ISBN 978-89-5884-947-6 98560

UML 기반의 DRAGON 분석 및 적용

일자: 2007년 10월 31일
부서: 고성능연구망사업단 연구망개발팀
제출자: 홍원택, 김민아, 공정욱
{wthong, petimina, kju}@kisti.re.kr

한국과학기술정보연구원
305-806 대전광역시 유성구 어은동 52번지
TEL (042)869-0676 / FAX (042)869-0679
www.kisti.re.kr

UML 기반의 DRAGON 분석 및 적용

작성자: 홍원택, 김민아, 공정욱

연구망개발팀, 고성능연구망사업단, 한국과학기술정보연구원
{wthong, petimina, kju}@kisti.re.kr

최종수정일: 2007년 10월 31일

Abstract

UML (User Mode Linux) 기반의 DRAGON(Dynamic Resource Allocation via GMPLS Optical Networks)은 다수의 네트워크 장비 및 종단 시스템의 설치 없이 한 대의 시스템 내에서 DRAGON의 기본적인 기능을 테스트할 수 있게 해 준다. 현재 DRAGON의 Control plane이 에뮬레이션될 수 있고, Data plane은 개발이 진행 중이다. 본 보고서에서는 이러한 UML 기반의 DRAGON을 이용하여 종단 시스템에서의 LSP 설정 상태, vlsr에서의 RSVP Resv 메시지 및 RSVP데몬에서 제공하는 로그 등을 확인하였다.

Topics

1. 서론
2. DRAGON 개요
3. UML 이란?
4. UML 기반의 DRAGON
5. 설치 및 테스트
6. 결론
7. 참고문헌
8. 부록

1. 서론

최근, 독점적인 네트워크 사용의 필요성이 증대되면서 람다 네트워킹 기술이 연구 및 교육 망 분야에서 활발히 사용되는 추세이다. 람다 네트워킹을 통해 종단간 대용량 데이터를 전송하려는 그리드, e-Science 응용 연구자들은 고 성능, 저 지연의 특성을 갖는 독점적인 대역을 제공 받을 수 있다. 일반적 관점에서의 람다 네트워킹의 목적은 특정 목적을 같이 하는 연구 그룹들이 그들 자신의 "lightpath"를 제공받는 것이다. 이러한 "lightpath"를 통해 연구자들은 고 대역의 응용에 특화된 독점적인 네트워크를 구성할 수 있게 된다.

"lightpath" 프로비저닝의 요구가 증대되면서 정적 프로비저닝 서비스와는 달리 원하는 시점에 "lightpath"를 동적으로 할당해 주는 서비스에 대한 요구가 증가되었다. 현재 요구 기반 대역 할당 서비스와 관련된 연구 프로젝트들이 있다. DRAGON (Dynamic Resource Allocation via GMPLS Optical Networks) 프로젝트는 광 장비에 기반한 코어/에지 네트워크에서 동적인 패스 프로비저닝 서비스를 제공한다. DRAGON은 주로 GMPLS-enabled 네트워크 환경에 적합한 방법으로 응용에 특화된 토폴로지를 생성하는 부가적인 다양한 API들을 제공한다. UCLP (User Controlled LightPaths) 프로젝트는 일반 사용자의 관점에서 트래픽 엔지니어링이 가능한 방법을 제공한다. 즉, 일반 사용자는 UCLP 소프트웨어를 이용하여 단일 혹은 복수의 도메인에서 Layer 1에서의 광 패스를 동적으로 프로비저닝할 수 있다. 위에서 언급된 연구 프로젝트들은 주로 인터넷 환경에서의 서비스 질을 높이고, 네트워크 대역을 조절할 수 있는 방법론을 제공한다.

본 문서는 DRAGON을 바로 실제 네트워크에 적용하기 어려운 경우, 실험용으로 도입 가능한 UML(User Mode Linux) 기반의 DRAGON에 대한 소개 및 분석에 관한 것이다. 구성은 DRAGON의 개요에 대해 알아보고, UML의 개념을 파악한다. 이어서 UML 기반의 DRAGON의 구성 요소 및 동작 환경을 알아보고, 설치 및 테스트 과정을 수행한다.

2. DRAGON 개요

DRAGON은 Shared Cyberinfrastructure Division (SCI) Experimental Infrastructure Network (EIN) 프로그램으로서 NSF (United States National Science Foundation) 에서 지원을 받는 프로젝트이다. DRAGON 프로젝트는 다양한 기술들로 이루어진 네트워크 에서 특정 목적으로만 사용할 수 있는 경로를 제공하기 위한 기반, 기술들, 그리고 소프트웨어를 개발하는 것이다. DRAGON 프로젝트의 최종 목적은 일반적인 이용에 요구되는 수준의 보안 및 인증을 갖추면서, 여러 네트워크들 사이를 기반으로 하여, 사용자 어플리케이션의 요청에 대하여 직접 동적으로 광 패스를 만들어 주는 것이다.

DRAGON 프로젝트는 GMPLS를 지원하는 광통신 스위치들로 이루어진 네트워크 환경을 가정한다. Label Switching Routers (LSRs) 역할을 하는 광전송 및 스위칭

장치는 패킷(packet), 람다(lambda), 광섬유의 교차 연결 (cross connect) 수준의 특정 목적으로만 사용할 수 있는 네트워크 자원을 제공한다. 이러한 광통신 스위치로 이루어진 네트워크를 위해서 제안된 모든 광학적 가능성은 연결에 기반을 두고 있으며, GMPLS 의하여 자원 관리를 할 수 있다. 실제 종 단간 연결 환경을 제공하기 위하여, 광통신 장치로만 이루어진 네트워크에 실험적인 대학 네트워크를 연결하여 여러 도메인 환경과 여러 다양한 네트워크 기술들로 이루어진 환경을 제공하여 시험한다.
DRAGON 프로젝트를 기반으로 핵심 GMPLS를 지원하는 광통신 네트워크의 동적 제어, 여러 도메인 사이에서의 라우팅, e-Science 어플리케이션들이 종 단간 연결을 위한 광 패스를 만드는 것 등의 시연이 진행되었다. DRAGON은 안정적인 여러 네트워크 사이의 알림과 라우팅, 안정적인 자원할당, 권한 부여와 평가 그리고 단순화된 상호 연결된 API 를 제공한다. DRAGON 프로젝트의 시스템은 다음의 요소들로 구성되고, 추가적인 개발이 이뤄지고 있다.

- NARB (Network Aware Resource Broker) : NARB는 도메인을 표현하며, 사용자 요청이 있을 때 여러 도메인 사이에 라우팅 경로를 제공하기 위해 필요한 기능들을 제공한다. 이것은 IGP/EGP (Interior Gateway Protocol/Exterior Gateway Protocol) 라우팅 프로토콜들과 동작하며, 여러 도메인 사이의 가능한 서비스들을 알리고, 여러 도메인 사이의 경로를 계산하고, 스케줄링, 프락시 알리는 서비스 와 요청한 네트워크 서비스에 대한 보안 기능들을 포함한다. 이러한 NARB 기능이 지원이 되면, 도메인 토폴로지 개념화과 도메인 사이의 알림이 가능하다.

- VLSR (Virtual Label Switch Router) : GMPLS가 지원되지 않는 통신 장치들과 GMPLS를 지원하는 통신 장치들을 통합하여 하나의 GMPLS 자원 할당을 제공하는 종단간 연결을 만든다. VLSR은 표준의 GMPLS 프로토콜을 통신장치에 따른 프로토콜로 바꾸어 주고, GMPLS를 지원하지 않는 장치들을 동적으로 재설정할 수 있게 해준다.

- ASTB (Application Specific Topology Builder) : 어플리케이션이 네트워크 서비스를 요청하기 위해 제공되는 API이다.

- ESA (End-System Agent) : 최종 사용자 시스템에서 돌아가는 소프트웨어로서 자원 할당을 이룬 서비스의 데이터 제어를 종료시키는 역할을 한다. 그리고 최종 사용자 시스템을 위하여 자원 할당 동작을 시작하게 한다.

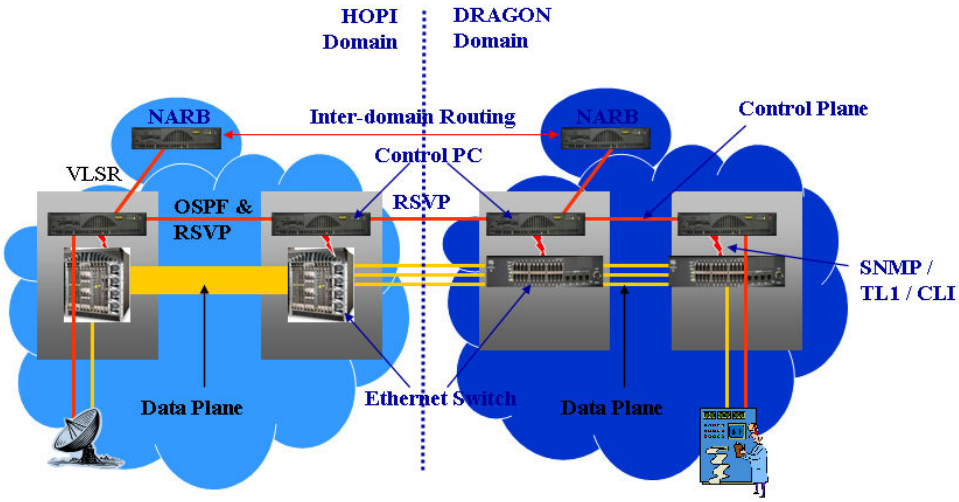

그림 1 DRAGON 프로젝트의 시스템 구성

DRAGON 프로젝트는 정교한 설계와 다양한 기능들을 제공함으로써, 동적인 광 패스를 제공해 줄 수 있지만, 기본적으로 요구되는 최소한의 통신 장치 및 종단 서버들이 필요하게 되어 기본적인 기능들을 테스트하는 데에도 많은 어려움을 유발시킨다. 최근 이러한 번거로움을 피하고, 한 대의 시스템에서 DRAGON의 일부 기능을 시험할 수 있는 방법을 UML(User Mode Linux) 기반 DRAGON의 형태로 제공함으로써 DRAGON을 처음 접하는 사용자들에게 편의를 제공한다.

3. UML(User Mode Linux) 이란?

일반적으로 리눅스 커널은 비디오 카드, 키보드, 하드 디스크 드라이브 등과 같은 하드웨어와 직접적인 통신을 하고, 구동중인 프로그램들은 커널을 통해 하드웨어를 조작할 수 있다. 그림 2에서 좌측은 이러한 일반적인 리눅스 커널을 포함하는 스택을 보여준다. 반면, UML 커널은 하드웨어와 직접적인 통신을 하는 대신 다른 프로그램들처럼 본래의 리눅스 커널과 통신을 한다. 이러한 UML 커널을 이용하면 프로그램들은 본래의 리눅스 커널하에서 운영되는 것처럼 UML 기반으로 동작할 수 있다. (그림 2에서 우측 그림 참고)

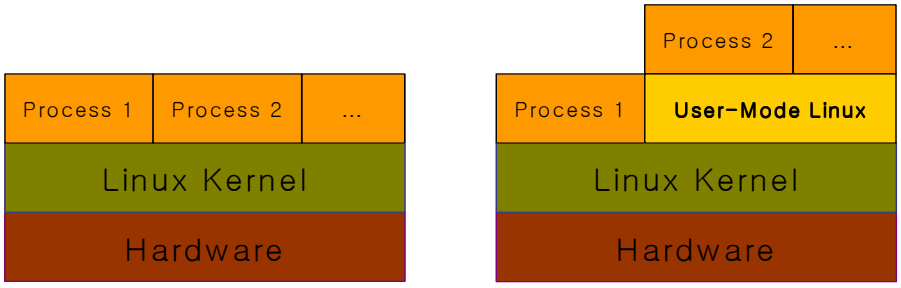

그림 2 일반적인 스택(좌)과 UML이 포함된 스택(우)의 비교

이러한 UML 개념은 여러 가지 장점을 갖고 있다. 설령 UML에 문제가 생긴다 하더라도 본래의 커널은 여전히 안정적으로 구동된다. UML 커널은 root권한이 없는 일반 사용자도 구동시킬 수 있고, 다른 일반적인 프로세스처럼 UML을 디버깅할 수도 있다. 또한 동시에 다양한 버전의 배포판들을 구동시킬 수 있으므로, 테스트 용도로 많은 장점을 제공한다.

4. UML 기반의 DRAGON

DRAGON의 GMPLS 제어 평면은 네트워크에 걸쳐 다수의 유닉스 PC상에서 구동되도록 설계되었다. 이웃하는 노드들과는 raw IPv4 소켓을 기반으로 GRE/IPIP/IPsec 터널(RSVP 시그널링)을 이용하여 통신한다. 이러한 기본 동작을 지원하기 위해서는 다수의 인스턴스들이 하나의 OS상에서 구동될 수 없다. 또한, 하나의 호스트 상에서 구동되는 단일의 인스턴스로는 많은 부분들을 수행할 수 없으므로 UML(User-Mode Linux)를 사용하는 것이 고려되어 적용되었다. 현재 UML 환경에서 DRAGON의 제어평면 소프트웨어를 구동하는 것이 가능하고, 전반적인 GMPLS 네트워크가 단일의 리눅스 PC상에서 시뮬레이션될 수 있다. 다만, 제한범위는 데이터 평면(Data plane)이 아닌 제어 평면(Control plane)만이 지원가능하다.

VNE(Virtual Network Experiments)라고 불리는 공개 소스 패키지가 변경되어 UML환경에서 DRAGON을 구동할 수 있도록 구성되었다. 그림 3에서처럼 XML 파일이 사용되어 시뮬레이션될 GMPLS 네트워크의 토폴로지를 표현할 수 있고, 각각의 UML 인스턴스들이 실험에 사용될 복수개의 호스트들로 매핑된다. 본래의 VNE 소프트웨어는 네덜란드 Amsterdam 대학교의 Systems and Network Engineering research 그룹에서 개발되었다.

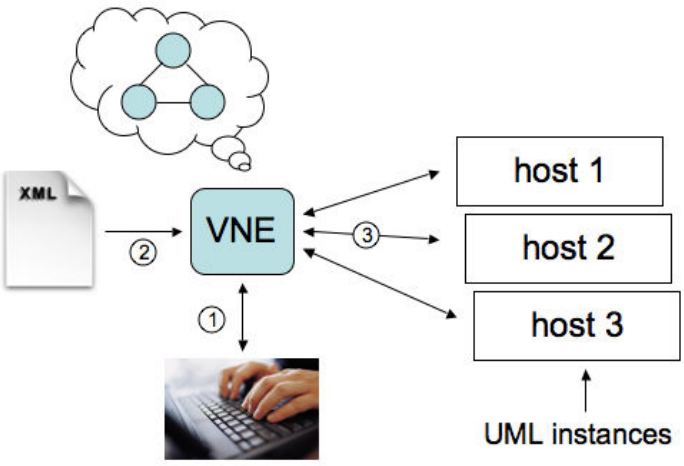

그림 3 VNE (Virtual Network Experiments) 개요

5. 설치 및 테스트

본 절에서는 설치에 앞서 필요한 요구사항들, 설치 과정들을 살펴보고 기본적인 테스트 절차 및 결과를 알아본다.

5.1 요구 사항들

- 리눅스를 구동할 수 있는 x86 기반의 PC
  . 기본 예제들을 구동하는데 요구되는 최소 512MB 메모리
  . 복잡한 시뮬레이션을 위해서는 1GB 메모리
  . 각 UML 인스턴스는 약 32MB의 메모리를 필요로 함
- 리눅스 배포판
  . 테스트는 Debian 3.1("sarge"), Ubuntu 6.10("edgy eft"), Redhat Fedora Core 3에서 수행되었슴. 참고로, 본 문서에서는 Debian으로 수행함.
  . Linux 2.4 또는 2.6 커널을 구동하는 시스템이 권장됨
- 기타 요구사항
  . VNE(Virtual Network Experiments) 소프트웨어 패키지. VNE는 추가적으로 python 2.4, xterm, GNU screen을 요구함.
  . UML kernel binary.
  . DRAGON의 GMPLS 제어평면 소프트웨어 및 기타 유틸리티들을 포함하고 있는 UML root 파일시스템.

5.2 설치 과정 및 설정

- 리눅스 설치
- python 설치
  . Debian/Ubuntu 에서는 *apt-get install python*
- uml-utilities 패키지 설치
  . Debian/Ubuntu 에서는 *apt-get install uml-utilities*
  . 또는 http://user-mode-linux.sourceforge.net/에서 다운로드하여 설치
- xterm 설치
  . Debian/Ubuntu 에서는 *apt-get install xterm*
- GNU screen 설치
  . Debian/Ubuntu 에서는 *apt-get install screen*
- pre-compiled UML kernel binary 설치
  . uml_linux-2.6.20 다운로드
  . 예를들어, /a/uml/debian31와 같은 디렉토리에 파일을 설치
  . pre-compiled UML kernel을 사용하지 않을 경우에는 소스 파일을 컴파일하여 설치
- UML root 파일시스템 설치

. Debian-3.1-x86-root_fs.bz2를 다운로드. 이 파일은 Debian 3.1 x86 root 파일 시스템과 dragon-sw를 포함하고 있슴

. 해당 파일을 /a/uml/debian31과 같은 파일에 설치하고 “bzip2 -d"와 같은 유틸리티를 이용하여 압축해제

- VNE 설치

. VNE-dragon-snapshot.2007Jun15.tar.gz파일을 다운로드하여 VNE(Virtual Network Experiments) 파이선 패키지를 설치.

- 그 외 설정

. 실험을 위해서 root 파일 시스템과 UML kernel binary의 위치를 정함.

* root 파일 시스템의 위치: /a/uml/debian31/Debian-3.1-x86-root_fs
* UML kernel binary의 위치: /a/uml/debian31/linux

. ~/.vnerc 파일을 생성. 이 파일은 uml기반의 dragon을 구동하는데 필요한 기본 환경 정보를 포함하고 있고, VNE 배포판에는 VNEDefaults.cfg 파일안에 파라미터들이 포함됨.

```
[DEFAULT]
kernel     = /a/uml/debian31/linux
filesystem = /a/uml/debian31/Debian-3.1-x86-root_fs
fstype     = cow
xmllint    = /usr/bin/xmllint
screen     = /usr/bin/screen
xterm      = /usr/X11R6/bin/xterm
uml_switch = /usr/bin/uml_switch
memorysize = 128
```

. screen, xterm과 같은 유틸리티들의 위치 정보를 확인하고, 설정 파일을 수정

. 시스템의 메모리 사이즈를 고려하여 "*memorysize*" 파라미터의 수정

. UML kernel binary로의 심볼릭 링크를 설정

* “*ln -s /a/uml/debian31/uml_linux-2.6.20 /a/uml/debian31/linux*" 수행
* 이러한 심볼릭 링크를 이용하면 다른 UML kernel 버전을 테스트할 경우에 변경을 손쉽게 반영할 수 있슴.

. ~/uml 디렉토리 생성

* uml 디렉토리는 COW(copy-on-write) 파일들이 위치하는 곳으로, COW 파일은 각각의 UML 인스턴스들에 대해 생성됨
* uml 디렉토리는 VNE를 구동하기 전에 미리 생성되어 있어야 함.

5.3 사용 방법

- X11 포워딩을 제공하는 UML host로의 ssh 또는 Console에서 접속
- XML 포폴로지 파일을 입력으로 하여 VNE 구동

. 기본적인 LSP를 프로비져닝하는 방법을 포함하는 아래와 같은 예제들을 참고

bridged_triangle.xml —

config-4domains.xml —

config-narb-inter-uni-csa.xml —

config-narb-intra-p2p-csa.xml —

config-narb-intra-uni-csa.xml —

config-stress.xml —

config.xml —

example.xml —

test_config.xml —

testconfig.xml —

- VNE의 대화형 CLI 모드에서 아래와 같은 기본 명령어를 제공.

. config - 콘솔로 설정과 관련된 세부 정보를 제공. 종단 시스템 및 각 네트워크 노드들에 대한 상세 정보를 포함

. start all or start host - UML 인스턴스들을 구동

. xterm all or xterm host - 구동된 UML 인스턴스들을 사용자들이 다룰 수 있도록 xterm들을 실행. UML 인스턴스마다 하나씩 xterm이 실행됨

. stop all or stop host - UML 인스턴스들을 정지

. quit - VNE 환경을 떠남

5.4 테스트

그림 4에서는 위에서 언급된 토폴로지 예제들 중에서 config.xml 파일의 토폴로지 구성을 보여준다. 현재 개발된 UML 기반의 DRAGON 소프트웨어는 Control plane만을 실험해 볼 수 있는 제약이 있으므로, 아래와 같은 구성으로 토폴로지가 형성된다. 향후 진행될 Data plane이 완성되면, vlsr(Virtual Label Switch Router)들이 실제로 제어할 스위치들에 각각 매핑되어 테스트할 수 있게 된다.

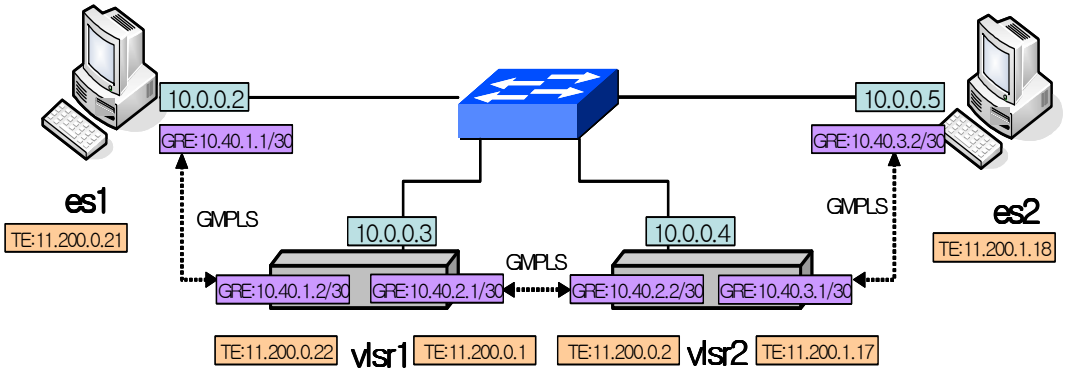

그림 4 config.xml의 토폴로지 구성

그림 5에서는 위의 토폴로지를 구성하는 예제 파일의 내용을 보여준다. 중요 부분은 진한 이탤릭체로 표시하였다.

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE config SYSTEM "dtd-VNEControlCfg.dtd">

<config>
<devices>
<switch name="switch1">
<interface name="eth0" bridge="br0" macaddr="00:00:AA:BB:00:01"/>
<interface name="eth1" bridge="br0" macaddr="00:00:AA:BB:00:02"/>
<interface name="eth2" bridge="br0" macaddr="00:00:AA:BB:00:03"/>
<interface name="eth3" bridge="br0" macaddr="00:00:AA:BB:00:04"/>
</switch>

<host name="es1" dragonconfig="--start --element='p2p-csa' --loopback='10.0.0.2'
--gretunnel='gre1, 10.0.0.3, 10.0.0.2, 10.40.1.2, 10.40.1.1, 10.40.1.0/30, 11.200.0.21,
123'">
<interface name="eth0">
<ipv4 netmask="255.0.0.0" broadcast="10.255.255.255" addr="10.0.0.2"/>
</interface>
</host>
<host name="vlsr1" dragonconfig="--start --element='vlsr' --loopback='10.0.0.3'
--gretunnel='gre1, 10.0.0.2, 10.0.0.3, 10.40.1.1, 10.40.1.2, 10.40.1.0/30, 11.200.0.22,
123, gre2, 10.0.0.4, 10.0.0.3, 10.40.2.2, 10.40.2.1, 10.40.2.0/30, 11.200.0.1, 123'">
<interface name="eth0">
<ipv4 netmask="255.0.0.0" broadcast="10.255.255.255" addr="10.0.0.3"/>
</interface>
</host>
<host name="vlsr2" dragonconfig="--start --element='vlsr' --loopback='10.0.0.4'
--gretunnel='gre3, 10.0.0.5, 10.0.0.4, 10.40.3.2, 10.40.3.1, 10.40.3.0/30, 11.200.0.17,
123, gre2, 10.0.0.3, 10.0.0.4, 10.40.2.1, 10.40.2.2, 10.40.2.0/30, 11.200.0.2, 123'">
<interface name="eth0">
<ipv4 netmask="255.0.0.0" broadcast="10.255.255.255" addr="10.0.0.4"/>
</interface>
</host>
```

그림 5 config.xml 파일(I)

```
<host name="es2" dragonconfig="--start --element='p2p-csa' --loopback='10.0.0.5'
--gretunnel='gre3, 10.0.0.4, 10.0.0.5, 10.40.3.1, 10.40.3.2, 10.40.3.0/30, 11.200.0.18,
123'">
<interface name="eth0">
<ipv4 netmask="255.0.0.0" broadcast="10.255.255.255" addr="10.0.0.5"/>
</interface>
</host>
</devices>
<connections>
<connection>
<device name="switch1" interface="eth0"/>
<device name="es1" interface="eth0"/>
</connection>
<connection>
<device name="switch1" interface="eth1"/>
<device name="vlsr1" interface="eth0"/>
</connection>
<connection>
<device name="switch1" interface="eth2"/>
<device name="vlsr2" interface="eth0"/>
</connection>
<connection>
<device name="switch1" interface="eth3"/>
<device name="es2" interface="eth0"/>
</connection>
</connections>
</config>
```

그림 6 config.xml 파일(II)

그림 7에서처럼 config.xml 파일을 XML 포폴로지 파일로 입력 받아 VNE 환경을 로딩한다. VNE 대화형 CLI 모드에서 "start all"을 실행하여 각각의 UML 인스턴스들을 구동한다. 마지막으로 "xterm all"을 실행하여 각각의 UML 인스턴스를 제어할 수 있는 xterm들을 띄운다. xterm들이 로딩된 상태에서 "enter"를 누룬 후, 디폴트 입력 ID인 "root"로 로그인한다.

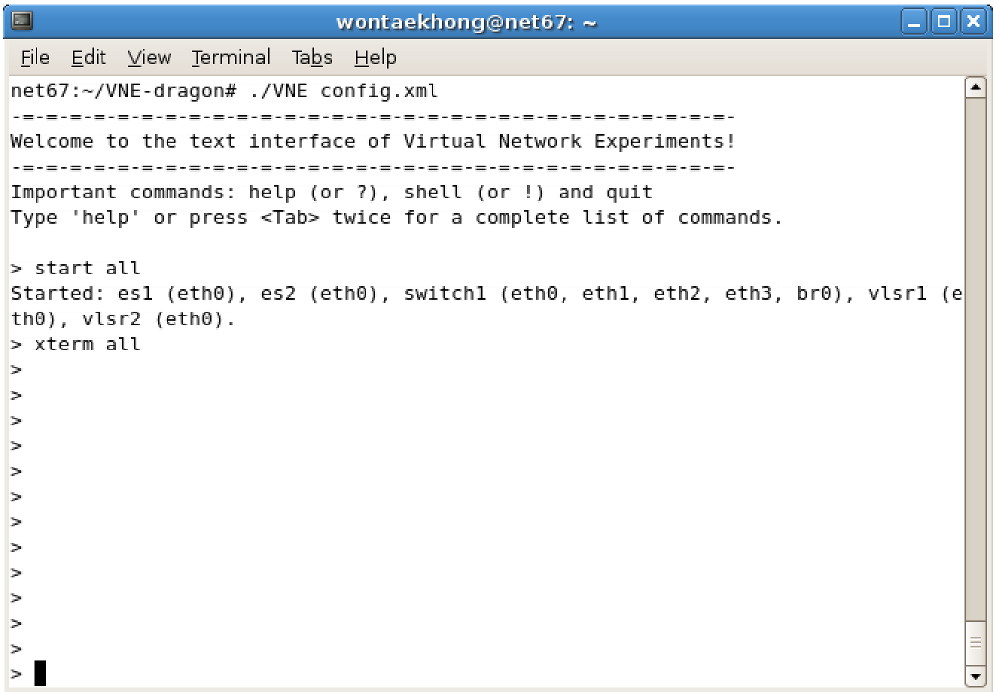

그림 7 VNE를 구동하는 UML 호스트 시스템

그림 8에서 그림 11까지는 각각의 UML 인스턴스들을 제어하는 xterm들을 캡처하여 보여주고 있다. 그림 8에서 아래와 같이 수행하면 간단한 종단간 프로비저닝을 수행할 수 있다.

```
es1% telnet 127.0.0.1 2611
edit lsp test
set source ip-address 10.0.0.2 lsp-id 1000 destination ip-address 10.0.0.5 tunnel-id 2000
set bandwidth gige swcap l2sc encoding ethernet gpid ethernet
set vtag any
exit
commit lsp test
```

"commit lsp test"를 수행하면 LSP가 생성되고, "show lsp test"와 "show lsp"를 이용하여 LSP의 상태를 확인할 수 있다.

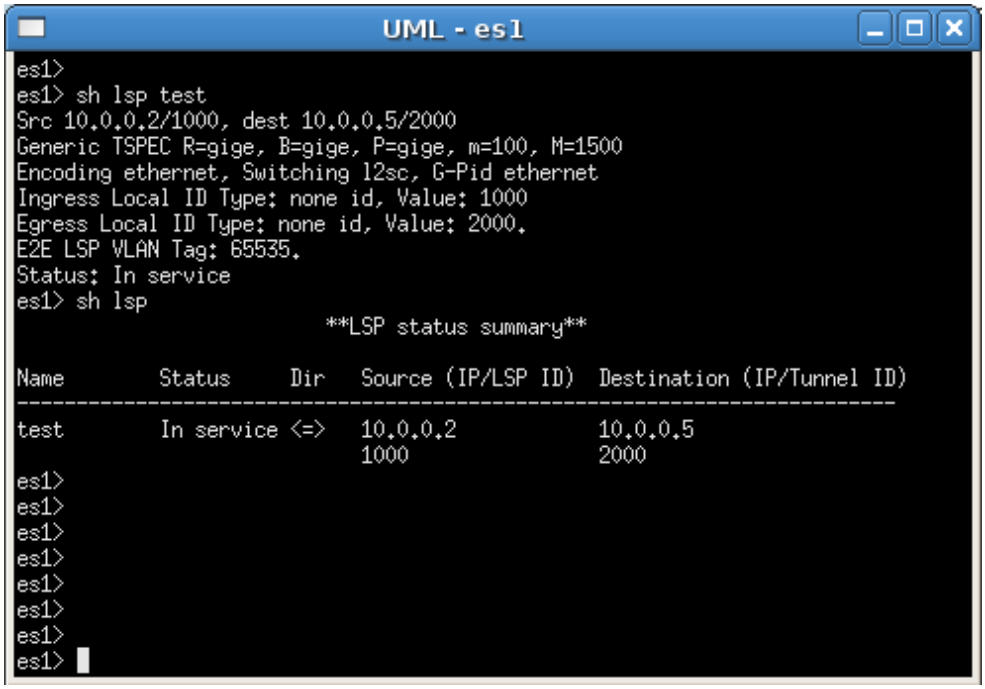

그림 8 es1에서의 LSP 상태

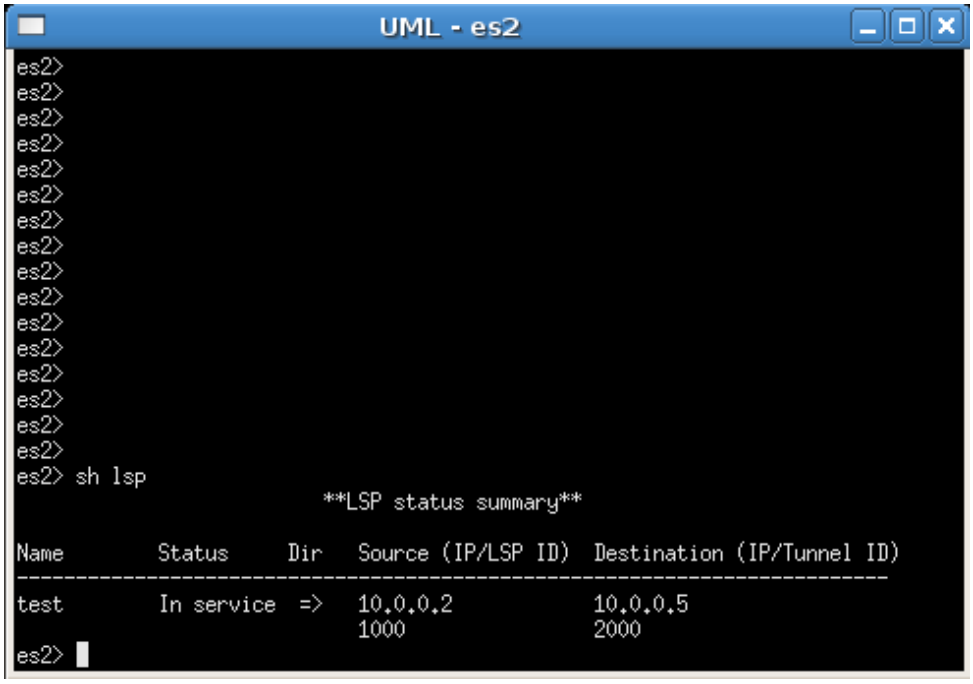

그림 9 es2에서의 LSP 상태

그림 10에서는 vlsr1에서 tcpdump로 캡쳐한 패킷 내용들 중에서 RSVP Resv 메시지의 일부를 보여준다. 일반적으로 tcpdump에서 캡쳐하려는 기본 패킷 크기는 80byte이므로 RSVP Resv 메시지를 제대로 캡쳐하기 위해서는 패킷 크기를 조절할 필요가 있다. 참고로 테스트에서는 "tcpdump -v -s 5000"의 파라미터를 사용하여 RSVP Resv 메시지를 캡쳐할 수 있었다.

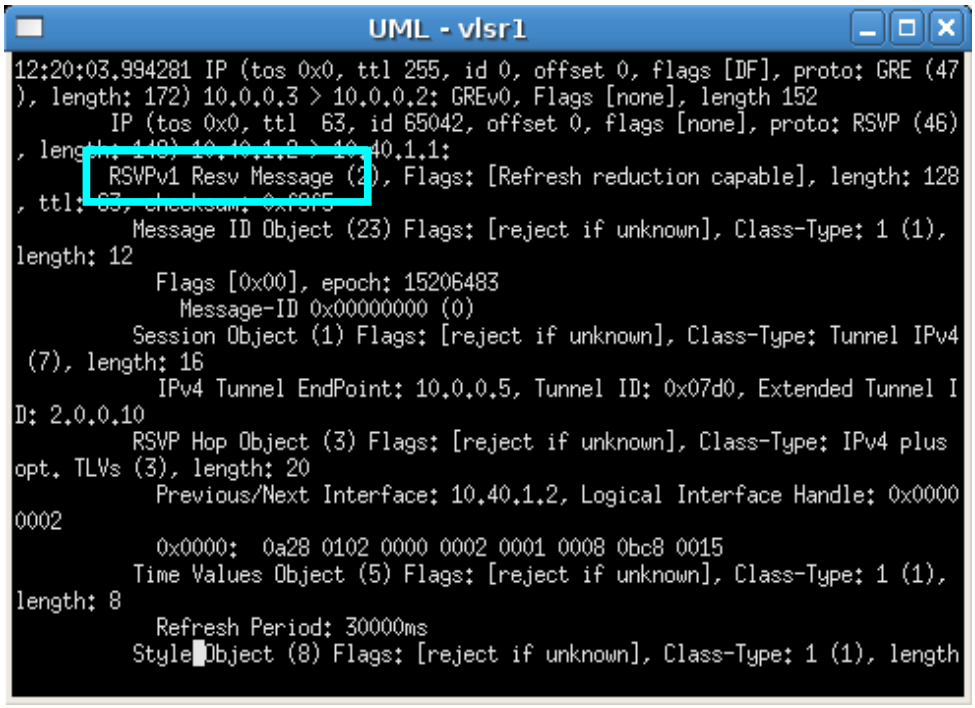

그림 10 vlsr1에서 캡쳐한 "RSVP Resv“ 패킷 내용

그림 11에서는 vlsr2에서의 RSVP 데몬의 로그 내용 중 일부를 보여준다. 로그는 /var/log/RSVPD.log로 남게 된다. EXPLICIT_ROUTE의 내용 중 vlsr2 중심의 TE 주소가 전달되고, IP4:11.200.0.2/32--IP4:11.200.1.17/32--IP4:11.200.0.18/32와 같은 연결 정보를 알게 된다. 각각 UML 인스턴스들인 es1, vlsr1, vlsr2, es2에서 RSVPD.log 파일을 확인할 수 있으며, 각 노드들 중심의 EXPLICIT_ROUTE 내용을 확인할 수 있다.

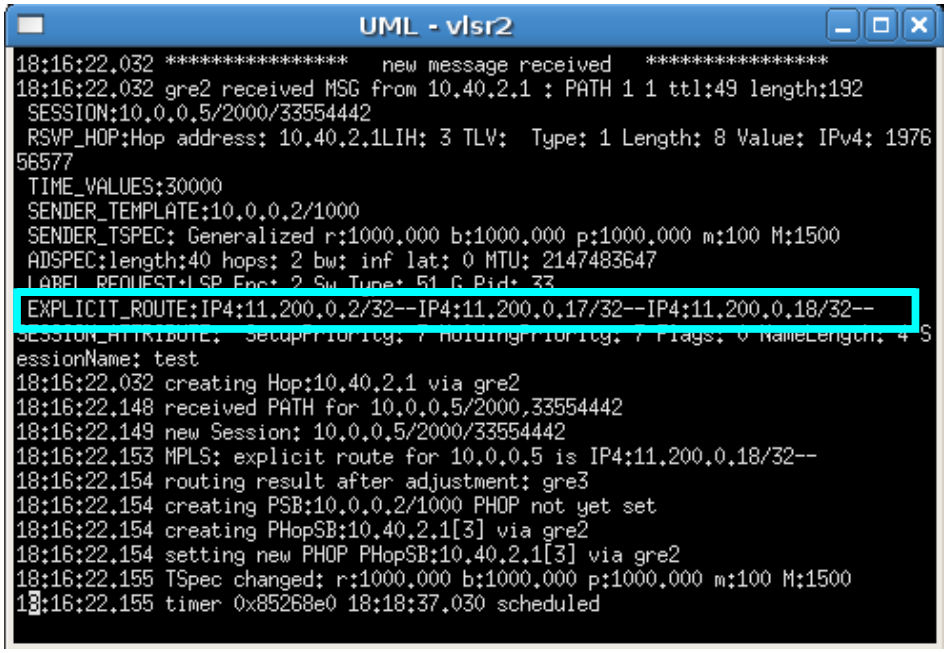

그림 11 vlsr2에서의 RSVP 데몬 로그

6. 결론

본 보고서에서는 UML기반의 DRAGON을 소개하고, 설치 및 기본적인 테스트를 수행하였다. DRAGON은 GMPLS 기반의 네트워크 도메인에서 Control plane으로서 적용될 수 있지만, 기본적인 기능 테스트 및 처음 도입을 위해서도 많은 네트워크 장비 및 시스템들을 요구한다. 이러한 상황에서 UML 기반의 DRAGON은 한 대의 시스템으로 기본적인 기능 테스트를 수행할 수 있게 해 준다.

현재 UML기반의 DRAGON을 사용하는데 있어서 약간의 제약이 따른다. Control plane이 구현되어 있고, Data plane은 구현되고 있는 중이다. 최종적으로는 Data plane을 통해 프로비져닝된 LSP상에서 종단간 시스템상의 "ping" 테스트가 이뤄질 수 있을 것이다. 그러한 단점을 고려하더라도, 예제에서 볼 수 있었듯이 종단 시스템에서의 LSP 설정 상태, vlsr에서의 RSVP Resv 메시지 및 RSVP데몬에서 제공하는 로그 등을 확인할 수 있었다.

7. 참고문헌

[1] DRAGON Project_1, "http://dragon.maxgigapop.net/twiki/bin/view/DRAGON/WebHome"

[2] DRAGON Project_2, "http://dragon.east.isi.edu/twiki/bin/view/Main/WebHome"

[3] Andreas Steffen, Eric Marchionni, Patrik Rayo. "Advanced Network Simulation"

[4] User-mode Linux Kernel Homepage, "http://user-mode-linux.sourceforge.net/"

[5] Virtual Network Experiments, "http://ndl.uva.netherlight.nl/trac/vne"

[6] Jeroen van der Ham, Chris Tracy, "Emulation of GMPLS using Virtual Network Experiments & DRAGON (http://www.glif.is/meetings/2007/joint-session/vdham-gmpls.pdf)", 7th Annual Global LambdaGrid Workshop

8. 부록

본 절에서는 앞의 예제에서 사용된 "config.xml"을 생성하는데 사용되는 dtd-VNEControlCfg.dtd 파일과 genDragonConfig 파일을 첨부한다. "config.xml"와 같은 토폴로지 파일에서의 각 속성을 이해하기 위해서는 genDragonConfig 파일에 포함된 주석 및 파라미터들을 살펴본다.

■ dtd-VNEControlCfg.dtd 파일

```
<!-- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
VNEControl Configuration File DTD version 0.1

$Id: dtd-VNEControlCfg.dtd,v 1.9 2004/07/01 13:19:00 uml Exp $

Part of "VNE" (Virtual Networking Experiments)

Jeroen van der Ham <jeroen@os3.nl>
Gert Jan Verhoog      <gjv@os3.nl>

Analythical Network Project,
Master System and Network Administration,
University of Amsterdam, the Netherlands
<http://www.os3.nl>
- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -->

<!ELEMENT config        (defaults?,devices, connections?)>

<!ELEMENT defaults      (kernel?,filesystem?,hostmount?)>
<!ELEMENT kernel        EMPTY>
<!ELEMENT filesystem    EMPTY>
<!ELEMENT hostmount     EMPTY>

<!ELEMENT devices       (host|hub|switch|router)*>
<!ELEMENT host          (kernel?,filesystem?,hostmount?,gateway?,interface*)>
<!ELEMENT hub           (kernel?,filesystem?,hostmount?,gateway?,interface*)>
<!ELEMENT switch        (kernel?,filesystem?,hostmount?,gateway?,interface*)>
<!ELEMENT router        (kernel?,filesystem?,hostmount?,gateway?,interface*)>
<!ELEMENT gateway       EMPTY>
<!ELEMENT interface     (ipv4?, ipv6?)>
<!ELEMENT ipv4          EMPTY>
<!ELEMENT ipv6          EMPTY>
```

```
<!ELEMENT connections   (connection*)>
<!ELEMENT connection    (device*)>
<!ELEMENT device        EMPTY>

<!ATTLIST kernel     name      CDATA #REQUIRED
options   CDATA #IMPLIED>
<!ATTLIST filesystem name      CDATA #REQUIRED
type      CDATA #IMPLIED>
<!ATTLIST hostmount  name      CDATA #REQUIRED>
<!ATTLIST host       name      ID    #REQUIRED
dragonconfig CDATA #IMPLIED>
<!ATTLIST hub        name      ID    #REQUIRED>
<!ATTLIST switch     name      ID    #REQUIRED>
<!ATTLIST router     name      ID    #REQUIRED>
<!ATTLIST gateway    ipv4      CDATA #IMPLIED
ipv6      CDATA #IMPLIED>
<!ATTLIST interface  name      CDATA #REQUIRED
macaddr   CDATA #IMPLIED
bridge    CDATA #IMPLIED>
<!ATTLIST ipv4       addr      CDATA #REQUIRED
netmask   CDATA #IMPLIED
broadcast CDATA #IMPLIED>
<!ATTLIST ipv6       addr      CDATA #REQUIRED>
<!ATTLIST device     name      IDREF #REQUIRED
interface CDATA #REQUIRED>
```

■ genDragonConfig 파일

```
#!/usr/bin/perl -W

use strict;
use Getopt::Long;

#my $PREFIX            = "/usr/local/dragon/etc/";
my $PREFIX            = "/var/dragon/";
my $DRAGON_CONF       = $PREFIX."dragon.conf";
my $ZEBRA_CONF        = $PREFIX."zebra.conf";
my $OSPFD_CONF        = $PREFIX."ospfd.conf";
my $OSPFD_INTRA_CONF  = $PREFIX."ospfd-intra.conf";
my $OSPFD_INTER_CONF  = $PREFIX."ospfd-inter.conf";
my $RSVPD_CONF        = $PREFIX."RSVPD.conf";
my $NARB_CONF         = $PREFIX."narb.conf";
my $GRE_CONF          = $PREFIX."setup_gre_tunnels.sh";

my $DRAGONPREFIX      = "/usr/local/dragon/etc/";
my $NARB_TOPOLOGIES   = $DRAGONPREFIX."narb_topologies";

my $element = "vlsr";
my $narbDomain = "1";
my $switchIp = "0.0.0.0";

my @SAVE_ARGV = @ARGV;

my %opts = ('element'    => $element,
            'narbdomain' => $narbDomain,
            'switch-ip'  => $switchIp,
            );

GetOptions (₩%opts,
            "element=s",    # string: either vlsr, narb, p2p-csa or uni-csa
            "hostname=s",   # host name, e.g. clpk-vlsr
            "loopback=s",   # loopback address (ospf router-id), e.g. 140.173.2.232
            "gretunnel=s",  # see format below**
            "narb=s",       # intra-domain NARB IP address for VLSR's
dragon.conf/RSVPD.conf
            "narbdomain=s", # NARB domain IP -- e.g. 140.173.0.0 or 10.0.0.0
            "narbintra=s",  # intra-domain NARB GRE tunnel (GRE tunnel between
NARB--VLSR)
```

```
            "narbinter=s",  # inter-domain NARB GRE tunnel (GRE tunnel between
NARB--NARB)
            "inter-domain-te-link=s", # remote TE address for inter-domain TE link
            "passiveint=s", # passive interface for OSPFD -- for inter-domain TE links
(VLSR--VLSR interdomain)
            "switch-ip=s",  # IP address for switch that VLSR is controlling (0.0.0.0 for no
switch/simulation)
            "copy-hostfs",  # if this switch is passed, mount the hostfs and copy the
config files over..
            "start",        # pass this option to automatically start dragon daemons
            );

# **GRE tunnel config format:
#   GRE tunnel name (e.g. gre1)
#   GRE tunnel remote address (public)
#   GRE tunnel local address  (public)
#   GRE tunnel remote address (private)
#   GRE tunnel local address  (private)
#   GRE tunnel network (/30)
#   TE address (local)
#   VLSR switch port assocation (local)
#
# %gre hash keys:
#     name = GRE tunnel name (gre1 or higher)
#     pub_rem = GRE tunnel remote address (public)
#     pub_loc = GRE tunnel local address  (public)
#     prv_rem = GRE tunnel remote address (private)
#     prv_loc = GRE tunnel local address  (private)
#     prv_net = GRE tunnel private network
#     te_addr_local = TE address (local)
#     switch_port = VLSR switch port assocation (local)
#

my @gre_tunnel_config = split(/,₩s+/, $opts{'gretunnel'});
my $num_gre_tunnels = scalar(@gre_tunnel_config) / 8;
my %gre;
for my $gre_num (1..$num_gre_tunnels) {
my $name = shift @gre_tunnel_config;
$gre{$name}{'name'} = $name;
$gre{$name}{'pub_rem'} = shift @gre_tunnel_config;
$gre{$name}{'pub_loc'} = shift @gre_tunnel_config;
```

```
$gre{$name}{'prv_rem'} = shift @gre_tunnel_config;
$gre{$name}{'prv_loc'} = shift @gre_tunnel_config;
$gre{$name}{'prv_net'} = shift @gre_tunnel_config;
$gre{$name}{'te_addr_local'} = shift @gre_tunnel_config;
$gre{$name}{'switch_port'} = shift @gre_tunnel_config;
}

#
# open the proper config files, depending on network element type
# (e.g. NARB requires different set of config files than VLSR)
#
if ($opts{'element'} =~ /^(vlsr|p2p-csa)$/i) {
open DRAGON_CONF, "> $DRAGON_CONF";
open ZEBRA_CONF,  "> $ZEBRA_CONF";
open OSPFD_CONF,  "> $OSPFD_CONF";
open RSVPD_CONF,  "> $RSVPD_CONF";
open GRE_CONF,    "> $GRE_CONF";
}
if ($opts{'element'} =~ /^uni-csa$/i) {
open DRAGON_CONF, "> $DRAGON_CONF";
open RSVPD_CONF,  "> $RSVPD_CONF";
open GRE_CONF,    "> $GRE_CONF";
}
if ($opts{'element'} =~ /^narb$/i) {
open ZEBRA_CONF,       "> $ZEBRA_CONF";
open OSPFD_CONF,       "> $OSPFD_INTRA_CONF";
open OSPFD_INTER_CONF, "> $OSPFD_INTER_CONF";
open NARB_CONF,        "> $NARB_CONF";
open GRE_CONF,    "> $GRE_CONF";
}

#
# standard beginnings for these files...
#
print ZEBRA_CONF <<END if ($opts{'element'} =~ /^(vlsr|p2p-csa|narb)$/i);
hostname $opts{'hostname'}
password dragon
enable password dragon
END

print DRAGON_CONF <<END if ($opts{'element'} =~ /^(vlsr|p2p-csa|uni-csa)$/i);
```

```
hostname $opts{'hostname'}
password dragon
END

print OSPFD_CONF <<END if ($opts{'element'} =~ /^(vlsr|p2p-csa|narb)$/i);
hostname $opts{'hostname'}
password dragon
enable password dragon
log stdout
log file /var/log/ospfd.log
END

print OSPFD_INTER_CONF <<END if ($opts{'element'} =~ /^narb$/i and
defined($opts{'narbinter'}));
hostname $opts{'hostname'}
password dragon
enable password dragon
log stdout
log file /var/log/ospfd-inter.log
END

print RSVPD_CONF "api 4000₩n" if ($opts{'element'} =~ /^(vlsr|p2p-csa|uni-csa)$/i);

print NARB_CONF <<END if ($opts{'element'} =~ /^narb$/i);
domain-id { ip $opts{'narbdomain'} }
cli { host $opts{'hostname'} password dragon }
END

print GRE_CONF <<END if ($opts{'element'} =~ /^(vlsr|p2p-csa|uni-csa|narb)$/i);
#!/bin/sh

/sbin/modprobe ip_gre

END

#
# required for proper operation when running under UML...
# i think the daemons had problems bind()'ing without this...
#
system("ifconfig lo inet 127.0.0.1");
```

```
#
# set hostname for UML...(cosmetic)
#
if (defined($opts{'loopback'})) {
system("echo $opts{'hostname'} > /etc/hostname");
system("hostname --file /etc/hostname");
}

#
# configure the system's GRE tunnels on the fly...
# setup ospfd.conf and RSVPD.conf interface configurations...
#
foreach my $name ( sort keys %gre ) {
print GRE_CONF <<END;
ip tunnel del $gre{$name}{'name'}
ip tunnel add $gre{$name}{'name'} mode gre remote $gre{$name}{'pub_rem'} local
$gre{$name}{'pub_loc'} ttl 255
ip link set $gre{$name}{'name'} up
ip addr add $gre{$name}{'prv_loc'}/30 dev $gre{$name}{'name'}
ip route add $gre{$name}{'prv_rem'}/32 dev $gre{$name}{'name'}

END

if (defined($opts{'narbinter'}) and $opts{'narbinter'} =~ /$name/i) {
        print OSPFD_INTER_CONF <<END if ($opts{'element'} =~ /^narb$/i);
interface $gre{$name}{'name'}
description GRE tunnel $gre{$name}{'name'}
ip ospf network point-to-point
END
} else {
        print OSPFD_CONF <<END if ($opts{'element'} =~ /^(vlsr|p2p-csa|narb)$/i);
interface $gre{$name}{'name'}
description GRE tunnel $gre{$name}{'name'}
ip ospf network point-to-point
END
}

# skip RSVPD configuration on VLSR for VLSR--NARB GRE tunnel:
next if $name eq $opts{'narbintra'};

# normal VLSR and end-system (in peer mode) operation
```

```
print RSVPD_CONF "interface $name tc none mpls₩n"
	if ($opts{'element'} =~ /^(vlsr|p2p-csa)$/i and $gre{$name}{'te_addr_local'} ne
"0.0.0.0");

# if this is a VLSR, a local TE address of 0.0.0.0 signals that this GRE is used for
servicing a UNI client
print RSVPD_CONF "interface $name tc none mpls p/1₩n"
	if ($opts{'element'} =~ /^vlsr$/i and $gre{$name}{'te_addr_local'} eq "0.0.0.0");

# if this is a UNI-mode end system, assume we are connected to VLSR port 1
print RSVPD_CONF "interface $name tc none mpls p/1₩n"
	if ($opts{'element'} =~ /^uni-csa$/i);
}

print OSPFD_CONF <<END if ($opts{'element'} =~ /^(vlsr|p2p-csa|narb)$/i);
router ospf
ospf router-id $opts{'loopback'}
END

print OSPFD_INTER_CONF <<END if ($opts{'element'} =~ /^narb$/i and
defined($opts{'narbinter'}));
router ospf
ospf router-id $opts{'loopback'}
END

#
# inter-domain GRE tunnels between 2 VLSRs need to be put into passive mode
# (we have to represent the TE link, but we don't want to flood LSAs across this interface)
#
if (defined($opts{'passiveint'})) {
foreach my $passive_int ( split(/[, ]/, $opts{'passiveint'}) ) {
	print OSPFD_CONF <<END if ($opts{'element'} =~ /^(vlsr|p2p-csa|narb)$/i);
passive-interface $passive_int
END
}
}

foreach my $name ( sort keys %gre ) {
if (defined($opts{'narbinter'}) and $opts{'narbinter'} =~ /$name/i) {
	print OSPFD_INTER_CONF "	network $gre{$name}{'prv_net'} area 0.0.0.1₩n" if
($opts{'element'} =~ /^narb$/i);
```

```
} else {
        print OSPFD_CONF "    network $gre{$name}{'prv_net'} area 0.0.0.0₩n" if
($opts{'element'} =~ /^(vlsr|p2p-csa|narb)$/i);
}
}

print OSPFD_CONF "    ospf-te router-address $opts{'loopback'}₩n" if ($opts{'element'}
=~ /^(vlsr|p2p-csa|narb)$/i);

#
# eventually we may want to use switch-ip 127.0.0.1 and talk to a 'fake' switch
# (implemented by a perl script) that is called via NET-SNMP's snmpd.
#
if ($opts{"element"} =~ /^vlsr$/i) {
foreach my $name ( sort keys %gre ) {
#
        # this section is not necessary for the GRE tunnel between the NARB and a
VLSR.
# the 'narbintra' option is used to specify this GRE tunnel, so it knows whether
# or not this section can be skipped.  we want to bring up an OSPF adjacency
# between the intra-domain NARB ospfd and the VLSR's ospfd, but the NARB should
# not advertise any TE links to the VLSR...
#
        print OSPFD_CONF <<END if ($opts{'element'} =~ /^(vlsr|p2p-csa|narb)$/i and
$name !~ $opts{'narbintra'});
ospf-te interface $gre{$name}{'name'}
level gmpls
data-interface ip $gre{$name}{'te_addr_local'} protocol snmp switch-ip 0.0.0.0
switch-port 123
!     data-interface ip $gre{$name}{'te_addr_local'} protocol snmp switch-ip
$opts{"switch-ip"} switch-port $gre{$name}{'switch_port'}
swcap l2sc encoding ethernet
! 1Gbps = 125000000 bytes/sec
max-bw 125000000
max-rsv-bw 125000000
max-lsp-bw 0 125000000
max-lsp-bw 1 125000000
max-lsp-bw 2 125000000
max-lsp-bw 3 125000000
max-lsp-bw 4 125000000
max-lsp-bw 5 125000000
```

```
max-lsp-bw 6 125000000
max-lsp-bw 7 125000000
vlan 100 to 200
metric 10
exit
END
}
} elsif ($opts{"element"} =~ /^p2p-csa$/i) {
foreach my $name ( sort keys %gre ) {
# the configuration for an end system in peer mode is relatively straightforward...
        print OSPFD_CONF <<END if ($opts{'element'} =~ /^(vlsr|p2p-csa|narb)$/i);
ospf-te interface $gre{$name}{'name'}
level gmpls
data-interface ip $gre{$name}{'te_addr_local'}
swcap l2sc encoding ethernet
! 1Gbps = 125000000 bytes/sec
max-bw 125000000
max-rsv-bw 125000000
max-lsp-bw 0 125000000
max-lsp-bw 1 125000000
max-lsp-bw 2 125000000
max-lsp-bw 3 125000000
max-lsp-bw 4 125000000
max-lsp-bw 5 125000000
max-lsp-bw 6 125000000
max-lsp-bw 7 125000000
metric 10
exit
END
}
}

print RSVPD_CONF "narb $opts{'narb'} 2609₩n"
if ($opts{'element'} =~ /^vlsr$/i and defined($opts{'narb'}));

#
# some example local IDs for testing...
#
print DRAGON_CONF <<END if ($opts{'element'} =~ /^vlsr$/i and
defined($opts{'narb'}));
set local-id port 1
```

```
set local-id port 9
set local-id group 1000 add 1
set local-id group 1000 add 9
set local-id group 1001 add 2
set local-id group 1001 add 10
set local-id tagged-group 101 add 1
set local-id tagged-group 101 add 9
set local-id tagged-group 102 add 2
set local-id tagged-group 102 add 10
END

#
# configure UNI client as local-id port 1 (p/1 also hardcoded in RSVPD.conf above)
#
print DRAGON_CONF <<END if ($opts{'element'} =~ /^uni-csa$/i);
set local-id port 1
END

#
# configure intra-domain NARB if this is a VLSR or an end system in peer mode
# (so that the dragon daemon can contact the NARB to request an ERO)
#
print DRAGON_CONF <<END if ($opts{'element'} =~ /^(vlsr|p2p-csa)$/i and
defined($opts{'narb'}));
configure narb intra-domain ip-address $opts{'narb'} port 2609
END

#
# configure NARB linkage to intra-domain OSPFD API
#
print NARB_CONF <<END if ($opts{'element'} =~ /^narb$/i and
defined($opts{'narbintra'}));
intra-domain-ospfd { address localhost port 2617 originate-interface
$gre{$opts{'narbintra'}}{'prv_loc'} area 0.0.0.0 }
END

#
# configure NARB linkage to inter-domain OSPFD API
#
if (defined($opts{'narbinter'}) and $opts{'element'} =~ /^narb$/i) {
# doesn't look like we need multiple inter-domain-ospfd stanzas
```

```
# foreach my $narbinter ( split(/[, ]/, $opts{'narbinter'}) ) { }
my @narbinter = split(/[, ]/, $opts{'narbinter'});
print NARB_CONF <<END;
inter-domain-ospfd { address localhost port 2607 originate-interface
$gre{$narbinter[0]}{'prv_loc'} area 0.0.0.1 }
END
}

# import NARB topology from $NARB_TOPOLOGIES directory, based on hostname.
# eventually we will have a better way to automatically generate this...
if ($opts{'hostname'} =~
/^(red-narb|blue-narb|yellow-narb|green-narb|narb1|narb2)$/ and
$opts{'element'} =~ /^narb$/i and defined($opts{'narbinter'})) {
open I, "$NARB_TOPOLOGIES/$opts{'hostname'}.narbtopo.conf";
while(<I>) {
        print NARB_CONF;
}
close I;
}

# need one of these statements for each inter-domain TE link.
# associate each inter-domain link with a peering NARB.
if (defined($opts{'narbinter'}) and $opts{'element'} =~ /^narb$/i) {
my @inter_domain_te_links = split(/[, ]/, $opts{'inter-domain-te-link'});
my $z = 0;
foreach my $narbinter ( split(/[, ]/, $opts{'narbinter'}) ) {
print NARB_CONF <<END;
inter-domain-te-link { id $inter_domain_te_links[$z] narb-peer
$gre{$narbinter}{'prv_rem'} port 2609 }
END
$z++;
}
}

# close files
if ($opts{'element'} =~ /^(vlsr|p2p-csa)$/i) {
close DRAGON_CONF;
close ZEBRA_CONF;
close OSPFD_CONF;
close RSVPD_CONF;
close GRE_CONF;
```

```
}
if ($opts{'element'} =~ /^uni-csa$/i) {
close DRAGON_CONF;
close RSVPD_CONF;
close GRE_CONF;
}
if ($opts{'element'} =~ /^narb$/i) {
close ZEBRA_CONF;
close OSPFD_CONF;
close OSPFD_INTER_CONF;
close NARB_CONF;
close GRE_CONF;
}

#
# set $GRE_CONF to be executable
#
system("/bin/chmod +x $GRE_CONF");

#
# if the --copy-hostfs option was passed, copy the generated configs to the host
filesystem
#
if ($opts{'copy-hostfs'}) {
system("mount none /mnt -t hostfs -o /a/uml/hostfs");
mkdir "/mnt/$opts{'hostname'}" if (!-d "/mnt/$opts{'hostname'}");
system("/bin/cp $DRAGON_CONF       /mnt/$opts{'hostname'}/");
system("/bin/cp $ZEBRA_CONF        /mnt/$opts{'hostname'}/");
system("/bin/cp $OSPFD_CONF        /mnt/$opts{'hostname'}/");
system("/bin/cp $OSPFD_INTRA_CONF /mnt/$opts{'hostname'}/");
system("/bin/cp $OSPFD_INTER_CONF /mnt/$opts{'hostname'}/");
system("/bin/cp $NARB_CONF         /mnt/$opts{'hostname'}/");
system("/bin/cp $RSVPD_CONF        /mnt/$opts{'hostname'}/");
system("/bin/cp $GRE_CONF          /mnt/$opts{'hostname'}/");
open Z, "> /mnt/$opts{'hostname'}/genDragonConfig.sh";
print Z "#!/bin/sh₩n/usr/local/dragon/bin/genDragonConfig.pl ";
foreach my $val ( sort keys %opts) {
        print Z "--$val=₩"$opts{$val}₩" ";
}
print Z "₩n";
close Z;
```

```
system("umount /mnt");
}

#
# start the network element's daemons in the proper mode
# (if --start was passed)
#
if ($opts{'start'}) {
#
# setup GRE tunnels
#
system( $GRE_CONF );
#
# start the proper set of daemons
#
if ($opts{'element'} =~ /^(vlsr|p2p-csa)$/i) {
        system("/usr/local/dragon/bin/dragon.sh start-vlsr");
}
if ($opts{'element'} =~ /^uni-csa$/i) {
        system("/usr/local/dragon/bin/dragon.sh start-uni");
}
if ($opts{'element'} =~ /^narb$/i) {
        system("/usr/local/dragon/bin/dragon.sh start-narb");
}
}
```