운 시스템 콜 심볼 등록을 위해 "LINUXSRCROOT/arch/arm/kernel/calls.S"이 수정되어야 한다.
- LINUXSRCROOT/include/linux/sys.h : 기존 시스템 콜 수와 새로 등록한 embeddedRTP 관련 시스템 콜의 수를 고려하여 최대 시스템 콜의 수를 조정한다.

나. embeddedRTP에서 애플리케이션 호출

EmbeddedRTP에서 애플리케이션의 호출은 시그널 메커니즘을 이용한다. 시그널 메커니즘은 리눅스 시스템에서 범용으로 사용하는 메시지 전송 메커니즘으로서 미리 정의되어 있는 시그널만을 전송할 수 있고, 데이타는 전송할 수 없다. 또한 전송할 수 있는 시그널 종류가

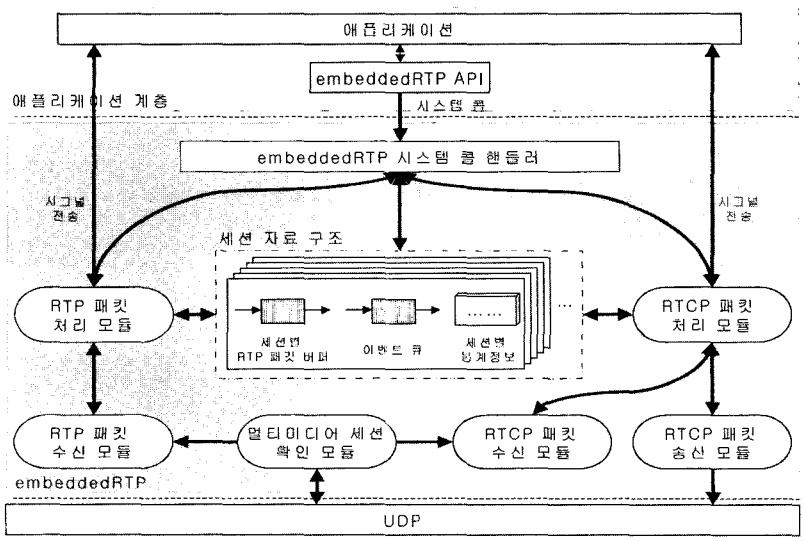

그림 4 EmbeddedRTP의 전체 모듈 구성도

많지 않기 때문에 사용 범위가 제한적이고, 전송받은 시그널의 저장 메커니즘이 없기 때문에 전송받은 시그널을 처리하기 전에 다른 시그널을 전송받으면 이전에 받은 시그널은 무시된다는 단점이 있다. 그러나 이벤트의 발생 여부만을 전달하기 위해 시그널은 부족함이 없다. 시그널이 저장되지 않는 문제는 이벤트 큐를 생성하여, 각종 이벤트 발생시 이를 큐에 저장하고 해당 프로세스에 시그널을 전송하여 해결하였다. EmbeddedRTP에서 사용하는 이벤트 큐는 가장 단순한 큐 형태인 FIFO (First-In First-Out)를 유지하며, 각 큐 엔트리에는 멀티미디어 세션 ID와 이벤트 종류가 저장된다.

3.1.2 멀티미디어 세션 관리

EmbeddedRTP는 RTP를 사용하는 모든 애플리케이션에게 서비스를 제공해야 하기 때문에 멀티미디어 세션 관리는 필수적이다. RTP 프로토콜 스택이 애플리케이션에 포함되어 있는 경우에는 해당 애플리케이션이 세션을 관리하면 되었으나, embeddedRTP에서는 커널이 모든 멀티미디어 세션을 관리해야 한다. 멀티미디어 세션은 그림 5와 같은 1차원 배열과 원형 큐를 접목한 자료구조로 관리된다.

세션 정보는 멀티미디어 세션 식별자인 1차원 배열의 인덱스를 이용하여 빠르게 액세스할 수 있다. 또한 front, rear 인덱스를 이용하여 원형 큐의 형태를 유지함으로써 오버플로우에 대비할 수 있다. 그러나 동시에 오픈 가능한 멀티미디어 세션 수와 메모리 낭비라는 상충관계를 고려해야 하는 어려움이 있다.

하나의 멀티미디어 세션을 저장하기 위해 표 1의 rtp 구조체를 사용하였는데, 이 구조체는 RTP 패킷 송수신

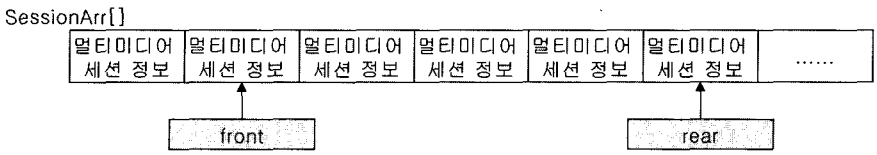

그림 5 멀티미디어 세션 관리를 위한 자료구조

표 1 멀티미디어 세션 데이타 저장 구조체

```
struct socket_udp_   {
   uint16_t          rx_port;        // 송신 포트
   uint16_t          tx_port;        // 수신 포트
   ttl_t             ttl;            // 패킷의 TTL
   fd_t                    fd;               // 소켓 파일 디스크립터
   struct socket     *sock           // 소켓이용 관련 구조체
   struct in_addr     addr4;         // 소켓이용 관련 구조체
};
struct rtp {
   uint8_t       inited;             // RTP 초기화 여부
   /* 송수신을 위한 데이터 */
   socket_udp        *rtp_socket;            // RTP socket
   socket_udp        *rtcp_socket;           // RTCP socket
   char              *addr;                  // 서버 주소

   /* 이벤트 큐 관련 자료구조 */
   int                pid;                   // 세션 생성 프로세스ID
   RTP_EVENT    eventq[RTP_EVENT_Q_SIZE];    //이벤트 큐
   int                eventq_head;           // 이벤트 큐 head 인덱스
   int                eventq_tail;           // 이벤트 큐 tail 인덱스
   int                eventq_count;          // 저장된 이벤트 수

   /* RTP 패킷 버퍼를 위한 자료구조*/
   uint8_t   *rtp_chunk;                     // 패킷 버퍼 청크
   int                rtp_chunk_index;            // 탐색 지점 인덱스
   int                rtp_bitmap[BITMAP_SIZE];    // 비트맵

   /* 통계정보 계산을 위한 데이터 */
   uint32_t    my_ssrc;              // SSRC
   int                invalid_rtp_count;     // 유효하지 않은 RTP 패킷 수
   int                invalid_rtcp_count;    // 유효하지 않은RTCP 패킷수
   int                sending_bye;           // BYE 패킷 전송 여부
                                  ......
};
```

을 위한 데이타, 이벤트 큐 관련 자료구조, RTP 패킷 버퍼 관련 자료구조, 그리고 통계정보 산출을 위한 데이타로 구성된다. 송수신을 위한 데이타는 송수신에 필요한 서버 주소, 포트 번호, 소켓 디스크립터로 구성되고, 이벤트 큐 관련 자료구조는 이벤트 큐와 원형 큐 유지를 위한 포인터, 그리고 세션을 생성한 프로세스의 ID로 구성된다. 패킷 버퍼 관련 자료구조는 패킷을 저장할 패킷 버퍼 청크(chunk)와 이에 대응하는 비트맵으로 구성되며, 통계정보 산출을 위한 데이타는 RTCP의 수신자 보고 메시지를 작성하기 위한 데이타들로 구성된다.

3.2 상세 모듈 설계 및 구현

3.2.1 embeddedRTP API

본 논문에서는 RTP를 이용하는 애플리케이션 개발의 편의성을 제공하기 위해 embeddedRTP API 라이브러리를 정의하였는데, 애플리케이션은 이 함수들을 통해 안전하게 embeddedRTP 관련 시스템 콜을 호출할 수 있다. 본 논문에서 정의한 embeddedRTP API 함수는 표 2와 같다.

표 2 embeddedRTP API 리스트

API	기 능
ertp_init	RTP 세션 데이타 생성
ertp_set_my_ssrc	멀티미디어 세션의 SSRC(Synchronization Source) 설정
ertp_set_option	송수신 소켓의 옵션 설정
ertp_send_bye	BYE 메시지를 전송하여 세션을 종료
ertp_done	세션 자료구조를 해제
ertp_get_packet	RTP 패킷 버퍼로부터 전송받은 패킷을 복사
ertp_get_info_of_q	RTP 패킷 버퍼에 대한 정보 획득
ertp_event_get	embeddedRTP에 관련 이벤트 정보 획득

EmbeddedRTP의 API들은 멀티미디어 세션의 생성 및 종료, RTP 패킷 버퍼로 부터의 패킷 복사, 사용한 RTP 패킷의 RTP 패킷 버퍼로부터의 제거 기능을 제공한다. 패킷의 수신 및 송신, 패킷 버퍼의 유지·관리 기능은 embeddedRTP가 관리해주기 때문에 애플리케이션에서 관여할 필요가 없다.

3.2.2 시스템 콜 등록

표 2의 API를 지원하려면 8개의 시스템 콜이 필요로 하지만, 본 논문에서는 entry.S 또는 calls.S파일에 시스템 콜들 중에서 파라미터 타입과 리턴 타입이 비슷한 것을 묶어 6개의 새로운 시스템 콜 심볼들을 추가하였다. 그리고 새롭게 추가되는 심볼과 추후 확장을 고려하여 최대 시스템 콜 수를 i386시스템의 경우에는 270, ARM 시스템의 경우에는 256으로 설정하였는데, 이는 구현에 사용된 리눅스(커널 버전 2.4.20)가 i386시스템의 경우에는 258개, ARM 시스템의 경우에는 238개의 시스템 콜 심볼이 이미 등록되어 있기 때문이다.

하나의 시스템 콜 심볼이 여러 개의 시스템 콜을 처리하여야 할 경우에는 BSD기반 소켓 API에서 사용하는 기법과 동일하게 시스템 콜 핸들러는 콜 번호를 통해 해당 함수로 분기하도록 하였다. 표 3은 추가된 시스템 콜 심볼들 중의 하나인 "embeddedRTP_t1"의 시스템 콜 핸들러를 보여주고 있는데, 파라미터로 넘어오는 call의 값에 따라 rtp_send_bye 또는 rtp_done 함수를 호출하도록 되어 있다.

3.2.3 멀티미디어 세션 확인 모듈

UDP 모듈은 검사합 등 UDP 패킷에 관련된 모든 처리를 한 후, 마지막으로 소켓 버퍼에 UDP 패킷을 저장한다. 이때 멀티미디어 세션 확인 모듈은 UDP 패킷의 송수신 포트번호와 일치하는 멀티미디어 세션이 있는지 확인한다. UDP 패킷이 멀티미디어 세션에 포함되는 RTP/RTCP 패킷인 경우 패킷의 종류에 따라 RTP/RTCP 패킷 수신 모듈을 호출하고 UDP 모듈을 종료하게 된다.

표 3 embeddedRTP_t1의 시스템 콜 핸들러

```
asmlinkage uint8_t * sys_embeddedRTP_t1(int call, int index)
{
        struct rtp * krtp;

        krtp = get_rtp_session(index);
        switch(call)
        {
        case 2:                         //rtp_send_bye()
                rtp_send_bye(krtp);
                return NULL;
        case 3:                         //rtp_done()
                rtp_done(krtp);
                return NULL;
        }
        return NULL;
}
```

3.2.4 RTP 패킷 수신 모듈

RTP 패킷 수신 모듈은 UDP 소켓 버퍼로부터 RTP 패킷을 읽어와 RTP 패킷 헤더 구조체를 설정한 이후 RTP 패킷의 유효성을 검사하고, 이때 유효한 RTP 패킷이 전송되었다면 RTP 패킷 처리 모듈이 호출된다. RTP 패킷의 유효성 검사는 먼저 버전 필드을 검사하고, 패킷 헤더의 각종 정보를 이용하여 패킷의 길이 등을 확인한다.

수신된 RTP 패킷은 표 4와 같은 rtp_packet 구조체에 의하여 관리되는데, rtp_packet 구조체는 패킷 버퍼를 구성하는 rtp_packet_data 구조체와 RTP 패킷 헤더 정보를 저장할 수 있는 rtp_packet_header 구조체로 나뉘어진다.

3.2.5 RTP 패킷 처리 모듈

RTP 패킷 처리 모듈은 rtp_packet_data 구조체를 처리하여, RTP 패킷을 세션별 RTP 패킷 버퍼에 삽입하고, 통계정보를 갱신한 후 애플리케이션에게 RTP 패킷이 수신되었음을 알린다.

가. 패킷 저장

수신된 RTP 패킷은 그림 6과 같이 패킷 버퍼 청크(packet buffer chunk)와 이에 대응하는 비트맵(bit-map)으로 구성된 자료구조를 이용하여 저장된다. 이 자

표 4 RTP 패킷 저장 구조체

```
typedef struct rtp_packet_data {
  struct rtp_packet     *rtp_pd_next; // 다음 패킷 링크
  struct rtp_packet     *rtp_pd_prev; // 이전 패킷 링크
  uint32_t              *rtp_pd_csrc; // CSRC(Contributing Source) 포인터
  uint8_t               *rtp_pd_data; // 데이터 포인터
  int                   rtp_pd_data_len;// 실제 데이터 길이
  uint8_t               *rtp_pd_extn; // 확장 헤더 포인터
  uint16_t              rtp_pd_extn_len;// 확장 헤더 길이
  uint16_t              rtp_pd_extn_type;// 확장 헤더의 종류
  int                   rtp_pd_buflen;    // 전체 바이트 수
  int                   rtp_pd_have_timestamp;//타임스탬프 유무
  uint64_t              rtp_pd_timestamp;     // 타임스탬프 값
} rtp_packet_data;

typedef struct rtp_packet_header {
#ifdef WORDS_BIGENDIAN
        unsigned short   ph_v:2;              // RTP 패킷 버전
        unsigned short   ph_p:1;              // 패딩 플래그
        unsigned short   ph_x:1;              // 헤더 확장 플래그
        unsigned short   ph_cc:4;             // CSRC 카운터
        unsigned short   ph_m:1;              // 마커 비트
        unsigned short   ph_pt:7;             // 페이로드 종류
#else
        ............
#endif
        uint16_t        ph_seq;               // 시퀀스 번호
        uint32_t        ph_ts;                // 타임스탬프
        uint32_t        ph_ssrc;              // SSRC
} rtp_packet_header;

typedef struct rtp_packet {
  rtp_packet_data pd;
  rtp_packet_header ph;
} rtp_packet;
```

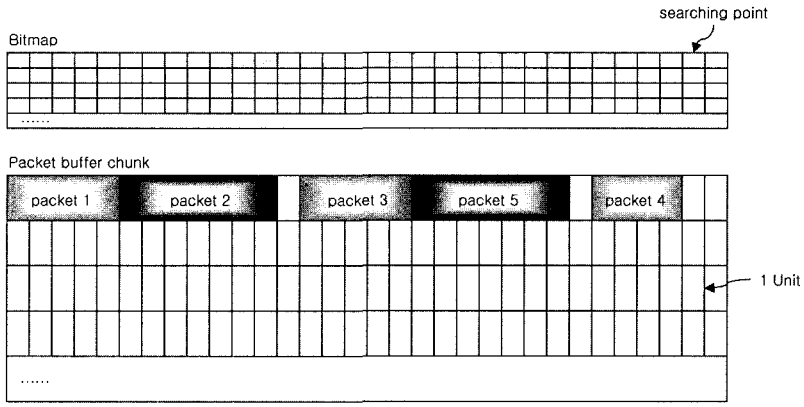

그림 6 RTP 패킷 저장 자료구조

료구조에는 표 4의 rtp_packet 구조체와 페이로드가 같이 저장된다. 그림 6에서 패킷 버퍼 청크는 일정한 데이타 크기를 갖는 유닛(unit)들로 구성되고, 비트맵은 패킷 버퍼 청크 각 유닛의 사용 여부를 나타내는 비트스트림(bit-stream)이다. RTP 패킷 처리 모듈이 RTP 패킷을 전송받으면, 비트맵의 검색 지점(searching point)으로부터 패킷의 저장 가능 여부를 검사한다. 검색 지점으로부터 패킷이 저장될 수 있다면, 검색 지점에 해당하는 패킷 버퍼 청크로부터 패킷을 저장한다. 패킷 저장이 불가능한 경우에는 검색 지점을 데이타가 저장 가능한 다음 유닛으로 변경하여 재검색한다. 패킷을 저장한 후에는 사용된 패킷 버퍼 청크에 대응하는 비트맵 값을 사용 중으로 설정한다. 저장된 패킷이 사용됐을 때는 대응하는 비트맵 값이 리셋된다.

이 자료구조는 버디 시스템(buddy system)[16]의 메모리 청크와 비트맵 구조를 응용한 방법이다. 버디 시스템은 프리 리스트 헤더를 이용하기 때문에 할당할 메모리의 위치를 찾는 과정이 매우 간단하지만, RTP 패킷 저장에 바로 사용되기에는 부적합하다. 이는 RTP 패킷의 페이로드 크기가 1과 1500사이라는 점을 감안했을 때 메모리 효율성이 떨어지기 때문이다. 예를 들어, 크기가 1030바이트인 RTP 패킷을 저장하기 위해서 버디 시스템은 2048바이트를 할당해야 하기 때문에 1018 바이트의 메모리가 낭비될 수 밖에 없다. 본 논문에서 제안한 자료구조는 버디 시스템의 이러한 메모리 효율성 문제를 해결하기 위해 유닛의 크기를 조절할 수 있도록 하였다. 즉, 하나의 유닛을 100 바이트로 설정했을 때에는 11개의 유닛을 사용하고 70 바이트를 메모리를 낭비하게 되나, 200 바이트로 설정했을 때에는 170 바이트를 낭비하게 되어 유닛의 크기가 작을수록 메모리 효율성이 높아진다. 그 반면, 한 유닛의 크기가 작을수록 패킷을 저장할 메모리의 위치를 검색해야 하는 비트맵이 커지기 때문에 검색 시간이 증가하게 된다. 따라서 메모리 효율성과 검색 시간의 상충관계(trade-off)로 인하여 유닛의 크기는 타깃 시스템의 자원과 애플리케이션에 따라 커널 컴파일 과정에서 적절하게 조절되어야 한다.

나. RTP 패킷 정렬

서버가 패킷을 1번부터 5번까지 순차적으로 전송했을 때, RTP는 순차전송을 보장하지 않는 UDP를 이용하기 때문에 클라이언트가 보낸 순서대로 패킷을 받지 못할 수 있다(그림 6 참조). 따라서 RTP를 통해 전송받은 데이타들이 재생되는 순서를 보장받기 위해서는 서버가 전송한 패킷 순서를 유지할 필요가 있다. 이를 위해 표 4의 rtp_packet 구조체를 이용한 원형큐가 사용되며, 원형큐 head와 tail 포인터 각각은 패킷 순서의 처음과 끝을 가리킨다.

도착된 패킷은 무조건 tail 포인터가 가리키는 패킷의 다음 대신, RTP 패킷의 시퀀스 번호에 따라 버퍼의 적절한 곳에 저장된다. 만일 전송받은 패킷의 시퀀스 번호가 저장된 패킷의 시퀀스 번호와 같다면 중복 전송으로 판단하고 이를 폐기한다. 이처럼 도착 패킷의 순서를 유지함으로써 애플리케이션은 head 포인터가 가리키는 패킷부터 순서대로 가져다 쓸 수 있다. RTP 패킷을 저장할 적절한 위치를 찾는 알고리즘의 의사 코드(pseudo code)는 표 5와 같다.

3.2.6 RTCP 패킷 송신 및 수신 모듈

RTCP 패킷 수신 모듈은 UDP 소켓 버퍼로부터 RTCP 패킷을 읽어와 RTCP 패킷 헤더 구조체를 설정한다. 그 이후 RTCP 패킷의 버전 필드의 값을 검사하고 패킷 헤더의 각종 정보를 바탕으로 패킷의 길이를 체크하여 RTCP 패킷의 유효성을 검사한 후, 유효한 RTCP 패킷을 전송 받았다면 RTCP 패킷 처리 모듈을 호출한다. 이 과정은 RTP 패킷 수신 모듈과 동일하다.

RTCP 패킷 송신 모듈은 수신자 보고(Receiver Report: RR) 메시지, 소스 설명(Source Description:SDES) 메시지, 세션 종료(BYE) 메시지를 생성하여 RTCP 패킷을 서버로 전송하는 역할을 한다. 이 모듈은 미리 계산된 RTCP 전송 주기에 따라 RTCP 패킷을 보낼 시점이 되면 계산되어 있는 통계정보를 바탕으로 패킷을 생성하여 전송하고, 패킷 송신 후에는 통계정보를 초기화한 다음 모듈 수행을 종료한다.

3.2.7 RTCP 패킷 처리 모듈

RTCP 패킷 처리 모듈은 RTCP 패킷 헤더의 SSRC (Synchronization Source) 필드를 이용하여 멀티미디어 세션인지 여부를 확인한다. 멀티미디어 세션이 확인되면, 그림 7의 처리과정에 의해 처리된다.

패킷에 포함된 RTCP 메시지 타입에 따라 해당 처리 루틴을 호출한다. RTCP 메시지를 처리한 후에는 애플리케이션에 RTCP 메시지 수신 이벤트를 알리기 위해 이벤트를 이벤트 큐에 저장하고 애플리케이션으로 시그널을 전송한다. 하나의 RTCP 패킷에는 2개 이상의 RTCP 메시지 포함될 수 있기 때문에 하나의 RTCP 메시지 처리 후에도 RTCP 메시지가 패킷에 존재하는지 검사하여야 한다. 만일 잔여 메시지가 존재한다면, 위의 처리하는 과정을 반복한다.

4. 실행 결과 및 성능 분석

4.1 실험 환경 설정

본 논문에서는 embeddedRTP의 동작 여부와 성능 평가를 위한 도구로서 멀티미디어 스트리밍 서비스를 채택하였다. 스트리밍 서버와 미디어 플레이어는 각각 애플(Apple)사의 다윈 스트리밍 서버(Darwin Strea-

표 5 RTP 패킷 순차 정렬 위치 검색 의사 코드

```
if(Queue is empty ){
        *Head = *Tail = NewPacket;
}else if( There is one packet ){
        if( Head's SeqNum == NewPacket's SeqNum){
                Discard NewPacket; /* Duplicated Packet */
                break;
        }
        Insert NewPacket in Queue;
        if( Head's SeqNum < NewPacket's SeqNum)
                Set *Tail to NewPacket;
        else if( Head's SeqNum > NewPacket's SeqNum)
                Set *Head to NewPacket;
}else if( NewPacket should be inserted at the both end of Queue  )
 /* In case of "Head's SeqNum < Tail's SeqNUM",
    NewPacket's SeqNum > Tail's SeqNum OR NewPacket's SeqNum < Head's SeqNUM
    In case of "Head's SeqNum > Tail's SeqNUM",
    NewPacket's SeqNum > Tail's SeqNum AND NewPacket's SeqNum < Head's SeqNUM
*/
        Add NewPacket at the end of Queue;
        if( Head's SeqNum > NewPacket's SeqNum )
                Set *Head to NewPacket;
        else if( Tail's SeqNum < NewPacket's SeqNum )
                Set *Tail to NewPacket;
}else{    /* Insert new packet in the middle of Queue */
        Set *current to *Head;
        Do{
                if( Current's SeqNum == NewPacket's SeqNum){
                        Discard NewPacket;/* Duplication Detected */
                        break;
                }
                if( Proper location is found ){
                        Add NewPacket in the middle of Queue;
                        break;
                }
                Move *Current to Next Packet;
        }while( All Packets in Queue )
}
```

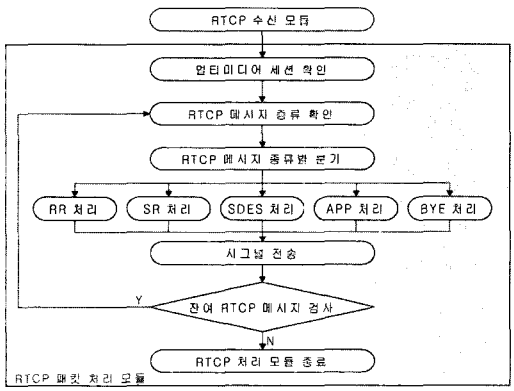

그림 7 RTCP 패킷 처리 모듈의 처리과정

ming Server)[17]와 MPEG4IP 프로젝트[18]의 MPEG4IP 플레이어를 이용하였다. 스트리밍 서버는 일반 데스크톱 PC에서 실행하였고, MPEG4IP 플레이어는 기존의 RTP 모듈을 embeddedRTP로 대체한 후, PDA에서 구동시켰다. 서버 PC와 클라이언트인 PDA는 무선랜을 통해 통신하도록 하였고, PDA의 운영체제로는 대표적인 임베디드 리눅스인 Familiar[19]를 이용하였다.

4.2 성능 분석

본 절에서는 embeddedRTP의 성능 분석을 위해 패킷 처리속도, 패킷 저장시 메모리 요구량, 코드 사이즈, 멀티미디어 세션 확인에 필요한 오버헤드를 측정하였다. RTP 패킷 처리속도와 패킷 버퍼링시 메모리 요구량, 코드 사이즈의 성능 비교 대상은 MPEG4IP 프로젝트에서 사용하고 있는 UCL(University College London) RTP 라이브러리[10]이다.

4.2.1 RTP 패킷 처리속도

프로토콜 스택의 가장 중요한 성능 요소는 패킷 처리속도이다. RTP 패킷은 초당 수십 개의 패킷이 송수신되지만 RTCP 패킷은 약 5초에 하나씩 송수신 되기 때문에 RTCP 패킷 처리시간은 배제하고 RTP 패킷 처리속도만을 측정하였다. 여기서 패킷 처리속도란 RTP 패킷이 UDP 모듈에서 처리가 끝난 후, 멀티미디어 세션임이 확인되는 시점부터 애플리케이션이 RTP 패킷의 모든 처리를 끝내는 시점까지를 말한다. RTP 패킷 처리속도 측정 결과, embeddedRTP와 UCL RTP 라이브러리는 각각 초당 평균 3460개와 444개의 패킷을 처리했다. 즉, embeddedRTP의 RTP 패킷 처리속도가 UCL

RTP 라이브러리보다 약 7.8배 빠르다. 이 결과는 embeddedRTP가 패킷을 저장할 때 정적으로 할당받은 메모리를 사용하는 반면, UCL RTP 라이브러리는 패킷이 도착했을 때마다 패킷 저장 자료구조를 동적으로 생성하기 때문이다. 또한, RTP 패킷 저장시 embeddedRTP는 커널 수준에서 수행되는 반면, UCL RTP 라이브러리는 애플리케이션 계층에서 수행되는 것도 이유가 될 수 있다.

4.2.2 패킷 저장시 메모리 요구량

본 절에서는 embeddedRTP와 UCL RTP 라이브러리에서 패킷을 저장할 때에 사용되는 메모리 요구량에 대해 기술한다. EmbeddedRTP의 패킷 저장과 순서 정렬 메커니즘은 3.6절에서 기술하였다. UCL RTP 라이브러리는 RTP 패킷을 전송받았을 때 관리 자료구조와 패킷 저장을 위한 메모리 영역을 동적으로 할당받아 패킷을 저장하고, 전송받은 데이타의 송신 순서 유지를 위해 이 패킷들을 이중 연결 리스트로 연결한다.

EmbeddedRTP와 UCL RTP 라이브러리의 버퍼 메커니즘에 의한 메모리 요구량을 비교하기 위해 동일 메모리량 사용시 페이로드 크기에 따른 버퍼링 가능 패킷수와 저장할 패킷수가 일정할 때 페이로드 크기에 따른 메모리 요구량을 측정하였다.

그림 8은 동일한 크기(32,000바이트)의 메모리를 사용할 경우, 페이로드 크기에 따른 저장 가능 패킷수를 측정한 그래프이다. 이때 패킷별 최대 메모리 할당량은 IP의 MTU(Maximum Transmission Unit)인 1500바이트와 세션 관리를 위해 필요한 메모리를 고려하여 1600바이트로 가정하였고, 실험에서 사용한 동영상들의 평균 페이로드 크기가 200바이트의 배수 전후로 나타났으므로, 유닛의 크기는 200 바이트로 가정하였다.

UCL RTP 라이브러리는 페이로드 크기에 관계없이 할당하는 메모리 크기가 같기 때문에 그림 8에서 보는 바와 같이 저장 가능한 패킷수가 RTP 패킷의 페이로드 크기에 관계없이 일정하다. EmbeddedRTP는 패킷을 저장할 위치를 검색할 때는 부가적인 메모리 복사를 하지 않기 위해 패킷 최대 크기만큼 저장 가능한 버퍼 위치를 찾지만 패킷을 저장한 후에는 페이로드 크기에 따라 사용되지 않은 메모리 영역을 재사용할 수 있도록 비트맵에 해당 영역을 미사용 구간으로 표시함으로써 메모리 효율성을 높였다. EmbeddedRTP는 패킷 버퍼 청크를 유닛으로 나누어 사용하기 때문에 계단형 그래프가 도출되었다.

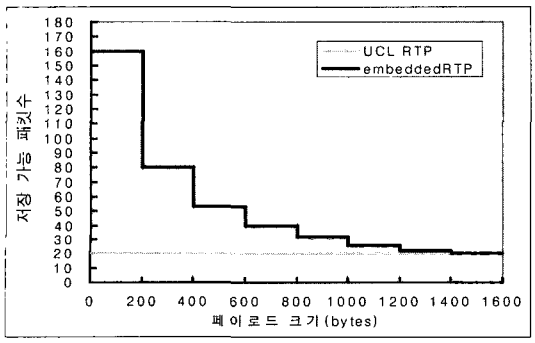

그림 8 페이로드 크기에 따른 저장 가능 패킷수

위의 결과에서 저장해야할 패킷의 개수가 일정할 때, UCL RTP 라이브러리의 경우에는 페이로드 크기에 관계없이 메모리 요구량이 일정한 반면, embeddedRTP는 페이로드의 크기에 따라 적절한 유닛들을 사용하여 저장하기 때문에 페이로드가 작은 패킷이 많을수록 적은 메모리를 사용함을 알 수 있다.

EmbeddedRTP와 UCL RTP 라이브러리의 실제 메모리 요구량을 측정하기 위해서는 표 6과 같은 특성을 갖는 동영상 샘플들을 이용했다.

표 6 실제 메모리 요구량 측정에 이용된 샘플의 특성

항목	영상 크기(pixel)	비트율(kbps)	프레임율(fps)
샘플 1	176 x 110	125	15
샘플 2	320 x 240	277	30
샘플 3	160 x 120	101	30
샘플 4	160 x 120	68	30

표 6의 동영상 샘플을 실시간 스트리밍으로 재생했을 때, 패킷당 평균 페이로드 크기, 평균 버퍼링된 패킷수, embeddedRTP가 사용한 평균 유닛수, 그리고 이를 이용하여 도출한 UCL RTP 라이브러리와 embeddedRTP의 메모리 요구량을 비교하면 표 7과 같다. 이때 동영상 재생 시작시의 미디어 버퍼링 시간은 메모리 사용량에 큰 영향을 미치게 되는데, 일반적으로 버퍼링 5~10초 정도이다. 본 실험에서는 동영상 플레이에 영향을 미치지 않으면서 필요한 메모리 요구량을 줄이기 위해 미디어 버퍼링 시간은 2초로 하였고, 유닛의 크기는 앞의 경우와 마찬가지로 200 바이트로 하였다.

표 7에서 보다시피 embeddedRTP가 UCL RTP 라이브러리 메모리 요구량의 28%~65.8%를 이용하여 같은 일을 수행함을 알 수 있었다. 이와 같이 샘플 동영상의 특성에 따라 메모리 요구량은 크게 달라진다. 좀 더 구체적으로 샘플 2와 샘플 4의 측정값을 비교해 보면 평균 버퍼링 패킷수는 비슷하지만 평균 페이로드 크기가 약 570 바이트 차이가 난다. 이 경우 UCL RTP 라이브러리는 페이로드 크기에 상관없이 메모리를 사용하기 때문에 각 샘플에 필요한 메모리 요구량이 거의 비슷하게 측정되었으나, embeddedRTP는 샘플 2의 경우

표 7 UCL RTP와 embeddedRTP의 메모리 요구량

항목	페이로드 크기 (bytes)	평균 버퍼링 패킷수	embeddedRTP가 사용한 평균 유닛수	메모리 요구량 (bytes)		대비(%)
				UCL RTP	embeddedRTP	
샘플 1	798	27	4.683	40,087	25,031	62.4
샘플 2	849	57	4.935	85,580	56,315	65.8
샘플 3	389	87	2.778	131,180	47,469	36.2
샘플 4	280	58	2.099	86,594	24,231	28.0

페이로드 크기가 샘플 4의 경우보다 훨씬 커 더 많은 메모리를 요구하였다. 하지만 embeddedRTP와 UCL RTP 라이브러리의 메모리 요구량을 비교해보면, embeddRTP가 샘플 2에서는 UCL RTP 라이브러리와 비교하여 약 65.8%, 샘플 4의 경우에는 약 28%의 메모리를 필요로 하였다.

또 샘플 1과 샘플 4의 측정값을 비교해 보면 샘플 1이 샘플 4에 비해 평균 페이로드 크기가 큰 반면 평균 버퍼링 패킷수는 적다. 이 두 샘플의 경우, embeddedRTP의 메모리 요구량은 거의 비슷하나, UCL RTP 라이브러리는 매 패킷마다 일정한 크기의 메모리를 할당하기 때문에 메모리 요구량은 평균 버퍼링 패킷수에 따라 큰 폭의 차이가 있다.

샘플 3은 다른 샘플에 비해 평균 페이로드 크기가 비교적 작고 평균 버퍼링 패킷수는 많기 때문에 embeddedRTP와 UCL RTP의 메모리 요구량에서 있어서 가장 큰 차이가 났다.

4.2.3 코드 사이즈

내장형 시스템은 실제 메모리 크기가 일반 PC에 비해 훨씬 작기 때문에 내장형 시스템용 소프트웨어의 코드 사이즈는 중요하다. 표 8은 embeddedRTP의 코드 사이즈와 UCL RTP 라이브러리에서 수신에 필요한 코드들의 사이즈를 비교한 결과이다.

EmbeddedRTP의 코드 사이즈는 커널 수준에 구현된 embeddedRTP 모듈과 embeddedRTP 모듈을 사용하기 위한 embeddedRTP API의 합계로 측정하였다. 측정 결과에 의하면 embeddedRTP의 코드 크기가 UCL RTP에 비하여 약 42%정도임을 알 수 있다. 코드 사이즈를 축소할 수 있었던 가장 큰 이유는 RTP 스트리밍 송신에 필요한 자료구조를 제거하였고, 변환기 및 혼합기 등 임베디드 시스템에 적합하지 않은 코드를 제거한 것이다.

표 8 코드 사이즈

항목	embeddedRTP(bytes)		UCL RTP (bytes)
	embeddedRTP 모듈	embeddedRTP API	
코드 사이즈	25,080	804	61,132
	25,884		

4.2.4 멀티미디어 세션 확인에 따른 오버헤드

EmbeddedRTP는 시스템으로 전송된 모든 UDP 패킷에 대해 멀티미디어 세션에 속한 RTP 패킷인지 여부를 검사해야한다. 따라서 멀티미디어 세션에 포함되지 않은 RTP 패킷과 UDP 패킷에 대해 불필요한 멀티미디어 세션 확인에 따른 오버헤드를 측정하였다. 오버헤드는 최악의 경우 패킷당 평균 0.014 ms가 걸리는 것으로 측정되었다. 이는 RTP 패킷의 평균 처리속도인 0.289 ms의 약 5%에 해당하는 수치로서 무시할 수 있다. 이처럼 멀티미디어 세션 확인에 다른 오버헤드가 그다지 크지 않다는 점과 일반적으로 UDP를 사용하면서 멀티미디어 서비스에 이용되지 않는 TFTP, DHCP, ECHO 등의 프로토콜이 높은 성능을 요구하지 않는 점을 고려하면 embeddedRTP로 인해 발생하는 멀티미디어 세션 확인 오버헤드가 이들 프로토콜의 성능에 그다지 영향을 미치지 않을 것으로 예상된다.

5. 결론 및 향후 연구 과제

본 논문의 목적은 멀티미디어 서비스의 대표적인 전송 프로토콜인 RTP를 내장형 시스템에 적합하도록 설계하고 이를 커널 수준에서 구현함으로써 RTSP, H.323, SIP와 같은 멀티미디어 서비스 지원 프로토콜과 멀티미디어 플레이어 등에서 RTP 프로토콜을 사용할 때, 시스템 자원을 적게 소모하면서도 멀티미디어 데이타의 효율적인 전송을 보장하는데 있다.

본 논문에서는 우선 RTP와 그 문제점에 대하여 간략히 기술하고, 국내외에서 이루어지고 있는 RTP 관련 연구의 동향에 대하여 조사하였다. 그 다음 embeddedRTP의 설계와 구현에 대하여 설명하였고, 다윈 스트리밍 서버와 embeddedRTP를 사용하도록 수정된 MPEG4IP 플레이어를 이용하여 embeddedRTP의 동작 여부를 PDA상에서 확인하고 성능을 검증하였다.

EmbeddedRTP의 성능 측정 결과를 보면, 커널 수준의 embeddedRTP가 비교 대상인 UCL RTP 라이브러리에 비해 코드 사이즈는 약 58% 정도 줄일 수 있는 반면, 패킷 처리속도는 약 7.8배가 빨라졌고, RTP 패킷의 페이로드의 크기가 작을수록 패킷 저장시의 메모리

요구량이 적어지는 것으로 분석되었다. 또한 모든 UDP 패킷에 대해 멀티미디어 세션 확인 과정을 거침으로써 발생하는 오버헤드는 최악의 경우 패킷당 0.014 ms가 걸려, UDP를 사용하면서 멀티미디어 서비스에 이용되지 않는 TFTP와 같은 프로토콜들이 높은 성능을 요구하지 않는다는 것을 감안하면 그리 크지 않은 것으로 분석되었다.

논문에서 구현된 내용을 보다 발전시키기 위해서는 다음과 같은 몇 가지 사항이 추가로 연구되어야 할 것이다. 첫째, 확장성 있는 RTP 패킷 버퍼의 구현이다. 본 논문에서 구현된 embeddedRTP는 RTP 패킷 버퍼를 정적으로 구성함으로써 동적 할당에 비해 패킷 처리 속도는 향상됐지만 버퍼의 확장성이 없다. 즉, 시스템에 RTP 패킷이 갑자기 몰리는 경우 패킷 손실이 발생할 수 밖에 없다. 이 문제를 해결하기 위해서는 RTP 패킷 버퍼 이외에 오버플로우 패킷 버퍼 등을 사용하여 확장성 있는 RTP 패킷 버퍼를 구현해야 한다. 둘째, RTP 패킷 버퍼의 구성 파라미터의 동적 조절 기법에 관한 연구가 필요하다. 패킷 버퍼 청크를 구성하는 유닛의 크기는 메모리 사용의 효율성과 버퍼 탐색시간 사이에서 상충관계를 갖기 때문에, 애플리케이션, 타깃 시스템의 스펙 등에 따라 최적의 성능을 낼 수 있는 유닛 크기를 결정할 수 있는 방안의 모색이 필요하다. 특히 유닛의 크기를 커널 컴파일 시에 결정하지 않고, 애플리케이션 계층에서 동적으로 조정할 수 있도록 지원하여야 한다. 셋째, 메모리 복사 횟수의 최소화이다. 현재 UDP 소켓 버퍼로부터 embeddedRTP, 그리고 embeddedRTP로부터 애플리케이션으로의 복사가 필요하기 때문에 기존의 UCL RTP와 비교해서 메모리 복사 횟수를 줄이지 못했다. 향후 UDP 소켓 버퍼를 RTP 패킷 버퍼로 사용하거나 애플리케이션 메모리 영역을 RTP 패킷 버퍼로 사용하는 등 메모리 복사 횟수를 줄일 수 있는 기법에 대한 연구가 필요하다. 마지막으로, 구현된 embeddedRTP를 다양한 애플리케이션과 네트워크 환경에 적용하고, 스트레스 테스트를 통하여 안정성에 대한 검증이 필요로 하다.

참 고 문 헌

[1] ITU-T Recommendation, H.323: "Packet Based Multimedia Communications Systems," Feb. 1998.

[2] J. Rosenberg, et al., "SIP : Session Initiation Protocol," RFC 3261, Jun. 2002.

[3] H. Schulzrinne, et al., "Real Time Streaming Protocol(RTSP)," RFC 2326, Apr. 1998.

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

[5] 남상준, 이병래, 김태우, 김태윤, "실시간 멀티미디어 데이타 전송을 위한 SRTPIO 모듈 설계 및 구현", 한국정보과학회 논문지, 제28권 제4호, pp. 621-630, Dec. 2001.

[6] 문영준, 유인태, "RTP 기반 멀티미디어 데이타 전송을 위한 동적 QoS 제공 방안 및 구현", 한국정보과학회 2002년 추계 학술대회, 제29권 제2호, pp. 562-564, Sep. 2002.

[7] 온타임텍, "Technology-Streaming", "http://www.ontimetek.com/tec_server.asp".

[8] 팜캐스트, "팜캐스트 기술," "http://palmcast.co.kr/technique/contents1-2.htm".

[9] Lucent Labs., "Lucent Technologies Software Distribution," "http://www.bell-labs.com/topic/swdist".

[10] University College London, "UCL Common Multimedia Library," "http://www-mice.cs.ucl.ac.uk/multimedia/software/common/index.html".

[11] Lawrence Berkeley National Laboratory, "UCB/LBNL Video Conferencing System(vic)," "http://www-nrg.ee.lbl.gov/vic/"

[12] Gaibisso, C., et al. "Mobile code implementation of the RTP protocol in Java: design choices and evaluation," ITS '98 Proceedings. SBT/IEEE International, vol.2, pp. 644-649 Aug. 1998.

[13] 박상현, 박상윤, 김명준, 엄영익, "응용 독립적인 RTP 통신 모듈의 설계 및 구현", 한국정보처리학회 논문지, 제6권 제9호, pp. 2512-2523, Sep. 1999.

[14] Sun Microsystems "Mobile Media API(MMAPI)," "http://java.sun.com/products/mmapi/"

[15] Sun Microsystems, "Java 2 Platform, Micro Edition," "http://java.sun.com/j2me/"

[16] U. Vahalia, "UNIX Internals : the New Frontiers," Prentice-Hall, Oct. 1996.

[17] Apple, "Apple-Public Source-Darwin Streaming Server", "http://developer.apple.com/darwin/projects/streaming/"

[18] MPEG4IP, "MPEG4IP - Open Streaming Video and Audio," "http://mpeg4ip.net".

[19] Handhelds.org, "The Familiar Project", "http://familiar.handhelds.org".

선 동 국

정보과학회논문지 : 컴퓨팅의 실제
제 10 권 제 3 호 참조

김 태 웅

2002년 2월 중앙대학교 컴퓨터 공학사 취득. 2004년 2월 중앙대학교 컴퓨터 공학 석사 취득. 2004년 2월~현재 (주)텔리언 재직중. 관심분야는 네트워크 및 네트워크 운용 관리, 멀티미디어 시스템

김 성 조

정보과학회논문지 : 컴퓨팅의 실제
제 10 권 제 3 호 참조