ALMSock : 응용 계층 멀티캐스트 프로토콜의 개발 및 지원 프레임워크

이 영 희[†] · 이 중 수[††] · 이 경 용[†††]

요 약

IP 멀티캐스트 서비스 실현의 기술적, 비용적인 한계로 인하여 그에 대한 대안으로 응용 계층 멀티캐스트(Application Layer Multicast)가 대두되고 있다. 응용 계층 멀티캐스트 프로토콜은 현재까지 각 멀티캐스트 서비스의 요구사항에 맞게 다양하게 개발되어 왔으나, 지금까지의 연구는 각 응용 프로그램의 요구사항에 맞는 프로토콜의 디자인이나 효율적인 멀티캐스트 그룹 관리 기법에만 치중되어 왔을 뿐, 효율적인 프로토콜의 개발 방법에 관한 연구나 기존의 여러 프로토콜의 다중 사용에 관한 연구는 그 진척이 미비한 상황이었다. 본 논문에서는 새로운 응용 계층 멀티캐스트 프로토콜의 개발을 용이하게 하는 응용 계층 멀티캐스트 프로토콜의 개발을 위한 API를 제시하고, 한 시스템에서 여러 개의 응용 계층 멀티캐스트 프로토콜의 다중 동작과 관리를 용이하게 하는 프로토콜 통합 관리를 위한 API와 프레임워크를 제안한다.

ALMSock : A Framework for Application Layer Multicast Protocols

Younghee Lee[†] · Joongsoo Lee[††] · Kyoungyong Lee[†††]

ABSTRACT

Due to the deployment problem of the IP Multicast service, the Application Layer Multicast (or Overlay Multicast) has appeared as an alternative of the IP Multicast. However, even though plenty of the Application Layer Multicast (ALM) Protocols were designed and their applications were developed according to the diverse requirements of each multicast service, researches on the ALM Protocols are focused on only a protocol design or an efficient multicast group management algorithm. And there is little effort to provide a unified guideline for development of the ALM Protocols and provide an environment for running multiple protocols simultaneously in a system. In this paper, we propose socket APIs to be a reference in developing new ALM Protocols which enables a system to support multiple protocols in a system with other ALM Protocols and which gives an environment to support efficient protocol management.

키워드 : 멀티캐스트(Multicast), 오버레이 멀티캐스트(Overlay Multicast), 응용 계층 멀티캐스트(Application Layer Multicast), API

1. 서 론

IP 멀티캐스트 서비스 실현의 기술적, 비용적인 문제[1]로 인하여 네트워크 계층(Network Layer)이 아닌 응용 계층(Application Layer)에서 end-host들 간에 유니캐스트 방식을 이용하여 멀티캐스트 서비스를 제공하는 응용 계층 멀티캐스트(Application Layer Multicast 혹은 Overlay Multicast)가 그 대안으로 대두되고 있다. 응용 계층 멀티캐스트는 그 이름이 제시하는 바와 같이 네트워크 내의 end-host가 응용 계층에서 멀티캐스트 기능을 수행하므로, 그 동안 IP 멀티캐스트 서비스 실현의 장애물이 돼 왔던 라우터 교체와 같은 Network Infrastructure의 변화 없이도 멀티캐스트 서비스를 제공할 수 있다는 장점이 있다. 또한 모든 멀티캐스트 기능이 end-host의 응용 계층에 구현되어 있기 때문에 인터넷의 end-to-end 개념을 유지할 수 있다는 장점 역시 지니고 있다.

응용 계층 멀티캐스트 프로토콜은 현재까지 그 설계나 구현에 대한 표준이 정해져 있지 않고, 멀티캐스트 서비스에 대한 다양한 요구사항 즉, 파일 전송시의 대역폭 혹은 온라인 게임이나 실시간 멀티미디어 데이터 전송시의 전송 지연 등으로 인한 프로토콜 개발의 복잡함, 어려움에도 불구하고 여러 응용 계층 멀티캐스트 프로토콜이 어떤 최소한의 표준도 없이 각 응용 프로그램의 요구사항에 맞게 독립적으로 개발되어 왔다. 이는 새로운 응용 계층 멀티캐스트 프로토콜의 개발과 각 프로토콜의 성능 비교 또한 어렵게 하고 있다. 또한 응용 계층 멀티캐스트 프로토콜은 IP Multicast에 비해 기술적, 경제적인 이점을 가짐에도 불구하고 그 성

※ 본 연구는 한국과학재단 특정기초연구(R01-2003-000-10562-0)와 정보통신연구진흥원의 정보통신 선도기술개발사업(A1100-0401-0114, 웹 서비스 기반 미들웨어 기술 개발)의 지원으로 진행되었음.
† 종신회원 : 한국정보통신대학교 교수
†† 준 회 원 : 한국정보통신대학교 대학원
††† 준 회 원 : 한국정보통신대학교 대학원
논문접수 : 2004년 1월 30일, 심사완료 : 2004년 6월 14일

능이 IP Multicast에 미치지 못한다는 문제점을 가지고 있다[2]. 이는 응용 계층 멀티캐스트 프로토콜이 멀티캐스트 패킷 전송 시 같은 링크에 대해서 동일한 데이터 패킷을 여러 번 중복 전송함으로 인한 호스트의 패킷 처리 부담이 발생하기 때문이다.

응용 계층 멀티캐스트 프로토콜은 멀티캐스트 네트워크 내의 한 멤버 호스트가 여러 멀티캐스트 세션에 가입하고 있는 경우를 가정했을 경우, 시스템 관리자는 시스템에서 독립적으로 동작하고 있는 각 프로토콜을 효율적으로 관리해 줄 수 있는 Framework이 필요한 현실이다.

본 논문에서는 새로운 응용 계층 멀티캐스트 프로토콜의 개발 시 지침이 될 공통 API(Common Application Program Interface)를 제안함과 동시에 다양한 형태를 지닌 기존의 응용 계층 멀티캐스트 프로토콜이 어느 시스템에 장착되더라도 안정적으로 동작할 수 있게 해주고 효율적인 프로토콜 관리를 도와주는 Framework을 제안한다.

본 논문의 구성은 다음과 같다. 2장에서는 본 연구와 관련된 관련 연구에 대해 살펴보고, 3장에서는 본 논문에서 제안된 Framework인 ALMSock의 구조와 각각의 특징에 대해서 살펴본다. 4장에서는 본 논문의 ALMSock API를 사용한 간단한 예제와 ALMSock에 탑재된 Forwarding Engine의 성능에 대해 살펴보고 그 결과에 대해 분석한다. 마지막으로 5장에서는 결론과 향후 연구방향을 제시한다.

2. 관련 연구

네트워크 내의 시스템을 오버레이(overlay)하여 논리적인 네트워크를 구성하는 오버레이 네트워크의 프로토콜 개발에 관한 문제에 대해서는 이미 몇몇 연구가 진행되어 왔다.

특히 동일 목적을 가진 여러 프로토콜의 공통 API를 정의한 [4]의 연구에서는 Chord[11], CAN[9], Bayeux[12]와 같이 파일 공유와 저장(File Sharing and Storage) 혹은 Location Lookup과 라우팅을 위한 여러 *구조화된 Peer-to-Peer 오버레이 프로토콜*(Structured Peer-to-Peer Overlay)로부터 각 프로토콜의 공통적인 특성을 DHT(Distributed Hash Table), CAST(Group multicast and Anycast), DOLR(Decentralized Object Location & Routing) 등의 3가지 특성으로 추상화하고, 이 세 가지 특성을 다시 KBR(Key Based Routing) 기반으로 추상화하는 3단계의 추상화 과정을 거쳐 각 구조화된 Peer-to-Peer 시스템을 위한 오버레이 프로토콜 개발에 사용될 수 있는 공통 API를 KBR 기반으로 재정의하였다.

또한, 오버레이 멀티캐스트 프로토콜 기반의 응용 프로그램 개발을 위한 API를 정의하여 제공한 [5]의 연구에서는 Hypercast 2.0[6, 7]을 기반으로 하는 오버레이 멀티캐스트 응용 프로그램 개발을 위한 API인 overlay socket을 제공하며 overlay socket을 이용한 오버레이 멀티캐스트 네트워크 프로그래밍에 대하여 논의하고 있다. Overlay socket은 복잡한 오버레이 멀티캐스트 프로토콜의 하부 구조에 대한 지식이 없이도 오버레이 멀티캐스트 응용 프로그램을 작성할 수 있는 overlay socket API를 기반으로 하며, 이 API를 응용하여 Hypercast 2.0 프로토콜 기반의 멀티캐스트 응용 프로그램뿐만 아니라 다른 오버레이 멀티캐스트 프로토콜의 응용 프로그램 작성에도 응용할 수 있는 환경을 제공하였다. [15]의 연구에서는 Relayed Multicast를 위한 control protocol의 framework을 제안하였다. 특히 멀티캐스트 데이터 전달을 제어하는 Session Manager라는 특별한 목적의 모듈을 framework에 탑재하여 IP 멀티캐스트 서비스가 완벽히 실현되지 않은 현재의 인터넷 환경에 적합한 데이터 전달 과정을 제시하였다.

패킷 처리 지연의 최소화하는 효율적인 데이터 패킷 전송 방법을 제안하고 평가한 [3]의 연구에서는, 한 호스트 내의 장치(device)끼리 멀티미디어 데이터와 같은 대용량 데이터를 빠른 전송률로 전송 시 생기는 데이터 복사(Data Copy) 혹은 문맥교환(Context Switching)을 입출력 시 발생하는 비효율성의 원인이라 정의하고, 데이터 복사 회피(Data Copy Avoidance)와 데이터 전달 회피(Data Passing Avoidance) 기술을 제안하여 효율적인 데이터 전달 방법을 제안하였다. 특히 사용자 영역과 커널 영역의 불필요한 데이터 복사를 피하는 방법으로 커널 영역 버퍼링을 제안하여 호스트의 데이터를 보다 효율적으로 전송할 수 있는 방법을 제안하였다.

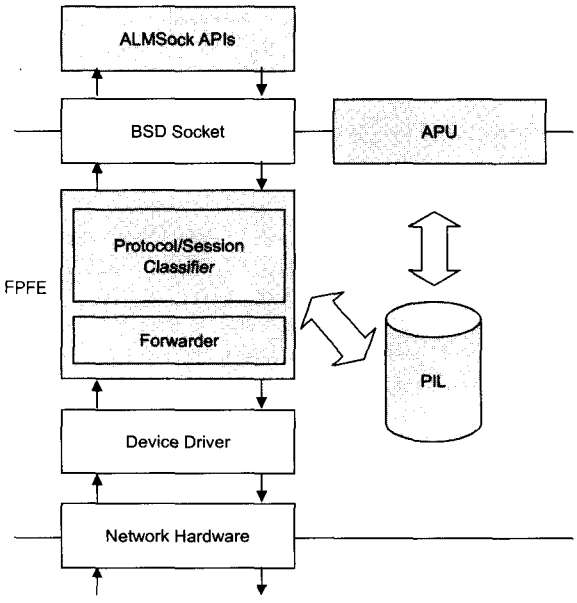
3. ALMSock의 구조

3.1 ALMSock의 구조

본 논문에서 제안된 Framework인 ALMSock에서는 응용 계층 멀티캐스트 프로토콜 개발을 위한 소켓 API(Socket API)를 지원함과 동시에 다중 프로토콜 지원을 위한 API를 제공한다. ALMSock API는 버클리 타입의 소켓(BSD Socket)을 기반으로 함으로써 각 프로토콜을 사용하여 동작하는 응용 프로그램의 요구사항에 맞게 수정될 수 있고 다중 프로토콜 지원을 위해 변경될 수 있는 유연성을 지녔다. 또한 각 응용 계층 멀티캐스트 프로토콜의 기본 정보와 라우팅 정보가 기술되어 있는 PIL(Protocol Information List)과 PIL에 기술되어 있는 각 프로토콜 정보의 수정을 위하여 APU(API for PIL Update)을 두어 한 시스템에서 여러 프로토콜이 함께 동작할 수 있는 다중 프로토콜 지원을 위함과 동시에 효율적인 프로토콜의 관리를 위한 장치를 하였다. 또한 응용 계층 멀티캐스트 프로토콜의 단점 중 하나로 지적되어 왔던 응용 계층(Application Layer)에서의 멀티캐스트 패킷 전송으로 인한 호스트 내의 패킷 처리 지연을 해결하기 위해 커널 영역에 FPFE(Fast Packet Forwarding

Engine, 빠른 패킷 전달 모듈)을 장착하여 호스트 내에서 패킷 처리 시간을 최소화하였고, 각 프로토콜은 FPFE를 공유하여 사용하는 형태를 지닌다.

ALMSock이 동작하는 전체 구조는 (그림 1)과 같다. 호스트가 패킷을 수신하면 FPFE는 패킷이 사용하고 있는 프로토콜 종류와 멀티캐스트 세션 정보를 PIL(Protocol Information List)의 정보를 바탕으로 구분한 후, PIL에 기술되어 있는 라우팅 정보를 이용하여 패킷을 다음 도착지로 멀티캐스트 하거나 호스트 내의 해당 응용 프로그램으로 전달한다. 프로토콜의 정보와 라우팅 정보를 가지고 있는 PIL은 호스트에 수신된 제어 패킷에 의해 APU(API for PIL Update)를 이용하여 갱신된다.



(그림 1) ALMSock의 동작 개요

ALMSock을 구성하는 각 구성 요소의 개괄적인 기능은 다음과 같다.

① **ALMSockAPI** : 새로운 응용 계층 멀티캐스트 프로토콜 개발을 위한 API로 데이터 전송, 프로토콜 관리, 멀티캐스트 그룹 관리 등을 위한 API를 제공하며, 버클리 타입의 소켓을 기반으로 하여 작성되었다. 응용 계층 멀티캐스트 프로토콜 개발자들은 ALMSock API를 이용하여 프로토콜을 개발 또는 기존의 프로토콜을 수정할 수 있다.

② **PIL(Protocol Information List)** : ALMSock Framework에 탑재되는 일종의 *Routing Table*로써 응용 계층 멀티캐스트 네트워크의 라우팅 정보를 저장하고 있으며, 다중 프로토콜 지원을 위하여 호스트에서 동작중인 여러 프로토콜의 정보가 기술되어 있다.

③ **APU (API for PIL Update)** : ALMSock Framework이 다중 프로토콜 지원을 위하여 필요한 PIL의 정보 변경을 위한 API로서, 제어 패킷(control packet) 수신 시 PIL에 기술되어 있는 프로토콜의 정보를 변경한다.

④ **FPFE(Fast Packet Forwarding Engine)** : ALMSock Framework에 탑재되는 *빠른 패킷 전달 모듈(packet forwarding engine)*로써 기존의 응용 계층 멀티캐스트 프로토콜에서 사용된 응용 계층의 패킷 전달 모듈을 커널 영역에 탑재하여 호스트 내의 패킷 처리 시간을 최소화 한 효율적인 패킷 전달을 담당한다.

3.2 ALMSock API

3.2.1 API 정의를 위한 고려사항 및 접근방법

ALMSock API는 기존의 여러 프로토콜에 적용 시 적용 범위를 넓히기 위하여 단계별 구체화 과정을 거쳐 정의되었으며, 기본적으로 다음과 같은 사항이 고려된다.

- 기존의 응용 계층 멀티캐스트 프로토콜의 특성을 고려하여 기존 프로토콜을 수용할 수 있는 형태를 지녀야 한다.
- 데이터 전송 시 UDP/TCP 전송 모두를 지원할 수 있는 형태를 지녀야 한다.
- 프로토콜 정보 변경을 위한 제어 패킷(control packet)은 멀티캐스트 전송이 아닌 유니캐스트로 전송한다.
- forward, multicast, broadcast 등과 같은 기능들은 3.5절에서 설명할 *빠른 패킷 전달 모듈*의 기능이므로 본 절에서는 제외한다.

또한, API를 정의하기 위하여 Hypercast[6, 7], ALMI[8], YOID[10] 등 기존 프로토콜의 구현 내용을 분석한 결과를 바탕으로, 응용 계층 멀티캐스트 프로토콜의 기능을 구체화 과정을 거쳐 각 프로토콜의 공통된 API를 추출하게 된다. 이 결과로 각 프로토콜의 기능은 기본적으로 다음과 같은 3가지 부분으로 1차 구체화 될 수 있다.

- **CAST** : Data Transfer
- **GMGT** : Group Management
- **PMGT** : Protocol Management

CAST는 *send, receive, multicast, broadcast* 등 데이터의 전달을 담당하기 위한 부분이고, GMGT는 *join/leave group, join request/reply* 등의 멀티캐스트 그룹 관리를 담당하며, 마지막으로 PMGT는 멀티캐스트 네트워크 내의 각 호스트들이 가지고 있는 라우팅 정보의 관리를 담당하는 부분이다.

다음 과정으로 위의 3가지 추상적 특성들을 보다 구체화하여 2차 특성을 정의하게 되었으며, 그 특성을 바탕으로 다시 3차 특성이 정의되었고, 다시 이 3차 특성을 바탕으로 API가 정의되었다. 정의된 특성과 각 구체화 단계는 다음의 <표 1>과 같다.

〈표 1〉 API 정의를 위한 추상화 및 구체화 단계

Level 0	Level 1	Level 2
CAST	socket send receive	socket open, close, bind, connect, accept send UDP, TCP packet receive UDP, TCP
GMGT	join group leave group poll	join request, join reply join, leave get join request, get join reply poll(target node)
PMGT	update list	update protocol information

3.2.2 응용 계층 멀티캐스트 프로토콜 개발을 위한 API

<표 1>의 Level 2로부터 파생된 응용 계층 멀티캐스트 프로토콜 개발을 위한 API인 ALMSock API의 프로토타입은 부록에 기술되어 있다. 프로토콜 설계자는 ALMSock API를 이용하여 프로토콜의 각 기능들을 정의할 수 있으며 부가적인 기능을 가감할 수 있다.

3.3 Protocol Information List(PIL)

Protocol Information List(PIL)는 다중 프로토콜 지원을 위하여 한 시스템에서 동작하고 있는 여러 응용 계층 멀티캐스트 프로토콜의 정보를 저장하고 있는 일종의 라우팅 테이블(routing table)이다. 호스트에 수신된 패킷은 패킷이 사용하고 있는 프로토콜 분류를 위해 PIL을 참조한다. PIL 내의 프로토콜 정보는 멀티캐스트 네트워크 내의 제어 패킷(control packet) 등에 의해 변경되고, 다음의 (그림 2)과 같은 자료구조를 가진다.

```
struct ProtocolInformationList
{
        struct ProtocolInformationList *prev ;
        struct ProtocolInformationList *next ;

        int groupID ; /* group ID */
        char *name ; /* Protocol Name */
        int pf ; /* Protocol Family, tcp or udp */

        int dataPort ; /* Port Number for Data Packet */
        int controlPort ; /* Port Number for Control Packet */

        struct Address rootIPAddress ; /* root IP Address */
        struct Address parentIPAddress ; /* parent IP Address */
        struct Address childIPAddress ; /* children IP Address */

        int num_fo ; /* Number of Fan-Out */
}

struct Address /* Data Structure for addresses */
{
        struct Address *prev ;
        struct Address *next ;

        u_int32_t address ;
}
```

(그림 2) PIL의 자료구조

3.4 프로토콜 정보 변경을 위한 API(API for PIL Update, APU)

ALMSock에서는 멀티캐스트 멤버의 가입/탈퇴 등의 이유로 인한 멀티캐스트 네트워크의 토폴로지 변화에 대처하고 호스트의 다중 프로토콜 지원을 위하여 프로토콜 정보의 갱신을 위한 API인 APU를 제공한다. APU는 프로토콜의 정보 변경, 예를 들어 사용하는 포트 번호의 변경, root의 IP 주소 변경, routing 정보 변경 등의 상황 발생시 수신된 control packet에 의해 호출되며 PIL의 각 field를 변경한 후에 변경 후의 내용을 시스템에 적용시킨다. 프로토콜 정보 변경을 위한 API는 다음의 (그림 3)과 같이 정의된다.

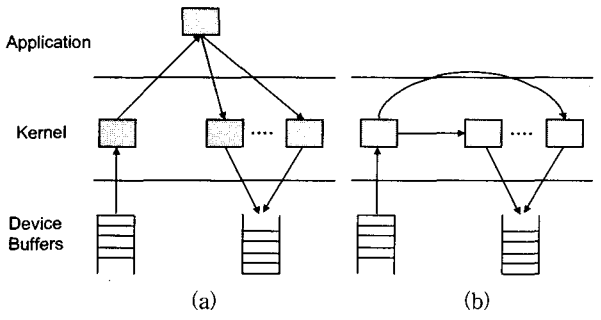
```
/* update IP Address of target
 * gid is a Group ID
 * target may be root, parent, children
 */
int update_address(int gid, int target, struct Address new_address);

/* update port number of target
 * target may be data and control
 */
int update_port(int target, int new_port);
```

(그림 3) 프로토콜 정보 변경을 위한 API의 프로토타입

3.5 빠른 패킷 전달 모듈(Fast Packet Forwarding Engine, FPFE)

ALM 네트워크 내의 멤버 호스트에서 패킷을 멀티캐스트 트리(tree) 혹은 메쉬(mesh) 상의 자식 노드(child node)에게 전달해주는 역할을 하는 패킷 전달 모듈(Packet Forwarding Engine, FPFE)은 응용 계층 멀티캐스트라는 이름이 말해주듯이 일반적으로 응용 계층(사용자 영역, user space)에 존재한다. 하지만 [3]에서 기술된 바와 같이 커널 영역(kernel space)에서의 데이터 버퍼링(buffering)은 *커널 영역 버퍼*와 *사용자 영역 버퍼* 사이의 불필요한 복사 부담(copy overhead)을 줄여줄 수 있다. (그림 4)에서 볼 수 있듯이 사용자 영역에서의 패킷 버퍼링은 사용자 영역과 커널 영역 사이의 불필요한 복사 부담을 초래하고 이는 호스트의 패킷 처리 지연(packet processing delay)을 유발한다.



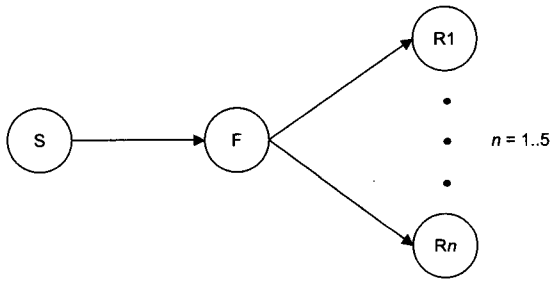
(그림 4) (a) 사용자 영역 버퍼링과 (b) 커널 영역 버퍼링

이러한 이유로 ALMSock에서는 사용자 영역에 위치하는

기존의 패킷 전달 모듈을 커널 영역에 위치시켜 커널 영역과 사용자 영역 사이의 복사 수를 줄임으로써 호스트 내의 패킷 처리 시간을 감소시킨다. 이것은 기존의 패킷 전달 모듈에 비해 향상된 성능을 제공한다. 또한 패킷 전달의 향상된 성능은 전체 네트워크의 성능 향상에도 영향을 미칠 것으로 기대된다.

3.5.1 빠른 패킷 전달 모듈(FPFE)의 성능

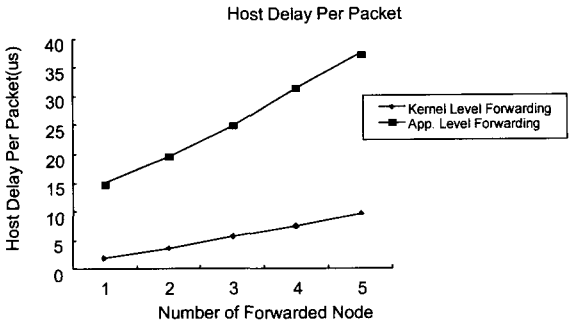
ALMSock Framework에 탑재된 패킷 전달 모듈은 사용자 영역이 아닌 커널 영역에서의 패킷을 전달하는 구조를 지녀 패킷당 처리 시간의 향상을 꾀하였다. 이를 알아보기 위해 (그림 5)과 같은 응용 계층 멀티캐스트 네트워크 환경을 구성하고 패킷을 전달할 노드(fan-out)의 개수에 따른 *사용자 영역 패킷 전달 모듈*과 본 논문에서 제안된 *커널 영역 패킷 전달 모듈*의 패킷당 처리 시간을 측정, 비교하였다. 또한 응용 계층 멀티캐스트 프로토콜의 특성상 멀티캐스트 노드가 전용 라우터가 아닌 범용 컴퓨터임을 감안하여 노드에 일정량의 CPU 부하를 가하여 실험하였다.



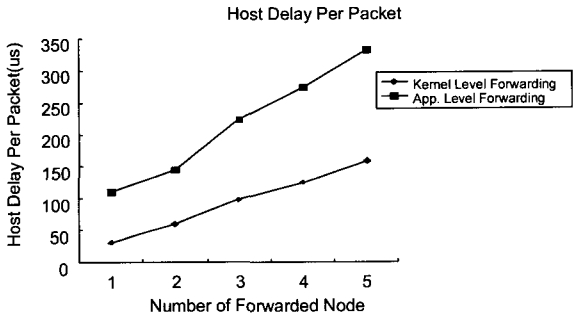
(그림 5) 실험에 사용된 멀티캐스트 그룹의 논리적 구성

(그림 5)의 노드 S는 패킷을 전송하는 송신 노드(Sending Node), F는 S로부터 수신된 패킷을 R_1~R_n에게 멀티캐스트 해주는 전달 노드(Forwarding Node)를 나타내며 R_1~R_n은 패킷을 수신하는 수신 노드(Receiving Node)를 나타낸다. 본 논문에서는 ALMSock Framework의 패킷 전달 모듈의 성능을 측정하기 위하여, F에서의 패킷 처리 시간을 기존의 *사용자 영역 패킷 전달 모듈*을 사용했을 때의 패킷 처리 시간과, 본 논문에서 제안된 *커널 영역 패킷 전달 모듈*을 사용했을 때의 패킷 처리 시간을 비교하였다. 또한 수신 노드인 R_n의 개수가 늘어날 때의 패킷 처리 시간의 변화를 측정하였고, 응용 계층 멀티캐스트 그룹의 멤버 노드가 멀티캐스트 전용 시스템이 아닌 범용 시스템인 점과 시스템 내에 다른 응용 프로그램이 동작하고 있을 수 있는 환경을 감안하여, 호스트 내의 CPU 점유율을 달리한 비교도 함께 수행하였다. 본 실험의 결과는 (그림 6), (그림 7)와 같다.

(그림 6)과 (그림 7)는 멀티캐스트 그룹 내의 호스트가 하나의 패킷을 멀티캐스트할 때 수신노드가 증가함에 따라 걸리는 시간의 변화를 보여주고 있다. (그림 6)에서는 호스트가 MSN의 사용시와 같이 멀티캐스트 이외의 다른 작업을 수행하지 않을 경우의 멀티캐스트 패킷 처리 시간을 보여주며 (그림 7)에서는 호스트가 멀티캐스트와 다른 작업을 병행했을 때의 멀티캐스트 시간을 보여주고 있다. (그림 6)과 (그림 7)에서 볼 수 있듯이 ALMSock Framework의 *커널 영역 패킷 전달 모듈*을 사용했을 경우 기존의 *사용자 영역 패킷 전달 모듈*을 사용했을 때와 비교하여 패킷당 처리 시간이 눈에 띄게 감소하는 것을 알 수 있다.



(그림 6) 패킷당 처리 시간



(그림 7) 패킷당 처리 시간(90% CPU 점유율)

3.5.2 문제점 및 해결책

ALMSock Framework에서는 기존의 여러 응용 계층 멀티캐스트 프로토콜을 지원하기 위한 방법의 하나로, 응용 계층에 존재하는 각각의 패킷 전달 모듈을 커널 영역으로 이동시켜 각 멀티캐스트 프로토콜들이 패킷 전달 모듈을 공유하는 형태로 설계되었다. 이러한 이유로 패킷 전달 모듈은 각 패킷이 사용하고 있는 멀티캐스트 프로토콜의 종류와 세션을 구분할 능력을 가져야만 하는 문제점을 지닌다. 이러한 문제점들을 해결하기 위하여, ALMSock Framework에 탑재된 패킷 전달 모듈은 FPFE 내에 프로토콜 분류기(protocol classifier)를 탑재하고, 패킷이 사용하고 있는 포토 번호와 세션 번호 등의 정보를 사용하여 각 프로토콜의 종류와 세션 번호를 인식하게 된다.

4. 프로토콜 개발 방법 및 평가

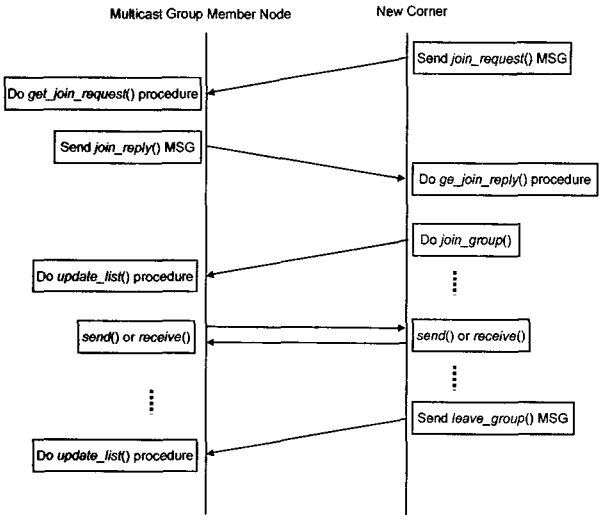
4.1 프로토콜 개발을 위한 API의 사용 예

ALMSock API를 이용한 응용 계층 멀티캐스트 프로토콜

개발 방법에 관한 간단한 예는 다음의 (그림 8)과 같다.

멀티캐스트 그룹에 가입을 원하는 새로운 노드는 *join_request* 메시지를 멀티캐스트 그룹에 보낸다. *join_request* 메시지를 받은 멀티캐스트 그룹의 멤버 노드가 가입의 가/부에 관한 메시지인 *join_reply* 메시지를 새로운 노드에게 전송하면 새로운 노드는 *join_reply* 받아 그룹에 가입을 시도한다. 새로운 노드가 멀티캐스트 그룹에 가입이 완료되면 멀티캐스트 그룹의 멤버 노드 혹은 루트 노드는 PIL의 멀티캐스트 트리에 관한 주소 정보 등을 갱신하고 멀티캐스트 그룹에 속한 전체 노드에게 PIL 정보를 전송한다. 새로운 노드의 멀티캐스트 그룹 가입이 완료된 후 멀티캐스트 그룹은 *send*() 혹은 *receive*()를 이용하여 멀티캐스트 데이터를 송신 혹은 수신한다.

만약 멀티캐스트 그룹에서 탈퇴하고 싶은 노드가 있을 경우, 그 노드는 *leave_group* 메시지를 멀티캐스트 그룹에 보내며, *leave_group* 메시지를 받은 멀티캐스트 그룹의 멤버 노드 혹은 루트 노드는 PIL의 멀티캐스트 트리에 관한 주소 정보를 갱신 후 다시 PIL을 멀티캐스트 그룹에 속한 전체 노드에게 전송한다.



(그림 8) 프로토콜의 개발 절차의 예시

4.2 API의 평가

본 논문에서 제안된 ALMSock API의 평가를 위해 ALMSock API를 이용한 프로토콜 개발의 간단한 예제와 ALMSock API가 기존의 응용 계층 멀티캐스트 프로토콜인 YOID와 ALMI에의 적용성을 평가하였다. ALMSock API를 이용한 프로토콜 개발의 예제는 (그림 9)과 같다.

```
#define NumOfTrial 5 /* Number of trials to join a multicast group */

int *gid ;
int *dataPort, *controlPort; /* data and control port */
int *pf ; /* Protocol Family */
int *type ; /* Transmission Type */
struct Address *rootIPAddr ;
int *numFO ; /* Number of Fan-Out */
struct Address *nextNode ;
int result = 0 ;
int i = 0 ;

result = almsock_init() ; /* initialize a global variables using PIL */
if(result < 0){ /* if this function fails to load protocol
                   information */
        printf("Fail to Load Protocol") ;
        exit(result) ;
}
loadPIL() ; /* load basic protocol information from PIL */
while(!(result = get_join_reply())){ /* when a node tries to join
                                        multicast group */
        i++ ;
        if(i == NumOfTrial){
                printf("Cannot join to Multicast group!!") ;
                exit(-1) ;
        join_request(gid) ;
      wait(time) ;
}
if(get_join_request()){ /* when a node in a mcast group receives a
                           join_request message */
      calc_position() ;
      update_list() ;
      join_reply(newcomer) ;
}
if((result = send(data, AF_INET, dest)) < 0 ){ /* send data to
                                                  multicast group */
      printf("fail to send data\n") ;
      exit(-1) ;
}
result = receive(buffer, buf_length) ; /* receive data from a multicast
                                          group */
```

(그림 9) ALMSock API를 이용한 프로토콜 개발의 예제 (simplified)

또한, ALMSock API의 평가를 위해 ALMSock API의 기존 프로토콜에 대한 적용성을 평가하기 위하여 <표 2>에서 볼 수 있듯이 여러 레벨의 구체화 과정을 통하여 정의된 ALMSock API를 기존의 프로토콜에 적용해 본 결과 ALMI나 YOID 등의 기존 프로토콜의 대부분의 기능에 적용 가능하였다.

〈표 2〉 ALMSock의 적용(GMGT)

ALMSock	YOID	ALMI
join_request	ytmp_proto_join_send	AlmiSession.send
get_join_request	ytmp_proto_join_recv	AlmiSession.recv
join_reply	ytmp_proto_join_process	AlmiSession.send
get_join_reply	ytmp_proto_join_recv	AlmiSession.recvJoin
join_group	ytmp_proto_notification_join_process	AlmiSession.recvJoin
leave_group	ytmp_proto_leave_send ytmp_proto_leave_recv	AlmiSession.recvLeave
poll	ytmp_proto_keep_alive_send ytmp_proto_keep_alive_recv	AlmiSession.recvSrcQuery

<표 2>와 같이 ALMSock API의 각 인터페이스 함수들은 YOID와 ALMI에 적용되기에 충분하다는 것을 알 수 있고 여타의 다른 응용 계층 멀티캐스트에도 적용이 가능할 것이다.

하지만 GMGT의 경우에는 응용 프로그램의 요구사항에 따라 다양한 멀티캐스트 그룹 관리 알고리즘이 존재하기 때문에 ALMSock API는 각 프로토콜의 모든 요구사항을 만족시키지는 못한다. 그렇기 때문에 ALMSock Framework에서는 새로운 ALM 프로토콜 개발에 쓰일 수 있는 프로토타입 (prototype)을 제공하여 프로토콜 개발 시 변경 혹은 확장 사용할 수 있는 유연성을 제공한다.

CAST의 경우에는 multicast나 forward와 같은 기능은 FPFE에서 이미 제공을 하기 때문에 ALMSock API에서는 send나 receive와 같은 함수를 제공함으로써 대부분의 기존 프로토콜에 적용 가능하다는 것을 알 수 있다.

마지막으로 PMGT의 경우에는 라우팅 정보와 같은 프로토콜 관련 정보를 PIL에서 관리를 하며, ALMSock API의 *update_PIL*()에서 PIL의 정보를 수정할 수 있는 기능을 제공함으로써 ALMSock API가 기존 프로토콜의 대부분에 적용 가능하다고 말할 수 있다.

5. 결론 및 향후 연구

본 논문에서는 새로운 응용 계층 멀티캐스트 프로토콜의 개발과 기존의 응용 계층 멀티캐스트 프로토콜이 하나의 시스템에서 동작할 수 있게 해주는 Framework을 제공함으로써 프로토콜 개발의 편의성을 제공함과 동시에 각 응용 프로그램의 요구사항에 맞게 설계, 구현된 기존의 응용 계층 멀티캐스트 프로토콜 혹은 오버레이 멀티캐스트 프로토콜이 하나의 시스템에서 동작할 수 있는 환경을 제안하였다. 또한 빠른 패킷 전달 모듈을 커널 영역에 탑재하여 IP 멀티캐스트에 비해 응용 계층 멀티캐스트 프로토콜의 단점으로 지적된 비효율성을 보완하였으며, 패킷 전달 모듈의 공유로 인하여 발생하는 문제점인 프로토콜 구분 문제는 프로토콜 정보 저장과 라우팅을 위한 테이블인 PIL의 사용으로 해결하였다.

ALMSock Framework에는 응용 계층 멀티캐스트 프로토콜의 개발을 위한 모든 API가 정의돼있지는 않다. 이는 멀티캐스트 프로토콜의 사용 목적이나 사용자의 요구사항 등이 다양하여 각 요구사항에 따라 멀티캐스트 프로토콜의 개발 방법이 달라지기 때문이다. 하지만 본 논문에서 제안하고 정의한 응용 계층 멀티캐스트 프로토콜의 공통 분모인 API를 이용하였을 경우 보다 구조화 된 프로토콜의 개발과 개발된 프로그램의 가독성을 높여 다른 목적으로 개발될 프로토콜로의 용이한 이전도 기대되므로 본 논문에서 제안한 ALMSock API와 Framework을 사용하여 향후 개발될 응용 계층 멀티캐스트 프로토콜에의 적용을 기대한다.

ALMSock Framework의 패킷 전달 모듈은 실시간 데이터 전송시의 패킷 처리 시간의 향상 등과 같이 네트워크의 전송 지연과 멀티캐스트 기능을 수행하는 멀티캐스트 멤버 호스트의 부하를 줄이는 부분에 초점을 맞추었기 때문에 네트워크 성능에 영향을 미치는 또 다른 변수인 대역폭(Bandwidth)에 대한 고려는 그 대상에서 제외되었고, 선입선출(FIFO)의 자료구조만을 사용하여 데이터 전송의 우선순위(priority)에 관한 고려도 배제되었다. 향후 연구에서는 네트워크의 전송 지연뿐만이 아니라 대역폭 할당 등과 같이 네트워크의 성능에 영향을 미치는 여러 변수뿐만이 아니라 데이터 전송의 우선순위 역시 고려한 Framework과 패킷 전달 모듈의 설계가 필요하다.

참 고 문 헌

[1] Diot, C., Levine, B. N., Lyles, B., Kassem, H. and Balensiefen, D., "Deployment issues for the IP multicast service and architecture," *Journal of Network, IEEE*, Vol.14, Issue 1, pp. 78-88, Jan.-Feb., 2000.

[2] Banerjee S. and Bhattacharjee B., "A Comparative Study of Application Layer Multicast Protocols," in *Submitted for Review*, 2002.

[3] Brustoloni, J. C. and Steenkiste P., "Evaluation of Data Passing and Scheduling Avoidance," in *Proceedings of the IEEE 7th International Workshop on Network and Operating System Support for Digital Audio and Video*, pp.95-105, May, 1997.

[4] Dabek F., Zhang B., Druschel P., Kubiatowicz J. and Stoica I., "Towards a Common API for Structured Peer-to-Peer Overlays," in *Proceedings of the 2nd International Workshop on Peer-to-Peer Systems (IPTPS '03)*, Berkeley, CA, February 2003.

[5] Liebeherr J., Wang J. and Zhang G., "Programming Overlay Networks with Overlay Sockets," in *Proceedings of 5th International Workshop on Networked Group Communications (NGC 2003)*, Munich, Germany, pp.242-253, Sep., 2003.

[6] *The Hypercast Project*, http://www.cs.virginia.edu/~hypercast.

[7] Liebeherr J. and Beam T. K., "Hypercast : A protocol for maintaining multicast group members in a logical hypercube topology," In *Proceedings of First International Workshop on Networked Group Communications (NGC 99)*, In Lecture Nodes in Computer Science, Pisa, Italy, Vol.1736, pp.72-89, November, 1999.

[8] Pendarakis D., Shi S., Verma D. and Waldvogel M., "ALMI :

An application level multicast infrastructure," In *Proceedings of 3rd Usenix Symposium on Internet Technologies and Systems*, San Francisco, CA, pp.49-60, March, 2001.

[9] Rantnasamy S., Francis P., Handley M., Karp R. and Shenker S., "A Scalable Content-Addressable Network," In *Proceedings of ACM SIGCOMM*, San Diego, CA, pp.161-172, August, 2001.

[10] *The Yoid Project*, http://www.icir.org/yoid/.

[11] Stoica I., Morris R., Karger D., Kaashoek F. and Balakrishnan H., "Chord : A Scalable Peer-to-Peer Lookup Service for Internet Applications," *In Proceedings of ACMM SIGCOMM*, San Diego, CA, pp.149-160, August, 2001.

[12] Zhuang S. Q., Zhao Y., Joseph A. D., Katz R. H. and Kubiatowicz J., "Bayux : An Architecture for Scalable and Fault-tolerant Wide-Area Data Dissemination," In *Proceedings of the Eleventh International Workshop on Network and Operating System Support for Digital Audio and Video, (NOSSDAV 2001)*, Port Jefferson, NY, pp.11-20, January, 2001.

[13] Shi S. and Turner J., "Routing in overlay multicast networks," In *Proceedings of Infocom*, June, 2002.

[14] Yair Amir and Claudiu Danilov, "Reliable Communication in Overlay Networks," IEEE DSN, 2003,

[15] Koh, S. et al., "Framework of Control Protocol for Relayed Multicast," LNCS 2713, pp.576-581, June, 2003.

부록 : Specification of an ALMSock APIs

NAME

almsock_init

SYNOPSIS

int almsock_init(void)

DESCRIPTION

Initialize socket related variables in the ALMSock Framework

NAME

almsock_open

SYNOPSIS

int almsock_open(int domain, int type, int protocol)

DESCRIPTION

Create a socket for application layer multicast and returns a socket descriptor.

PARAMETERS

domain - AF_INET

type - tells what kind of socket this is. SOCK_STREAM or SOCK_DGRAM

protocol - protocol number

NAME

almsock_bind

SYNOPSIS

int almsock_bind(int sock, struct sockaddr *addr, int addrlen)

DESCRIPTION

Associate a socket with a port on local machine

PARAMETERS

sock - socket descriptor

addr - pointer to a *struct sockaddr* that contains information about port and IP address

addrlen - length of a *struct sockaddr*

NAME

almsock_listen

SYNOPSIS

int almsock_listen(int sock, int backlog)

DESCRIPTION

Waiting for incoming connections and handle them in some way

PARAMETERS

sock - socket descriptor

backlog - number of connections allowed on the incoming queue

NAME

almsock_connect

SYNOPSIS

int almsock_connect(int sock, struct sockaddr *remote_addr, int remote_port)

DESCRIPTION

Tries to connect a remote host

PARAMETERS

sock - socket descriptor

remote_addr - address of a remote host

remote_port - port number of a remote host

NAME

almsock_accept

SYSNOPSIS

int almsock_accept(int sock, void *addr, int *addrlen)

DESCRIPTION

Accept a connection from a queue and create a new sock-

et descriptor to use for this single connection.

PARAMETERS

sock - socket descriptr

addr - a pointer to a local *struct sockaddr_in*. This is where the information about the incoming connection will go.

addrlen - size of *struct sockaddr_in*

NAME

almsock_close

SYNOPSIS

int almsock_close(int sock)

DESCRIPTION

Close the connection of an ALMSock Framework

PARAMETERS

Sock - socket descriptor

NAME

send

SYNOPSIS

int send(char *packet, int protocol_family, Address dest, int flag)

DESCRIPTION

Sends packet to destination Dest. If success it returns the number of bytes sent. Otherwise, it returns the error state. Both TCP and UDP transmission is available as protocol_family. If a reliable communication is needed set flag to 1, otherwise set flag to 0.

PARAMETERS

packet - packets to send

protocol_family - TCP or UDP

dest - destination address

flag - 0 : default, 1 : reliable communication

NAME

receive

SYNOPSIS

int receive(char *buffer, int length, int flag)

DESCRIPTION

Receives from multicast group and store it to reserved buffer. If success, it returns the number of byte received. Otherwise, it returns error state.

PARAMETERS

buffer - buffer to store packet

length - buffer length

flag - 0 : default reception, 1 : reliable packet reception

NAME

send_nack

SYNOPSIS

int send_nack(char *message, int length)

DESCRIPTION

Send a NACK message to a parent node if there is an error in receiving packet when reliable communication is set.

PARAMETERS

message - an error message

length - message length

NAME

join_request

SYNOPSIS

int join_request(Address member)

DESCRIPTION

Send *join_request* message to a node in a multicast group. The root node or a member node in a multicast group may receive *join_request* message.

PARAMETERS

Member - a member node in a multicast group

NAME

get_join_request

SYNOPSIS

int get_join_request()

DESCRIPTION

Receive a *join_request* message from a new comer.

NAME

join_reply

SYNOPSIS

int join_reply(Address newcomer)

DESCRIPTION

Send a *join_reply* message to a new comer.

NAME

get_join_reply

SYNOPSIS

get_join_reply()

DESCRIPTION

Receive a join_reply message from a multicast group member.

NAME

join_group

SYNOPSIS

int join_group(int groupID, Address addr)

DESCRIPTION

Join to a multicast group.

PARAMETERS

groupID - the unique ID of a multicast group

addr - the IP address of a parent node in future.

NAME

leave_group

SYNOPSIS

int leave_group()

DESCRIPTION

Leave from a multicast group

NAME

poll

SYNOPSIS

int poll(Address addr)

DESCRIPTION

Check whether a node is alive or not.

PARAMETERS

addr - the IP address to be checked

NAME

update_list

SYNOPSIS

int update_list(type field)

DESCRIPTION

Modify the field in PIL (Protocol Information List) when there is a change in protocols.

PARAMETERS

field - this may be a field in the PIL

이 영 희

e-mail : yhlee@icu.ac.kr

1976년 서울대학교 공업교육학과(학사)

1980년 서울대학교 공업교육학과(석사)

1984년 프랑스 UTC (Universite de Technologie de Compiegne) 전산학과(박사)

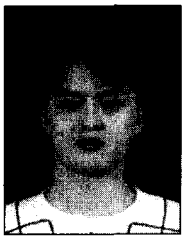
1976년~1980년 동양정밀 중앙연구소 연구원

1984년~1997년 한국전자통신연구원 센터장(책임연구원)

1986년~1987년 IBM T. J. Watson 연구소 초빙연구원

1997년~현재 한국정보통신대학교 교수

관심분야 : 컴퓨터 네트워크, 차세대 인터넷, Ubiquitous Computing



이 중 수

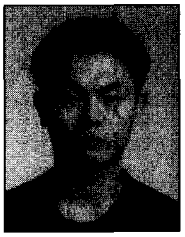
e-mail : jslee@icu.ac.kr

1999년 경북대학교 전자공학과(학사)

2001년 한국정보통신대학교(석사)

2001년~현재 한국정보통신대학교 박사과정

관심분야 : Active Network, Overlay Multicast Network



이 경 용

e-mail : leeky@icu.ac.kr

1995년~2002년 전북대학교 컴퓨터공학과 (학사)

2002년~현재 한국정보통신대학교 석사과정

관심분야 : Overlay Multicast Network, Ad hoc Network