제18회 한국정보처리학회 추계학술발표대회 논문집 제9권 제2호 (2002.11)

MPLS 프로토콜 지원을 위한 네트워크 OS의 구조 및 API 설계

박재형*, 김영희**, 이동길**, 장종현**, 최완**
*전남대학교 전자컴퓨터정보통신공학부
**한국전자통신연구원 네트워크연구소 인터넷기술연구부
e-mail:hyeoung@chonnam.ac.kr, {yhkim,dglee}@etri.re.kr

Design of Architectures and API between MPLS Protocol and Network OS

Jaehyung Park*, Young Hee Kim**, DongGill Lee**, Jonghyun Jang**, Wan Choi**
*Dept of Electronics, Computer, and Inforamtion Engineering, Chonnam National University
**Internet Technology Dept., Network Laboratory, ETRI

요 약

최근 인터넷 환경에서는 고속의 패킷 처리 능력 및 새로운 프로토콜에 대한 요구들은 또한 인터넷 통신 시스템인 라우터의 개발에도 많은 요구를 가하고 있다. 전송 속도의 초고속화와 다양한 서비스를 지원하여야 하는 라우터의 개발에 있어서 효율성 및 프로토콜의 이식성을 높이는 방법을 요구한다. 이와 같은 요구사항에 발맞추어 네트워크 운영체제(Network Operating System) 개념이 등장하였다. 본 논문에서는 이런 특징을 갖는 네트워크 OS에 MPLS 관련 프로토콜을 이식하기 위한 요구사항을 분석하여 네트워크 응용 프로그램 인터페이스(API)를 설계하고자 한다.

1. 서론

최근 인터넷은 전송 속도의 초고속화와 다양한 인터넷 서비스를 위한 새로운 프로토콜이 개발 제공되고 있는 추세이다. 이러한 차세대 인터넷의 요소 기술은 20Gbps 스위칭 라우터, MPLS (MultiProtocol Level Switching), VPN(Virtual Private Network), QoS(Quality of Service), IPv6, Multicast 통신 및 네트워크 관리 등의 보다 고급화된 서비스를 제공하기 위한 핵심 기술로써 서비스와 세션을 제어하는 프로토콜 기술 및 네트워크 OS 개념이 등장하였다.

네트워크 OS에서는 라우팅 프로토콜을 포함하는 상위 제어 프로토콜과 운영 관리 프로그램에 효율적이고 적합한 표준 API를 제공함으로써 새로운 프로토콜의 이식을 용이하게 하고 라우팅 시스템의 고성능을 유지하게 하여 궁극적으로는 새로운 프로토콜의 개발과 발맞추어 차세대 라우터를 용이하게 구현할 수 있는 방법을 제공한다.

본 논문에서는 이런 특징을 갖는 네트워크 OS에 MPLS 프로토콜을 이식하기 위한 MPLS 프로토콜의 구조를 설계하고 네트워크 응용 프로그램 인터페이스(API)를 제안한다. MPLS 관련 프로토콜 정보들의 자료 구조 및 MPLS S/W 구조에 대해서 분석하여 MPLS 프로토콜을 이식하기에 적합한 MPLS S/W에서 쓰이는 데이터 구조 및 네트워크 OS용 네트워크 API를 제시한다.

2. Zebra OS

본 절에서는 네트워크 OS의 기본 모델이 되는 Zebra OS에 대해서 살펴본다. Zebra OS ARS(Advanced Routing Software)는 Unix 호환 하드웨어 플랫폼에 라우팅 기능을 지원하는 소프트웨어 패키지로서 IP Infusion 사에서 제공하는 소프트웨어이다. Zebra OS는 Unix 호환이 되는 Linux, FreeBSD, NetBSD, Solaris 그리고 몇 개의 Real Time OS에서도 지원이 되는 플랫폼 독립적인 라우팅 소프트웨어이다.

2.1 Zebra OS의 구조

Zebra OS는 시스템의 OS를 포함한 하드웨어 플랫폼에 독립적인 이식성이 높은 구조로 여러가지 측면에서 높은 유연성을 갖는다. 독자적인 thread-based 구현으로 각 기능에 대한 모든 수행은 embedded CPU의 하나의 프로세스 내에서 관리된다. Zebra OS가 자동적으로 thread를 관리를 해주며 개별적인 라우팅 프로토콜 내에서 thread에 대한 우선순위와 관리를 쉽게 할 수 있다. 이러한 thread들은 컴파일 시에 OS에 의존적인 타스크 및 프로세스로 대치된다.

Zebra OS는 1996년 IP Infusion사에서 개발한 것으로 고성능의 라우팅 소프트웨어로 네트워크 장비 제조사에 알맞은 패키지로 각광받고 있다. Zebra OS가 갖는 가장 큰 특징은 core 및 edge 라우터, access 라우터에도 적용가능하며 IPv4 및 IPv6 라우팅 프로토콜도 지원하는 확장성과 각 프로토콜 모듈별로 독립적으로 타 프로토콜에 영향을 주지 않으면서 수정/보완이 가능한 모듈화된 구조이며, Zebra OS ARS 모듈의 임의의 하나가 장애가 발생하더라도 다른 프로토콜이 여전히 라우터로 동작할 수 있는 신뢰성을 갖고 있다.

그림 1은 제어 평면에 있는 각각의 프로토콜 데몬들과 관리 및 패킷 전달 평면에 있는 Zebra OS의 구성요소를 보여준다.

RIP, OSPF, BGP 모듈은 라우팅 프로토콜로서 임의의 프로토콜일 수 있으며, 각 프로토콜 모듈 및 VTY(Zebra OS 명령어 인터페이스), SNMP는 Zebra OS외부의 OS의 TCP/IP 스택과 인터페이스가 있다. TCP/IP 스택은 Zebra OS가 라우팅 테이블 엑세스하는 것과 마찬가지로 시스템에서 수행하는 기본적인 라우팅 기능을 수행할 때 이용된다. 형상 정보 및 운용관리는 CLI 명령과 초기설정 화일을 읽고 씀으로써 수행된다. 타스크 간의 통신은 Zebra OS가 각 모듈과의 통신, 시스템 커널과의 통신으로 수행된다.

IPv4 라우팅 데몬은 ripd, ospfd와 bgpd이고 IPv6 데몬은 ripngd, ospf6d, bgpd이다. 이러한 데몬들은 모두 라이브러리 서비스를 사용한다. Zebra OS API인터페이스를 통해서 Zebra OS Network Service Module(NSM)과 라우팅 정보(RIB)를 얻어온다. Zebra OS NSM 모듈은 시스템 커널과 직접 인터페이스한다.

2.2 Zebra NSM의 구조

Zebra OS NSM은 라우팅 프로토콜 데몬들과 통신하여 RIB(Routing Information Base)를 관리하고, 경로 및 주소에 대한 추가/삭제를 커널 기반 명령을 수행한다. 패킷 포워딩은 FIB(Forwarding

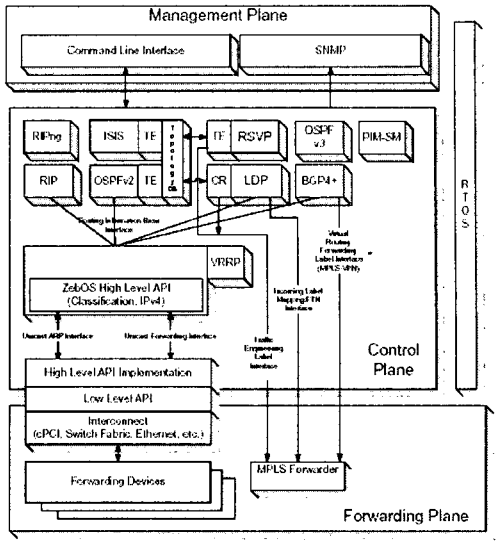

그림 1 Zebra OS의 구조

Information Base)를 통해서 이루어진다. 라우팅 프로토콜 모듈과 Zebra OS NSM 모듈간의 통신은 socket을 이용하여 프로세스간 통신을 제공한다. 이 때, Zebra OS NSM은 Server로 동작하고 라우팅 프로토콜은 Client로 동작하여 Zebra OS 모듈 인터페이스에서 제공하는 라이브러리 함수들을 이용하여 연결을 맺은 후에 서로 통신을 하게 된다. 이 때, 통신하는 메시지는 Zebra OS에 정의되어 있다.

그림 2는 라우팅 프로토콜 데몬과 Zebra OS NSM 그리고 시스템 커널의 FIB와 Zebra OS Protocol과 API의 관계를 보여준다.

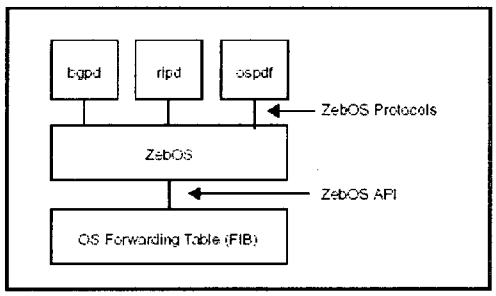

그림 2 Zebra OS Protocol과 API

Zebra OS Protocol은 Zebra OS와 라우팅 프로토콜 데몬들이 사용하는 일종의 IPC로서 RIB, interface, interface 주소 변경시에 Zebra OS와 프로토콜 데몬들간에 사용되는 프로토콜인 kernel update, RIB 변경 정보를 프로토콜 데몬에서 Zebra OS로 전달하는데 사용되는 프로토콜인 protocol

update, BGP 데몬이 IBGP 경로의 nexthop을 FIB로부터 탐색하려 할 때 사용하는 nexthop lookup을 수행한다.

Zebra OS API는 시스템 커널의 FIB를 변경하고자 할 경우에 사용하며 C 구조체 및 함수가 정의되었다.

3. MPLS Protocol 구조

MPLS는 고속스위칭을 하기 위해 특정 패킷의 주소나 포트와 같이 일정한 규칙을 설정하고 이에 해당되는 패킷에 대해서는 레이블(label)을 부여해 MPLS망에서는 IP 주소가 아닌 레이블을 기반으로 하는 고속스위칭을 하는 기술이다(label swapping). 레이블을 할당하는 것은 MPLS망의 진입점인 LER(Label Edge Router)에서 하고 망 내부의 라우터인 LSR(Label Switched Router)는 레이블만을 보고 라우팅을 하게 된다. 하지만 레이블이 할당되지 않는 패킷의 경우는 LSR은 일반적인 라우터와 동일한 동작으로 하게 된다.

망 내부에 있는 라우터들이 패킷에 포함된 레이블로 전달하기 위해서는 각 라우터들에게 레이블을 알려주는 프로트콜이 필요하다. 현재 MPLS에서는 별도의 프로토콜의 이용하는 방법을 이용하고 있는데 이 프로토콜이 LDP(Label Distribution Protocol)이다.

LDP는 다음과 같은 기본적인 기능을 지원한다.

① LSR peer들이 상대방을 발견하고 통신 채널을 설정하기 위한 LSR Discovery 메커니즘 제공

② 다음과 같은 네가지 클래스의 메시지 처리

-DISCOVERY

-ADJACENCY : 세션의 initialization, keepalive, shutdown 처리

-LABEL ADVERTISMENT : 레이블 바인딩에 대한 advertisement, request, withdrawal, release

-NOTIFICATION : 오류 정보를 알려주기 위해서 사용

③ 신뢰성 있는 메시지 전송을 위해서 DISCOVERY 메시지를 제외하고는 TCP 상에서 동작

위와 같은 동작을 수행하는 LDP는 라우터 시스템에서 그림 3에서와 같이 타 블록과 인터페이스가 존재한다. 그림 3에서 실선은 LDP 패킷 흐름을 표시하고, 점선은 각 블록간의 제어 흐름을 표시한다. LDP의 타 블록과의 인터페이스를 살펴보면 다음과 같다.

- 시스템 콜에 관련된 기반 Real Time OS와의

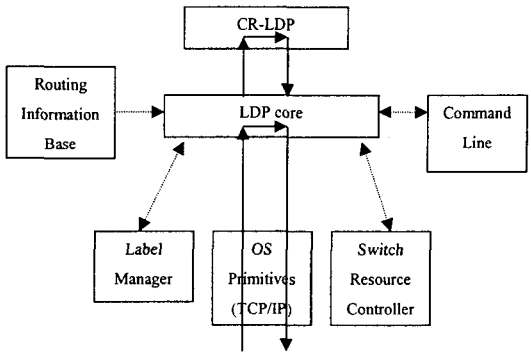

그림 3 MPLS Protocol의 인터페이스

인터페이스 및 TCP/IP 패킷 전달과 관련한 인터페이스

- 레이블 스위칭을 위해 인터페이스 카드들간의 연결을 지원하는 스위치 자원 관리자와의 자원 정보 할당 및 해제와 자원 상태에 대한 보고 인터페이스

- Routing 프로토콜에 의해서 생성/수정/삭제된 정보를 가져오기 위한 RIB와의 인터페이스 및 라우터의 논리적인 인터페이스에 대한 생성/수정/삭제와 관련된 인터페이스

- 운영자의 명령어의 인터페이스 또는 SNMP 망관리 인터페이스

- Label space의 생성/삭제 및 해당 인터페이스에 대한 Label 요구/할당/해제 등과 관련된 label manager와의 인터페이스

- CR-LDP 블록과는 CR-LDP 관련 TLV의 처리에 관한 함수 인터페이스 및 CR-LDP 메시지 송수신을 지원해주기 위한 인터페이스

MPLS 프로토콜을 네트워크 OS에 적용하기 위해서는 앞에서 설명한 LDP/CR-LDP 프로토콜에서 외부 블록간의 인터페이스를 지원하여야 한다.

4. 네트워크 OS와 MPLS 프로토콜의 구조 및 API

본 절에서는 Zebra OS에 근간한 네트워크 OS와 MPLS 프로토콜의 구조 및 API를 설계한다.

그림 4는 네트워크 OS와 MPLS 프로토콜 스택과의 구조와 인터페이스를 보여준다. Zebra OS의 NSM이 데이터 포워딩 평면의 계층 3 정보를 전달하는 역할을 맡고 있는 반면에, LSM(Label Service Module)은 데이터 포워딩 평면의 MPLS 계층 정보를 전달하는 역할을 맡고 있으며, Label 자원을 사용하는 모든 프로토콜이 LSM과 인터페이스를 가지고 있다.

RIB 인터페이스는 Zera OS NSM에서 지원되었

던 kernel update와 protocol update를 통해서 이루어진다. 이 때, 사용되어지는 API들은 다음과 같다.

- ZEBOS_INTERFACE_ADD/DEL
- ZEBOS_INTERFACE_ADDRESS_ADD/DEL
- ZEBOS_INTERFACE_UP/DOWN
- ZEBOS_IPV4/6_ROUTE_ADD/DEL

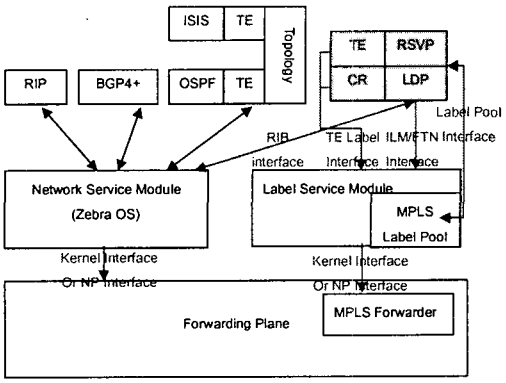

그림 5 MPLS 프로토콜 구조 및 Interface

FTN(FEC-To-Nexthop) 테이블과 ILM(Incoming Label Mapping) 테이블 인터페이스는 각 프로토콜 블록에서 생성/삭제하는 레이블 자원에 대한 LSM과의 인터페이스이다. 또한, TE 레이블에 대한 테이블 인터페이스도 포함된다. 이 때, 사용되는 인터페이스는 다음과 같다.

- mpls_clean_fib_for() : 해당 프로토콜로부터 생성된 FTN, ILM 엔트리 모두 삭제
- mpls_ilm_entry_add/del() : ILM 테이블에 엔트리 추가 및 갱신 / 삭제
- mpls_ftn_entry_add/del() : FTN 테이블에 엔트리 추가 및 갱신 / 삭제
- mpls_tel_entry_add/del() : TEL 테이블에 엔트리 추가 및 갱신 / 삭제

Kernel Interface 및 NP Interface는 생성된 레이블을 MPLS 데이터 포워딩 평면에 전달하는 인터페이스로 다음과 같은 API들이 있다.

- npfIPv4MPLSFTNTableEntryAdd/Del()
- npfIPv4MPLSFTNTableEntryFlush()
- npfIPv4MPLSFTNTableEntryCreate/Close()
- npfIPv4MPLSILMTableEntryAdd/Del()
- npfIPv4MPLSILMTableEntryFlush()
- npfIPv4MPLSILMTableEntryCreate/Close()
- npfIPv4MPLSTELTableEntryAdd/Del()
- npfIPv4MPLSTELTableEntryFlush()
- npfIPv4MPLSTELTableEntryCreate/Close()

MPLS Label Pool은 MPLS 프로토콜 블록에서 사용하는 label 자원을 할당/해제를 지원한다. Label Pool은 하나의 시스템에서 수행중인 여러 개의 프로토콜에 대해서 유일한 레이블을 할당할 수 있도록 하는 목적이 있다.

LSM에서 관리하는 자료 구조는 그림 5에서 보여진 것과 같이 Label Table을 관리하고 시스템의 상태에 포워딩 엔진에 있는 테이블을 적절히 조정함으로써 패킷에 대한 정확한 포워딩을 할 수 있도록 한다.

- FTN

FEC	OutIf	NextHop	OutLabel	Opcode

– Opcode : PUSH, DLVR-TO-IP

- ILM

InLabel	InIf	OutIf	NextHop	OutLabel	Opcode

– Opcode : POP, SWAP, POP-FOR-VPN

그림 4 Label 관련 테이블들

MPLS 프로토콜 스택을 위해서 하나의 LSM 모듈을 둠으로써, 프로토콜 스택과의 OAM 모듈간의 인터페이스를 줄일 수 있다. 그 이유는 MPLS의 여러 프로토콜들이 자신이 생성/삭제한 레이블 스위칭을 위한 정보를 포워딩 엔진에 전달할 때, 포워딩 엔진에 대한 정보를 해당 프로토콜이 OAM 모듈과의 인터페이스를 통해서 교환하여야 하기 때문이다.

5. 결론

본 논문에서는 네트워크 OS에 MPLS 프로토콜을 이식하기 위한 구조로써 LSM의 구조를 제시하였고 그들간의 API를 설계하였다. 설계된 LSM 구조는 기존의 NSM과 별도의 블록으로 구성하여 블록의 모듈성을 높이고, MPLS 프로토콜 스택에서의 타 블록간의 인터페이스를 줄일 수 있다. 향후 연구과제로는 포워딩 엔진과의 인터페이스에 대한 API의 상세화가 필요하다.

참고문헌

[1] IP Infusion Inc., ZebOS Advanced Routing Suite Version 2.0 : ZebOS Developer Guide, September 2001.

[2] IP Infusion Inc., ZebOS ARS Version 2.1 : Developer Guide for ZebOS Network Service Module including the Network Processor API, December 2001.

[3] L. Anderson, *et al.*, LDP Specification, IETF RFC 3036, January 2001.

[4] B. Jamoussi, Constraint-Based LSP Setup using LDP, draft-ietf-mpls-cr-ldp-05.txt, July 2000.

[5] E. Rosen, *et al.*, Multiprotocol Label Switching Architecture, IETF RFC 3031, January 2001.