Multiprotocol Label Switching System을 위한 Label Distribution Protocol 구현

정회원 박 재 현*

Implementation of the Label Distribution Protocol for the Multiprotocol Label Switching

Jae-Hyun Park* *Regular Member*

요 약

본 논문에서 Multiprotocol Label Switching 시스템을 위한 Label Distribution Protocol의 설계와 구현에 관해서 서술한다. 먼저 Gigabit Switched Router를 만들기 위해서 필요한 LDP의 구현 시 고려해야 될 사항에 대해 살피고, 상세 설계를 제안 한다. 결과로써, IETF 표준에 의거한, LDP를 구현하는 데이터 구조 및 처리 절차들을 제시한다. 본 논문에서는 구현 시 Carrier Class Product에 적용하기 위해 고려해야 할 점들을 제시한다. 구현된 프로토콜은 상용망에서 요구되는 IP 라우팅 테이블 항목 개수인 4만개의 엔트리들 수용 요구를 만족 시킨다. 또한 본 시스템은 표준 Unix API를 사용하여 구현됨으로써 이식성을 가지고 있다. 이러한 구현 시 고려할 사항들과 표준에 의거하여 구현을 함으로써, 우리는 본 LDP가 표준에 기준한 다른 상용제품과 연동성을 확보할 것을 기대한다. 이 LDP의 프로토타입을 구축하고, 이 구현 결과의 프로세스 대수를 사용한 형식적 명세와 분석 그리고 성능 분석을 함으로써 메커니즘의 타당성을 검증하고 있다.

ABSTRACT

In this paper, we describe the design and implementation of the Label Distribution Protocol (LDP) for Multiprotocol Label Switching System. We review the implementation issues of LDP that is required to make a gigabit switched router, and propose a detail design of it. We present the data structures and procedures for the LDP as a result, which are based on IETF standard. We present design issues for applying this to carrier class products. The implemented protocol could afford 40,000 entries of the IP routing table that is required for deploying this system to commercialized data network. Furthermore this system implemented using the standard API of Unix, as a result, it has portability. By implementing LDP based on the international standard and these implementation issues, we expect that the implemented LDP will be interoperable with other commercialized products. We prove the validity of the design of the LDP through prototyping, and also verify the prototype with the specification using the process algebra and the performance analysis.

중요어: MPLS, Label Distribution Protocol, Process Algebra, Performance Analysis

* 삼성전자 정보통신본부 네트웍 사업부(jaehyun.park@ieee.org),
논문번호 : 98267-0703, 접수일자 : 1998년 7월 3일

Ⅰ. 서 론

ADSL(하향10Mbps), Cable Modem(30Mbps)과 같은 많은 대역폭을 요구하는 가입자 망 기술들이 재택 가입자들, 작은 사무실들, 기업 망들을 위해서 준비되고 있다. 또한 이동 전화 같은 단말의 데이터 서비스 요구도 신규수요를 창출하고 있다. 이러한 다양한 가입자 망들을 통해, 접근 망 제공자들에 의해서 모아진 인터넷 트래픽을 지역적 혹은 전역적인 인터넷 제공자의 가장 가까운 제공 점으로 전송하기 위해서는 공중 데이터 망의 구축이 필요하다.

이러한 요구에 부응하는 초고속 공중 데이터 망을 구축하기 위해서는 고속 패킷 교환기의 사용이 필수적이며, 이를 개발하기 위한 노력이 선진 각 사들에서 결실을 맺어 제품들로 출시되고 있다. 이러한 데이터 교환기(Alcatel 1000, 1100, Ericsson AXD 301, 311, Lucent Packetstar 6400)는 기존의 음성 교환기의 가입자인 64kbps에 비해, 대역폭을 많이 사용하면서도(ADSL 2Mbps, Cable Modem 10Mbps 정도가 현실적인 수치임) 굉장히 폭발적인 트래픽 특성을 갖는 인터넷 가입자를 비용면에서 효과적으로 수용할 수 있다. 이러한 교환기의 구성 요소로써, 공통적으로 MPLS를 사용하는 다계층 스위치드 라우팅 기술(Multi-layered Switched Routing Technique)을 수용하고 있으며, 이를 사용하여 각종 표준과 사실상 표준에 준하는 장비들은 정연한 방법으로 그들의 데이터 통신망을 만들거나 확장하는 도구를 제공한다.

시장에서 성공하고 있는 대부분의 공중망 용 제품들은 Internet Engineering Task Force (IETF) 표준인 Multiprotocol Label Switching (MPLS)을 장착하고 있다. 이러한 통합되고 유연하고 비용면에서 효과적인 장비를 사용하여, 고속 인터넷 망 서비스 제공자들은 망을 구성하고, 중심 장소에서 트래픽 집성(Concentration)을 제공한다. 이러한 장비의 백본망 안으로의 도입으로, ISP는 LAN 속도로 인터넷 접근을 제공할 수 있다. 또한 통합 망 관리는 전체 망의 완전한 제어를 제공한다. 이러한 완전한 제어를 통해서, 전송 망은 또한 ISP에게 여러 IP망을 투명하게 지원하고 그렇게 함으로써 VPN 서비스를 기관들이나 회사들에게 제공한다. 다시 말하면 기존의 IPOA, MPOA 과 같은 Overlay Network에 비해서, MPLS의 도입으로 인해 사업자들은 망의 확장성, QoS, 그리고 멀티캐스팅과 VPN과 같은 신규 서비스의 수용, 관리성에서 강점을 제공받게 된다[1].

본 논문에서는 ATM기반 인터넷 서비스 시스템인 Multiprotocol Label Switching (MPLS) 시스템 개발[7]에 있어서, 핵심 프로토콜인 Label Distribution Protocol의 구현 방법을 제시한다. 2절에서 MPLS 시스템에 있어서, LDP의 역할을 이해하기 위해, MPLS 시스템의 구조와 동작 시나리오를 설명한다. 3절에서는 LDP 자체의 이해를 위해, MPLS 소프트웨어 시스템의 구조를 정의하고, LDP를 자체를 설명하고, 시스템의 동작 모델을 제시한다. 4절에서 설계 시 고려할 점들을 제시하고, 설계를 제시하고, 구현에 관해 설명하고, 결과인 데이터 구조들과 처리절차들을 제시한다. 5절에서 프로세스 대수를 사용한 형식적 명세를 통해 구현된 프로토콜의 안전성을 간략히 보이고, 성능 분석을 한다. 마지막으로 6절에서 결론을 맺는다.

Ⅱ. MPLS System 구조와 동작 시나리오

이 절에서는 먼저 시스템 구조(architecture)를 살피고, 다음 동작 시나리오에 관해 살펴본다.

1. MPLS System 구조

MPLS System은 그림 1에서 보이는 바와 같이, 크게 Label Switched Router (LSR)과 Label Edge Router(LER)로 구분된다. LSR은 기존의 인터넷 트래픽을 처리가 할 수 있는 Layer1, Layer2의 메커니즘을 지닌, ATM 교환기에 Internet Protocol을

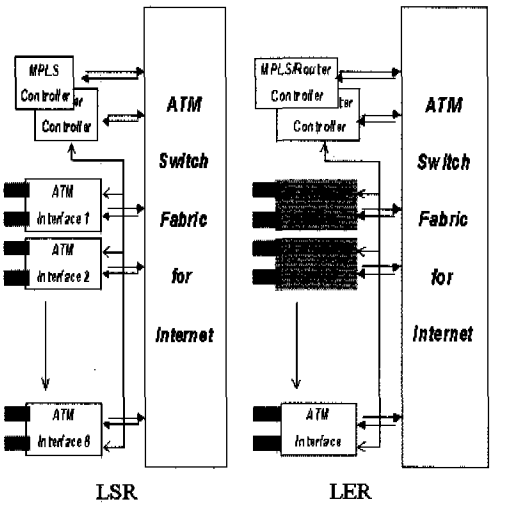

그림 1. MPLS System 구조도

연동할 수 있는 일종의 시그널링 보드인 MPLS Controller를 추가하고, 이 제어기와 IPC를 주고 받으면서 LSP의 설정 및 관리를 맡은 소프트웨어를 기존의 ATM 호처리 블록에 추가로 구현함으로써 만들어진다[2].

LER은 LSR에 High Performance Packet Forwarding Engine을 탑재한 Smart Router Interface와 관련 소프트웨어들이 추가된 것이다. 본 논문은 초고속 정보통신망 구축 시 도입되는 ATM 교환기에 인터넷 서비스를 수용하기 위해 2001년까지 개발하는 MPLS Controller 서브 시스템의 개발에 관해 기술하고 있으므로, LSR에 필요한 MPLS Controller의 개발에 초점을 두고 있다. 그러나 본 논문에서 제시된 LDP는 LER에도 적용할 수 있다.

2. MPLS 시스템의 동작 시나리오

MPLS 시스템은 IETF 표준 초안[3]에 명시된 LDP를 따르며, IPOA를 통해 접속한 이웃 제어기들 사이에 IETF 표준인 RIP, OSPF, BGP를 동작시켜 얻어진 IP 라우팅 정보를, 스위치의 2계층 정보로 변환 시켜, 전달 경로 설정을 스위칭 하드웨어의 호처리/자원관리 블럭에 보내어 스위치드 라우팅을 실현하는 시스템이다 [3]. 여기서 상기 모든 프로토콜 들은 모두 IETF의 표준 또는 표준 초안을 따른다.

MPLS 시스템의 동작 목적인 고속 패킷 전달 혹은 스위칭을 위해 행하는, MPLS 시스템의 동작 시나리오를 그림 2를 통해서 알아보자. 가장 일반적인 경우인 Downstream on Demand Ordered [3]의 경우의 고속 패킷 전달 혹은 스위칭을 위한 패킷 전달 경로인 Label Switched Path들의 설정 주요 동작들을 예시하면 다음과 같다. 먼저 1단계에서 라우팅 프로토콜을 통해서 자신을 통해서 전달할 수 있는 서브 망의 주소들을 전파한다. 다음으로 2단계에서 필요한 동일한 LSP를 통해 전송할 수 있는 각 서브 망의 주소에 해당되는 Forwarding Equivalence Class (FEC) 별 경로를 설정하기 위한 레이블을 요구하는 레이블 요구 메세지를 발생 시킨다. 3단계에서 받은 레이블 요구 메세지를 처리하여, 레이블 사상 메세지를 발생시킨다. 레이블 사상 메세지를 받은 LDP는 받은 레이블을 교환기에 넘겨주어, 해당 Label Switched Path를 설정 하도록 한다. 마지막으로 4단계에서 받은 패킷을 정합의 물리층의 속도로 전달 혹은 스위칭 한다. 다음 절에서는 위의 동작 시나리오의 중심 역할을 하는 LDP에

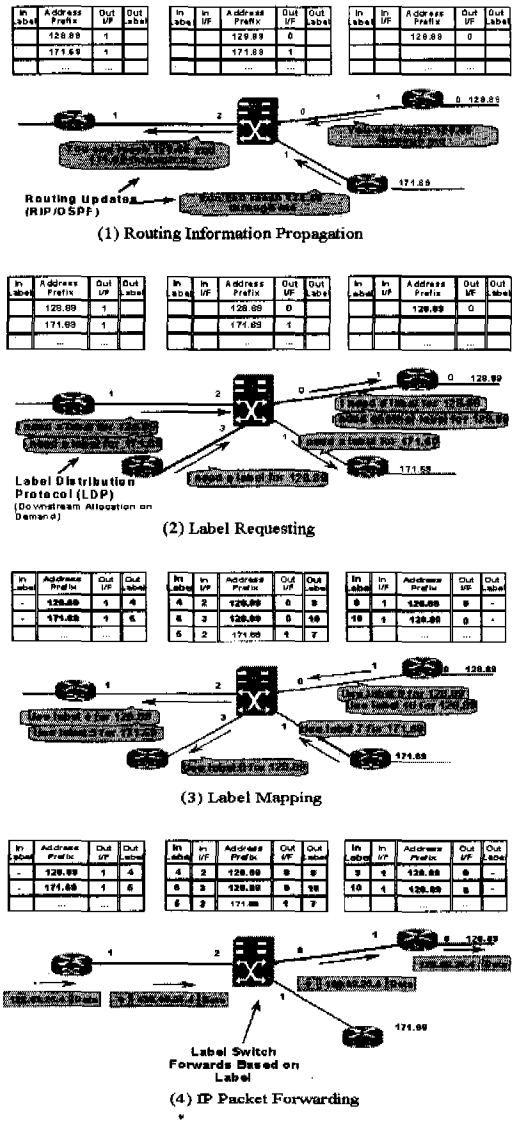

그림 2. MPLS System 동작 시나리오 [1]

관해 알아보겠다.

Ⅲ. Label Distribution Protocol

본 절에서는 LDP 자체의 이해를 위해, MPLS 제어기 시스템의 구조를 정의하고, LDP를 자체를 설명하고, 시스템의 동작 모델을 제시한다.

1. MPLS 제어기 시스템 구조

MPLS 제어기는 그림 3과 같이 Label Distribution Protocol[3], 그리고 LSP의 실현을 위해 ATM

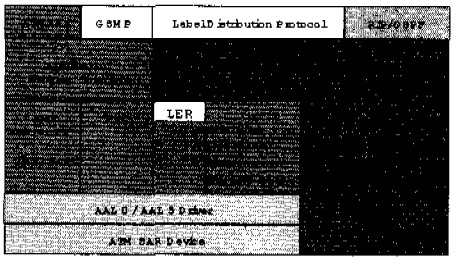

그림 3. MPLS 제어기 시스템 구조

교환기의 호처리 블록과 LDP간의 연동에 사용되는 General Signal Management Protocol (GSMP) [5], 시그널링 경로를 만드는 데 사용되는 Classical Internet Protocol over ATM (IPOA), 기존의 TCP/IP Protocols, 3계층 라우팅 정보를 전달하여, 라우팅 테이블을 만드는 라우팅 프로토콜(RIP 혹은 OSPF)로 구성된다. 이제 각각 구성 요소들에 관해서 자세히 알아보자. 참고로 이들 기능의 상세 상호작용들은 다음 4절에서 기술한다.

1) LDP

Label Distribution Protocol은 IETF의 Working Draft인 draft-ietf-mpls-ldp-06.txt 혹은 미래의 표준을 따라서 동작하여야 한다. 본 기능은 이를 구현하기 위한 프로시저들과 데이타 구조를 포함한 소프트웨어 블럭 이다.

LDP는 Label Switched Routers가 망 계층의 경로설정 정보를 직접 데이터 링크 계층 스위치된 경로에 사상함으로써, 망을 통한 레이블 스위치된 경로를 설정하는 일련의 처리 과정들과 메세지들의 집합이다[3]. 하나의 LSP는 패킷들의 흐름 하나를 위하여 만들어진다. LSP들은 각각의 LSR이 하나의 흐름을 위한 입력 레이블들을 주어진 흐름을 위한 다음 홉으로 할당된 출력 레이블로 레이블들의 사상을 실현한 것들의 연속으로 구성된다.

LDP 메세지에는 3가지 종류가 있다. 1) 망상에 하나의 LSR의 존재성을 알리고, 유지하기 위해 사용되는 발견 부류의 메세지 (UDP를 통함), 2) LSR 동료들간에 인접을 성립하고, 유지하고, 끝내는 인접 부류의 메세지 (TCP를 통함), 3) 흐름들을 위한 레이블 사상들을 생성하고, 변경하고, 지우는데 사용되는 알림 부류의 메세지(TCP를 통함)가 있다.

2) Routing Protocols

Routing Protocol Function은 라우팅 테이블을 생성하는 기능이다. 이는 RIP 또는 OSPF (선택적으로 BGP, PIM)이며, 만들어진 Routing Table은 FIB를 만드는데 사용된다. 이 기능은 Routing Table에 변동사항이 있을 때, 이를 LDP에 알려 주는 기능을 기존의 라우팅 소프트웨어에 추가하므로써 만들어진다.

3) IPOA

상기 프로토콜들은 IPOA (RFC1577,RFC1483)위에서 동작 한다.

4) GSMP

GSMP는 LDP와 ATM Switch의 호처리 및 자원관리 소프트웨어간의 통신을 위해 사용하는 Protocol 이다. 교환기 상에서 연결을 설정하거나 해제하고, 스위치 포트 정보와 형상정보를 요구하여 관리한다. 또한 통계 정보를 유지하고, Switch상의 비동기 이벤트를 인지한다.

2. MPLS 제어기의 동작 모델

일반적인 MPLS 제어기의 동작 모델을 통해, 위의 구성 요소들의 동작을 간단히 정리하면 그림 4과 같다. 모델의 동작은 다음 절의 메시지 흐름 도표(Message Flow Chart)를 통해서 상세히 보이겠다.

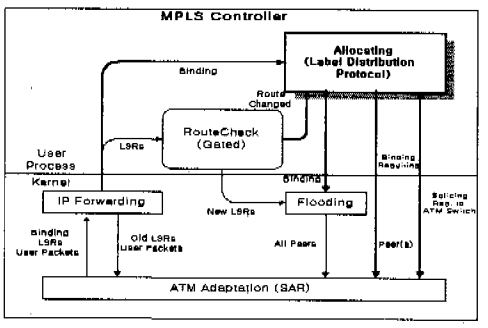

그림 4. MPLS 제어기 동작 모델[4]

Ⅳ. MPLS LDP 설계 및 구현

이 절에서는 설계 시 고려할 점들에 관해 살피고, 표준에 의거하여 설계된 LDP의 메시지 흐름 절차를 통해, 실제 구현된 기능들에 관해 설명하고, 구현 결과인 데이터 구조와 처리절차를 제시한다.

1. MPLS LDP 설계 시 고려할 점들

상용시 효과적인 LDP를 설계하기 위해 고려할 점들은 다음과 같다.

- 표준의 만족: LDP는 IETF의 Working Draft 인 draft-ietf-mpls-ldp-06.txt 혹은 미래의 표준을 따라서 동작하여야 한다. 또한 기존의 시장의 선점한 제품과의 상호 연동성을 확보하여야 한다.
- 성능과 신뢰성: LDP 및 Software는 Carrier Class 제품의 성능과 신뢰성을 지녀야 한다. 성능을 위해서, 구현 시 LDP 세션을 위하여, Nonblocking TCP session을 사용해야 하며, 세션 설정의 재시도는 Exponential Back-off Timer에 따라 이루어져야 한다. 또한 신뢰성을 위해서 이중화 기능을 지원하여, 장애발생 시 서비스 중단 없이 예비장치가 서비스를 계속 수행 하여야 한다.
- 확장성: LDP는 기존의 상용 망의 라우팅 테이블의 항목수인 40,000 개의 라우팅 항목을 처리할 수 있어야 한다[6]. 또한 새로운 기능의 추가나 표준의 변화에 쉽게 적응시키기 위하여 쉽게 확장할 수 있어야 한다.
- 이식성: LDP는 표준 Unix API를 사용하여, 다양한 환경에 쉽게 이식될 수 있도록 작성되어야 한다. 또한 하드웨어의 데이타 워드의 바이트 저장 순서와 무관하게 동작될 수 있어야 한다.
- 유지보수성: LDP는 MPLS Controller Subsystem의 서비스 Upgrade, Test시 서비스 중단이 없도록 설계 되어야 한다.

2. MPLS LDP의 메시지 흐름 절차

LDP의 동작모드는 크게 Downstream on Demand Ordered와 Downstream on Demand Independent 그리고 Downstream Unsolicited Ordered와 Downstream Unsolicited Independent가 있다[3].

1) Downstream on Demand Ordered Mode

Downstream on Demand Ordered Mode의 LDP의 동작을 그림 5의 메시지 흐름 도표에서 도식적으로 나타내었다. LDP는 시스템 초기화 시에 GSMP Master(도표에서 GSMP.m 으로 명기)를 통해, 시스템의 MPLS 포트들에 대한 초기화 명령을 내린다. 이것에 대한 응답으로 사용 가능한 Label의 범위를 받는다. 한편 시스템 초기화 시 OSPF가 동작하여 만든 Routing Information Table을 LDP에 전달한다. 이는 LDP가 Forwarding Information Base를 만드는데 사용된다. 이렇게 시스템 초기화를

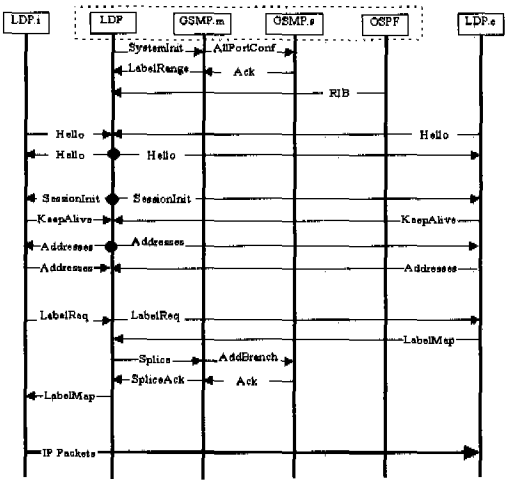

그림 5. Downstream on demand ordered mode로 동작하는 LDP의 Message Flow

마치면, LDP 들은 Hello Message를 통해 (IP 주소, Label Space) 정보를 서로에게 알린다. 그리고 나서 IP 주소의 값이 큰 LDP(그림의 예에서는 LDP)가 LDP 세션의 초기화 메시지를 통해, Label Advertisement Mode를 비롯한 세션의 동작과 관련된 각종 파라메터들을 주고, 파라메터들이 수용 가능하면 이웃한 LDP(LDP.i와 LDP.e)들이 응답으로 KeepAlive 메시지를 줌으로써 세션들을 성립 시킨다. 그리고 나서 시스템 정합들의 IP 주소를 Addresses 메시지를 통해 서로 주고 받는다. 이 초기화 절차들은 모든 LDP 동작 모드에서 동일하게 사용된다.

이 모든 초기화 절차를 끝낸 후, 다음과 같은 메시지 흐름들을 통해 LSP를 성립시킨다. 자신이 Ingress LER일 경우의 LDP(그림에서 LDP.i로 도시)는 기존의 IP 라우터로부터 자신에게 들어오는 IP 데이터 패킷을 처리하기 위한 Ingress LSP를 구축하기 위해, 레이블 요구 메세지를 발생 시킨다. LSR의 경우는 LDP레이블 요구 메세지를 받아서, 자신의 Forwarding Information Base (FIB)를 확인한 다음, 레이블 요구 메세지를 발생시키고, 상태를 유지시킨다. 그러면 Egress LER의 LDP (LDP.e)는 레이블 요구 메시지를 받아서 기존의 라우터로 나가는 Egress LSP 중 자신이 설정할 부분을 완성하고, 레이블 사상 메시지를 발생 시킨다. LSR의 LDP (LDP)는 돌아오는 레이블 사상 메세지를 처리하여, 자신의 데이타 구조에 반영하고, LSP의 실현을 위해, 교환기 안의 패킷 전달 경로인 스플라이스를 스위치에 설정하도록, GSMP Master (GSMP.m)를 사

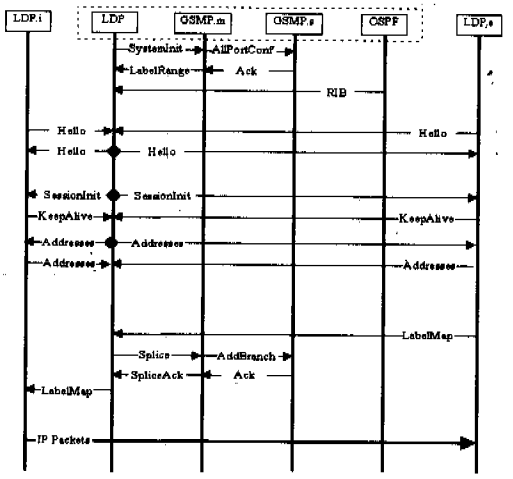

그림 6. Downstream unsolicited ordered mode로 동작하는 LDP의 Message Flow

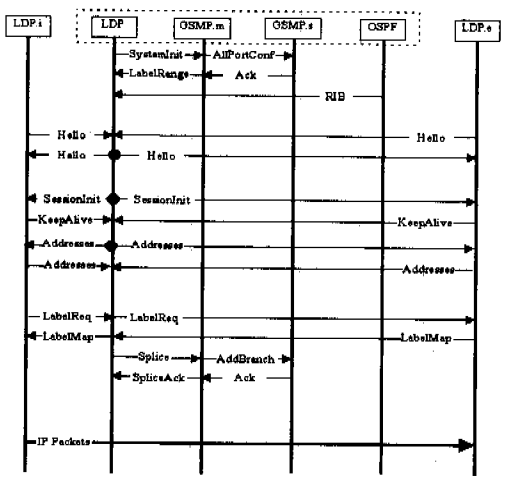

그림 7. Downstream on demand independent mode로 동작하는 LDP의 Message Flow

용하여 AddBranch 메시지를 스위치 제어기에 보낸다. 이때 스위치 제어기는 GSMP Slave (GSMP.s)를 통해 통신을 한다. 성공적으로 스플라이스가 설정되면, 레이블 요구 메시지를 보낸 세션으로 레이블 사상 메세지를 발생시켜 돌려 준다. 상기의 단계들을 거쳐 성립된 LSP를 거쳐 IP 패킷이 전달된다.

2) Downstream Unsolicited Ordered Mode

Downstream Unsolicited Ordered Mode의 LDP의 동작은 그림 6의 메시지 흐름 도표에서 도식적으로 잘 나타나 있다. 앞 절에서 설명한 초기화 절차를 끝낸 후, 다음과 같은 메시지 흐름들을 통해 LSP를 성립시킨다. Egress LER의 LDP (LDP.e)는 Forwarding Information Base (FIB)를 확인하고, 출력 레이블을 할당하고, 이를 GSMP 메시지를 통해 스위치의 제어부에 보내 기존의 라우터로 나가는 Egress LSP 중 자신이 설정할 부분을 완성하고, 레이블 사상 메시지를 발생 시킨다. LSR의 LDP (LDP)는 돌아오는 레이블 사상 메세지를 처리하여, 자신의 데이타 구조에 반영하고, LSP의 실현을 위해, 교환기 안의 패킷 전달 경로인 스플라이스를 스위치에 설정하도록, GSMP Master (GSMP.m)를 사용하여 AddBranch 메시지를 스위치 제어기에 보낸다. 이때 스위치 제어기는 GSMP Slave (GSMP.s)를 통해 통신을 한다. 성공적으로 스플라이스가 설정되면, 레이블 요구 메시지를 보낸 세션으로 레이블 사상 메세지를 발생시켜 Ingress 쪽으로 보내준다. 상기의 단계들을 거쳐,성립된 LSP를 거쳐 IP 패킷이 전달된다.

3) Downstream On Demand Independent Mode

Downstream On Demand Independent Mode의 LDP의 동작은 그림 7의 메시지 흐름 도표에서 도식적으로 잘 나타나 있다. 앞에서 설명한 초기화 절차를 끝낸 후, Egress LER의 LDP를 제외한 모든 LDP(그림에서 LDP.i와 LDP로 도시)는 FIB에 있는 FEC들에 대응하는 레이블 요구 메세지를 발생 시킨다. LDP가 레이블 요구 메세지를 받으면, 자신의 FIB를 확인한 다음, 즉각 대응하는 입력 레이블을 할당해서 상태를 데이터 구조에 저장하고, 레이블 사상 메시지를 발생시켜 응답해준다. 한편 Egress LER의 LDP (LDP.e)는 레이블 요구 메시지를 받아서 기존의 라우터로 나가는 Egress LSP 중 자신이 설정할 부분을 완성하고, 레이블 사상 메시지를 발생 시킨다. LSR의 LDP (LDP)는 돌아오는 레이블 사상 메세지를 처리하여, 자신의 데이타 구조에 반영하고, LSP의 실현을 위해, 교환기 안의 패킷 전달 경로인 스플라이스를 스위치에 설정하도록, GSMP Master (GSMP.m)를 사용하여 AddBranch 메시지를 스위치 제어기에 보낸다. 성공적으로 스플라이스가 설정되면, 상기의 단계들을 거쳐 성립된 LSP를 거쳐 IP 패킷이 전달된다. 스플라이스의 설정에 실패하게 되면, 이전에 레이블 요구 메시지를 보낸 세션으로 레이블 취소 메세지를 발생시켜 보내 준다.

4) Downstream Unsolicited Independent Mode

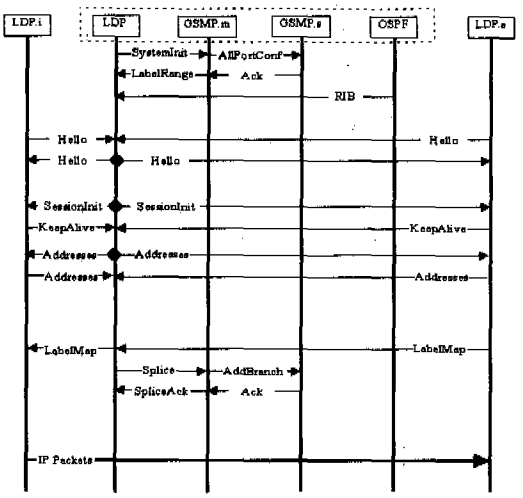

그림 8. Downstream unsolicited independent mode로 동작하는 LDP의 Message Flow

Downstream Unsolicited Independent Mode의 LDP의 동작은 그림 8의 메시지 흐름 도표에서 도식적으로 잘 나타나 있다. 동일한 초기화 절차를 끝낸 후, 다음과 같은 메시지 흐름들을 통해 LSP를 성립시킨다. Ingress LER의 LDP (LDP.i)를 제외한 모든 LDP는 Forwarding Information Base (FIB)를 확인하고, 출력 레이블을 할당하고, 이를 GSMP 메시지를 통해 스위치의 제어부에 보내 기존의 라우터로 나가는 LSP 중 자신이 설정할 부분을 완성하고, 레이블 사상 메시지를 발생 시킨다. LSR의 LDP (LDP)는 레이블 사상 메세지를 처리하여, 할당된 레이블 정보를 자신의 데이타 구조에 저장하고, 처리 상태를 저장한 다음, LSP의 실현을 위해, 교환기 안의 페킷 전달 경로인 스플라이스를 스위치에 설정하도록, GSMP Master (GSMP.m)를 사용하여 AddBranch 메시지를 스위치 제어기에 보낸다. 이때 스위치 제어기는 GSMP Slave (GSMP.s)를 통해 통신을 한다. 성공적으로 스플라이스가 설정되면, 상기의 단계들을 거쳐 성립된 LSP를 거쳐 IP 페킷이 전달된다.

3. MPLS LDP 구현 기능들

각각의 구성요소(component)들을 만들기 위해 요구되는 세부 기능들을 정의해 보면 다음과 같다.

1) LDP

Label Distribution Protocol을 구현하는 세부 기능들은 다음과 같다[1].

- IETF 표준에 정의된 LDP를 위한 세가지 종류의 메세지들을 위한, UDP, TCP 통신 경로를 설정하고, 유지하고, 종료하는 작업들 하는 처리과정
- 상기 세 가지 메세지를 작성해서, 보내는 처리과정
- LDP 상태 기계를 유지하는 처리과정
- LSR 동료들의 목록을 유지하고, 상호 등록하는 처리과정
- Label 사상을 생성하고, 변경하고, 없애는 처리과정
- Stream Merge에 대비하기 위한 처리과정
- Label 사상을 ATM 스위치의 호처리 및 망연동 모듈에게 알리기 위한 처리과정
- 경로 설정표(Routing Table)를 전달 정보 기반 (Forwarding Information Base: FIB) 구축을 위해서 전달받는 초기화 처리과정

다음의 기능들은 추후 구현을 고려하고 있다.

- Loop-Free를 유지하기 위한 처리과정
- LSR 동료들을 상호 인증하는 인증 처리과정
- LDP 상태 기계의 자체 신뢰성 증가를 위한 처리과정
- LSR 동료의 죽음과 재시동에 대비한 설계를 통한 신뢰성을 증대시키는 처리과정
- 멀티캐스팅의 처리과정
- QoS를 고려한 Stream Merge 처리과정
- Management Information Base에 필요한 망관리 통계 정보를 수집 정리, 망관리 기능의 요구 시 전달하는 처리과정

2) Routing Protocols Interfaces

LDP의 Routing Protocol Interface를 구현하는 세부 기능은 Routing Table의 변동사항에 대해 LDP 데몬에게 알리는 처리과정으로 구성되어 있다.

3) GSMP 및 교환기 의존적인 부분

LDP가 LSP의 설정을 위해 사용하는 General Switch Management Protocol 및 교환기 의존적인 부분들을 구현하는 세부 기능들은 다음과 같다[5].

- 컨트롤러와 스위치의 동기화 기능
- 스위치상의 연결을 설정 해제 요청 기능
- 스위치상의 포트 서비스 작동 여부 결정 및 리셋 요청 기능
- 스위치의 Configuration 상태 정보 요청 기능
- 스위치에서 발생하는 비동기 이벤트 인지 기

능

- 스위치의 포트별 Label Space ID와 Label Space ID에 대해 가용한 Label Space 관리 정보 요청 기능
- LDP에서 생성된 LIB 정보를 파라메터로 한 ATM 스위치의 Splice 형성요청을 LDP로부터 받아서, 해당 Connection(Splice)을 ATM 교환기에 요청을 하고, 받은 결과 상태 정보를 LDP에 알리는 기능
- 교환기의 포트 Down등의 장애 시, LDP에게 이를 알려 하는 기능. Port의 실장, 탈장 시 이것을 LDP에게 알려 주는 기능

4. MPLS LDP 구현 결과

다음에서 구현 결과인 데이터 구조들과 처리 과정들에 관해서 기술한다.

1) 데이터 구조들

LDP를 구현하는 데이터 구조들은 LDP Signaling 연결을 관리하는, LDP Session 관리 블럭과 하나의 Forwarding Equivalence Class 항목을 담고 있는 FIB 항목과 FEC에 할당된 레이블 또는 할당할 레이블을 저장할 LIB로 구성되어 있다.

- Session Control Block: LDP 세션관리를 위한 제어 블럭은 다음의 데이터 구조로 구성된다.

```
typedef struct __SC_t {
  __u16                 Local_Logical_Label_Space;
  __u32                 L3_NextHop_LSR_Addr;
  __u16                 NextHop_Logical_Label_Space;
  __u8                  Role;
  __u8                  State_Machine;
  int                   Socket_ID;
  __u32                 Msg_ID;
  __u16                 LDP_version;
  Init_Parameters_t     LDP_Init_Para;
  LdpTimer_t            HoldTime;
  Label_Range_t         Label_Range;
  struct __SC_t         *next;
} Session_Control_t;
```

여기서 Local_Logical_Label_Space와L3_NextHop_LSR_Addr, NextHop_Logical_Label_ Space는 LDP의 Hello 처리과정으로부터 얻어지는 정보를 저장한다. Role은 LDP 세션 초기화 과정에서 자신이 TCP 세션을 여는 시도를 하는 Active 역할을 할지, 반대인 Passive 역할을 할지 여부를 저장한다. 이의 판단은 자신의 IP 주소와 L3_NextHop_LSR_Addr를 비교하여, 큰 값을 갖는 쪽이 Active 역할을 하도록 하는 방식으로 이루어진다. State_Machine은 4.4.3 절에서 프로세스 대수로 명세 될, 세션의 상태 기계이다. Socket_ID는 세션의 소켓 ID를 저장한다. Msg_ID는 LDP 메세지의 PDU에 포함되어 있는 ID로써 LDP 메세지의 순서 번호(Sequence Number)를 나타낸다. LDP_version은 LDP 자신의 버전을 나타내고, 세션을 시작할 때 세션의 교섭(Negotiation)에 있어서 매개 변수로 사용한다. 기타 매개변수들은 LDP_Init_Para에 저장되며, 협상이 끝난 변수들도 여기에 갱신되어 저장된다. HoldTime은 Exponential Back-off에 의해서 LDP 세션 설정의 재시도를 하고, 세션의 설정 후에는 Keep Alive Timer를 위해서 사용된다. Label_Range는 이 세션에서 할당 가능한 레이블의 범위를 저장하고, 세션 성립 후에는 협상이 끝난 값들로 갱신된다. next는 단일 연결 목록(Single Linked List)를 유지하기 위해 사용한다. 이 구조에 저장된 값들을 사용하는 것에 대해서는 뒤의 LDP 처리절차 설명에서 좀더 하겠다.

- Forwarding Information Base: FIB는 다음의 데이터 구조로 표현된 항목들의 집합으로 구성된다[2].

```
typedef struct __FIB_t {
  __u32         Destination_IP;
  __u32         Class_of_Service;
  __u32         Forwarding_Equivalance_Class;
  __u32         NextHop_LSR_Addr;
  __u16         Local_Logical_Label_Space;
  __u16         NextHop_Logical_Label_Space;
  Label_Information_Base_t      *LIBE;
  struct __FIB_t        *next;
} Forwarding_Information_Base_t;
```

여기서 Destination_IP는 라우팅 테이블로 부터 만들어진 정보로써, 하나의 FEC에 대한 주소 접두어(Prefix)이고, Class_of_Service는 CR-LDP에서 CoS를 위해 사용하기 위해 예약해둔 변수다. Forwarding_Equivalence_Class는 FEC를 관리하기 위한 ID이고, NextHop_LSR_Addr은 Routing Table의 Gateway 정보에서 만들어 진다. Local_Logical_Label_Space는 레이블이 할당될 Label Space ID를 나타내고, NextHop_Logical_Label_Space는 레이블을 할당 받을 Peer LSR의 Label Space ID를 나타낸다. LIBE는 다음에 좀더 자세히 설명되는데, 레이블 정보를 저장하는 변수이다.

- Label Information Base: LIB는 할당된 레이블 값을 저장하거나 또는 레이블을 할당 받기 위해서 처리절차를 수행하는데 있어 필요한 정보들을 저장하는 다음의 데이터 구조로 표현된 항목들의 집합으로 구성된다.

```
typedef struct __LIB_t {
  __u32          Incoming_Port;
  ATMLabel_t     Incoming_Label;
  __u8           Incoming_State_Machine;
  __u32          Incoming_Msg_ID;
  __u32          Outgoing_Port;
  ATMLabel_t     Outgoing_Label;
  __u8           Outgoing_State_Machine;
  __u32          Outgoing_Msg_ID;
  struct __LIB_t      *next;
} Label_Information_Base_t;
```

Incoming_Port는 레이블 요청 메세지를 받아서 자신이 레이블을 할당해야 하는 포트 번호를 저장한다. 이 정보는 LDP 초기화 시 GSMP를 통해서 가져오는, 시스템 구성 (Configuration) 정보로 부터 얻어진다. Incoming_Label은 자신이 할당한 입력 포트의 레이블 값을 저장한다. Incoming_State_Machine은 해당 FEC의 Incoming Label을 할당하기 위한 상태 기계이고, Incoming_Msg_ID는 들어오는(Incoming) 레이블 요청 메세지의 순서 번호(Sequence Number)이다.

Outgoing_Port는 레이블 요청 메세지를 보내어, 동료(Peer) LSR로부터 레이블 사상 메세지를 받아서, 할당 받은 레이블을 사용할 포트 번호를 저장한다. 이 정보는 LDP 초기화 시 GSMP를 통해서 가져오는, 시스템 구성 (Configuration) 정보로 부터 얻어진다. Outgoing_Label은 자신에게 할당된 출력 포트의 레이블 값을 저장한다. Outgoing _State_ Machine은 해당 FEC의 Outgoing Label을 할당 요청을 위한 상태 기계이고, Outgoing _Msg_ID는 나가는(Outgoing) 레이블 요청 메세지의 순서 번호 (Sequence Number) 이다.

2) 메시지 처리 절차들

본 절에서는 앞서 설명한 메시지 흐름 도표들에서 나타난 메시지들이 각각의 모드들로 동작하는 LDP에 도착했을 때, 앞서 언급한 데이터 구조들에 저장된 어떤 정보들을 검색하고 사용해서 메시지들이 처리되는지 간략하게 설명한다. 레이블 광고 방식(Label Advertisement Mode)에 따라, Label Distribution Protocol를 구현하기 위한 처리절차 들은 크게 두 가지 방식으로 분류 된다.

• Downstream on Demand Label Advertisement

레이블 요구 메세지를 받은 경우, 자신의 FIB를 찾아보고 할당이 가능한 경우에 레이블을 할당하여, 해당 FEC의 레이블 정보 베이스에 저장하고, 앞서 설명한 것처럼 각각의 레이블 제어 방법(Label Control Scheme)에 따라, 레이블 요구 메시지나 레이블 사상 메시지를 보낸다. 레이블 사상 메시지를 받는 경우, 자신의 FIB를 찾아서 할당된 레이블을 저장하고, 이 스플라이스를 교환기에 설정하도록 요구한다. Ordered Control의 경우, 설정이 효과적으로 수행되면, 자신의 Upstream LSR에 대해서 레이블 사상 메세지를 발생시킨다. Independent Control의 경우, 설정이 효과적으로 수행되지 못한 경우, 자신의 Upstream LSR에 대해서 레이블 취소 메세지를 발생 시킨다. 자세한 처리과정은 아래 그림 9와 같다.

- When Initialization Msg received
  - Make LDP session, and send ACK
- When FIB Initialization Msg received
  - Make FIBE, and send ACK
- When Label Request Msg received
  - Get Peer LSR Address, Peer Label Space, and Prefix from the Msg
  - *Find matched FIBE by using above three keys*
  - Find corresponding Session Control Block by using keys
  - Allocate Label
  - Set Input Label as the Label
  - Send Label Req or Mapping Msg including the Prefix and the Label
- When Label Mapping Msg received
  - Get Peer LSR Address, Peer Label Space, and Prefix from the Msg
  - Find matched FIBE by using above three keys
  - Set Output Label as the Label
  - If ordered control, Send Label Mapping Msg including the Prefix and the Label

그림 9. Downstream on Demand

• Downstream Unsolicited Label Advertisement

Independent Control의 경우는 Ingress LER을 제외한 모든 LSR이, 그리고 Ordered Control의 경우는 Egress LER은 Downstream LSR이 자신의 FIB를 찾아보고 입력 레이블 할당이 요구되는 경우에 레이블을 할당하여, 해당 FEC의 레이블 정보 베이스에 저장하고, 레이블 사상 메시지를 Upstream LSR로 보낸다. 레이블 사상 메시지를 받는 경우, 자신의 FIB를 찾아서 할당된 레이블을 저장하고, 이 스플라이스를 교환기에 설정하도록 요구한다. Ordered Control의 경우, 설정이 효과적으로 수행되면, 자신의 Upstream LSR에 대해서 레이블 사상 메세지를 발생 시킨다. Independent Control의 경우, 설정이 효과적으로 수행되지 못한 경우, 자신의 Upstream LSR에 대해서 레이블 취소 메세지를 발생시킨다. 자세한 처리과정은 아래 그림 10와 같다.

3) LDP의 프로세스 대수 기법을 사용한 명세

본 절에서는 개발되어진 LDP의 상세한 동작을 대수 형식적 기술 기법인 프로세스 대수(Process Algebra)의 일종인 Labelled Transition System[8, 9]을 사용하여 기술한다. 본 기법은 프로토콜을 상태들과 프로토콜의 행위에 따른 이들 상태들간의

- **When Initialization Msg received**
  - **Make LDP session, and send ACK**
- **When FIB Initialization Msg received**
  - **Make FIBE, and send ACK**
- **For Each FIBE (for the case of Egress LER in Ordered Control or All LSR except for Ingress LER in Independent Control),**
  - **Get Peer LSR Address, Peer Label Space, and Prefix from the FIBE**
  - **Find corresponding Session Control Block by using keys**
  - **Allocate Label**
  - **Set Input Label as the Label**
  - **Send Label Mapping Msg including the Prefix and the Label**
- **When Label Mapping Msg received**
  - **Get Peer LSR Address, Peer Label Space, and Prefix from the Msg**
  - **Find matched FIBE by using above three keys**
  - **Set Output Label as the Label**
  - **If ordered control, Send Label Mapping Msg including the Prefix and the Label**

그림 10. Downstream Unsolicited

천이로써 정의한다. 기호 체계를 간략히 설명하면, 각 항에서 A ≡ a . B 가 의미하는 것은 A 상태의 프로세스는 프로토콜 행위 a 후에 B 상태의 프로세스로 천이한다는 것을 의미하며, A ≡ a . A + b . B는 A 상태는 행위 a 후에 A 상태로 되거나, 행위 b 뒤에 B 상태로 되는 것을 의미한다. 괄호는 일반적인 대수에서의 의미와 같으며, | 연산은 연산에 참여하는 프로세스가 독립적으로 동작하나 상호 통신을 하는 관계를 표현하고, \ 연산은 내부 통신 통로의 이름을 나타내기 위한 연산자이다.

LDP는 HELLO 상태와 공존하는 LDPMAIN 상태로 정의되며, HELLO 프로세스에서 획득한 ldp_id를 LDPMAIN 프로세스에 전달하는 내부 연결이 있다. HELLO 프로세스는 [3]에 정의된 바와 일치하며, SESSION_ldpid는 각 세션의 상태를 나타내는 프로세스로써, [3]의 세션 초기화 상태 기계 (Session Initialization State Machine)의 상태들 중 Non에 대응된다. 나머지 INITIALIZE 프로세스, OPENSENT 프로세스, OPENREC 프로세스, 그리고 OPERATION 프로세스도 [3]에 정의된 바와 같다.

SESS 프로세스는 세션 초기화 메시지를 받았을 때, 메시지 처리과정을 명세하고 있다.

OPERATION 프로세스는 모든 초기화가 끝난 후, 초기화 시 설정된 각각 LDP 세션의 레이블 분배 모드와 레이블 제어 모드와 LDP의 망에서의 역할에 따라, 레이블 요구 메시지와 레이블 사상 메시지를 발생시키는 처리과정을 명세하고 있다. PROCESSPDU 부분은 LDP 메시지를 받았을 때, 각 메시지의 PDU를 처리하는 과정을 명세하였다. 이 명세에서 우리는 각각의 예외적인 경우가 어떻게 처리되는 지를 볼 수 있다. REQ는 레이블 요구 메시지의 처리과정을 상세히 명세하고 있으며, MAP는 레이블 사상 메시지의 처리과정을 상세히

LDP ≡ (HELLO | LDPMAIN) \ ldp_id

HELLO ≡ (one_third_time_out . send(hello) + recv (hello)) . reset_timer_for_ldp_id . acceptable_mesg . (session_found_in_SCB . HELLO + session_not_found in_SCB . send(ldp_id) . HELLO) + expiry . remove adjacency . HELLO

LDPMAIN ≡ recv(ldp id) . SESSINIT ldpid

SESSINIT_ldpid ≡ session_connected . INITIALIZE

INITIALIZE ≡ active . send(session_init) . OPENSENT + passive . recv(mesg) . (session_init_msg_received . SESS + (except session init msg + timeout). SESSINIT ldpid)

OPENSENT ≡ recv(mesg) . session_init_msg_received . send (keepalive) . OPENREC + (timeout + recv(mesg) . except session init msg) . send(notify). SESSINIT ldpid

SESS ≡ too_short_msg . send(notify) . SESSINIT_ldpid + notfind_adjacency . send(notify) . session_closed . SESSINIT_ldpid + find_adjacency . (accept_parameters . send(session_init) . send(keep_alive) . OPENREC + not_accept_parameters . send(notify) . session closed . SESSINIT ldpid)

OPENREC ≡ recv(keepalive) . send(address_msg) . OPERATION + (timeout + recv(mesg) . except keep alive_msg) . send (notify) . SESSINIT

OPERATION ≡ recv(mesg) (except_shutdown. PROCESSPDU + (timeout + shutdown_msg_received) . send (shutdown) . SESSINIT) + on_demand . (i_am_ingress_lsr . ordered_control + independent) . label_required_for_fec . send(label_req) . OPERATION + unsolicited . (i_am_egress_lsr . ordered_control + independent_control) . label_map_for_fec . send(label_map) . OPERATION + hold_time_expiry . send(keepalive) . OPERATION + no_map_mesg_within_time . unsolicited . independent_control . send(label withdraw) . OPERATION

PROCESSPDU ≡ too_short_pdu . send(notify) . OPERATION + process fixed hdr . B

B ≡ format_error . send(notify) . OPERATION + process_mesg_hdr . (map_mesg . MAP + req_mesg . REQ + withdraw_mesg . send(delete_branch) . send(label_withdraw) . UNSPLICE + other meages . OTHERS)

UNSPLICE ≡ recv(success_delete_branch) . OPERATION + recv(fail delete branch) . send(alarm) . OPERATION

REQ ≡ unsolicited . send(notify) . OPERATION + on_demand . (format_error . send(notify) . OPERATION + fec . (format_error . send(notify) . OPERATION + search session . C))

C ≡ no_session_found . send (notify) . OPERATION + fib search . (no fib entry. send(notify) . OPERATION + search set lib . D)

D ≡ illigal_req . send(notify) . OPERATION + ordered_control . (i_am_not_egress_lsr . send(fwd_label_req) . OPERATION + i_am_egress_lsr . EGRESSMAP) + independent_control . (i_am_not_egress_lsr . SENDMAP + i_am egress_lsr . EGRESSSENDMAP)

SENDMAP ≡ alloc_vcc . (can_not_alloc_vcc . send(notify) . OPERATION + alloced_vcc . set_inlabel_state . send(label_map) . OPERATION)

EGRESSMAP ≡ alloc_vcc . (can_not_alloc_vcc . send(notify) . OPERATION + send(add_branch_for_egress_lsr) . ODR MAP EGRESS)

ODR_MAP_EGRESS recv(success_add_branch) . send(label_map) . OPERATION + recv(fail addbranch) . send(notify) . OPERATION

EGRESSSENDMAP ≡ alloc_vcc . (can_not_alloc_vcc . send(notify) . OPERATION + send(add branch for egress lsr) . INDP MAP EGRESS)

INDP_MAP_EGRESS ≡ recv(success_add_branch) . send(label_map) . OPERATION + recv(fail_addbranch) . send(label_withdraw) . OPERATION

MAP ≡ format_error . send(notify) . OPERATION + fec . (fec_not_found . send(notify) . OPERATION + label . (send(notify) . OPERATION + session search . E)

E ≡ session_not_found . send (notify) . OPERATION + fib_search . (no_fib_entry . send(notify) . OPERATION + ordered_control . search set lib . F + independent control . search set lib . F')

F ≡ illigal_map . send(notify) . OPERATION + alloc_vcc . (can_not_alloc_vcc . send(notify) . OPERATION + send(addbranch) . H)
H ≡ recv(success_addbranch) . send(fwd_label_map) . OPERATION + recv(fail_addbranch) . send(notify) . OPERATION
F ≡ illigal_map . send(notify) . OPERATION + send(addbranch) . H
H ≡ recv(success_addbranch) . OPERATION + recv(fail_addbranch) . send(label_withdraw) . OPERATION
OTHERS ≡ notification_msg . process_notification . OPERATION + address_withdraw_msg . process_address_withdraw . OPERATION + label_withdraw_msg . process_label_withdraw . OPERATION + label_release_msg . process_label_release . OPERATION + label_abort_req_msg . process_ label_abort_req . OPERATION + keepalive_msg . reset_keepalive_timer . OPERATION + vender_private_extentions_msg . process_ vender_private_extentions . OPERATION + ldp_experiments_msg . process_ldp_experiments . OPERATION

명세하고 있다. OTHERS는 상세히 기술하지 않은 나머지 메시지들을 명세하고 있는데, 이는 명세의 완전성 (Completeness)을 위해 정의되었다. 나머지 프로세스들은 내부 프로세스들로써 프로토콜의 동작 상황을 상세히 명시하기 위해 정의되었다.

이 정의들은 현재 LDP 표준[3]에 의거하여 작성되었기 때문에, 프로토콜 적합성 검증을 위해서 사용될 수 있다. 또한 본 정의는 구현에 사용된 함수 수준의 프로토콜 행위를 표현하고 있다. 따라서 이것은 성능분석에 있어 행위 모델로 다음절에서 사용한다.

5. 구현 환경

프로토타입의 개발은 Unix의 Clone인 Linux 중 RedHat5.0에 Linux Kernel 2.0.25와 IPOA Package인 Linux-atm 0.31를 설치하고, 그 위에서 응용 프로그램으로 진행되었으며, 하드웨어는 PentiumPro™ PC와 ATM Card는 ENI 155MMF 보드가 사용되었다. 따라서 Unix 표준 API인 Socket Interface를 사용해서 TCP/UDP Service Access Point (SAP) 접속 부분이 작성되었다. 이 프로토타입은 후에 타겟 보드인, 400Mhz로 동작하는 MPC750 Board에서 동작할 예정이며, 이를 위해서 pSOS™와 Epilogue 환경으로 옮겨질 예정이다.

V. 구현 결과 분석

본 절에서는 구현된 프로토콜의 안정성과 이것의 성능분석에 관해서 언급한다.

1. 프로토콜의 안정성

앞 절에서 기술한 구현된 프로토콜의 명세를 검토해보면, 다음과 같은 프로토콜의 특성들을 검정할 수 있다. 명세된 프로토콜의 상태 천이 절차에 따라, 초기 상태로부터 정의된 모든 상태들에 도달 가능하다. 따라서 이 정의는 도달성을 가졌다. 또한 항상 초기상태로 돌아갈 수 있는 경로가 있으므로 생존성을 가졌다. 이는 프로토콜에서 Deadlock이 발생하지 않음을 의미한다. 또한 같은 경로를 반복하며 다른 상태로 천이하지 않는 비생산적인 주기가 없으므로 Livelock이 존재하지 않는다. 따라서 이 구현된 프로토콜은 안전성(Safety)을 지닌다.

2. 성능 분석

구현된 프로토타입의 성능을 분석하기위해 가장 빈번히 사용되어 성능에 주요한 영향을 미칠 것으로 예상되는 레이블 요구 메시지와 레이블 사상 메시지의 처리과정을 분석하여 전체 처리 속도의 기대값을 구해본다. 이를 위하여 각각의 메시지 처리에 필요한 C Language 라인의 수를 서브 블록별로 각각 확인하고, SAR를 드라이브하기 위한 Interrupt Service Routine(ISR)을 수행하는 명령들의 수를 확인한다. 그리고 타겟 보드의 클럭 수를 감안하여 C 언어의 라인 당 처리속도와 명령의 처리속도를 계산하여, 이들을 각각 곱하고 더함으로써 하나의 메시지를 처리하기 위해 요구되는 시간을 다음과 같이 계산한다.

(Total Time To Process 1 Message) = (# of Instruction for ISR) x (Execution Time / Instruction) + (# of C Lines for each Message) x (Execution Time / C line)

이 시간에 GSMP의 처리, Routing, Management Overhead를 25% 추가하여 계산하면, 하나의 메시지를 처리하는데 요구되는 시간의 기대값이 이러한 구현에서 대략 2.92 msec 임을 알 수 있고, 따라서 구현된 Unsolicited로 동작하는 LDP는 2분 정도의 시간 안에, On Demand로 동작하는 LDP는 4분 정도의 시간 안에 40,000개의 라우팅 항목의 처리가 가능할 것으로 예상 된다. 상세한 계산 내역은 표 1에 잘 나타나 있다. 참고로 Message Processing Time 항목에 언급된 주요기능들은 앞의 4.4.2 절에서 기술한 각각의 메시지 처리 절차들의 세부 항목에 해당함을 밝혀둔다.

3. 논의

본 연구를 통해 프로토타입으로 만들어진 LDP는 다음과 같이 상용화를 위해 고려해야 할 점들을 충족시키고 있다.

- 표준의 만족: Label Distribution Protocol은

Execution Time Constraints	
CPU clock cycle	400 MHz
Cycle Time / Each Cycle	2.5E-09
Average # of clock / Instruction (Assume zero wait states for RAM)	2
Execution Time / Instruction	0.000000005
Average Instructions / a C line	5
Execution Time / C line	0.000000025
SCA ISR Execution Time	
E (# of Inst in ISR)	200
Execution Time for ISR	1.0 us
Add 15% as error margin	1.150 us
Message Processing Time (Estimated No. of C lines)	
Main	11
Function Body	8
Session Search = # of Sessions * 3 lines	192
FIB Search (average) = 40000 Entries / 2 * 4 lines	80000
LIB Search (average) = 512 vcc / 2 * 2 lines	512
VCC allocation (average) = 512 vcc / 2 * 3 lines	768
Making Req PDU (FxdHdr+MtsdHdr+FEC)	14
Making Map PDU (FxdHdr+MtsdHdr+FEC+Label)	20
Req Func.	80763
Map Func.	81519
Average C lines for 1 Msg	81141
Total Time for 1 Msg in LDP	2.03 ms
Add 15% as error margin	2.33 ms
Message Handling Capability	
Total time to process 1 message	2,335.104 us
Routing, GSMP, Performance Monitoring, Fault Managment, Conf. Mang	2,918.880 us
Handling Capability (LDP Msg / Sec)	343 messages
# splice / second (On Demand Case)	343 Splice
# splice / second (On Demand Case)	171 Splice
Total time to process 40,000 entries (Unsolicited Case)	1.95 minutes
Total time to process 40,000 entries (On Demand Case)	3.89 minutes

표 1. 성능 분석

IETF의 Working Draft인 draft-ietf-mpls-ldp-06.txt에 맞추어 설계되었고, 그 프로토타입 결과는 프로세스 대수를 사용한 명세함으로써, 표준을 따라서 동작하는가 여부를 검증할 수 있다. 따라서 시장을 선점한 제품과의 상호 연동성을 가질 수 있을 것으로 예상된다.

- 성능과 신뢰성: LDP 및 Software는 Carrier Class 제품의 신뢰성과 성능을 지녀야 한다. 성능을 위해서, 구현 시 LDP 세션을 위하여, Nonblocking TCP session[10]을 사용하였으며, 세션 설정의 재시도는 Exponential Back-off Timer[10]에 따라 이루어진다. 성능 분석 결과, Unsolicited로 동작하는 LDP는 2분 정도의 시간 안에, On Demand로 동작하는 LDP는 4분 정도의 시간 안에 40,000개의 라우팅 항목의 처리가 가능할 것으로 예상된다. 또한 프로토타입된 프로토콜의 상세 명세에 대해 안정성을 검정함으로써 프로토콜의 신뢰성을 보장한다.
- 확장성: LDP는 기존의 상용 망의 라우팅 테이블의 항목수인 40,000 개의 라우팅 항목을 처리할 수 있다[6]. 성능분석을 통해 추후 확장의 애로점이 FIB 의 검색임을 확인하였다. 따라서 FIB 항목의 증가 시 Patricia Tree를 적용함으로써 쉽게 확장할 수 있음을 알아내었다. 또한 새로운 기능의 추가나 표준의 변화에 쉽게 적응시키기 위하여, 모듈화 되어 구성되었으며, 이들 구현된 기능이 프로세스 대수를 사용하여 명세됨으로써, 상호 연동성을 보장하면서 쉽게 기능을 확장할 수 있는 기반을 마련했다.
- 이식성: LDP는 표준 Unix API를 사용하여, 다양한 환경에 쉽게 이식될 수 있도록 작성되었다. 표준 BSD Socket Interface[10]를 사용하여, TCP/UDP Service Access Point 부분이 구현되었다. 또한 하드웨어의 데이타 워드의 바이트 저장 순서와 무관하게 동작될 수 있도록 표준 함수[10]들을 사용하여 작성되었다.

VI. 결 론

본 논문에서 우리는 초고속 인터넷 교환기로 주목받고 있는, 다계층 스위칭 라우터의 구현에 필수적인 MPLS Controller에 있어서, 핵심 시그널링 프로토콜인 Label Distribution Protocol의 설계 및 구현에 관해서 기술했다. 본 고에서 LDP 구현을 위한 MPLS 소프트웨어 시스템의 구조를 제시하고, LDP를 자체를 설명하고, 시스템의 동작 모델을 제시하였다. 또한, 설계 시 고려할 점들을 제시하고, 구현 결과인 데이터 구조들과 메시지 처리 절차들을 제시하였다. 구현된 Label Distribution Protocol은 IETF의 Working Draft인 draft-ietf-mpls-ldp-06.txt 표준에 따라 구현되고, 이를 프로세스 대수적 방법으로 명세함으로써 상호 연동성 확보의 기반을 마련 하였다. 또한 성능 분석 결과 구현된 프로토콜은 Carrier Class 제품에 적용 가능한 성능을 가지고 있다는 점을 확인했다. 즉 이러한 구현 구조를 사용 시, 기존의 상용 망의 라우팅 테이블의 항목수인 40,000 개의 라우팅 항목을, Unsolicited로 동작하는 구현된 LDP는 2분 정도의 시간 안에, On Demand로 동작하는 구현된 LDP는 4분 정도의 시간 안에 처리하도록 구성할 수 있다는 점을 보였다. 더불어

성능 분석을 통해 추후 성능개선 시 수정할 부분을 추출하였다.

그리고 프로토타입된 프로토콜의 상세 명세를 가지고 안정성을 간략히 검정함으로써, 프로토타입된 프로토콜의 신뢰성을 확보하였다. 또한 LDP 세션 설정 시 Nonblocking TCP session[10]을 사용하며, 세션 설정의 재시도는 Exponential Back-off Timer [10]에 따라 이루어지게 함으로써, Wide Area Network에서의 본 LDP 적용 시 지연에 따른 문제점 발생을 방지 하였다. LDP는 BSD Unix API를 사용하여, 다양한 환경에 쉽게 이식될 수 있도록 구현되었다.

참 고 문 헌

[1] 박재현, "MPLS LDP 기술 및 개발 방안", 1999년도 1차 ATM-KIG Workshop - ATM기반 MPLS 기술 발표자료집, pp. 99--116, 5월 1999, 용인.

[2] Goran Hagard and Mikael Wolf, "Multiprotocol Label Switching in ATM Networks," Ericsson Review, No. 1, pp. 32--39, 1998.

[3] L. Andersson, P. Doolan, N. Feldman, A. Fredette, and B. Thomas, "LDP Specification," Internet Draft draft-ietf-mpls-ldp-05.txt, Internet Engineering Task Force, June 1999.

[4] Siamack Ayandeh and Yanhe Fan, "MPLS Routing Dynamics," Internet Draft draft-ayandeh-mpls-dynamics-00.txt, Internet Engineering Task Force, March 1998.

[5] P. Newman, W. Edwards, R. Hinden, E. Hoffman, F. Ching Liaw, T. Lyon, and G. Minshall, "Ipsilon's General Switch Management Protocol Specification Version 2.0," Request For Comments RFC 2297, Internet Engineering Task Force, March 1998.

[6] Mikael Degermark, Andrej Brodnik, Svante Carlsson, and Stephen Pink, "Small Forwarding Tables for Fast Routing Lookups," ACM Sigcomm Computer Communication Review, Vol. 27, No. 4, pp. 3-14, October 1997.

[7] Bruce Davie, Jeremy Lawrence, Keith McCloghrie, Yakov Rekhter, Eric Rosen, and George Swallow, "MPLS using LDP and ATM VC Switching," Internet Draft draft-ietf-mpls-atm-02.txt, Internet Engineering Task Force, April 1999.

[8] R. Milner, *Communication and Concurrency*, Prentice Hall, 1989.

[9] 전자통신연구원, *정보통신 프로토콜 공학*, 전자통신연구원, 1998.

[10] W. Richard Stevens, *UNIX Network Programming, Volume 1: Networking APIs - Sockets and XTI*, Prentice Hall, 1997.

박 재 현(Jae-Hyun Park) 정회원

1988년 2월 : 중앙대학교 전자계산학과 졸업

1991년 2월 : 한국 과학기술원 전산학과 석사

1995년 8월 : 한국 과학기술원 전산학과 박사

1995년 8월~현재 : 삼성전자 정보통신 본부 데이터네트웍 개발팀 ATM 시스템 MPLS 개발담당

<주관심 분야> ATM Switch Arch., 상호연결 네트웍, Multiprotocol Label Switching