MPLS 네트워크 시뮬레이터의 설계 및 구현

정회원 안 개 일*, 전 우 직*

Design and Implementation of MPLS Network Simulator

Gaeil Ahn*, Woojik Chun* *Regular Members*

요 약

인터넷의 빠른 성장과 멀티미디어 응용과 같이 실시간을 요구하는 서비스의 등장으로 인하여 기존의 인터넷 방식에 획기적인 변화가 요구 되고 있다. 이런 변화를 수용할 수 있는 중요한 대안으로 제안된 기술중의 하나가 바로 MPLS이다. 현재 MPLS 기술을 이용한 연구가 많은 응용분야에서 진행되고 있지만, 새로운 연구를 분석하고 평가할 수 있는 MPLS 시뮬레이터는 아직 개발되지 않았다. 본 논문에서는 이러한 MPLS상에서의 연구를 뒷받침해 줄 수 있는 시뮬레이션 도구인 MPLS 시뮬레이터에 대한 설계와 구현 그리고 응용에 관하여 서술한다. MPLS 시뮬레이터는 레이블 스왑핑 기능, LDP, CR-LDP, 그리고 MPLS 표준에서 정의된 다양한 레이블 분배 기능을 지원한다. 본 논문에서 제안하는 MPLS 시뮬레이터를 통하여 LSP가 어떻게 설정되는지를, 그리고 레이블된 패킷들이 어떻게 동작하는지를 시뮬레이션할 수 있다. MPLS 시뮬레이터의 유용성을 보여주기 위하여 두 가지 응용이 시뮬레이션 된다. 하나는 MPLS 표준에서 정의된 레이블 분배 기법들, Flow Aggregation, ER-LSP, 그리고 LSP Tunnel 등에 대한 시뮬레이션이다. 또 다른 하나는 Haskin에 의하여 제안된 경로 보호/복원 메커니즘에 대한 시뮬레이션이다. 각각의 시뮬레이션 결과는 분석 평가되며, 각 응용들이 어떻게 동작하는지는 그래픽 방법을 사용하여 보여준다.

ABSTRACT

The explosive growth of the Internet and the advent of real-time services such as multimedia application have required an epoch-making change in the Internet. As an alternative that can take in it, MPLS was proposed. Recently, many efforts and activities on MPLS have been initiated, which prompt the necessity of MPLS simulator that can evaluate and analyze newly proposed MPLS techniques. This paper describes design, implementation, and application of a MPLS simulator, which supports label swapping operation, LDP, CR-LDP, and various sorts of label distribution function. It enables researchers to simulate how a LSP is established and terminated, and how the labeled packets act on the LSP. In order to show MPLS simulator's usefulness, two MPLS applications are simulated. One is the basic MPLS function defined in MPLS standards; label distribution schemes, flow aggregation, ER-LSP, and LSP Tunnel. The other is the path protection/restoration scheme proposed by Hakin. The results are evaluated and analyzed, and their behaviors are shown by means of graphical manner.

I. 서 론

인터넷 사용자의 폭발적인 증가와 웹 서비스를 통한 다양한 응용 분야의 등장으로 인하여 인터넷 트래픽의 양은 기하급수적으로 증가하고 있다. 한편, 음성 및 화상 등의 멀티미디어 데이터의 실시간 전송을 요구하는 응용들은 서비스의 질을 보장할 수 있는 새로운 데이터 통신을 요구하고 있다. 이러한 요구사항을 수용하기 위하여 시급히 해결해야 하는 과제는 인터넷 망의 고속화와 품질 보장 그리고 트래픽 제어이다.

Multiprotocol Label Switching(MPLS)[5][8][12]는

* 충남대학교 컴퓨터공학과 프로토콜공학 연구실(fog1@ce.cnu.ac.kr, chun@ce.cnu.ac.kr)
논문번호 : 00325-0816, 접수일자 : 2000년 8월 16일

이런 문제점들을 해결하기 위하여 제안된 기술중의 하나로서 IP 스위칭 기술, Tag 스위칭, ARIS (Aggregate Routed-Based IP Switching) 그리고 CSR(Cell Switch Router)의 개념들을 정리하고 표준화하여 만들어졌다[4].

MPLS는 목적지 주소가 아닌 레이블(label)을 사용하여 다음 홉으로 직접 패킷을 전달할 수 있는 고속의 패킷 스위칭 기능을 제공하며, 또한 QoS를 고려한 Explicit-Routed Label Switched Path(ER-LSP)를 설정할 수 있는 기능도 제공한다. 이러한 MPLS 기술들을 이용하여 트래픽 엔지니어링 및 자원 예약을 통한 보장형 서비스(Guaranteed Service), QoS를 지원하는 VPN 등 차세대 인터넷에서 요구하는 기반 기능을 제공할 수 있는 장점을 가지고 있다.

현재 이러한 MPLS 기술을 이용한 연구가 많은 응용분야에서 진행되고 있지만, 이러한 연구를 뒷받침할 MPLS 시뮬레이터가 아직 개발되지 않았다. 그렇기 때문에 현재 수행되고 있는 MPLS상에서 많은 연구는 실제의 MPLS 망을 구축하여 성능을 평가해야만 하며, 수 많은 라우터로 구성된 실제의 MPLS 망을 구성하는 작업은 시간과 비용의 측면에서 현실적이지 못하다.

본 논문에서는 MPLS 네트워크 시뮬레이터에 대한 설계 및 구현에 관하여 서술한다. MPLS 시뮬레이터의 유용성을 보여주기 위하여 두 가지 응용이 시뮬레이션 된다. 하나는 MPLS 표준에서 정의된 레이블 분배 기법들, Flow Aggregation, ER-LSP, 그리고 LSP Tunnel 등에 대한 시뮬레이션이다. 또 다른 하나는 실질적인 MPLS 응용으로서 Haskin[6]에 의하여 제안된 경로 보호/복원 메커니즘에 대한 시뮬레이션이다. MPLS 네트워트의 경로 보호/복원이란 설정된 LSP내에 링크 에러나 노드에서의 혼잡이 발생한 트래픽을 우회 시키는 것을 말한다[2][15].

본 논문의 구성은 다음과 같다. 먼저, 2장에서는 MPLS의 기본 개념 및 본 시뮬레이터를 구현하기 위하여 사용된 NS 시뮬레이터와 NAM에 대한 간단한 소개를 하며, 3장에서는 MPLS 시뮬레이터의 설계에 대하여 설명한다. 4장에서는 MPLS 시뮬레이터의 내부 구조에 대하여 자세히 설명한다. 5장과 6장에서는 본 논문에서 제안하는 네트워크 시뮬레이터를 사용하여 MPLS 망의 기능을 평가한 몇 가지 사례를 보인다. 7장에서는 MPLS 시뮬레이터의 연구 동향 및 적용 방안을 서술하며, 8장에서 결론을 맺는다.

Ⅱ. 관련 연구

1. MPLS

일반적으로 IP 프로토콜상에서 라우터가 패킷을 다음 홉(hop)으로 전달하기 위해서는 패킷들을 Forwarding Equivalence Classes(FEC)로 분류하고 각각의 FEC를 다음 홉으로 사상하는 기능이 모든 라우터에서 행해진다. 그러나 MPLS상에서는 이러한 일들이 MPLS 망에 들어갈 때 한번만 발생한다. 즉, 어떤 한 FEC에 속하는 패킷은 레이블(label)이라고 하는 식별자가 그 패킷에 삽입되어 레이블된(labeled) 패킷이 되고, 이 레이블된 패킷을 받은 Label Switching Router (LSR)은 네트워크 계층의 헤더를 조사할 필요 없이 곧바로 그 레이블 값을 인덱스로 사용하여 다음 홉을 결정한다. MPLS는 이러한 레이블 스위칭 방식을 사용하기 때문에 패킷을 L3가 아닌 L2수준에서 고속 스위칭할 수 있는 특징을 가지고 있다.

Label Distribution Protocol(LDP)[10]는 LSR이 네트워크 계층의 라우팅 정보를 데이터 링크 계층의 스위치된 경로로 직접 사상함으로써 LSP(Label Switched Path)를 설정할 수 있도록 정의된 프로시저와 메시지들의 집합이다. LDP는 레이블 할당 방식, LSP 트리거 방식, 레이블 분배 모드, 레이블 보유 모드에 따라서 몇 개의 옵션을 정의하고 있다.

먼저, 어느 LSR이 레이블을 할당하느냐에 따라서 상위 LSR 레이블 할당(Upstream Label Allocation), 요구 시 상위 LSR 레이블 할당 (Upstream on Demand Label Allocation), 하위 LSR 레이블 할당 (Downstream Label Allocation), 그리고 요구 시 하위 LSR 레이블 할당 (Downstream on Demand Label Allocation) 방식으로 나누어 진다. 그리고 LSP 설정을 언제 트리거할 것인지에 따라서 데이터 기반 트리거(Data-driven-trigger), 제어 기반 트리거 (Control-driven-trigger), 요구 기반 트리거 (Request-driven-trigger)로 나누어 진다. 또한 이웃하는 LSR에게 레이블 매핑(Mapping) 메시지를 보낼 수 있는 조건에 따라서 독자적인 레이블 분배 제어 모드 (Independent Label Distribution Control)와 순차적인 레이블 분배 제어 모드(Ordered Label Distribution Control)로 나누어 진다. 마지막으로 하위 LSR이 레이블을 분배할 때 이웃 LSR로부터 받은 레이블을 무조건 유지할 것인지에 따라서 자유적인 레이블 보유 모드(Liberal Label Retention Mode)와 보

수적인 레이블 보유 모드(Conservative Label Retention Mode)로 나누어 진다.

Constraint-based Routing LDP(CR-LDP)[1]는 기존의 LDP를 확장하여 Constraint-based Routing에 기반한 명시적인(explicit) 경로 설정을 제공한다. CR-LDP는 Ingress LSR에 의해 시작되는 단대단 CR-LSP의 설정 메커니즘과 자원 예약에 대한 수단을 명시하고 있다.

2. NS와 NAM

본 MPLS 네트워크 시뮬레이터는 UC Berkeley에서 개발한 Network Simulator (NS)[16][17]라는 시뮬레이터를 확장하여 구현하였다. NS는 현재 VINT 프로젝트에서 버전 2까지 개발되었다.

NS는 이벤트 스케줄러와 네트워크 구성 요소들로 구성되어 있다. 이것들은 OTcl과 C++ 언어로 작성되었다. OTcl[11]은 Tcl[3] 스크립터 언어에 객체지향 개념을 추가한 언어이다. OTcl과 C++ 언어를 서로 링크 시킬 수 있도록 하기 위하여 tclcl이 정의되어 있다. NS에서 C++언어는 패킷의 행동 및 상태 정보를 관리할 때 사용되며, OTcl 언어는 시뮬레이션을 위한 이벤트를 설정할 때 사용된다.

NS 시뮬레이터에서 시뮬레이션의 결과는 Network Animator(NAM)[18] 이라는 도구를 사용하여 GUI로 볼 수 있다. NAM은 위상 배치, 패킷 레벨의 애니메이션, 그리고 다양한 데이터의 조사를 제공하는 Tcl/Tk로 작성된 애니메이션 도구이다.

Ⅲ. MPLS 시뮬레이터의 설계

1. 구현 목적 및 범위

본 논문은 MPLS의 핵심 기능을 지원하는 MPLS 시뮬레이터를 개발하여 MPLS상에서의 실험 환경을 쉽게 구축하고, 또한 필요하다면 쉽게 확장할 수 있도록 하여 원하는 실험을 용이하게 시뮬레이션할 수 있게 하는 것을 목적으로 한다.

MPLS 시뮬레이터는 다음을 고려하여 설계하였다.

① 특정 NS 버전에 종속적이지 않도록 NS 시뮬레이터의 코드 수정을 최소화 함
② MPLS 시뮬레이터의 확장성을 지원할 수 있도록 객체지향 개념을 충실히 따름
③ 실제의 LSR을 구현하는 것에도 도움을 줄 수 있도록 설계함

MPLS 시뮬레이터의 구현 범위는 다음과 같다.

① 레이블 스위칭 기능
② LDP/CR-LSP 메시지 전송/처리 기능

LSP 설정에 관련된 MPLS 시뮬레이터의 기능은 다음과 같다.

① LSP 트리거 방식 -- 제어 기반 트리거 및 데이터 기반 트리거 방식 지원
② 레이블 할당 방식 -- 제어 기반 트리거에서는 하위 LSR 레이블 할당만을 지원, 데이터 기반 트리거에서는 상위 LSR 레이블 할당 방식 및 요구 시 하위 LSR 레이블 할당 방식 모두 지원
③ 레이블 분배 제어 -- 독자적인 레이블 분배 제어 모드는 하위 LSR 레이블 할당 방식으로 동작할 때만 지원, 순차적인 레이블 분배 제어 모드는 요구 시 하위 LSR 레이블 할당 방식과 상위 LSR 레이블 할당 방식 모두 지원
④ 레이블 보유 모드 -- 보수적인 레이블 보유 모드만을 지원
⑤ CR-LDP에 기반한 ER-LSP 설정
⑥ Flow Aggregation 기능

2. MPLS 노드의 구조

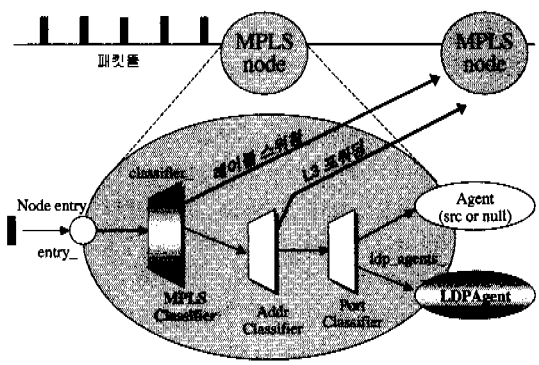


그림 1. MPLS 노드의 구조

MPLS 시뮬레이터는 IP에 기반하여 동작하는 NS 시뮬레이터를 확장하여 구현되었다. NS에서 노드는 에이전트와 분류자(classifier)로 구성되어 있다. 에이전트는 프로토콜의 송신자와 수신자에 관련된 객체이며, 분류자는 다음 노드로 전달하기 위한 패킷의 분류를 책임지고 있는 객체이다.

NS 시뮬레이터의 기본 노드에 'MPLSClassifier' 분류자와 'LDPAgent' 에이전트가 추가된 새로운 MPLS 노드의 구조가 그림 1에 도시 되어 있다. 그림 1에서 *Node entry*는 노드에 도착하는 패킷을 처리할 요소를 알려주는 역할을 한다. 현재 MPLS 시

뮬레이터는 유니케스트 레이블 처리만을 지원하기 때문에 Node 맴버 변수인 *entry_*는 *MPLSClassifier* 하나만을 가리키고 있다. 또한 MPLSNode 멤버 변수인 *classifer_*도 MPLS 노드에 속하는 분류자를 알기 위하여 *MPLSClassifier*를 가리키고 있다.

*MPLSClassifier*는 레이블 스위칭을 하며 레이블된 패킷이 아니면 *AddrClassifier*로 패킷을 보낸다. *AddrClassifier*는 IP 패킷에 대한 L3 포워딩을 담당하며, *PortClassifier*는 노드에 속하는 에이전트들을 구별하는 역할을 담당한다. *LDPAgent*는 LDP 및 CR-LDP 메시지를 송신하거나 수신하여 LSP를 설정하는 역할을 한다. MPLSNode의 멤버 변수인 *ldp_agents_*는 MPLS 노드에 속하는 *LDPAgent*들을 가리킨다.

MPLS노드에서 패킷의 처리 과정은 다음과 같다.

① *MPLSClassifier*는 패킷을 받으면, 먼저 그 패킷이 레이블된 패킷인지 아닌지를 조사한다. 만약 레이블된 패킷이라면, *MPLSClassifier*는 레이블 스위칭 연산을 수행한 후 그 다음 노드로 직접 패킷을 전달한다. 만약 레이블되지 않은 패킷이지만 그 패킷에 대한 LSP가 이미 설정되어 있으면 이 때도 레이블 스위칭을 수행한다. 그 외는 *AddrClassifier*에게 패킷을 전달한다.

② *AddrClassifier*는 패킷의 목적지 주소에 기반하여 L3 포워딩을 수행한다.

③ 만약 그 패킷의 다음 노드가 자기 자신이라면 *PortClassifier*에게 전달한다.

Ⅳ. MPLS 시뮬레이터의 내부 구조

1. MPLS 노드에서 관리하는 테이블 구조

MPLS 노드는 LSP와 관련된 정보를 관리하기 위해서 Partial Forwarding Table(PFT), Label Information Base(LIB) 그리고 Explicit Routing information Base(ERB) 테이블을 가지고 있다. 그림 2는 레이블 스위칭을 위한 간단한 알고리즘과 테이블들의 구조를 도시한 그림이다. PFT는 포워딩 테이블에 속하는 테이블이며, LIB는 현재 설정된 LSP의 정보를 관리하는 테이블이며, ERB는 ER-LSP에 대한 정보를 관리하는 테이블이다. 각 테이블에 있는 'LIBPtr' 필드는 관련된 LIB의 엔트리의 위치 값을 저장하고 있다.

각 테이블이 사용되는 예는 다음과 같다. MPLS 노드가 IP 패킷을 받았을 때는 PFT 테이블을 참조

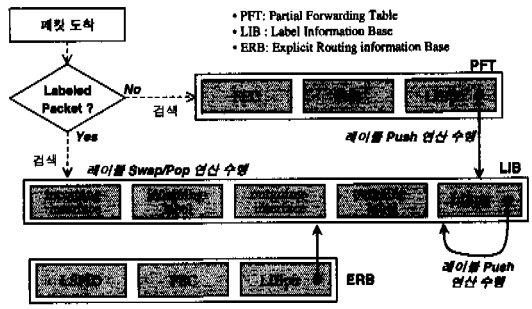


그림 2. 레이블 스위칭을 위한 알고리즘 및 테이블

하여 *LIBptr*이 널이면 L3로 포워딩하며, 그렇지 않으면 *LIBptr*이 가리키는 LIB 테이블의 엔트리에서 *outgoing-label*을 가지고 push 연산을 수행한다. 만약 그 LIB 테이블 엔트리에 있는 *LIBptr*이 널이 아니면 그 *LIBptr*이 가리키는 *outgoing-label*을 찾아 또 다시 push 연산 수행한다. 이 레이블 스택 연산은 *LIBptr*이 널이 아닐 때까지 계속 반복한다. 레이블 연산이 수행 되었으면 *outgoing-interface*가 가리키는 노드로 패킷을 전달한다.

레이블된 패킷을 받았을 때는 패킷의 MPLS shim 헤더의 레이블 값을 인덱스로 하여 LIB 테이블에서 *incoming-label*을 찾는다. 만약 존재하면 MPLS shim 헤더의 label을 *outgoing-label*로 대치하는 swap 연산을 수행하며, 그렇지 않으면 에러이므로 pop 연산을 수행한 후 그 패킷을 L3로 포워딩한다. LIB 테이블의 *outgoing-label* 값이 0인 경우에는 penultimate hop poping을 의미하며 pop 연산을 수행한다. 이러한 레이블 연산을 수행한 후에, 만약 그 LIB 엔트리에서의 *LIBptr*이 널이 아니면 위에서 설명한 것과 마찬가지로 레이블 스택 연산을 수행한다.

ERB 테이블은 ER-LSP에 대한 정보만을 유지할 뿐 패킷의 전달 과정에는 관여하지 않는다. 그래서 특정 트래픽을 설정된 ER-LSP에 바인딩하기를 원한다면, ERB 테이블에 있는 그 ER-LSP에 해당하는 엔트리의 *LIBptr*의 값을 구해서, 그 트래픽의 FEC와 그 구해진 *LIBptr*을 가지고 PFT 테이블에 새로운 엔트리를 생성해야 한다.

2. 레이블 할당 및 분배를 위한 API

그림 3은 MPLS 노드에서 LDPAgent가 LDP 메시지를 받았을 때 그 메시지를 처리하고, 다시 그 LDP 메시지를 다음 LSR에게 전달하기 위하여 발생되는 일련의 API 호출과정을 나타낸 그림이다.

LDPAgent 객체가 LDP 메시지를 받으면 *get-*-

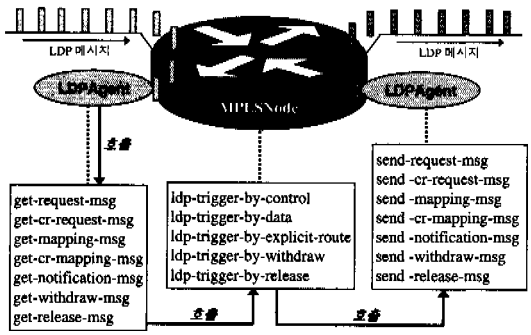


그림 3. LDP 처리를 위한 API 호출 과정

msg API를 호출한다. *get-*-msg* API는 받은 LDP 메시지를 처리한 후, 다음 LSR에게 그 LDP 메시지를 전달하기 위하여 MPLSNode 객체의 *ldp-trigger-by-** API를 호출한다. *ldp-trigger-by-** API는 LDP 메시지를 받을 다음 LSR을 고른 후 그 LSR과 연결된 LDPAgent 객체의 *send-*-msg*를 호출한다.

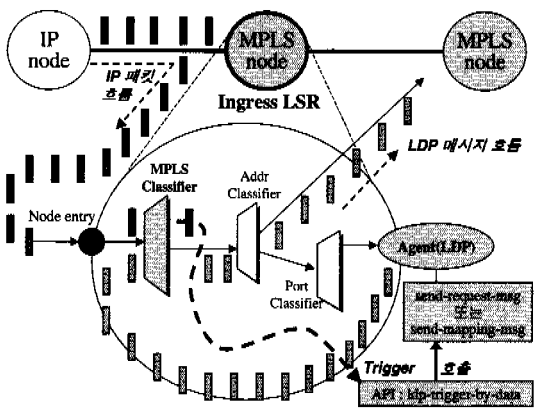


그림 4. Ingress LSR에서 데이터 기반 트리거

그림 4는 Ingress LSR이 IP 패킷을 받았을 때 그 패킷에 대한 LSP 설정을 위하여 LDP를 트리거하는 데이터 기반 트리거의 과정을 보여주는 그림이다. *MPLSClassifier*가 IP 패킷을 받았을 때 그 패킷에 대한 LSP가 설정되어 있는지를 조사한 후 만약 없다면, *ldp-triger-by-data* API를 호출하여 LSP를 설정한다.

3. API 및 구현 환경

다음은 정의된 API 이다.

- MPLSnode -- 새로운 MPLS 노드 생성
- configure-ldp-on-all-mpls-nodes -- 모든 MPLS 노드에 LDP에이전트를 설치
- enable-control-driven -- control-driven trigger로서 동작하도록 LSR을 설정
- enable-data-driven -- data-driven trigger로서 동작하도록 LSR을 설정.
- enable-on-demand -- 요구시 레이블 할당 모드로 동작하도록 LSR을 설정
- enable-ordered-control -- ordered mode로 동작하도록 LSR을 설정
- make-explicit-route -- 새로운 ER-LSP를 설정
- flow-erlsp-binding -- 특정 flow를 설정된 ER-LSP에 바인딩
- flow-aggregation -- fine flow를 coarse flow에 aggregation시킴.
- trace-mpls -- MPLS packet을 트레이스
- trace-ldp -- LDP packet을 트레이스
- enable-reroute -- 링크 에러에 있는 패킷들이 대체 경로로 우회할 수 있도록 LSR을 설정
- reroute-binding -- 특정 ER-LSP가 대체경로로 사용되도록 설정

MPLS 시뮬레이터는 NS의 버전 2.1인 ns-2.1b6을 사용하여 SunOS 5.6 인 Sun 시스템에서 개발되었다.

V. 사례 1: MPLS 기능 시뮬레이션

1. 시뮬레이션 환경

MPLS 기능 시뮬레이션을 위한 MPLS 망은 그림 5에 도시 되어 있다.

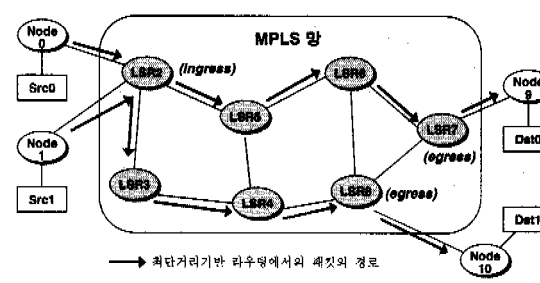


그림 5. 시뮬레이션을 위한 MPLS 망

생성된 노드 중 Node0, Node1, Node9, Node10은 IP 노드이며, 나머지는 MPLS를 지원하는 LSR이다. Node0에는 트래픽을 생성하는 에이전트인 Src0가, 그리고 Node1에는 Src1이 각각 탑재 되었

다. 또한 Node9에는 Src0가 생성한 트래픽을 수신하는 Dst0 에이전트가, 그리고 Node10에는 Src1이 생성한 트래픽을 수신하는 Dst1 에이전트가 각각 탑재 되었다. 최단거리기반 라우팅하에서 Src0가 Dst0를 향하여 패킷을 보내면 그림 5에서 도시 된 것처럼 실선 방향으로 패킷이 전송된다.

그림 5에 도시 된 MPLS 망을 구축한 코드가 그림 6에 도시 되어 있다. 그림 6에서 각 노드는 1Mbit 대역폭과 10ms의 지연 그리고 DropTail 큐를 가진 링크로 연결되었다. 또한 Src0와 Src1 에이전트는 각각 500byte의 패킷을 0.01초 마다 생성하고 있다. LSP 설정방식으로는 제어기반 트리거 방식으로 설정되어 있다.

이러한 환경에서 다음을 시뮬레이션 한다.

- 제어 기반 트리거 방식에서의 LSP 설정/해제
- 데이터 기반 트리거 방식에서의 LSP 설정/해제
- Flow Aggregation
- LSP 터널과 ER-LSP의 설정/해제

```
set ns [new Simulator]

# make IP nodes & MPLS nodes
set Node0 [$ns node]
set Node1 [$ns node]
set LSR2 [$ns MPLSnode]
set LSR3 [$ns MPLSnode]
.....

# connect nodes
$ns duplex-link $Node0 $LSR2 1Mb 10ms DropTail
$ns duplex-link $Node1 $LSR2 1Mb 10ms DropTail
.....

# create traffic source
set Src0 [new Agent/CBR]
$ns attach-agent $Node0 $Src0
$Src0 set packetSize_ 500
$Src0 set interval_ 0.01

set Src1 [new Agent/CBR]
$ns attach-agent $Node1 $Src1
$Src1 set packetSize_ 500
$Src1 set interval_ 0.01

# create traffic sink
set Dst0 [new Agent/Null]
set Dst1 [new Agent/Null]

# create LDP agents on all MPLSnodes
$ns configure-ldp-on-all-mpls-nodes
# set LSP establishment option
$ns enable-control-driven
# trace MPLS and LDP packets
$ns trace-mpls
$ns trace-ldp
```

그림 6. 그림 5의 망을 구축하기 위한 코드

그림 7은 실험하고자 하는 시뮬레이션에 대한 이벤트 스케줄링 코드이다. 그림 7에서 설정된 이벤트는 다음과 같다. 0.1초에 Src0와 Src1은 각각 패킷을 생성하며 0.2초에는 Egress LSR인 LSR7에서 LDP Withdraw 메시지를 사용하여 이미 설정된 FEC 9인 LSP를 해제하며, LSR8에서 FEC 10에 대해 설정된 LSP를 해제한다. 0.3초에 Ingress LSR인 LSR2에서 FEC가 9이거나 10인 트래픽들을 FEC가 6인 LSP에 바인딩시킨다. 0.5초에는 LSR6에서 FEC 6에 대해 설정된 LSP를 해제한다. 0.6초에 Src1은 패킷 생성을 멈춘다. 0.7초에 LSR5, 4, 8, 6, 7을 경유하고 FEC가 7이고 LSPID가 3000인

```
$ns at 0.1 "$Src0 start"
$ns at 0.1 "$Src1 start"
$ns at 0.2 "$LSR7 ldp-trigger-by-withdraw 9"
$ns at 0.2 "$LSR8 ldp-trigger-by-withdraw 10"
$ns at 0.3 "$LSR2 flow-aggregation 9 6"
$ns at 0.3 "$LSR2 flow-aggregation 10 6"
$ns at 0.5 "$LSR6 ldp-trigger-by-withdraw 6"
$ns at 0.6 "$Src1 stop"
$ns at 0.7 "$LSR2 make-explicit-route 7 5_4_8_6_7 3000"
$ns at 0.9 "$LSR2 flow-erlsp-install 9 -1 3000"
$ns at 1.1 "$LSR2 ldp-trigger-by-release 3000"
$ns at 1.2 "$LSR4 make-explicit-route 8 4_5_6_8 3500"
$ns at 1.3 "$LSR2 make-explicit-route 7 2_3_4_3500_7 3600"
$ns at 1.4 "$LSR2 flow-erlsp-install 9 -1 3600"
$ns at 1.6 "$LSR2 ldp-trigger-by-release 3600"
$ns at 1.7 "$LSR4 ldp-trigger-by-release 3500"
$ns at 1.8 "$LSR2 enable-data-driven"
$ns at 2.0 "$Src0 stop"
```

그림 7. 이벤트 스케줄링 코드

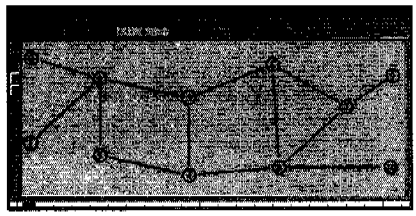
ER-LSP를 설정한다. 0.9초에 FEC가 9인 트래픽을 LSPID가 3000인 ER-LSP에 바인딩시키고, 1.1초에 그 설정된 ER-LSP가 해제한다.

1.2초에 LSR4에서 LSR4, 5, 6, 8을 경유하는 FEC가 8이고 LSPID가 3500인 ER-LSP를 설정한다. 또한 1.3초에는 LSR2에서 LSR2, 3, 4, 3500, 7을 경유하는 FEC가 7이고 LSPID가 3600인 ER-LSP를 설정한다. 명시된 ER 홉에서 3500은 LSPID 이다. 1.4초에 FEC가 9인 트래픽을 LSPID가 3600인 ER-LSP에 바인딩 시킨다. 1.6초에는 LSPID가 3600인 ER-LSP를 해제하며, 1.7초에 LSPID가 3500인 ER-LSP를 해제한다. 마지막으로 1.8초에 LSR2가 데이터 기반 트리거 방식으로 동작하도록 설정하며, 2.0초에 Src0는 패킷 생성을 멈춘다.

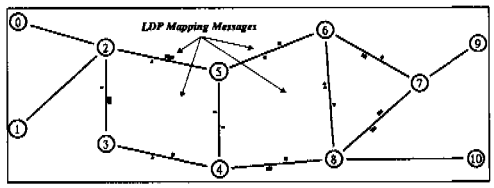
2. 시뮬레이션 결과

그림 6과 그림 7에 있는 코드를 MPLS 시뮬레이터상에서 시뮬레이션하여 NAM을 실행시키면 그림 8과 같은 애니메이션을 볼 수 있다.

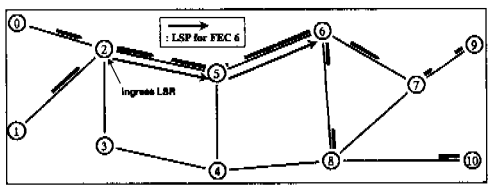
그림 8-(a)은 초기 화면이다. 그림 8-(b)는 제어기반 트리거 방식에서 동작하는 화면을 보여주는 그림이며, 모든 FEC에 대하여 LSP를 설정하기 위하여 LDP Mapping 메시지가 사용되고 있다. 그림 8-(c)는 FEC 9와 10에 대한 트래픽을 FEC 6에 aggregation 시킨 후의 패킷 흐름을 보여주는 그림이다. 이때 패킷들은 Ingress LSR인 LSR2에서 레이블된 패킷으로 변하며, LSR6에서 IP 패킷으로 다시 되돌아온다.



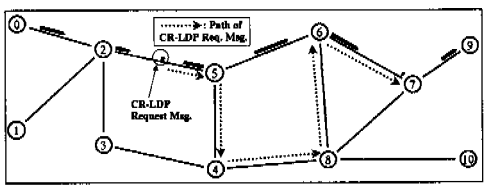
(a) 0.0초: 초기 화면



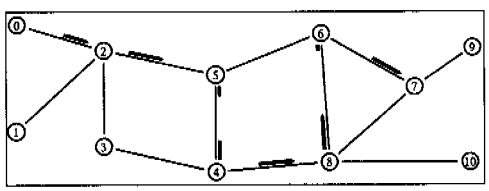
(b) 0.01초: 제어 기반 트리거로 인한 레이블 분배



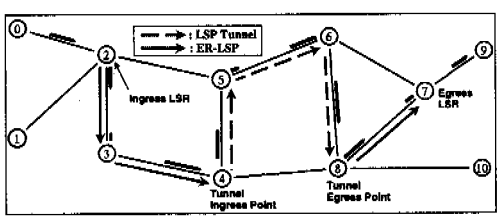
(c) 0.34초: Flow Aggregation



(d) 0.7초: ER-LSP 설정 위한 CR-LDP Request 메시지



(e) 0.97초: 설정된 ER-LSP에 바인딩된 트래픽 흐름



(f) 1.49초: LSP 터널을 사용하는 ER-LSP

그림 8. 시뮬레이션 결과의 에니메이션

그림 8-(d)는 LSPID가 3000인 ER-LSP를 설정하기 위하여 LSR2에서 CR-LDP Request 메시지를 보내는 상황이다. 그림 8-(e)는 FEC 9인 트래픽을 LSPID가 3000인 ER-LSP에 바인딩시킨 후의 트래픽 흐름을 보여주는 그림이다.

그림 8-(f)는 LSPID가 3500인 LSP 터널을 설정한 후에, 그 LSP 터널을 사용하는 LSPID가 3600인 ER-LSP에 FEC 9인 트래픽을 바인딩시킨 후의 패킷 흐름을 보여주는 그림이다. LSP 터널 및 ER-LSP의 설정 과정은 그림 8-(d)의 설정 과정과 비슷하므로 생략하였다.

그림 9는 그림 8-(f)에서 보여진 ER-LSP상에서 패킷들이 어떻게 레이블 스위칭 되는지를 트레이스한 결과이다. 각 필드는 시뮬레이션 시간, 그 패킷을 처리한 노드의 주소, 패킷의 발신지, 목적지 주소, 그 받은 패킷이 레이블된 패킷(L)인지 아닌지(U), incoming-label의 값, 수행된 레이블 연산, outgoing-interface, outgoing-label, TTL값, 그리고 Shim 헤더의 크기를 나타낸다.

표 1. LSP 설정 방식에 따른 LSP 설정 시간 비교

	트리거 메커니즘	LDP 메커니즘	시간 (ms)
모든 LSP 설정	Control-driven	Downstream	43
FEC 9인 LSP 설정	Data-Driven	Downstream-on-demand - independent mode - ordered mode Upstream	 56 77 56
FEC 9인 LSP 해제	Request-Driven	LDP withdraw message LDP release message	42 31

```
1.434000  2(0->9): U -1   Push(ingress)   3  11   32  4
1.448000  3(0->9): L 11   Swap            4  12   31  4
1.462000  4(0->9): L 12   Swap            8  11   30  4
1.462000  4(0->9): L 12   Push(tunnel)    5  12   32  8
1.476000  5(0->9): L 12   Swap            6  12   31  8
1.490000  6(0->9): L 12   Pop(penultimate) 8  0    30  4
1.504000  8(0->9): L 11   Pop(penultimate) 7  0    29  0
1.518000  7(0->9): U -1   L3              -1 -1    -1  0
```

그림 9. 그림 8-(f)에 대한 MPLS 패킷 트레이스

그림 9는 다음과 같이 해석될 수 있다. Ingress LSR인 LSR2에서 Push 연산이 수행되며 LSR3과 LSR4에서 Swap 연산이 일어난다. LSR4는 Tunnel Ingress Point이므로 레이블 스택 연산이 일어난다. LSR5에서 Swap 연산이 일어나며, LSR6과 LSR8에서는 Pop 연산이 수행된다.

표 1에서는 그림 5의 실험환경에서 각 LSP 설정 방식별로 LSP를 설정하거나 해제하는데 걸리는 시

간을 측정한 결과이다.

VI. 사례 2: 경로 보호/복원 시뮬레이션

MPLS 네트워트의 경로 보호/복원이란 설정된 LSP내에 링크 에러나 노드에서의 혼잡이 발생한 트래픽을 우회 시키는 것을 말한다. 그 설정된 LSP가 에러 상태이면, 트래픽은 대체 경로(alternative path)로 우회된다. 본 논문에서는 제안된 경로 보호/복원 메커니즘들 중 유용하다고 알려진 것들 중에서 Haskin[6]에 의하여 제안된 메커니즘을 시뮬레이션 한다. 이 시뮬레이션을 하는 이유는 그 메커니즘에 대한 성능이 아직 평가 되지 않았기 때문에 이것을 함으로써 본 MPLS 시뮬레이터의 유용성을 검증할 수 있기 때문이다.

1. 시뮬레이션 환경

그림 10은 Haskin의 경로 보호/복원 메커니즘을 나타낸 그림이다.

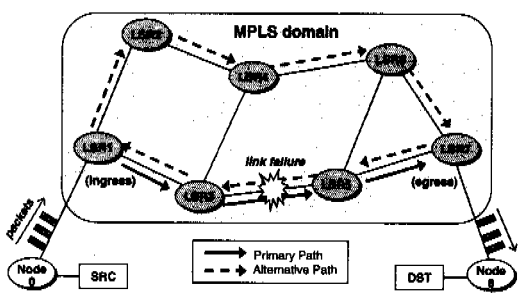


그림 10. Haskin의 경로 보호/복원 메커니즘

기본 경로(Primary Path)는 대체 경로에 의하여 보호되는 경로를 말한다. 그림 10에서 LSR1과 LSR7사이의 실선으로 표시된 LSP가 기본 경로이다.

대체 경로는 다음과 같이 설정된다.

① 목적지 LSR과 원천지 LSR사이에 기본 경로의 반대 방향으로 대체 경로를 설정한다.

② 기본 경로와 중첩되지 않도록 하면서 원천지 LSR과 목적지 LSR사이에 대체 경로를 설정한다. 설정된 두 LSP를 연결한다.

③ 기본 경로상에 있는 LSR이 에러를 탐지하면, 기본 경로인 LSP와 대체 경로인 LSP를 연결 시킴으로써 들어오는 패킷들을 모두 대체 경로로 보낸다. 그림 10에서 LSR3과 LSR5사이의 링크가 에러일 때 LSR3-1-2-4-6-7이 대체 경로이다.

그림 10의 환경은 다음과 같이 구성되어 시뮬레이션 된다. 각 노드는 1Mbit 대역폭과 10ms의 지연 그리고 DropTail 큐를 가진 링크로 연결된다. IP 노드인 Node0에 탑재된 SRC 에이전트는 크기가 200바이트인 패킷을 초당 500Kbit로 생성하여 Node8에 탑재된 DST에게 보낸다. 환경 구성을 위한 시뮬레이션 코드는 그림 6과 비슷하므로 생략한다.

그림 11는 그림 10에 도시된 Haskin의 메커니즘을 시뮬레이션하기 위한 이벤트 스케줄링 코드이다.

```
$ns at 0.1  "$SRC start"
$ns at 0.1 "$LSR1 make-explicit-route  7  2_4_6_7  7776 "
$ns at 0.2 "$LSR7 make-explicit-route  7  5_3_1_7776  7777"
$ns at 0.3 "$LSR1 reroute-binding     8  -1  7776"
$ns at 0.3 "$LSR3 reroute-binding     8  -1  7777"
$ns at 0.3 "$LSR5 reroute-binding     8  -1  7777"
$ns rtmodel-at 0.3 down $LSR3 $LSR5
$ns rtmodel-at 0.5 up   $LSR3 $LSR5
$ns at 0.6 "$SRC stop"
```

그림 11. 이벤트 스케줄링 코드

먼저, 0.1초에 SRC 에이전트는 패킷을 생성하고, LSR1은 FEC 7에 대해 LSR 2,4,6,7을 거치는 LSPID가 7776인 ER-LSP를 설정한다. 0.2초에 LSR7은 FEC 7에 대해 LSR 5,3,1을 거치고 LSPID가 7776인 ER-LSP를 사용하는 ER-LSP(즉, LSPID=7777)를 설정한다. 0.3초에 LSR1, LSR3, LSR5에 그 설정된 ER-LSP를 대체 경로로 사용한다. 0.3초에 LSR3과 LSR5사이의 링크에러가 발생하며, 0.5초에 복구된다. 마지막으로 0.6초에 SRC 에이전트는 패킷 전송을 멈춘다.

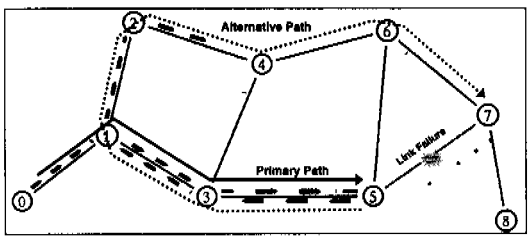
2. 성능 분석

Haskin의 메커니즘을 시뮬레이션한 결과가 그림 12와 그림 13에 나타나 있다. 그림 12-(a)는 링크 에러가 발생한 후의 경로 보호를 그리고 그림 12-(b)는 에러가 복구된 후의 경로 복원을 보여주는 애니메이션이다. 그림 11에서 정의된 ER-LSP를 설정하는 과정은 그림 8과 비슷하므로 생략한다.

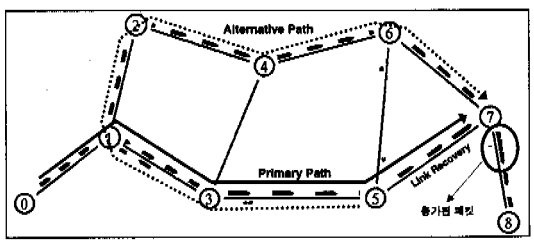
그림 13은 목적지 에이전트(즉 DST 에이전트)가 수신한 패킷에 근거하여 링크 에러의 위치별로 성능을 평가한 그림이다. LSR1과 LSR3사이의 링크, LSR3과 LSR5사이의 링크, 그리고 LSR5과 LSR7사이의 링크가 에러인 경우가 각각 시뮬레이션 되었다.

그림 13-(a)에서 목적지 에이전트는 링크 에러로 인하여 약 310ms 부터 약 60ms간 패킷을 받지 못

하고 있다. 링크가 복구된 약 520ms부터는 기본 경로 및 대체 경로로 오는 패킷을 동시에 모두 받기 때문에 약 40ms 동안 평균보다 더 많은 패킷을 받고 있다. 대역폭의 변화를 보면 목적지 LSR에 가까워질수록, 즉 LSR5와 LSR7사이의 링크가 에러일 때 대역폭의 변화가 가장 심한 것을 볼 수 있다.

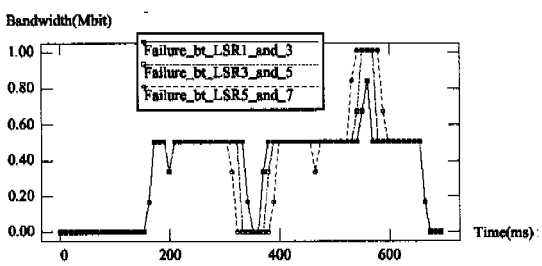


(a) 링크 에러 후 경로 보호

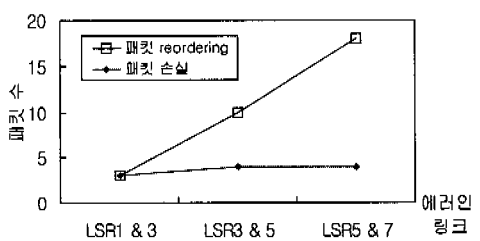


(b) 에러 복구 후 경로 보호

그림 12. Nam에 의해 출력된 에니메이션



(a) 대역폭 변화



(b) 손실된 패킷 및 reordering된 패킷 수의 변화

그림 13. 에러 위치별 성능 평가

이러한 이유는 링크 에러가 목적지 LSR에 가까워질수록 대체 경로의 길이가 길어지기 때문이다. 예를 들어, 그림 12에서 LSR1과 LSR3사이의 링크가 에러일 때의 대체경로는 LSR1-2-4-6-7이고 LSR5와 LSR7사이의 링크가 에러일 때의 대체경로는 LSR5-3-1-2-4-6-7이다. 따라서 LSR5와 LSR7 사이의 링크가 에러일 때의 대체 경로가 가장 길기 때문에 링크 에러가 발생한 후에 목적지 노드(Node8)는 다른 링크 에러 경우보다 더 오랫동안 패킷을 받지 못한다(그림 12-(a) 참조). 에러가 복구된 후에는 그림 12-(b)에 나타나 있듯이 기본 경로(LSR1-3-5-7)로 전달되는 패킷과 대체 경로에 남아 있는 패킷들을 동시에 받게 되는데, LSR5와 LSR7 사이의 링크가 에러일 때의 대체 경로가 가장 길기 때문에 다른 링크 에러 경우보다 더 오랫동안 증가된 패킷을 받는다.

Haskin의 메커니즘은 그림 13-(b)에서 도시되어 있듯이 링크 에러가 발생한 위치에 상관없이 패킷의 손실은 무시할 정도로 아주 적은 장점을 가지고 있다. 그러나 목적지 LSR에 가까워질수록, 즉 LSR5와 LSR7사이의 링크가 에러일 때 re-ordering 패킷의 수가 증가하는 문제점을 가지고 있다. 이러한 이유는 위에서 설명한 바와 마찬가지로 링크 에러가 목적지 LSR에 가까워질수록 대체 경로의 길이가 길어지기 때문에 그림 12-(b)에 나타나 있듯이 에러가 복구된 후 이전에 생성된 대체 경로상에 남아 있는 패킷이 더 늦게 생성된 기본 경로상에 있는 패킷보다 목적지에 늦게 도착하기 때문이다.

Ⅶ. 연구 동향 및 적용 방안

MPLS 시뮬레이터에 관련된 연구 동향 및 MPLS 시뮬레이터의 적용 방안은 다음과 같다.

1) 계층적 도메인에서 시뮬레이션
   본 시뮬레이터의 계층적 주소는 Sean Murphy[14]의 수정에 의해 지원된다.
2) 다양한 라우팅 프로토콜 지원
   MPLS가 특정 라우팅 프로토콜에 종속되지 않는 것처럼 본 MPLS 시뮬레이터도 특정 라우팅 프로토콜에 종속되지 않는다. 현재 ns 시뮬레이터상에서 구현되어 있는 거리 벡터 프로토콜과 링크 상태 프로토콜을 어떠한 수정 없이 모두 지원한다.
3) Virtual Private Network (VPN)
   기존에 가장 많이 제안되고 있는 Network-based VPN[7]은 VPN 정보(예, VPN ID, 위상 정보, 멤버십 정보)를 분배하기 위하여 보통 BGP/

OSPF와 같은 기존의 라우팅 프로토콜을 사용하며, 그 정보를 받은 각각의 서비스 제공자 LSR은 같은 VPN끼리 대응하는 서비스 제공자 LSR까지 터널을 설정한다. 이러한 VPN을 시뮬레이션 하려면, VPN 정보를 분배하기 위한 라우팅 프로토콜의 확장이 필요하다. 그러나 LSP 터널을 설정하는 것은 본 시뮬레이터의 어떠한 수정 없이 그대로 사용할 수 있다.

4) 차등 서비스(Differentiated Service)
ns 시뮬레이터 상에서 차등 서비스의 기능은 Sean Murphy에 의해 개발되었고 또한 본 논문에서 구현된 MPLS 시뮬레이터와도 결합되어 MPLS 망에서 차등 서비스의 시뮬레이션은 가능하다[14].

5) MPLS 경로 보호/복원
링크 또는 노드 에러를 탐지한 LSR이 직접 경로를 보호하는 경우(예 Haskin 기법)에는 6장에서 설명했듯이 본 MPLS 시뮬레이터의 어떠한 수정 없이도 시뮬레이션이 가능하다. 그러나 에러 및 복구를 탐지한 LSR이 그 정보를 특정 LSR에게 통보해 주는 메커니즘에서는 LDP 프로토콜의 수정이 요구된다.

6) MPLS Load Balancing
MPLS망에서 Load Balancing[9][13]이란 Ingress LSR에서 여러 개의 LSP를 미리 설정한 후에 자원의 효율성을 고려하면서 트래픽을 설정된 LSP에 분산시켜 전송하는 방식을 말한다. Ingress LSR에서 특정 트래픽을 두개 이상의 LSP에 매핑하는 메커니즘에서는 본 시뮬레이터의 레이블 스위칭 부분의 수정이 요구된다. 그러나 특정 트래픽을 하나의 LSP에만 매핑한다면 본 시뮬레이터의 수정없이 Load Balancing 알고리즘만 추가하면 된다.

Ⅷ. 결론 및 향후 연구 과제

본 논문에서는 MPLS의 핵심인 LDP/CR-LDP와 레이블 스위칭 기능을 구현함으로써 MPLS에서 정의된 다양한 레이블 할당 및 분배 방식 뿐만 아니라 ER-LSP의 설정 및 Flow Aggregation을 지원하는 MPLS 시뮬레이터를 설계하고 구현하였다. 그리고 간단한 시뮬레이션 사례를 통하여 MPLS 시뮬레이터의 정확성을 검증하였다.

또한 본 시뮬레이터의 유용성을 보여주기 위하여 Haskin에 의해 제안된 MPLS 경로 보호/복원 메커니즘을 시뮬레이션 하였다. 시뮬레이션 결과 Haskin의 메커니즘은 패킷의 손실은 거의 없지만 re-ordering 패킷의 수가 증가하는 문제점을 가지고 있었다. 본 시뮬레이션 결과는 다른 경로 보호/복원 메커니즘의 성능을 분석하기 위한 비교 대상으로써 사용될 수 있을 것이다.

MPLS의 기본 기능만이 구현된 실제의 MPLS 네트워크에서 어떠한 확장 및 수정 없이 모든 MPLS 응용(예, 트래픽 엔지니어링, VPN 등)을 지원하는 것이 불가능한 것처럼 본 논문에서 제안된 MPLS 시뮬레이터만을 가지고 모든 MPLS 응용들을 시뮬레이션할 수는 없다. 그러나 제안된 MPLS 시뮬레이터에는 이미 MPLS의 핵심 기능들이 구현되어 있기 때문에 시뮬레이션하고자 하는 MPLS 응용의 기능만을 본 시뮬레이터에 추가한다면 원하는 결과를 손쉽게 얻을 수 있을 것이다. 그러므로 본 시뮬레이터는 확장을 통하여 실제의 대규모 MPLS 망에서 실질적인 MPLS 응용들을 적용할 때 문제점을 파악하거나, 성능을 평가하기 위한 시뮬레이터로써 유용하게 사용될 수 있을 것이다.

향후 연구과제로써는 MPLS 멀티케스트 지원, QoS를 고려한 CR-LSP 지원, 그리고 제약기반 라우팅(constraint-based routing)을 지원하기 위한 QOSPF 및 QoS 라우팅 알고리즘의 추가이다.

참 고 문 헌

[1] Bilel Jamoussi, "Constraint-Based LSP Setup using LDP," *Internet Draft*, Oct. 1999.

[2] Brad Cain, Loa Andersson, Bilel Jamoussi, "Requirement Framework for Fast Re-route with MPLS," *Internet Draft*, Oct. 1999

[3] Brent B. Welch, "Practical Programming in Tcl and Tk," 2nd edition, *Prentice Hall PTR*, 1997.

[4] Bruce Davie, Paul Doolan, Yakov Rekhter, "Switching in IP Networks: IP Switching, Tag Switching, and Related Technologies," *Morgan Kaufmann Publishers, Inc.* 1998.

[5] Daniel O. Awduche, "MPLS and Traffic Engineering in IP Networks," *IEEE Communications Magazine*, pp.42-47. Dec. 1999.

[6] Dimitry Haskin, Ram Krishnan, "A Method for Setting an Alternative Label Switched Paths to Handle Fast Reroute," *Internet Draft*, Dec. 1999.

[7] E. Rosen, Y. Rekhter, "BGP/MPLS VPNs," *RFC2547*, March 1999.

[8] Eric C. Rosen, Arun Viswanathan, Ross Callon, "Multiprotocol Label Switching Architecture," *Internet Draft*, April 1999.

[9] I. Widjaja and A. Elwalid, "MATE : MPLS Adaptive Traffic Engineering", *Internet Draft* , October 1999

[10] Loa Andersson at al., "LDP Specification," *Internet Draft*, June 1999.

[11] MIT Lab for Computer Science, "OTcl - MIT Object Tcl," URL: http://www.neosoft.com/neowebscript/nws2.3/commands/otcl/README.html.

[12] R. Callon at al., "A Framework for Multiprotocol Label Switching," *Internet Draft*, Sep. 1999.

[13] S. Wright, R. Jaeger and M. Gibson, "Policy-Based Load-Balancing in Traffic-Engineering MPLS Networks", *Internet Draft*, June 2000

[14] Sean Murphy, "DiffServ and MPLS," URL: http://www.teltec.dcu.ie/~murphys/ns-work/mpls-diffserv/index.html

[15] Srinivas Makam, Ken Owens, Vishal Sharma, Changcheng Huang, "A Path Protection/Restoration Mechanism for MPLS Networks," *Internet Draft*, Oct. 1999.

[16] UCB/LBNL/VINT, "ns Notes and Documentation," July 1998, URL: http://www-mash.cs.berkeley.edu/ns.

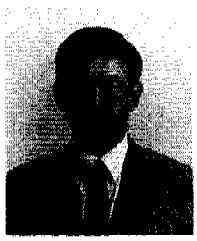
[17] UCB/LBNL/VINT, "ns manual," URL: http://www-mash.cs.berkeley.edu/ns/ns-man.html

[18] UCB/LBNL/VINT, "Network Animator (nam)," URL: http://www-mash.cs.berkeley.edu/nam.

안 개 일(Gaeil Ahn) 학생회원

1993년 2월 : 충남대학교 컴퓨터공학과 졸업(공학사)
1995년 2월 : 충남대학교 컴퓨터공학과 졸업 (공학석사)
1996년 3월~현재 : 충남대학교 컴퓨터공학과 박사과정
<주관심 분야> MPLS, 차등 서비스, 트래픽 엔지니어링,

전 우 직(Woojik Chun) 정회원



1982년 2월 : 서울대학교 컴퓨터공학과졸업(공학사)
1989년 2월 : 델라웨어대학교 (미국) 전산학과 (공학석사)
1992년 2월 : 델라웨어대학교 전산학과 (공학박사)
1984년~1987년 : 한국전자통신연구원
1992년~1993년 : 한국전자통신연구원
1993년 4월~현재 : 충남대학교 컴퓨터공학과 부교수
2000년 4월~현재 : (주)라오넷 대표이사
<주관심 분야> MPLS, 차등 서비스, NAT(Network Address Translation), 홈 네트워킹