Multiprotocol Label Switching System의 Label Distribution Protocol 상세설계 검증

정회원 박 재 현*

Validation of the Detailed Design of the Label Distribution Protocol for the Multiprotocol Label Switching System

Jae-Hyun Park* *Regular Member*

요 약

본 논문에서 Multiprotocol Label Switching 시스템을 위한 Label Distribution Protocol의 개발과 분석에 관해서 기술한다. 먼저 Gigabit Switched Router를 만들기 위해서, 상용화시 Carrier Class 제품에 적용하기 위한 LDP의 구현 시 고려해야 될 사항에 대해 살피고, 상세 설계를 제안 한다. IETF 표준에 의거한 LDP의 구현을 위한 상세 설계는 프로토콜 상태기계의 유도 트리와 프로세스 대수를 사용한 형식적 명세를 사용하여 제시한다. 본 논문에서는 제시된 유도트리와 프로세스 대수를 사용한 프로토콜 동작의 분석을 통해, 구현된 LDP의 상호 연동성과 완전성, 생존성, 도달성, 안전성을 검증한다. 또한 이를 사용하여 구현된 LDP가 기존 상용 제품들과의 연동성과 그 동작의 신뢰성을 확보할 것을 기대한다. 결과적으로 구현된 LDP의 프로토콜 동작들의 검증을 제공한다.

ABSTRACT

In this paper, we describe the development and analyses of the Label Distribution Protocol (LDP) for Multiprotocol Label Switching System. We review the implementation issues that are required to construct the LDP for a gigabit switched router, and propose a detailed design of the LDP. We present the detailed design using the deviation trees of the protocol state machine and a formal specification of the state machine using process algebra. These specifications are based on IETF standard. By analyzing the protocol behaviors of the deviation trees and the formal specification, we prove the interoperatability, the completeness, the liveness, the reachability, and the safety of the implemented LDP, and we expect that the reliability will be improved using these analyses. Using these validations, we expect the implemented LDP will be interoperable with other commercialized products. As a result, we validate the protocol behaviors of the implemented LDP with the proofs.

중요어: MPLS, Label Distribution Protocol, Message Sequence Chart, Deviation Tree, Process Algebra

Ⅰ. 서 론

하향으로 최대 9Mbps의 데이터를 전송할 수 있는 ADSL[11]과 하향 채널당 40Mbps정도의 모뎀이 나와 있으나, 이론적으로는 최대 100Mbps까지 데이터 전송할 수 있는 Cable Modem[12]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 전역적 인터넷 제공자의 가장 가까운 제공 점으로 전송하기 위해서는 공중 데이터 망의 구축이 필요하다.

* 영남대학교 전자정보공학부 정보통신전공 (jaehyun.park@ieee.org)
논문번호:00463-1215, 접수일자: 2000년 12월 15일

이러한 요구에 부응하는 초고속 공중 데이터 망을 구축하기 위해서는 고속 패킷 교환기의 사용이 필수적이며, 이러한 고속 패킷 교환기(Cisco GSR 12000 Series, Alcatel 1000/1100, Ericsson AXD 301/311, Lucent Packetstar 6400)는 기존의 음성 교환기의 가입자의 요구 대역폭인 64kbps에 비해, ADSL과 Cable Modem 같이 대역폭을 많이 사용하면서도 평상히 폭발적인 트래픽 특성을 갖는 인터넷 가입자를 비용면에서 효과적으로 수용할 수 있다. 이러한 고속 패킷 교환기는 소프트웨어로써, 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[2].

시장에서 성공하고 있는 대부분의 공중망 용 제품들은 Internet Engineering Task Force (IETF) 표준인 Multiprotocol Label Switching (MPLS)을 장착하고 있다. ISP 사업자들은 MPLS의 도입으로 인해, 기존의 IPOA, MPOA 과 같은 Overlay Network를 사용하는 경우와는 달리, 망의 확장성, QoS, 그리고 멀티캐스팅과 VPN과 같은 신규 서비스의 수용, 관리성에서 강점을 제공받게 된다[1].

본 논문에서는 Multiprotocol Label Switching (MPLS) 시스템 개발[9]에 있어서, 핵심 프로토콜인 Label Distribution Protocol의 상세 설계를 제시하고 이것을 검증한다. 먼저 IETF의 표준을 만족하는 LDP의 상세설계를 프로토콜 상태 기계의 유도 트리(Deviation Tree)와 프로세스 대수를 사용한 형식적 명세 방법으로 제시한다. 그리고 이들 명세를 분석하여, 구현된 LDP의 상호연동성과 강인성 및 일관성[5]을 분석적으로 검증하고, 신뢰성을 보완한다.

본 논문의 구성은 다음과 같다. II 절에서 MPLS 시스템에 있어서, LDP의 역할을 이해하기 위해, MPLS 시스템의 구조와 동작을 설명한다. III 절에서는 설계 시 고려할 점들을 제시하고, LDP의 상세 설계를 메시지 흐름 도표 (Message Sequence Chart)와 프로토콜 상태 기계의 유도 트리들, 그리고 프로세스 대수를 사용하여 제시한다. IV 절에서 상세 설계를 분석하여, 구현된 프로토콜의 완전성(Completeness)과 강인성(Robustness)을 증명하고, 교착상태 부재와 라이브락(Livelock) 부재, 도달성(Reachability), 생존성 (Liveness), 안전성 (Safety)을 검증하고 결과적으로 중요한 성질인 일관성을 증명한다. 마지막으로 V 절에서 결론을 맺는다.

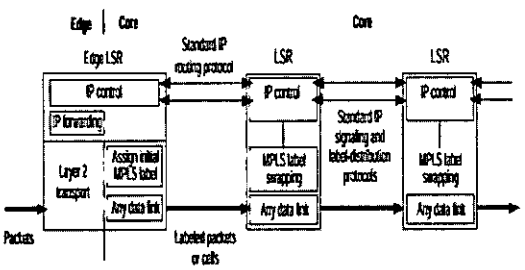

그림 1. MPLS Systems

II. MPLS 시스템 구조와 동작

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

1. MPLS 시스템 구조

MPLS System은 그림 1에서 보이는 바와 같이, 크게 Label Switching Router (LSR)과 Edge Label Switching Router(LER)로 구분된다. LSR은 기존의 인터넷 트래픽을 처리 할 수 있는 Layer 1, Layer 2의 메커니즘을 지닌, 교환기에 LDP 시그널링을 위한 제어 보드를 추가하여 만들어진다[2]. LER은 LSR에 Layer 3를 처리하는 Fast Packet Forwarding Interface와 관련 소프트웨어들이 추가된 것이다.

MPLS controller의 소프트웨어는 그림 2에 보이는 바와 같이, Label Distribution Protocol[3], 그리고 교환기와 LDP간의 연동에 사용되는 General Signal Management Protocol (GSMP)[5], 시그널링 경로를 만드는 데 사용되는 Classical Internet Protocol over ATM (IPOA), UNI 프로토콜과, IP의 Routing Protocol인 OSPF/RIP, 그리고 이 Routing Protocol과 LDP와의 연동 소프트웨어와 교환기 안에서 LSP의 관리를 맡은 소프트웨어 모듈들로 구성된다.

2. MPLS 시스템의 동작

MPLS 시스템은 IPOA를 사용하여 접속한 이웃 제어기들 사이에, IETF 표준인 RIP, OSPF, BGP를 동작시켜 얻어진 IP 라우팅 정보를, 스위치의 2계층 정보로 변환 시켜, 이 전달 경로 설정 정보를 스위칭 하드웨어의 호처리/자원관리 블럭에 보내어, 고속 패킷 전달 혹은 스위칭을 위한 패킷 전달 경로인 Label Switched Path들을 설정하여, 스위치드 라

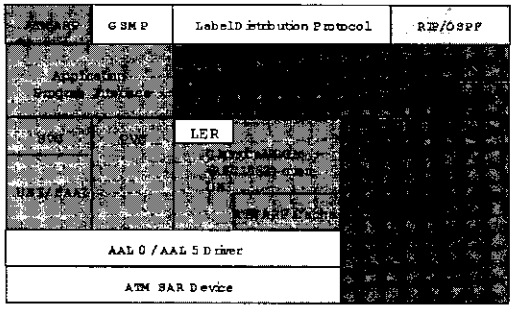

그림 2. MPLS 제어기 프로토콜 구조

우팅을 실현하는 시스템이다[3].

이들 동작들은 라우팅 정보 전파, 레이블 요구, 레이블 사상 단계로 이루어지며, 사상 단계가 끝나면, Label Switched Path를 통해 전송선 속도의 초고속 패킷 전달이 3계층 처리를 거치지 않고 이루어진다. LDP는 이러한 MPLS 시스템의 동작중 레이블 요구와 레이블 사상을 행하는 핵심적인 시그널링 프로토콜이다. 다음절에서는 이러한 LDP의 상세 설계를 메시지 흐름 도표와 프로토콜 상태 기계의 유도 트리들, 그리고 프로세스 대수를 사용하여 제시한다.

Ⅲ. MPLS LDP의 상세 설계

본 절에서는 공중망용 LDP의 구현을 위한 상위 설계의 결과로써, 메시지 흐름 절차 도표를 사용해, LDP의 구체적인 외부 프로토콜 동작의 명세를 제시한다.

1. LDP 상세 설계시 고려할 점들

상용시 효과적인 LDP를 설계하기 위해 고려할 점들은 다음과 같다. 표준 준수에 의한 상호 연동성 보장, Carrier Class 제품의 신뢰성과 성능 제공, 지수적으로 증가하는 망의 크기에 대응하는 확장성, 새로운 장비로의 이식성, 공중망에서의 서비스 중단 없이 서비스의 업그레이드 및 시험이 가능한 유지보수성과 같은 것들이 그것들이다[1]. 본 논문에서는 이들 중 다음의 두가지 점들을 상세 설계시 주안점들로 두고 있다.

- 표준의 만족: 기존의 시장을 선점한 제품과의 상호 연동성을 확보하기 위해, 상세설계된 LDP의 외부 프로토콜 동작들은 IETF의 RFC3036[4]을 따라야 한다.
- 신뢰성: LDP 및 Software는 Carrier Class 제품의 신뢰성을 지녀야 한다.

이제 LDP의 구현을 위한 상위 설계 결과로써, 프로토콜 상태기계의 외부 행위들을 명세하는 MSC 명세를 제공한다. 우리는 이를 다음 장에서 교착상태 부재를 증명하기 위해 사용한다. 또한 상세 설계의 결과로써, 프로토콜의 내부 행위까지 명세한 유도 트리를 제시한다. 이는 구현된 LDP의 상호 연동성 보장과 프로토콜 완전성 증명을 위해 다음 장에서 사용한다.

더불어 프로세스 대수를 사용한 형식적 명세를 제시한다. 이는 프로토콜 동작에서의 라이브락 부재를 증명하고, 초기 상태로부터의 도달성 을 증명하며, 모든 상태들로부터 초기상태로 도달이 가능한지 여부를 나타내는 생존성을 증명하는데 사용된다. 그리고 우리는 다음절에서 이를 메시지 흐름 도표와 더불어 설계된 프로토콜의 안전성을 증명하기 위해 사용한다.

2. MSC를 사용한 LDP 외부 행위 상위 설계

LDP의 프로토콜 외부 행위를 명세하는, LDP의 상위 설계를 제시한다. LDP의 동작모드는 Downstream on Demand Ordered와 Downstream on Demand Independent 그리고 Downstream Unsolicited Ordered와 Downstream Unsolicited Independent로 이루어진다[4]. 다음에서는 메시지 흐름 도표를 사용하여, 이들 각각의 프로토콜 동작 모드의 프로토콜 외부행위들을 명세하고, 프로토콜에 참여하는 개체들 (Entities)의 프로토콜 행위를 중심으로 각각의 MSC를 설명한다.

1) Downstream on Demand Ordered Mode

Downstream on Demand Ordered Mode의 LDP의 동작은 그림 3의 메시지 흐름 도표에서 도식적으로 잘 나타나 있다. LDP는 시스템 초기화 시에 GSMP Master(도표에서 GSMP.m으로 명기)를 통해, 시스템의 MPLS 포트들에 대한 초기화 명령을 내린다. 이것에 대한 응답으로 사용 가능한 Label의 범위를 받는다. 한편 시스템 초기화 시 OSPF가 동작하여 자신의 Routing Information Table을 LDP에 전달하고, 이는 LDP가 Forwarding Information Base를 만드는데 사용된다. 이렇게 LDP가 시스템 초기화를 마치면, LDP 들은 Hello Message를 통해 (IP 주소, Label Space) 정보를 서로에게 알린다. 그리고 나서 IP 주소의 값이 큰 LDP(그림의 예에

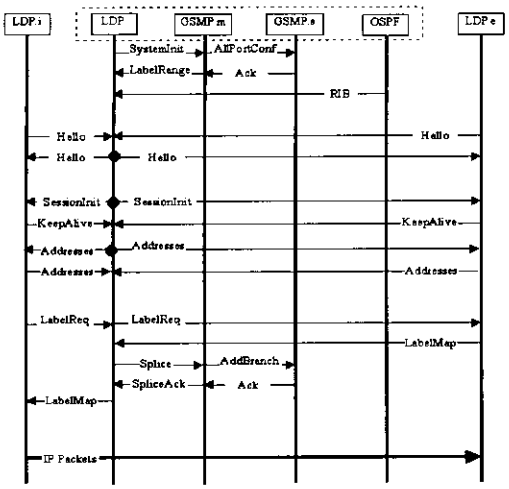

그림 3. Downstream on demand ordered mode로 동작하는 LDP의 Message Flow

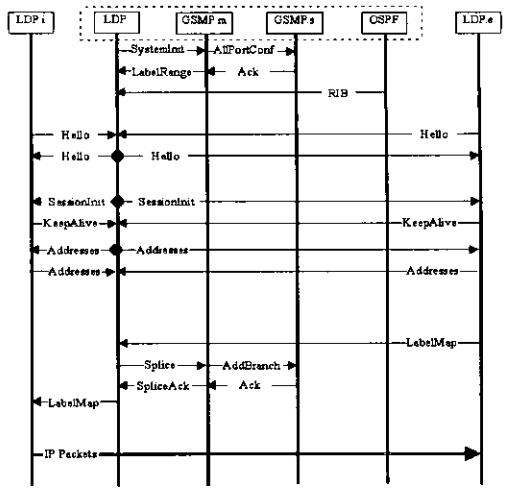

그림 4. Downstream unsolicited ordered mode로 동작하는 LDP의 Message Flow

서는 LDP)가 LDP 세션의 초기화 (LDP Session Initialization) 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 자신의 정합의 IP 주소를 Addresses 메시지를 통해 서로 주고받는다.

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 메시지를 발생시킨다. LSR의 경우는 LDP레이블 요구 메시지를 받아서, 자신의 Forwarding Information Base (FIB)를 확인한 다음, 상태를 변화시키고, 레이블 요구 메시지를 발생시킨다. 그러면 Egress LER의 LDP (LDP.e)는 레이블 요구 메시지를 받아서 기존의 라우터로 나가는 Egress LSP 중 자신이 설정할 부분을 완성하고, 레이블 사상 메시지를 발생시킨다. LSR의 LDP (LDP)는 돌아오는 레이블 사상 메시지를 처리하여, 자신의 데이타구조에 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 메시지를 발생시켜 돌려준다. 상기의 단계들을 거쳐 성립된 LSP를 거쳐 IP 패킷이 전달된다.

2) Downstream Unsolicited Ordered Mode

Downstream Unsolicited Ordered Mode의 LDP의 동작은 그림 4의 메시지 흐름 도표에서 도식적으로 잘 나타나 있다. 앞 절에서 설명한 Downstream on Demand Ordered Mode의 LDP의 시스템 초기화 동작과 Hello 및 LDP 세션 초기화 그리고 Addresses 메시지 흐름 과정과 동일한 과정을 통해서 초기화 된다. 이 초기화 절차를 끝낸 후, 다음과

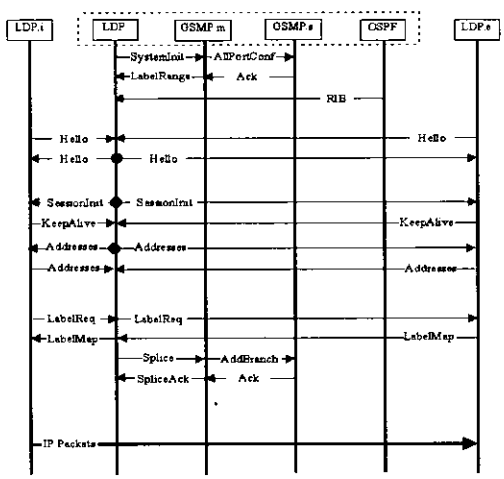

그림 5. Downstream on demand independent mode로 동작하는 LDP의 Message Flow

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 발생시킨다. LSR의 LDP (LDP)는 돌아오는 레이블 사상 메시지를 처리하여, 자신의 데이타 구조에 반영하고, LSP의 실현을 위해, 교환기 안의 패킷 전달 경로인 스플라이스를 스위치에 설정하도록, GSMP Master (GSMP.m)를 사용하여 Add Branch 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 메시지를 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 5의 메시지 흐름 도표에서 도식적으로 잘 나타나 있다. 동일한 초기화 절차를 끝낸 후, 다음과 같은 메시지 흐름들을 통해 LSP를 성립시킨다. Egress LER의 LDP를 제외한 모든 LDP(그림에서 LDP.i와 LDP로 도시)는 Forwarding Information Base (FIB)에 있는 Forwarding Equivalence Class (FEC)들에 대응하는 레이블 요구 메시지를 발생시킨다. LSR의 경우는 LDP 레이블 요구 메시지를 받아서, 자신의 Forwarding Information Base (FIB)를 확인한 다음, 즉각 대응하는 입력 레이블을 할당해서 상태를 데이터 구조에 저장한 다음, 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 발생시킨다. LSR의 LDP (LDP)는 돌아오는 레이블 사상 메시지를 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, 상기의 단계들을 거쳐 성립된 LSP를 거쳐

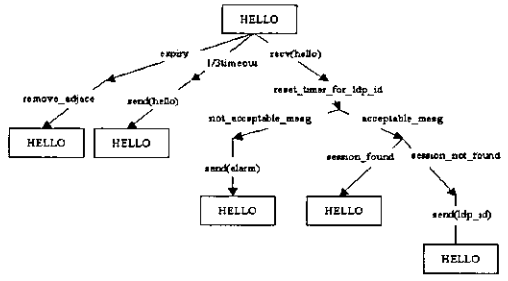

그림 6. Hello Protocol의 유도 트리

IP 패킷이 전달된다. 스플라이스의 설정에 실패하게 되면, 이전에 메시지를 주고받은 세션으로 레이블 취소 메시지를 발생시켜 보내 준다.

4) Downstream Unsolicited Independent Mode

Downstream Unsolicited Independent Mode의 LDP의 동작은 그림 6의 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 담은 레이블 사상 메시지를 업스트림의 MPLS 라우터로 보낸다. LSR의 LDP (LDP)는 레이블 사상 메시지를 처리하여, 할당된 레이블 정보를 자신의 데이타 구조에 저장하고, 처리 상태를 저장한 다음, 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, 상기의 단계들을 거쳐 성립된 LSP를 거쳐 IP 패킷이 전달된다. 스플라이스의 설정에 실패하게 되면, 이전에 메세지를 주고받은 세션으로 레이블 취소 메시지를 발생시켜 보내 준다.

2. 유도 트리를 사용한 LDP 행위 상세 설계

이제 LDP의 프로토콜 내부 행위까지 기술하는 상세 설계 결과인, LDP 상태기계의 상세한 동작을 상태 기계의 분석기법인 유도 트리(Deviation Tree)를 사용하여 제시한다. 이 방법은 프로토콜의 상태 기계를 상태들과 이들간의 모든 가능한 프로토콜 행위들에 의한 천이로 표현한다. 또한 다음절에서 이를 이용하여, 프로토콜 상태 기계의 완전성

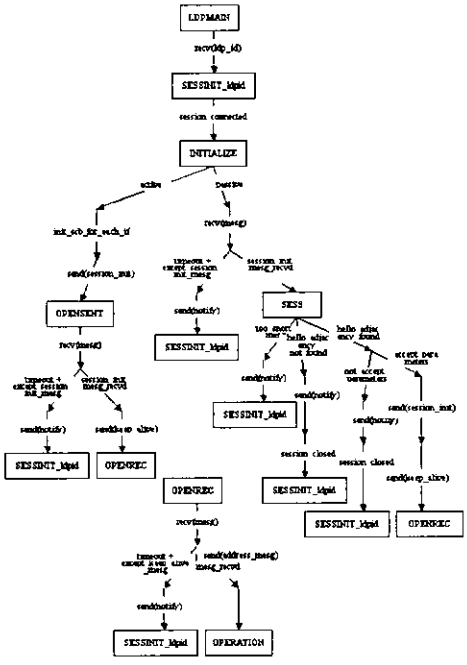

그림 8. LDP의 유도트리

(Completeness)의 증명을 위해 사용한다. 결과적으로 구현된 LDP의 상호 연동성 보장을 위해 사용한다.

표준에 의하면 LDP는 HELLO 상태와 공존하는 LDPMAIN 상태로 정의할 수 있으며, HELLO 프로세스에서 획득한 ldp_id를 LDPMAIN 프로세스에 전달하는 내부 연결이 있다. 그림 7에 도시된 유도트리에서, HELLO 프로세스는 [4]에 정의된 바를 그대로 표현한 것이다.

INITIALIZE 상태, OPENSENT 상태, OPENREC 상태, 그리고 OPERATION 상태는 [4]의 연결 초기화 상태 천이도 (Session Initialization State Transition Diagram)에서 정의된 바와 의미가 동일하다. SESSINIT_ldpid는 [4]에 정의된 NON EXISTENT와 동일한 상태로, 시그널링 세션 초기화을 위한 LDP ID [4]를 받았을 때 천이되는 상태를 나타내고 있다. SESS로 시작되는 프로세스는 세션 초기화 메시지를 받았을 때, 메시지 처리과정을 명세하고 있다. OPERATION 상태로부터 시작되는 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 상태로부터 시작되는 프로세스은

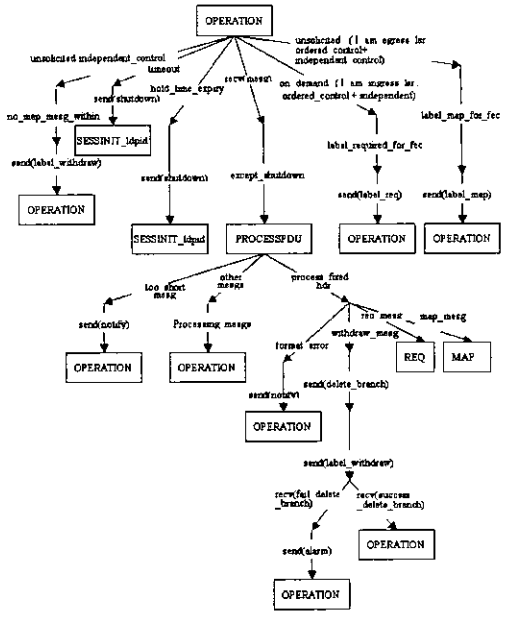

그림 7. LDP Operation State부터의 유도트리

LDP 메시지를 받았을 때, 각 메시지의 PDU를 처리하는 과정을 명세하였다. 이 명세에서 우리는 각각의 예외적인 경우가 어떻게 처리되는 지를 볼 수 있다.

REQ 상태로 시작되는 프로세스는 레이블 요구 메시지의 처리과정을 상세히 명세하고 있으며, MAP 상태로부터 시작하는 프로세스는 레이블 사상 메시지의 처리과정으로 REQ와 대칭적임을 밝혀둔다.

이 명세들은 천이를 일으키는 동작들이 구현에 사용된 함수 수준의 프로토콜 행위를 표현하고 있어, 구현 시 바로 적용이 가능하다. 또한 현재 LDP 표준[4]에 의거하여 작성되었기 때문에, 프로토콜 적합성 검증 및 보완을 위해서 사용한다. 다음절에서 이 유도트리를 프로세스 대수 기법으로 재기술하고, 이를 사용하여 증명한다.

3. 프로세스 대수를 사용한 LDP행위 상세 설계

본 절에서는 유도트리를 대수 형식적 기술 기법인 프로세스 대수(Process Algebra)의 일종인 Labeled Transition System[6,7,8]을 사용하여 재기술한 명세을 제시한다.

이 명세에 기술된 정의들은 현재 LDP 표준안[4]에 의거하여 작성되었기 때문에, 프로토콜 적합성 검증을 위해서 사용될 수 있다. 그리고, 이들 정의들은 상세 설계한 LDP의 라이브락 부재, 초기 상태

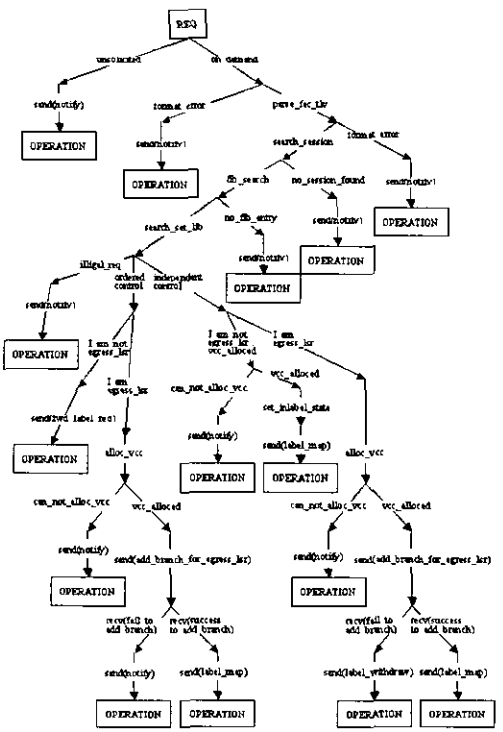

그림 10. LDP Request의 유도 트리

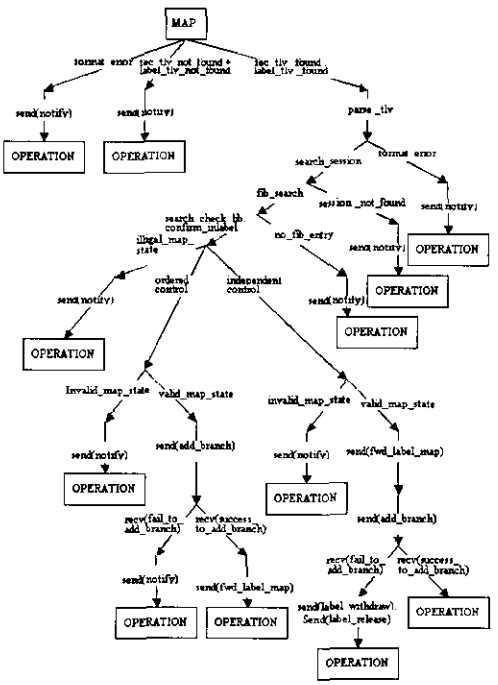

그림 9. LDP Map의 유도 트리

로부터의 도달성, 초기 상태로의 도달여부인 생존성을 증명하기 위해 사용한다. 따라서 본 명세는 LDP의 안전성 증명의 주축이라 할 수 있다. 또한 본 명세는 구현에 사용된 함수 수준의 프로토콜 행위를 표현하고 있다. 따라서 이것은 구현을 위해 바로 사용할 수 있다.

본 기법은 프로토콜을 상태들과 프로토콜의 행위에 따른 이들 상태들간의 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. 여기서 괄호는 일반적인 대수에서의 의미와 같으며, | 연산은 연산에 참여하는 프로세스가 독립적으로 동작하나 상호 통신을 하는 관계를 표현하고, \ 연산은 내부 통신 통로의 이름을 나타내기 위한 연산자이다.

LDP를 정의하면 다음과 같다. LDP는 HELLO 상태와 공존하는 LDPMAIN 상태로 정의할 수 있으며, HELLO 프로세스에서 획득한 ldp_id를 LDPMAIN 프로세스에 전달하는 내부 연결이 있다. HELLO 프로세스는 [4]에 정의된 바를 그대로 기술하였으며, SESSION_ldpid는 각 세션의 상태를 나타내는 프로세스로써, [4]의 세션 초기화 상태 기계 (Session Initialization State Machine)의 상태들 중 NON EXISTENT에 대응된다. 나머지 INITIALIZE 프로세스, OPENSENT 프로세스, OPENREC 프로세스, 그리고 OPERATION 프로세스는 [4]에 정의된 바와 동일하게 기술하였다.

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 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.

정 의 1 LTS를 사용한 구현 함수 수준의LDP의 명세는 다음과 같다.

```
LDP ≝ (HELLO | LDPMAIN) \ ldp_id

HELLO ≝ (one_third_time_out . send(hello) + recv (hello)) .
reset_timer_for_ldp_id . acceptable_mesg . (session_founded_in_SCB
. HELLO + session_not_founded_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_ldpid

OPERATION ≡ recv(mesg) (except_shutdown. PROCESSPDU +
(timeout + shutdown_msg_received) . send (shutdown) .
SESSINIT_ldpid ) + 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_mesgs . 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 ■
```

Ⅵ. 프로토콜의 강인성 및 일관성 검증

앞 절에서 기술한 구현된 프로토콜의 명세들을 사용하여, 다음과 같은 프로토콜의 특성들을 검증할 수 있다. 이러한 특성들을 증명하여, 프로토콜의 설계 시 요구되는 강인성과 일관성[5]을 증명할 수 있다.

먼저 제안한 메시지 흐름 도표를 사용하여, 개발한 프로토콜의 교착 상태의 부재를 증명하고, 유도트리와 프로세스 대수를 사용한 형식적 명세를 사용하여, 구현된 LDP의 프로토콜 완전성을 보인다.

그 다음 프로세스 대수를 사용한 형식적 명세를 사용하여, 프로토콜 동작에서의 폐흐름들은 모두 생산적 폐흐름임을 보여, 라이브락 부재를 증명하고, 초기 상태로부터의 도달성과 초기 상태로의 도달여부인 생존성을 증명한다. 이들 특성들의 증명을 기반으로 구현된 프로토콜의 안전성을 검증한다.

1. 교착상태 부재 (Deadlock-Free) 증명

제시된 MSC를 기반으로 설계된 프로토콜은 프로토콜의 행위시 교착상태가 발생하지 않음을 보인다. 교착 상태란 프로토콜 행위에 참여하는 개체들이 일어날 수 없는 일이 일어나기를 기다리며 프로토콜의 행위를 실행하지 않는 상태를 의미한다[5]. 따라서 LDP의 표준안[4]에 의하면, 프로토콜에 의해 정의되는 개체들간의 메시지들의 교착 상태가 있을 수 없기 때문에, LDP가 본질적으로 교착 상태를 발생시키지 않음을 보일 수 있다.

정 리 1 LDP는 본질적으로 프로토콜 행위에 참여하는 개체들이 교착상태를 발생시키지 않는다.

증 명: LDP는 다음과 같은 네가지 동작 모드가 있다. 이 네가지 경우들 모두가 메시지들을 순환적으로 상호 기다리는 교착 상태가 발생하지 않는다. 다음에서 각각의 동작모드에 대해서 증명한다.

가) *Downstream on Demand Ordered* 경우

MSC에 의하면, Ingress Router에서 레이블 요청 메시지를 처음 발생시키고, 이러한 메시지들은 Egress 라우터를 향해 나아간다. 여기에 대한 응답으로 Egress Router에서 레이블 사상 메시지를 처음 발생시키고 이것 연이어 발생하는 레이블 사상 메시지들을 발생시킨다. 이러한 메시지들의 연속적인 발생은 레이블들이 할당될 때까지 라우터와 라우터 사이에 순차적으로 발생한다. 이러한 순차적 발생하는 메시지들은 Ingress Router에서 발생하고 종료하므로, 순환적인 메시지들 발생 고리를 만들 수 없다. 따라서 이 경우는 교착 상태를 본질적으로 만들수 없다. 참고로 구현 측면에서 보면, 이러한 메시지들에 대한 기다림은 구현시 각 메시지의 기다림을 시그널링 채널 별로 나타내어지고 상태 기계에 의해서 표현된다. 그러므로 시그널 채널별로 상태기계를 가지고, 메시지가 채널 별로 처리된다. 따라서 메시지를 기다리면서 다른메시지를 처리하지 않는 식으로 구현되지 않는다.

나) *Downstream on Demand Independent* 경우

MSC에 나타난 바와 같이, 레이블 요구 메시지를 받으면, Egress Router로의 경로상 레이블 할당과는 무관하게, 요구받은 레이블을 즉시 제공하므로 순환적으로 메시지를 기다리는 교착상태는 발생하지 않는다.

다) *Downstream Unsolicited Ordered* 경우

Egress Router부터 시작되어 Ingress Router로 연속적으로 발생하는 레이블 사상 메시지에 의해서 레이블들을 연속적으로 할당된다. 따라서 프로토콜에 참여하는 개체들간에 메시지를 상호 순환적으로 기다리는 프로토콜 행위가 존재하지 않기 때문에 교착 상태는 발생하지 않는다.

라) *Downstream Unsolicited Independent* 경우

Downstream쪽 라우터에서 독립적으로 레이블을 할당하여 주므로, 프로토콜에 참여하는 개체들 간에 상호 순환적 메시지 수신 대기가 발생하지 않는다. 따라서 교착상태는 발생하지 않는다.

따라서, LDP는 본질적으로 교착 상태를 발생시키지 않는다.

2. 프로토콜의 완전성 및 강인성 증명

앞 절에서 기술한 구현된 프로토콜의 유도 트리를 검토해보면, 프로토콜의 완전성 및 강인성을 검증할 수 있다.

정 리 2 상세 설계된 LDP의 유도 트리는 발생가능한 모든 외부 프로토콜 행위들을 명세하여 완전성을 지니고 있으며, 예상되지 않은 문법이나 문맥의 메시지의 수신시 처리하는 프로토콜 행위들을 정의하여 강인성을 지니고 있다.

증 명: 가) *상세 설계된 LDP의 완전성 증명*

상세 설계한 LDP의 프로토콜 행위를 구현에 사용된 함수 수준으로 표현하는 유도 트리들 중 그림 7과 그림 8에 각각 도시된 Hello Protocol의 유도 트리와 LDP의 유도 트리는 IETF의 표준인 RFC3036 [4]의 Hello Protocol 프로토콜 행위 정의와 연결 초기화 상태기계 (Session Initialization State Machine)에 정의된 모든 가능한 이벤트들, 즉 모든 정의된 외부 프로토콜 행위들에 대해서 대응하는 상태로의 천이를 표현하고 있다. 또한 나머지 유도 트리들은 표준에 정의되어 있는 모든 나머지 외부 프로토콜 행위를 명세하고 있다. 이는 표준에 프로토콜 행위 정의 부분을 유도 트리 혹은 유도 트리를 대수적인 형식으로 표현한 LTS와 외부 프로토콜 행위 수준에서 일대일로 사상 시켜 봄으로써 확인이 가능하다. 따라서 상세 설계된 LDP는 RFC3036 표준에 정의된 모든 프로토콜 행위들에 반응을 하도록 설계되었음을 알 수 있다. 다시 말해서 주요 외부 프로토콜 행위인 메시지 수신에 있어서, 수신 메시지의 형식과 의미가 프로토콜 상태 기계의 상태와 일관성이 있을 때 처리하는 절차를 모두 정의하였다. 그리고 또 하나의 주요 외부 행위인 타임아웃에 대응하는 프로토콜 행위를 모두 정의하였다. 따라서, 상세 설계된 LDP가 완전성 (Completeness)를 가짐을 알 수 있다.

나) *상세 설계된 LDP의 강인성 증명*

강인성은 예상되지 않은 이벤트의 발생에 대해 타당한 행위를 하도록 설계되어야 하는 것을 의미한다[5]. LDP의 외부 이벤트는 바로 메시지의 수신이고, 상세 설계된 LDP는 모든 수신된 메시지의 형식과 의미가 상태기계의 상태와 일관성이 없을 때 처리하는 절차를 모두 정의하였다. 즉, 그림 7, 8, 9, 10, 11에서, 보이는 바와 같이, 상세 설계된 유도 트리에는 다음과 같은 예상되지 않은 메시지의 예외 처리 절차를 적어도 한가지를 정의 하였다.

1. 각 메시지의 수신시, 메시지의 형식을 검사하고 문제 발견 시 응답하는 절차를 정의하였다.

2. 메시지의 의미를 검사해서, 의미가 문맥상 다른 말로 상태기계의 상태와 일관성이 없을 때 처리하는 절차를 정의하였다.

따라서, 상세 설계된 LDP는 강인성을 가지고 있다는 사실을 증명할 수 있다.

따라서 모든 가능한 프로토콜의 이벤트들에 대해서 모두 정의를 하였으므로, 프로토콜은 모든 가능한 이벤트들에 반응하도록 완전성과 강인성을 지니도록 구현할 수 있었다. 또한 상세 설계한 LDP는 LDP 표준에 명세된 모든 이벤트들에 따른 프로토콜 행위와 상태 천이를 정의하였으므로, 표준과의 일치성(Conformance)을 가진다고 하겠다.

3. 상세설계된 LDP의 라이브락 부재와 도달성 그리고 생존성 증명

앞 절에서 제시한 프로세스 대수를 사용한 형식적 명세를 사용하여, 상세 설계된 LDP의 프로토콜 동작들의 폐흐름 (Loop)들이 모두 생산적 폐흐름임을 보여, 상세 설계된 LDP가 라이브락 부재(Livelock-Free)임을 대수적으로 증명하고, 모든 상태들이 초기 상태로부터의 도달 가능하다는 도달성(Reachability)을 지님과 모든 상태들로 부터 초기 상태로의 도달성 (Live -ness)을 가짐을 대수적으로 증명한다. 도달성과 생존성을 증명하기 위해서, 다음과 같은 정의를 도입한다.

정 의 2 기호 A+는 무행위(Empty Action)을 제외한 정의 1에 나타난 프로토콜 행위들에 속하는, 0번 이상의 연산 "." 또는 연산 "+" 이 적용된 프로토콜 행위들로 이루어질 수 있는 행위 리스트들의 집합이다. 그리고, 집합 A+의 하나 이상의 원소들을 담을 수 있는 변수를 하나 이상의 알파벳 소문자들을 기호로 만들어지는 기호로 나타낸다.

따라서 다음과 같이 변수의 개념을 도입하여, 도달성과 생존성을 증명하도록 간략화한 LTS 명세를 제공한다.

보조 정리 1 관심의 대상이 아닌 행위들을 변수로 대치하여, 정의 1의 형식적 명세를 간략화하면 다음과 같다.

LDP ≡ (HELLO | LDPMAIN) \ ldp_id

HELLO ⇨ g . HELLO

LDPMAIN ⇨ h . SESSINIT_ldpid

SESSINIT_ldpid ⇨ i . INITIALIZE

INITIALIZE ⇨ j . OPENSENT + l . SESS + m. SESSINIT_ldpid

OPENSENT ⇨ n . OPENREC + o . SESSINIT_ldpid

SESS ⇨ p . SESSINIT_ldpid + q . SESSINIT_ldpid + r . (s . OPENREC + t . SESSINIT_ldpid)
⇨ u . SESSINIT_ldpid + v . OPENREC

OPENREC ⇨ u . OPERATION + v . SESSINIT_ldpid

OPERATION ⇨ w (x. PROCESSPDU + y . SESSINIT_ldpid) + z . OPERATION + aa . OPERATION + bb . OPERATION + cc . OPERATION
⇨ x. PROCESSPDU + y .SESSINIT_ldpid
+ dd . OPERATION

PROCESSPDU ⇨ ee . OPERATION + ff . B
B ⇨ gg . OPERATION + hh . (ii . MAP + jj . REQ + kk . UNSPLICE + ll . OTHERS)

PROCESSPDU ⇨ ee . OPERATION + ii . MAP + jj . REQ + kk . UNSPLICE + ll . OTHERS

UNSPLICE ⇨ mm . OPERATION

REQ ⇨ nn . OPERATION + oo . D

D ⇨ oo . EGRESSMAP + pp . SENDMAP + qq . EGRESSSENDMAP)

SENDMAP ⇨ rr . OPERATION

EGRESSMAP ⇨ ss . OPERATION + tt . ODR_MAP_EGRESS

ODR_MAP_EGRESS ⇨ tt . OPERATION

EGRESSSENDMAP ⇨ uu . OPERATION + vv . INDP_MAP_EGRESS

INDP_MAP_EGRESS ⇨ ww . OPERATION

MAP ⇨ xx . OPERATION + yy . E

E ⇨ zz . OPERATION + aaa . F + bbb . F'

F ⇨ ccc . OPERATION + ddd . H

H ⇨ eee . OPERATION

F' ⇨ fff . H'

H' ⇨ ggg . OPERATION

OTHERS ⇨ hhh . OPERATION

예를 들어 HELLO 프로세스를 변수를 도입하여, 간략화 하면 다음과 같다.

HELLO ≡ (a + b) . c . (d . HELLO + e . HELLO)
+ f . HELLO
≡ g . HELLO

나머지 식들도 동일한 방법으로 간략화한다.

정 리 3 도달성: 상세 설계된 LDP의 모든 상태들은 초기 상태로부터의 도달 가능하다.

증 명: 상세 설계된 LDP의 프로세스 대수로 나타낸 형식적 명세를 간략화한 보조정리 1에 기술된 명세의 식들의 부호 "≡"의 왼쪽에 나타난 상태를 일반성의 상실함 없이, 다음과 같이 초기 상태로부터 프로토콜 행위 변수가 포함할 수 있는 프로토콜 행위들의 결과로 나타나는 천이로 다음과 같이 대치할 수 있다.

LDP ≡ (HELLO | LDPMAIN) \ ldp_id

HELLO ⇨ g . HELLO

LDPMAIN ⇨ h . SESSINIT_ldpid

SESSINIT_ldpid ≡ i . INITIALIZE

SESSINIT_ldpid . i ⇨ j . OPENSENT + k . (l . SESS + m. SESSINIT_ldpid)

SESSINIT_ldpid . jjjj = n . OPENREC + o . SESSINIT_ldpid

SESSINIT_ldpid . kkkk = p . SESSINIT_ldpid + q . SESSINIT_ldpid + r . (s . OPENREC + t . SESSINIT_ldpid)
= u . SESSINIT_ldpid + v . OPENREC

SESSINIT_ldpid . llll = u . OPERATION + v . SESSINIT_ldpid

SESSINIT_ldpid . mmmm = w (x. PROCESSPDU + y . SESSINIT_ldpid) + z . OPERATION + aa . OPERATION + bb . OPERATION + cc . OPERATION
= x. PROCESSPDU + y .SESSINIT_ldpid
+ dd . OPERATION

SESSINIT_ldpid . nnnn = ee . OPERATION + ff . B
B = gg . OPERATION + hh . (ii . MAP + jj . REQ + kk . UNSPLICE + ll . OTHERS)

SESSINIT_ldpid . oooo = ee . OPERATION + ii . MAP + jj . REQ + kk . UNSPLICE + ll . OTHERS

SESSINIT_ldpid . pppp = mm . OPERATION

SESSINIT_ldpid . qqqq = nn . OPERATION + oo . D

SESSINIT_ldpid . rrrr = oo . EGRESSMAP + pp . SENDMAP + qq . EGRESSSENDMAP)

SESSINIT_ldpid . ssss = rr . OPERATION

SESSINIT_ldpid . tttt = ss . OPERATION + tt . ODR_MAP_EGRESS

SESSINIT_ldpid . uuuu = tt . OPERATION

SESSINIT_ldpid . vvvv = uu . OPERATION + vv . INDP_MAP_EGRESS

SESSINIT_ldpid . wwww = ww . OPERATION

SESSINIT_ldpid . xxxx = xx . OPERATION + yy . E

SESSINIT_ldpid . yyyy = zz . OPERATION + aaa . F + bbb . F'

SESSINIT_ldpid . zzzz = ccc . OPERATION + ddd . H

SESSINIT_ldpid .iii = eee . OPERATION

SESSINIT_ldpid . jjj = fff . H'

SESSINIT_ldpid . kkk = ggg . OPERATION

SESSINIT_ldpid . lll = hhh . OPERATION

그러면 위와 같이 모든 상태들이 최소한 한 식에 나타난다. 따라서, 모든 상태들은 초기상태로 부터 도달 가능하다.

다음에서 구현된 프로토콜의 생존성 증명을 제시한다.

정 리 4 상세 설계된 LDP는 생존성을 가진다. 즉 상세설계된 LDP의 모든 상태들은 초기 상태로의 도달 가능하다.

증 명: 상세 설계된 LDP의 프로세스 대수로 나타낸 형식적 명세를 간략화한 보조정리 1에 기술된 명세의 식들 가운데, OPERATION 프로세스 식을 포함한 OPERATION식 이 전의 모든 프로세스 식들은 초기상태인 SESSINIT_ldpid 상태로 끝나는 항을 적어도 하나이상 가지고 있으므로, 모두 초기 상태로의 도달이 가능하다. 그리고, OPERATION 식 이후의 프로세스 식들은 적어도 하나의 OPERATION 상태로 끝나는 항을 적어도 하나가지고 있는데, OPERATION 상태를 위한 식의 오른쪽 편의 항중 하나의 항이 초기 상태(SESSINIT_ldpid)를 가지므로, 프로토콜 변수가 포함할 수 있는 행위들을 수행한 후, 초기상태로의 도달이 가능하다.

따라서, 모든 상태들은 초기상태로의 도달이 가능하다.

정 리 5 상세 설계된 LDP은 라이브락 부재 (Livelock-Free) 이다. 다시 말해서, 상세 설계된 LDP의 프로토콜 동작들의 폐흐름 (Loop)들이 모두 생산적 폐흐름 이다.

증 명: 정의 1에 제시된, 상세 설계된 LDP의 프로세스 대수로 나타낸 형식적 명세를 앞서 정의 2에서 도입한 프로토콜 행위 변수를 사용하여, 생산적인 최소 하나의 외부 프로토콜 행위 만을 남기고 간략화 하면 다음과 같다.

LDP = (HELLO | LDPMAIN) \ ldp_id

HELLO = a . send(hello) . b . HELLO + c . send(ldp_id) . HELLO) + d . remove_adjacency . HELLO

LDPMAIN = recv(ldp_id) . SESSINIT_ldpid

SESSINIT_ldpid = e . INITIALIZE

INITIALIZE = f . send(session_init) . OPENSENT + g . recv(mesg) . (h . SESS + i . SESSINIT_ldpid)

OPENSENT = j . send (keepalive) . OPENREC + k . send(notify). SESSINIT_ldpid

SESS = l . send(notify) . SESSINIT_ldpid + m . session_closed . SESSINIT_ldpid + n . send(keep_alive) . OPENREC + o . session_closed . SESSINIT_ldpid

OPENREC = o . send(address_msg) . OPERATION + p . send (notify) . SESSINIT_ldpid

OPERATION = recv(mesg) . q . PROCESSPDU + r . send (shutdown) . SESSINIT_ldpid + s . send(label_req) . OPERATION + t . send(label_map) . OPERATION + v . send(keepalive) . OPERATION + w . send(label_withdraw) . OPERATION

PROCESSPDU = x . send(notify) . OPERATION + y . B
B = z . send(notify) . OPERATION + aa . MAP + bb . REQ + cc . send(label_withdraw) . UNSPLICE + dd . OTHERS)

UNSPLICE = recv(success_delete_branch) . OPERATION + ee . send(alarm) . OPERATION

REQ = ff . send(notify) . OPERATION + gg . send(notify) . OPERATION + hh . send(notify) . OPERATION + ii . C

C = jj . send (notify) . OPERATION + kk send(notify) . OPERATION + ll . D

D = mm . send(notify) . OPERATION + nn . send(fwd_label_req) . OPERATION + oo . EGRESSMAP) + pp . SENDMAP + qq . EGRESSSENDMAP)

SENDMAP = rr . send(notify) . OPERATION + ss . send(label_map) . OPERATION

EGRESSMAP = tt . send(notify) . OPERATION + send(add_branch_for_egress_lsr) . ODR_MAP_EGRESS

ODR_MAP_EGRESS = uu . send(label_map) . OPERATION + vv . send(notify) . OPERATION

EGRESSSENDMAP = ww . send(notify) . OPERATION + send(add_branch_for_egress_lsr) . INDP_MAP_EGRESS

INDP_MAP_EGRESS = xx . send(label_map) . OPERATION + yy . send(label_withdraw) . OPERATION

MAP = zz . send(notify) . OPERATION + aaa . send(notify) . OPERATION + bbb . send(notify) . OPERATION + ccc . E

E = ddd . send (notify) . OPERATION + eee. send(notify) . OPERATION + fff . F + ggg . F'

F = hhh . send(notify) . OPERATION + iii . send(notify) . OPERATION + jjj . send(addbranch) . H

H ≡ kkk . send(fwd_label_map) . OPERATION + lll . send(notify) . OPERATION

F ≡ mmm . send(notify) . OPERATION + send(addbranch) . H'

H' ≡ recv(success_addbranch) . OPERATION + nnn . 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

따라서, 위의 모든 프로세스 식에서 보듯이, 모든 프로세스 식의 상태들은 적어도 하나의 생산적인 외부 프로토콜 행위를 행해야만 각각의 다른 상태들로 천이가 가능하므로, 상세 설계된 LDP의 프로토콜 동작들의 모든 폐흐름 (Loop)들은 생산적 프로토콜 행위를 적어도 하나를 수행하는 폐흐름이다. 따라서, 상세 설계된 LDP은 라이브락 부재 (Livelock-Free) 이다. ■

정 리 6 상세 설계된 LDP은 안전성을 가진다. 다시 말해서, 상세 설계된 LDP은 교착 상태 부재이고, 라이브락 부재 이다.

증 명: 정리 1과 정리 5에 의해서 증명된다.

정 리 7 상세 설계된 LDP은 일관성을 가진다. 다시 말해서, 상세 설계된 LDP는 안전성을 가지고, 적절한 종료 조건들이 만족되어야만, 프로토콜의 행위가 끝난다.

증 명: 정리 6과 정리 2, 3, 4에 의해서 증명된다.

V. 결 론

본 논문에서 MPLS 라우터의 구현에 필수 핵심 시그널링 프로토콜인 Label Distribution Protocol의 상세 설계 및 검증에 관해서 기술했다. IETF 표준 RFC3036에 정의된 LDP 구현을 위해, 표준의 만족과 신뢰성이 상용화를 위한 상세 설계 시 가장 중요하게 고려해야 할 점임을 제시하였다.

이러한 점들을 보장하기 위해, 우리는 먼저 메시지 흐름 도표를 사용하여 상위 설계를 제시하였다. 그리고 LDP의 상태 기계의 유도 트리를 제시하여, 구현에 적용되는 함수 수준의 상세 설계를 제공하였다.

두번째, 제시한 메시지 흐름 도표를 사용하여, 개발한 프로토콜의 교착 상태의 부재를 증명하고, 유도 트리를 사용하여, 구현된 LDP의 프로토콜 완전성 및 강인성을 보였다.

마지막으로, 유도 트리를 일대일로 변환한 프로세스 대수를 사용한 형식적 명세를 사용하여, 프로토콜 동작에서의 폐흐름들은 모두 생산적 폐흐름임을 보여, 라이브락 부재를 증명하고, 초기 상태로부터의 도달성과 초기 상태로의 도달여부인 생존성을 증명하였다. 이들 특성들의 증명을 기반으로 구현된 프로토콜의 안전성 및 일관성을 검증하였다.

결과적으로 구현된 LDP의 동작의 일관성, 강인성, 안전성을 확보하였고, 또한 상용화 시 신뢰성 있는 프로토콜을 확보할 것을 기대한다. 또한 본 논문에서 제시한 검증 기법들은 다른 프로토콜들의 검증에도 사용될 수 있음을 밝혀둔다.

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] 박재현, "Multiprotocol Label Switching System을 위한 Label Distribution Protocol 구현," 한국통신학회논문지, 제24권, 제12B호, pp.2249--2261, 1999 12월.

[3] 박재현, "Multiprotocol Label Switching System을 위한 Label Distribution Protocol 개발 및 안정성 분석," 2000년 제 10회 통신 정보 합동 학술대회, 제2권, pp.521--524, 2000 5월.

[4] L. Andersson, P. Doolan, N. Feldman, A. Fredette, and B. Thomas, "LDP Specification," RFC 3036, Internet Engineering Task Force, January 2001.

[5] Gerard J. Holzmann, *Design and Validation of Computer Protocols*, Prentice Hall, 1991.

[6] C. A. R. Hoare, *Communicating Sequential Processes*, Prentice Hall, 1985.

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

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

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

[10] P. Newman, W. Edwards, R. Hinden, E. Hoffman, F. Ching Liaw, T. Lyon, and G. Minshall, "Ipsilon's General Switch Manage-

ment Protocol Specification Version 2.0," Request For Comments RFC 2297, Internet Engineering Task Force, March 1998.

[11] 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.

[12] ADSL Forum, "General Introduction to Copper Access Technologies," General Tutorial, ADSL Forum, 1998.

[13] Devid Gingold, "Integrated Digital Services for Cable Networks," MS Thesis, Massachusetts Institute of Technology, USA, September 1996.

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~2000년 2월 : 삼성전자 정보통신 본부 데이터네트웍 개발팀 MPLS/ATM 개발담당

2000년 3월~현재 : 영남대학교 전자정보공학부 교수

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