실시간 멀티미디어 데이타 전송을 위한 SRTPIO 모듈 설계 및 구현

(The Design and Implementation of the SRTPIO Module for a Real-time Multimedia Data Transport)

남 상 준 † 이 병 래 † 김 태 우 †† 김 태 윤 †††
(Sang-Jun Nam) (Byung-Rae Lee) (Tai-Woo Kim) (Tai-Yun Kim)

요 약 최근 멀티미디어 서비스에 대한 사용자의 요구가 증가되고 있다. 그러나 서버 시스템은 이러한 멀티미디어 데이타를 사용자들에게 효율적으로 공급하지 못한다. 본 논문에서는 기존의 운영체제에 멀티미디어 데이타를 효율적으로 전송하기 위한 방안으로 RTP(Real-time Transport Protocol) 프로토콜을 SIO(Special Input/Output) 메커니즘과 함께 커널 영역에 내장하는 *SRTPIO(Special RTP Input/Output)* 모듈을 제안한다. SIO 메커니즘은 일반적인 서버 시스템에서 발생하는 사용자 영역과 커널 영역 사이의 데이타 복사와 문맥 교환을 줄임으로써 데이타 전송의 효율을 증가시킨다. SRTPIO 모듈은 RTP 프로토콜을 SIO 메커니즘과 커널 영역에서 통합해 효율적 멀티미디어 데이타 전송 구조를 지원한다.

Abstract Recently, users' demands for multimedia service are increasing. But, server systems offer inefficient multimedia data service to users. In this paper, to transport multimedia data in the server system more efficiently, we propose the *SRTPIO(Special RTP Input/Output)* module that process the RTP(Real-time Transport Protocol) data in the kernel with the SIO(Special Input/Output) Mechanism. The SIO mechanism improve a transfer speed because it reduces overheads associated with data copying and context-switching between the user mode and the kernel mode occured in general server system in the kernel-level. The SRTPIO module, integrating the SIO mechanism and the RTP data processing in the kernel, support efficient multimedia data transfer architecture.

1. 서 론

최근 멀티미디어 서비스에 대한 사용자의 요구에 따른 대단위 데이타 처리로 인해 서버 시스템의 부하가 증가하고 있다[1]. 이러한 요구를 충족시키는 서버 시스템은 지금까지의 한정적이고 일반적인 범위에서 벗어나, 실시간 처리 능력과 대용량의 데이타를 효율적으로 전송하는 새로운 서버 시스템이 요구되고 그림 1과 같은 서비스를 한다[2].

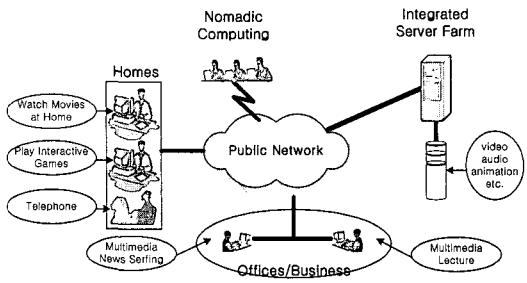


그림 1 멀티미디어 데이타에 대한 다양한 사용자의 요구

이러한 서버 시스템의 애플리케이션은 저장 장치와 네트워크 하부 시스템 사이에서 주기적으로 데이타를 주고받는다. 따라서 기대되는 작업에 대해 서버 시스템은 높은 성능으로 디자인되고 전송되어야 한다. 이 방안으로 멀티미디어 데이타 전송을 위한 RTP(Real-time

† 학생회원 : 고려대학교 컴퓨터학과
sjnam@netlab.korea.ac.kr
brlee@netlab.korea.ac.kr
†† 비 회 원 : 성공회대학교 컴퓨터정보공학부 교수
ktw@mail.skhu.ac.kr
††† 종신회원 : 고려대학교 컴퓨터학과 교수
tykim@netlab.korea.ac.kr
논문접수 : 2001년 3월 13일
심사완료 : 2001년 8월 13일

Transport Protocol)[3] 프로토콜이 사용되어지나 저장 장치와 네트워크 하부 시스템 사이의 효과적인 경로는 제어하지 못한다[4]. 대부분의 RTP 프로토콜을 지원하는 멀티미디어 응용 애플리케이션들은 사용자 영역과 커널 영역에서 과도한 데이타 복사와 문맥 교환이 발생한다[5]. 멀티미디어 데이타 전송과 같이 한번 접속된 서비스가 다른 변경 없이 연속적으로 전송된다면, 데이타 복사의 횟수를 줄이고 빠른 데이타 경로와 전송을 이룰 수 있다.

본 논문에서는 아직 멀티미디어 데이타 전송을 위해 커널 영역에서 수행하는 연구가 시작되지 않은 RTP 프로토콜을 SIO(Special Input/Output) 메커니즘과 함께 커널 영역에 내장함으로써 비디오와 오디오 같은 실시간 서비스에 만족하는 효율적 전송 모듈을 제안한다.

제안된 SIO(Special Input/Output) 메커니즘은 빠른 전송을 수행하기 위해 UDP 기반 전송 경로를 변경하여 이를 기반으로 리눅스 환경에서 구현과 성능 평가를 한다. 이러한 SIO 메커니즘과 RTP 프로토콜을 커널 영역에서 통합한 SRTPIO 모듈은 멀티미디어 데이타 전송의 서비스 품질을 향상시키고 입출력(Input/Output) 시스템의 성능을 향상시킨다.

본 논문의 구성은 다음과 같다. 2장에서 멀티미디어 서버 시스템의 입출력 메커니즘과 RTP 프로토콜을 살펴보고, 기존의 커널 기반 실시간 서버 시스템의 문제점을 분석한다. 3장에서 RTP 프로토콜과 리눅스 서버의 전송 메커니즘을 기반으로 SRTPIO 모듈을 제안한다. 4장에서 멀티미디어 환경의 일반적 리눅스 서버와 SRTPIO 모듈을 내장한 리눅스 서버와의 비교 및 성능 평가를 한다. 마지막 5장에서는 본 논문의 결론 및 향후 연구 과제를 제시한다.

2. 관련 연구

2.1 RTP 기반 멀티미디어 서버 시스템의 입출력 메커니즘

최근 인터넷상에서 멀티미디어 서비스들의 전송에 대한 과부하와 실시간 전송을 위한 연구에 관심이 집중되고 있다. 인터넷폰(VoIP)의 성장, 리얼 오디오(Real-Audio)상의 오디오와 비디오 스트림, Mbone (Multicast Backbone)이 실례이다[6]. RTP(Real-time Transport Protocol) 프로토콜은 멀티캐스트 또는 유니캐스트상에서 음성, 화상, 또는 모의 데이타와 같은 실시간 데이타를 전송하는 응용에 적합한 단대 단 트랜스포트 기능을 제공한다. 그러나 RTP 프로토콜은 자원 예약에 대한 내용은 다루지 않으며 특히 적시 데이타 전송(timely delivery), QoS 보장, 뒤바뀐 순서의 전송 방지와 같은 기능은 제공하지 않는다. RTP 프로토콜은 전송 계층과 네트워크 계층의 상위에서 작동하도록 디자인 된 프로토콜이다[3]. 그림 2는 기존에 사용되는 RTP 프로토콜의 스택이다.

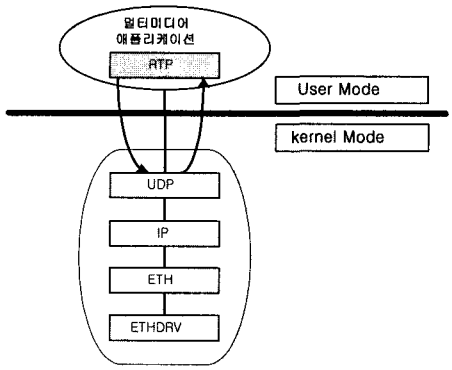


그림 2 RTP 기반 멀티미디어 데이타 전송 스택

그림 2에서 사용자 영역에서 멀티미디어 데이타와 함께 만들어진 RTP 헤더는 그 하위 계층인 UDP 프로토콜로 전달되고 네트워크 인터페이스로 전달되는 모습을 볼 수 있다. 바로 이러한 점에서 RTP 데이타의 생성과 하부 계층으로의 전송을 무수히 반복하는 멀티미디어 데이타의 경우는 과도한 데이타 복사와 문맥 교환이 발생한다.

RTP 프로토콜은 멀티미디어 데이타의 실시간 전송을 위해 만들어졌다. 애플리케이션상에서 RTP의 전송 속도를 향상시키는 연구와 QoS에 대한 연구[6, 7]는 계속 진행 중에 있으나 RTP 프로토콜의 전송 스택을 커널 영역으로 옮기는 연구는 아직까지 없다.

본 논문에서는 RTP 프로토콜이 사용자 영역에서 동작되어 성능이 늦어지는 현상을 방지하기 위해 RTP를 커널 영역에 포함시켜 RTP 데이타를 만들 때 과도한 복사가 일어나지 않도록 하는 메커니즘을 제안한다.

2.2 일반적인 서버 전송과 MARS(Massively-parallel And Real-time Storage)메커니즘

그림 3은 현재 UNIX 운영체제 시스템에서의 네트워크 입출력 시스템의 계층구조를 보여 준다. 애플리케이션에서 데이타를 요구하고 그 데이타가 네트워크 인터페이스로 전송되는 일반적인 과정을 나타낸다.

그림 3의 서버 시스템은 사용자 영역에서 네트워크

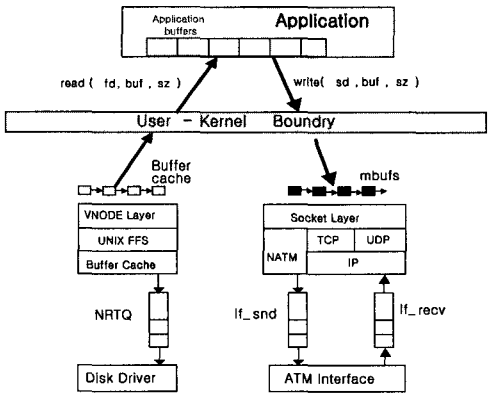


그림 3 일반적인 서버 시스템의 데이타 전송 메커니즘

장치까지 두 단계의 데이타 복사가 사용자 영역과 커널 영역 사이에서 발생한다.

첫 번째 복사는 read 호출이다. 이것은 커널 영역의 버퍼 캐쉬에서 사용자 영역 버퍼까지 데이타를 옮기는 경우이다. 두 번째 복사는 write 호출이다. 이 write 호출은 사용자 영역 버퍼의 데이타를 네트워크 버퍼로 보내는 역할을 한다. 그림 3의 메커니즘은 작은 크기의 입출력을 행하는 전통적인 텍스트와 바이너리 파일은 전송이 잘 이루어진다. 그러나 멀티미디어 데이타(오디오, 비디오, 애니메이션 등)와 같은 캐싱 속성을 가진 데이타들은 기대하는 것만큼의 효과를 보지 못한다[8]. 또한 이 서버 시스템은 메모리 공간을 많이 사용한다. 데이타는 종종 이것이 재 사용될 수 있음에도 불구하고 대체될 수 있다.

데이타 복사를 최소화하려는 시도는 이전부터 연구되어 왔다. 보호 영역(protection domains)을 통해 물리적 데이타 이동을 최소화하는 기술인 fbufs[9], 커널 영역에 식별자를 두어 저장 장치와 네트워크 장치 사이의 경로를 줄이는 Splice[10], MARS[8] 메커니즘 등이 있다. 이러한 연구들이 데이타 복사를 줄임으로써 데이타 전송 속도를 높이려는 것이나, 멀티미디어 데이타 전송시 RTP 프로토콜을 커널 영역으로 옮기는 연구는 제안되지 않았다.

일반적인 서버 전송 메커니즘의 단점인 잦은 데이타 복사의 보완이 MARS 메커니즘이다. 그림 4는 디스크에서 네트워크 입출력으로의 새로운 데이타 경로와 제어에 대해 보여주고, 특징을 살펴보면 다음과 같다.

디스크와 네트워크 입출력 사이의 데이타 전달을 Zero-Copy로 만들기 위해 mmbufs(Multimedia Memory Buffers)라는 새로운 버퍼 관리 메커니즘을 디자인했다. 그림 4에서 보듯이 새로운 데이타 경로는 mmbufs라는 새로운 구조를 통해 이루어진다[8].

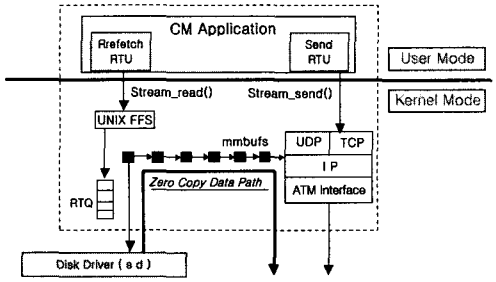


그림 4 MARS 메커니즘

MARS 메커니즘은 버퍼 캐쉬인 mmbufs를 커널 영역에서 관리함으로써 사용자 영역과 저장장치 사이의 전송 수단으로 사용된다. 그림 4와 같이 mmbufs를 통해 저장장치와 네트워크 사이의 Zero-Copy와 같은 빠른 데이타 경로가 공급될 수 있다[4, 8]. 그러나 MARS 메커니즘도 다음과 같은 문제점이 발생한다.

- mmbufs는 2개의 포인터 리스트로 구성되어 디스크에서 네트워크로 데이타를 전송한다. 이렇게 추가의 포인터 연산으로 인해 비효율이 발생한다.
- 새로운 mmbufs라는 커널 영역에 구조체 생성으로 과도한 메모리 할당이 이루어진다.
- 디스크 장치와 네트워크 장치 사이에 새로운 mmbufs라는 구조체의 생성으로 인한 다른 장치와의 호환성 결여 문제가 발생한다

본 논문에서는 처음으로 RTP 프로토콜을 커널 영역으로 옮기고 여기에 맞는 인터페이스와 데이타 복사를 줄이는 SIO 메커니즘을 만들어 멀티미디어 데이타 전송을 향상시키는 SRTPIO 모듈을 제안한다.

3. SRTPIO(Special RTP Input/Output) 모듈 설계 및 구현

2장에서 설명한 바와 같이 RTP 프로토콜은 현재 사용자 영역 애플리케이션상에서 동작되므로 커널 영역 내부에서 동작하는 것보다 데이타 복사와 문맥 교환에서의 지연 시간이 발생하는 단점이 있다. 이 장에서는 먼저 멀티미디어 데이타의 효율적 전송과 RTP 프로토콜과의 인터페이스를 맞추는 SIO 메커니즘을 제안하고, RTP 프로토콜을 SIO(Special Input/Output) 메커니즘을 통합해 커널 영역에 내장하는 SRTPIO 모듈을 제안한다.

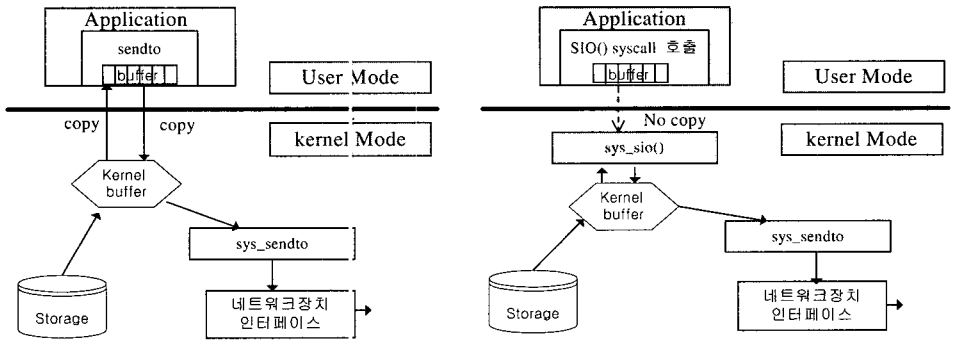
3.1 SIO(Special Input/Output) 메커니즘

커널 영역에서 RTP 프로토콜을 지원하고 커널 영역과 사용자 영역 사이의 인터페이스를 지원하기 위해 이 절에서는 SIO 메커니즘을 설계한다. 또한 SIO 메커니즘은 애플리케이션과 커널 영역 사이의 데이타 복사와 문맥 교환의 오버헤드를 줄이기 위해 일반적인 서버 시스템에 존재하는 데이타 전송 단계를 줄임으로써 전송 상의 향상을 보고자 한다.

오디오나 비디오 데이타와 같은 멀티미디어 데이타는 변경이 많이 일어나지 않는 연속적인 스트림의 전송이다. 이와 같은 전송은 항상 새로운 데이타를 생성해서 전송하는 것이 아니라 저장 장치로부터 읽어온 데이타를 연속적으로 전송하면 된다. 다음에 전송할 데이타가 무엇인지 아는 멀티미디어 데이타인 경우 일반적인 서버 시스템에서는 반복적으로 사용자 영역과 커널 영역을 넘나들면서 데이타를 이동시키며 불필요한 데이타 복사와 문맥 교환으로 인한 오버헤드가 발생하게 된다. 연속적인 데이타를 전송할 때 잦은 데이타 복사와 문맥 교환을 줄이기 위해 사용자 영역에서는 한번의 시스템 호출을 하고 연속적인 데이타는 커널 영역에서 수행하는 전송 메커니즘이라면 전송 시간을 단축시킬 수 있을 것이다. 특히 UDP 전송처럼 한번의 수신을 받고 다음 수신에 상관없이 송신을 하는 프로토콜의 경우 이러한 오버헤드는 더욱 심하다. RTP 데이타 또한 UDP 전송 계층을 이용하므로 이러한 오버헤드가 발생한다. 리눅스 기반의 일반적인 UDP 전송 메커니즘의 커널 경계 부근에서 데이타 전송 경로는 그림 5의 (a)와 같다. 그림 5의 (a)는 애플리케이션에서 UDP 프로토콜을 거쳐 네트워크 인터페이스로 데이타 전송 경로를 나타낸 것이다. 그림 5의 (a)는 sendto 함수를 호출하면서 커널 영역과 사용자 영역의 경계에서 데이타 복사가 발생하는 모습을 볼 수 있다. 일반적인 리눅스 시스템에서 UDP 기반 네트워크 전송 과정은 다음과 같다. UDP 전송을 위해 소켓과 바인드를 설정한 후 recvfrom 함수로 클라이언트의 접속을 기다리게 된다. recvfrom 함수로 들어온 클라이언트의 정보와 보낼 데이타 생성 후 sendto 시스템 호출을 통해 커널 영역 내부의 sys_sendto 함수를 호출한다. 이 sys_sendto 함수는 udp_sendto 함수에게 보내고, 이 함수는 UDP 헤더를 붙이고 ip_build_xmit 함수를 통해 네트워크 장치 인터페이스로 전송을 수행하게 된다[11,12,13,14]. 그러나 그림 5의 (b)와 같이 SIO 메커니즘을 사용한 전송 메커니즘에서는 sendto 함수를 커널 영역 내부에 추가함으로써 데이타 복사를 줄이는 과정을 볼 수 있다.

그림 5의 (a)와 차이가 있는 그림 5의 (b)의 전송 과정은 다음과 같다. UDP 전송을 위해 소켓과 바인드를 설정한 후 recvfrom 함수로 클라이언트의 접속을 기다리게 된다. recvfrom 함수로 들어온 클라이언트의 정보를 sendto 시스템 호출을 하는 것이 아니라 본 논문에서 제안하는 SIO 시스템 호출로 커널 영역 내부에 만든 sys_sio 함수를 호출하게 되는 것이다. SIO 시스템 호출에서 넘어온 클라이언트의 정보와 파라미터로 sys_sio 함수에서 데이타를 생성한 후 커널 영역 내부의 sys_sendto 함수로 넘겨주게 된다. 이렇게 넘겨준 패킷은 udp_sendto 함수를 통해 UDP 헤더를 붙이고 ip_build_xmit 함수를 통해 네트워크 장치 인터페이스로 전송을 수행하게 된다.

그림 5의 (b)와 같은 모델을 가진 운영체제는 현재 존재하고 있지 않다. 이와 같은 모델로 사용자 영역과 커널 영역 사이의 데이타 복사와 문맥 교환을 줄여 전송 효율을 증대시키는 동시에 기존의 커널 메커니즘 최



(a) 일반적인 리눅스 전송 메커니즘 (b) SIO 시스템 콜 추가 전송 메커니즘

그림 5 SIO 시스템 호출 추가 전과 후의 커널영역와 사용자영역 경계지점 전송 메커니즘

소한의 변경으로 호환성도 유지하였다.

이러한 SIO 메커니즘을 MARS 메커니즘과 비교해 볼 때 장점을 정리하면 다음과 같다.

- MARS 메커니즘의 mmbuf는 두 개의 포인터 리스트를 찾아다니면서 새로운 버퍼의 할당이나, 디스크의 읽기 작업이 시작되면 해당 mmbuf의 포인터를 찾아다니는 오버헤드를 이 SIO 메커니즘에서는 기존의 전송 방식을 그대로 따르면서 전통적인 전송 단계의 과정을 줄여줌으로써 전송 속도상의 효과를 보고자 하였다.
- MARS 메커니즘의 경우처럼 전송량이 많아질 경우 과도한 mmbuf의 할당이 아닌 기존의 캐쉬 버퍼를 사용함으로써 적응성을 유지하였다.
- SIO 메커니즘은 MARS 메커니즘과는 달리 다른 장치들과의 호환성 측면에서도 새로운 구조체를 만들지 않음으로써 유연성을 가지게 하였다.

이러한 SIO 메커니즘의 장점을 통해 RTP 지원 멀티미디어 데이타를 커널 영역에서 처리해 네트워크 인터페이스로 보내는 SRTPIO 모듈을 3.2절에서 제안한다.

3.2 SRTPIO(Special RTP Input/Output) 모듈 동작 원리

본 논문에서의 제안하는 SRTPIO 모듈은 RTP 프로토콜을 3.1절에서 제안한 SIO 메커니즘을 이용해 커널 영역으로 내장하는 것이다. RTP 프로토콜이 UDP 프로토콜 기반으로 전송이 수행되므로 UDP 기반의 SIO 전송 메커니즘과 잘 호환된다. 본 논문에서 제안하는 SRTPIO 통합 모듈의 구조는 그림 6과 같다.

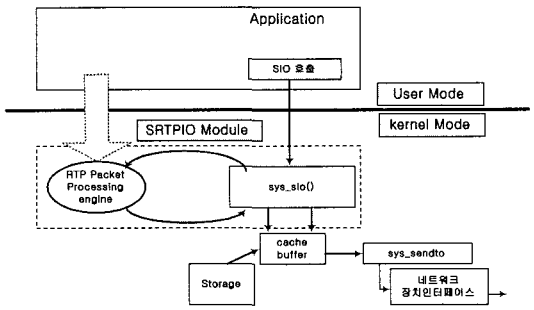


그림 6 SRTPIO 모듈 구성도

일반적인 RTP 기반 애플리케이션에서의 전송 과정은 다음과 같다. 클라이언트의 멀티미디어 서비스 요청을 받은 서버 시스템은 서비스 할 멀티미디어 데이타의 위치를 알아낸다. 이 위치의 데이타를 커널 영역의 버퍼에서 애플리케이션의 버퍼로 넘겨받고 RTP 패킷을 생성하여 커널 영역에 있는 하부 계층인 UDP 계층으로 넘겨준다. UDP 계층에서는 자신의 헤더를 붙인 후 네트워크 인터페이스로 옮겨준다. 이러한 과정을 반복으로 서비스 할 데이타는 전송이 되고, 사용자 영역과 커널 영역 사이에서 데이타 복사와 문맥 교환이 끊임없이 발생하게 된다.

그러나 이러한 단점을 제거한 그림 6의 SRTPIO 모듈은 클라이언트에서 멀티미디어 서비스 요청을 받은 서버 시스템은 서비스 할 멀티미디어 데이타의 위치와 정보를 SIO 호출을 이용해 커널 영역으로 내려 보내준다. 이 sys_sio 함수는 사용자 영역에서 받은 정보와 데이타를 RTP 패킷 프로세싱에게 보내 RTP 헤더를 붙인 RTP 패킷을 생성한다. 이 패킷을 같은 커널 영역에서 UDP 계층으로 넘겨주고 UDP 계층은 자신의 헤더를 붙인 후 네트워크 인터페이스로 옮겨주게 되는 것이다. 이러한 과정으로 다음 패킷부터 마지막 패킷까지 커널 영역에서 RTP 패킷 생성과 UDP 전송을 하는 일련의 작업을 처리하게 된다.

그림 6에서 보듯이 애플리케이션은 각각의 패킷이 만들어지던 RTP 프로토콜을 커널 영역 내부로 옮겼다. 이로 인해 저장 시스템과 사용자 영역 애플리케이션을 오가며 수행하는 데이타의 과도한 복사를 방지 할 수 있다. 또한 RTP 프로토콜을 커널 영역에 내장함으로써 비디오, 오디오 기반 애플리케이션의 단순성을 꾀하였다. 이러한 SRTPIO 모듈을 리눅스 시스템에 맞추어 설계하였다. 또한 SRTPIO 모듈은 RTP 패킷의 처리뿐만 아니라 UDP 기반 전송 단계를 SIO 시스템 호출을 이용하여 기존의 전송 단계를 줄임으로써 성능상의 개선을 볼 수 있다. 이러한 개선을 살펴보면 다음과 같다.

- 입출력의 개선
  애플리케이션에서 한번의 SIO 시스템 호출로 인해 데이타 전송을 커널 영역 내부로 옮김으로써 이에 대한 불필요한 데이타 복사와 문맥 교환을 줄였다.
- RTP 패킷 처리
  RTP 패킷을 생성함에 있어, 기존의 방식은 사용자 영역 애플리케이션에서 이루어지는 것을 커널 영역 내부의 모듈로 내장함으로써 처리 속도의 향상과 RTP 프로토콜 기반 응용 애플리케이션의 인터페이스를 단순하게 만들 수 있게 하였다.
- 커널 메커니즘의 수용
  최소한의 커널 메커니즘의 수정으로 효율적인 멀티미디어 데이타의 서비스를 지원한다.

3.3 SRTPIO 구현

그림 6과 같은 모듈을 가진 서버 시스템은 현재 존재하고 있지 않다. 멀티미디어 데이타 전송에 적합한 전송 메커니즘을 가지고 있지 않는 리눅스 시스템의 커널 영역에 새로운 SRTPIO 전송 모듈을 적용하여 구현하였다.

리눅스 시스템에 RTP 프로토콜을 커널 영역에 내장하였다. 리눅스의 UDP 전송 메커니즘은 3.1절의 그림 5의 (a)와 같이 동작하기 때문에 데이타 전송시 커널 영역과 사용자 영역 사이에서 데이타 복사와 문맥 교환이 발생한다. 이러한 단점을 피하기 위해 커널 영역 내부에 UDP 기반 RTP 패킷 전송을 담당하는 sio 시스템 호출을 등록한다. 커널 영역에서 수행하는 sio 함수의 RTP 패킷 전송 모듈의 알고리즘은 다음과 같다.

```
asmlinkage int sys_sio(fd, RTP_INFO)
{
    ...
    sioread(fd, buf, len);
    rtp_packet= rtp_hdr_build(RTP_INFO, buf, len);
    server_len= siosendto(rtp_packet);
    ...
    if (success)        flag == 1;
    else                flag == 0;
    return flag;
}
```

위 알고리즘은 크게 3부분의 함수로 구성되어 있다. 사용자 영역 애플리케이션에서 넘겨준 RTP 헤더의 정보를 가지고 sioread, rtp_hdr_build, siosendto의 함수로 멀티미디어 데이타 전송 메커니즘이 이루어지게 설계하였다.

SRTPIO 모듈의 동작 과정을 보면 다음과 같다.

① 클라이언트의 요청에 따라 서버의 애플리케이션이 열린다.

② 클라이언트와의 통신을 위한 설정이 세팅된다.

③ 서버 애플리케이션은 클라이언트의 요청에 맞는 멀티미디어 데이타의 정보를 sio 시스템 호출을 사용하여 커널 영역으로 전달한다.

④ sioread 함수는 전송하고자 하는 저장 장치의 위치 포인터를 가지며 데이타를 읽어 온다.

⑤ rtp_hdr_build 함수는 sioread 함수가 ④에서 읽어온 데이타와 사용자 영역 애플리케이션에서 넘어온 RTP 헤더 필드로 RTP 패킷을 만든다.

⑥ 만들어진 RTP 패킷을 3.1절에서 제시한 SIO 메커니즘 방식으로 siosendto 함수를 이용하여 RTP 패킷을 네트워크 인터페이스에게 보내게 된다.

위와 같은 동작 과정으로 본 논문에서 제안하는 SRTPIO 모듈은 사용자 영역과 커널 영역 사이의 불필요한 데이타 복사를 제거할 수 있으며, 사용자 영역 애플리케이션에서는 단지 커널 영역에 새로 만든 sio 시스템 호출만을 호출하면 된다.

서버 애플리케이션의 SRTPIO 모듈 호출 알고리즘은 다음과 같이 구현되었다.

```
/* file name : SRTPIOServ.c */
_syscall2(setting parameter)
int     main(void)
{
        RTP_INFO setting;
        socket setup;
        protocol, address, port establishment;
        ...
        bind establishment;
        start time;
      n = sio(fd, RTP_INFO);
        end time;
        if (n)      RTP Packet Service Success;
        else        RTP Packet Service Failed;
}
```

위의 서버 애플리케이션의 알고리즘을 보면 커널 영역에 내장한 SRTPIO 모듈을 sio 시스템 콜이 호출한다. 그리고 sio 함수에서 전송하고자 하는 파일의 포인터와 RTP 패킷의 정보를 보낸다. RTP_INFO의 내용은 RTP 패킷에서 주요한 정보를 나타내는 헤더의 필드를 의미하는데 버전 정보와 순서 번호와 같이 일괄적인 정보는 커널 영역의 SRTPIO의 모듈에서 설정하고 데이타 종류와 같은 필드는 애플리케이션에서 설정하고 커널 영역 내부로 정보를 넘겨준다. 넘겨준 정보와 포인터를 이용해 커널 영역에서는 모든 멀티미디어 서비스를 수행하고 결과값만을 리턴하게 된다. 이때의 애플리케이션은 리턴 값이 오는 동안 휴지 상태로 머물게 된다. 이로써 SRTPIO 모듈은 멀티미디어 데이타 전송시 SIO 메커니즘과 함께 RTP 패킷 전송시 사용자 영역과 커널 영역의 데이타 복사의 횟수를 줄여서 효율적 전송을 기대하고, 애플리케이션의 단순화로 인해 사용자 영역의 부하를 줄이게 되었다.

4. SRTPIO 모듈의 성능 평가 및 분석

위의 3장에서 제안한 SRTPIO 모듈의 성능 평가를 리눅스 시스템에서 수행하고 일반적인 리눅스 시스템과 SRTPIO 모듈 내장 리눅스 시스템의 전송 시간을 마이크로 세컨드 단위까지 비교하였다. 4.1절에서는 UDP 기반의 일반적인 전송 단계의 경로를 줄인 SIO 메커니즘

만의 성능을 알아보고, 4.2절에서는 성능이 향상된 SIO 메커니즘에 RTP 프로토콜을 커널 영역에 내장한 SRTPIO 모듈의 성능을 일반적인 리눅스 서버 시스템과 비교하여 본다. 이러한 실험을 통해 이 논문의 제안 모델인 SRTPIO 모듈의 성능 평가와 분석을 한다.

4.1 SIO 메커니즘의 성능 평가

리눅스 시스템에 SIO 메커니즘을 커널 영역에 내장하였다. 실험 환경의 구성을 위해 시스템 I은 서버 시스템으로 멀티미디어 데이타 패킷을 전송하고 시스템 II는 이 패킷을 수신함으로써 성능을 비교하였다. 두 시스템의 하드웨어 사양은 표 1과 같다.

표 1 시스템 및 운영체제의 소프트웨어 사양

	리눅스 시스템 I (Server)	리눅스 시스템 II (Client)
호스트 이름	netlab1.korea.ac.kr	mullab3.korea.ac.kr
CPU	인텔 팬티엄 III 600MHz	인텔 팬티엄 III 600MHz
RAM	128M	128M
Kernel Version	2.2.12-20	2.2.5-15
GCC Version	2.91.66	2.91.66
Libc6 Version	2.1.1	2.1.1

다음 표 2는 SIO 메커니즘 성능 평가에 사용한 데이타이다.

표 2 SIO 실험 환경

	패킷전체 크기	패킷당 전송간격	데이타의 종류	데이타의 생성
SIO 실험	32 byte	연속적 전송	일반적인 데이타	랜덤 생성

실험에 대한 세부 사항과 수행 시나리오는 다음과 같다.

- 현재까지의 리눅스 시스템은 RTP 프로토콜을 지원하지 않아 RTP 헤더 파일을 RFC 1889[3]의 권고에 따라 커널 영역에 내장하였다. 패킷 한 개의 크기는 오디오 스트림에 맞추어 32 바이트로 만들었다.
- SIO 메커니즘 성능 평가에서는 먼저 RTP 프로토콜을 추가하지 않고, 3.1절에서 제안한 SIO 메커니즘만으로 기존의 일반적인 리눅스와 SIO 메커니즘의 전송 속도를 비교하였다.
- 수행 시나리오는 일반적인 UDP 전송과 동일하게 하고 일반적 리눅스와 SIO 메커니즘 리눅스의 패킷 전송 시간을 비교하였다.
- 클라이언트의 요청을 받은 서버 시스템은 1개, 10개, 100개, 1000개, 10000개의 패킷을 각각 따로 전송하여 서버 측면에서의 전송 완료 시간을 비교하였고 실험 결과는 그림 7과 같다.

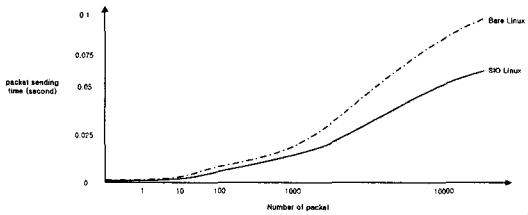


그림 7 일반적 리눅스와 SIO 메커니즘 리눅스의 패킷 수에 따른 전송 속도 비교

실험 결과는 각각 100회씩 반복하여 평균을 구한 값이다. 그림 7에서 보는 바와 같이 각각의 패킷 개수에 대해 일반적 리눅스의 전송 시간보다 SIO 메커니즘의 리눅스가 패킷 한 개의 전송에서부터 전송 패킷의 수가 많을수록 속도면에서 향상되는 그래프를 볼 수 있다.

그림 7에서 커널 영역에 SIO 메커니즘으로 UDP 전송을 수행한 리눅스의 전송횟수를 보면, 일반적 리눅스보다 약 6%~23%의 성능향상을 보였다. 더욱이 현재 비디오나 오디오 스트림이 많은 대용량 멀티미디어 데이타 전송을 고려할 때 성능 향상의 폭이 증가함을 알 수 있다.

데이타 전송 경로를 제어한 SIO 메커니즘은 사용자 영역 애플리케이션에서 서비스 할 패킷의 정보만 커널 영역으로 보내고 커널 영역에서 데이타를 전송함으로써 사용자 영역 애플리케이션과 커널 영역 사이의 데이타 복사와 문맥 교환의 횟수를 줄이게 된 결과이다.

4.2 SRTPIO 모듈의 성능 평가

4.1절에서 보듯이 SIO 메커니즘의 이점을 이용해 RTP 프로토콜을 리눅스 커널 영역에 내장해 SRTPIO의 성능이 향상됨을 알아본다. 성능 평가의 하드웨어 사양은 4.1절과 동일하다.

이 절에서는 4.1절의 실험보다 일반화 된 전송 시나리오를 구성하였다. 4.1절의 SIO 메커니즘은 서버 시스템의 전송 속도 향상만을 기대한 실험이므로 수신측의 패킷 손실을 고려하지 않았으나 이 실험에서는 RTP 패킷 전송 간격을 오디오 스트림의 간격인 30ms의 간격을 두어 전송하였다. 또한, 서버 시스템이 서비스하는 클라

이언트가 다수임을 가정하여 10개의 클라이언트가 서버 시스템에 동시에 접속하는 것으로 구성하였다. SRTPIO 모듈의 성능 실험의 환경을 정리하면 다음 표 3과 같다.

표 3 SRTPIO 실험 환경

	패킷 크기	패킷당 전송 간격	동시 접속 세션 수	세션당 패킷 개수
실험	32 bytes	30ms	10	1000

위의 표 3과 같이 실험 환경을 구성하여 실험을 하였다. 아래 그림 8은 일반적인 리눅스와 SRTPIO 모듈을 내장한 리눅스의 세션별 전송 속도를 비교한 그래프이다.

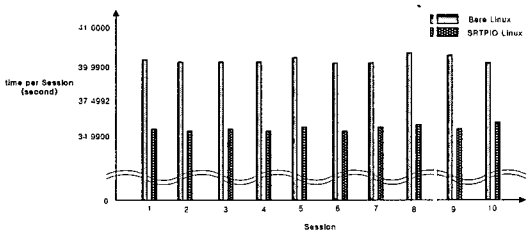


그림 8 일반적 리눅스, SRTPIO 모듈 리눅스의 세션별 전송 속도 비교

각각의 데이타 값은 100회씩 반복하여 평균을 구한 값이고 마이크로 세컨드 단위까지의 시간을 구했다. 그림 8에서 보듯이 SRTPIO 모듈을 내장한 리눅스가 일반적인 리눅스보다 각 세션별 성능이 뛰어남을 볼 수 있다. 각각의 10개 세션이 비슷한 시간으로 1000개의 패킷을 전송하는 것을 볼 수 있다. 위의 그림 8에서 SRTPIO 모듈을 내장한 리눅스가 일반적인 리눅스보다 30ms의 간격으로 패킷 전송을 처리할 때 약 12.5%의 성능 향상이 있음을 볼 수 있다. 12.5%라는 성능 향상이 실험에서는 패킷당 30ms의 단위로 전송하기 때문에 기본적인 지연시간이 있다. 실험에서 30ms의 간격으로 1000개의 패킷을 전송했으므로 적어도 30초의 전송 시간은 발생하는 것이다. 위의 결과와 4.1절의 SIO 메커니즘의 실험 결과에 의해 패킷의 수가 많아짐에 따라서 SRTPIO 모듈의 성능이 월등히 향상됨을 알 수 있다.

본 논문은 리눅스 환경 서버 시스템에서 멀티미디어 데이타 전송을 커널 영역에 내장할 경우 기존의 리눅스 환경의 전송보다 성능향상이 뛰어남을 보였다. 사용자 영역 애플리케이션에서 수행하는 RTP 프로토콜이 SIO 메커니즘과 함께 커널 영역에서 수행되므로 두 번 이상 발생하는 데이타 복사가 한번으로 줄었고 문맥 교환도 커널 영역에서 발생하므로 그 수행 속도가 애플리케이션보다 빠르게 일어난다.

표 4는 실험 결과를 바탕으로 일반적 리눅스, MARS 메커니즘, SIO 메커니즘과 본 논문의 제안인 SRTPIO 모듈 리눅스를 비교한 것이다.

표 4 기존 리눅스 시스템과 MARS, 제안된 SIO, SRTPIO 시스템간의 비교

(○: High △: Low ×: None)

	일반적 리눅스	MARS 메커니즘	SIO 리눅스	SRTPIO 리눅스
Read/Write 이득	×	○	○	○
전송 버퍼의 수정	×	○	△	△
전송 속도의 향상	×	○	○	○
RTP 패킷 처리	×	×	×	○
커널 메모리 소비	×	○	△	△
다른 장치와의 유연성	○	×	○	○

이러한 성능 향상을 보인 SRTPIO 모듈을 이용하여 멀티미디어 데이타 서버 시스템에 접목하면 새로운 멀티미디어 데이타 전송 구조를 지원할 수 있게 하였다.

5. 결론 및 향후 연구 과제

멀티미디어 서비스를 제공하는 서버 시스템이나 네트워크의 발전보다 훨씬 빠른 속도로 데이타의 양과 사용자의 요구가 증가하고 있다. 인터넷을 통한 전화, 화상회의, 원격 교육 등이 그것이다. 서버 시스템에서의 성능 향상 즉 라우터와 인터넷폰(VoIP) 전용 게이트웨이 등 실시간 전송을 보장해야 하는 멀티미디어 전용 서버의 연구가 현재 진행 중에 있고 또한 사용자의 요구를 충족시키기 위해 서버 시스템과 네트워크 대역폭이라는 두 가지 방향으로 연구가 진행되고 있다. 본 논문에서는 이러한 요구를 만족하기 위해 멀티미디어 데이타의 전송 지원을 위한 SRTPIO 모듈을 제안하였다. SRTPIO 모듈을 이용하여 실제 멀티미디어 데이타 생성과 전송은 커널 영역의 SRTPIO 모듈이 담당하게 되고, 애플리케이션은 휴지 상태로 들어갈 수 있도록 설계하였다. 따라서 커널 영역에서 사용자 영역으로의 데이타 복사가 필요하지 않고, 이를 수행하기 위한 멀티미디어 데이타 전송의 사용자 영역 프로세스를 스케줄링 할 필요가 없는 성능상의 향상이 있었다.

본 논문에서의 실험 결과에서 보듯이 리눅스 환경에

서 RTP 프로토콜을 사용자 영역 애플리케이션에서 수행하지 않고 커널 영역의 SIO 메커니즘과 함께 내장함으로써 기존의 리눅스 환경보다 전송 속도가 약 12.5%의 성능 향상이 있었다. 이는 인터넷 상황과 멀티미디어 데이타를 요구하는 각각의 서버 환경(VoIP 서버, 오디오서버, 비디오서버, 화상 회의용 서버 등)에 따라서 적절하게 응용할 수 있고 늘어나는 멀티미디어 데이타를 효과적으로 처리할 수 있게 되었다.

향후 연구 과제로는 제시한 SRTPIO 모듈 서버는 클라이언트와의 스트림 전송시 스트림 중에 연결을 잠시 중지시키지 못하는 점이 있다. 이러한 문제는 클라이언트의 애플리케이션에서 유지할 수도 있고, 식별자(Descriptor)를 두어 해결할 수도 있을 것이다. 현재 일반적인 서버 시스템의 네트워크 전송 단계는 기존의 텍스트 위주의 데이타를 처리할 수 있도록 제작되어 있어서 멀티미디어 데이타를 처리하기에는 적합하지 않다[9, 10]. 이러한 속도면에서 향상된 SRTPIO 모듈을 실시간 패킷에 우선 순위를 지원하는 스케줄러를 연구하여 합치면 애플리케이션 데이타 전송에 적합한 멀티미디어 서버 시스템의 핵심 기술로 활용할 수 있을 것이다.

참 고 문 헌

[1] Millind Buddhikot and Guru Parulkar, "Efficient Data Layout, Scheduling and Playout Control in MARS," ACM/Springer Multimedia Systems Journal, pp. 199-211, Volume 5, Number 3, 1997.

[2] *Millind Buddhikot, Guru Parulkar and* Gopalakrishnan, R., "Scalable Multimedia-On-Demand via World-Wide-Web (WWW) with QOS Guarantees," Sixth International Workshop on Network and Operating System Support for Digital Audio and Video, NOSSDAV96, Zushi, Japan, April 23-26, 1996.

[3] H. Schulzrinne and S. Casner, "RTP: the Real-time Transport Protocol," Audio-Video Transport Working Group, RFC 1889, January 1996.

[4] Jose Carlos Brustoloni, "Effects of Data Passing Semantics and Operating System Structure on Network I/O Performance," Ph.D. Dissertation, *Technical Report CMU-CS-97-176, School of* Computer Science, Carnegie Mellon University, September 1997.

[5] Moti N. Thadani and Yousef A. Khalidi, "An Efficient Zero-Copy I/O Framework for UNIX," Technical Report, SMLI TR-95-39, Sun Microsystems Lab, Inc., May 1995.

[6] Rosenberg, J. and Schulzrinne, H., "Timer Reconsideraion for Enhanced RTP Scahability," INFOCOM '98, Seventeenth Annual Joint Conference of the IEEE Computer and Communications Societies, Proceedings IEEE, pp. 233-241, Volume 1, 1998.

[7] Puneet Sharma, Deborah Estrin, Sally Floyd and Van Jacobson, "Scalable Timers for Soft State Protocols," INFOCOM '97, Sixteenth Annual Joint Conference of the IEEE Computer and Communications Societies, Driving the Information Revolution, Proceedings IEEE, pp. 222-229, Volume 1, 1997.

[8] Milind M. B., Dakang W., Guru M.P. and Xin J.C., "Enhancements to 4.4 BSD UNIX for Efficient Networked Multimedia in Project MARS," Multimedia Computing and Systems, Proceedings IEEE International Conference on, pp. 326-337, 1998.

[9] P. Druschel and L. L. Peterson. "Fbufs: A highbandwidth cross-domain transfer facility," In Proceedings of the Fourteenth ACM Symposium on Operating System Principles, pp. 189-202, Dec. 1993.

[10] Kevin Fall and Joseph Pasquale, "Improving Continuous-Media Playback Performance with In-kernel Data Paths," Proceedings of the International Conference on Multimedia Computing and Systems, May 14-19, Boston, Massachusetts. IEEE-CS, pp. 100-109, 1994.

[11] M. Beck, H. Bohme, M. Dziadzka, U. Junitz, R. Magnus and D. Verworner, "Linux Kernel Internals," 2nd Edition, Addison-Wesley, 1999.

[12] Richard M. S., Roland M. and Andrew O., "The GNU C Library Reference Manual," Edition 0.05, DRAFT last updated, 3 1993 for version 1.07 Beta.

[13] W. Richard Stevens, "TCP/IP Illustrated," Volume 3, Addison Wesley, April 1996.

[14] Daniel P. Bovet and Marco Cesati, "Understanding the Linux Kernel," O'Reilly, January 2001.

남 상 준

2000년 2월 한성대학교 컴퓨터공학과 공학사. 2000년 ~ 현재 고려대학교 컴퓨터학과 석사과정 재학중. 관심분야는 컴퓨터 네트워크, 실시간 시스템, 서버 QoS, 리눅스, 무선 네트워크 등.

이 병 래

1998년 2월 고려대학교 컴퓨터학과 학사. 2000년 2월 고려대학교 컴퓨터학과 석사. 2000년 ~ 현재 고려대학교 컴퓨터학과 박사과정 재학. 관심분야는 암호, 네트워크 보안, 이동통신

김 태 우

1984년 인하대학교 전자공학과 졸업. 1990년 인하대학교 전자계산학과 석사. 1996년 고려대학교 전산과학과 박사. 1984년 ~ 1986년 (주)금성사 컴퓨터사업부 시스템엔지니어. 1986년 ~ 1997년 시스템공학연구소 선임연구원. 1997년 ~ 현재 성공회대학교 컴퓨터정보공학부 부교수. 관심분야는 분산처리, 이기종컴퓨팅, 컴퓨터 네트워크, 유닉스 커널

김 태 윤

정보과학회논문지 : 정보통신
제 28 권 제 1 호 참조