

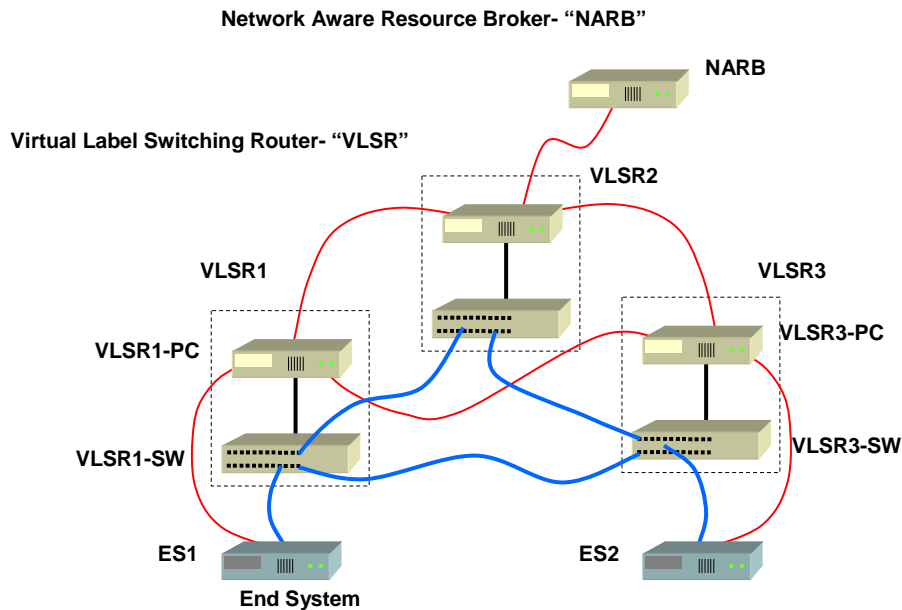
## Exercise #1: Designing a GMPLS Control Plane

Objective: In this exercise, we will layout and configure a GMPLS control plane for a small network.

There are four engineering teams. Each team will be responsible for one network domain, or “pod”. There are four pod domains: Red, Blue, Yellow, and Green with ASNs of 1, 2, 3, and 4 respectively. The networks constructed in each exercise will form the basis for subsequent exercises. Each network pod consists of three Virtual Label Switching Routers (VLSRs), two End Systems (ES), and one Network Aware Resource Broker (NARB).

### Step 1: Layout a diagram showing the desired data plane topology for the network.

We will be building a network consisting of three Ethernet switches (see diagram below) acting as network elements. Two PCs will act as End Systems. The team will develop a network diagram incorporating route diversity, where the end systems attach to the network, and all the interface and/or port assignments required to support the network.



Each device in the network will require a “management” IP address. For End Systems, this would be a normal IP address for that host. For the network elements, management addresses are used primarily to access the device for configuration. Management interfaces should have publicly routable IPv4 addresses. In this exercise, we will use the management address where a loopback address might be traditionally used for router IDs and such. In general, we use these

management addresses wherever a publicly reachable address for the device would normally be required.

In addition to these management interfaces, each device should have a separate “data plane” interface over which the dynamic circuits will be delivered. The End Systems provided in the workshop lab all have dual gigE ports, eth0 and eth1. In this exercise eth0 will be used for mgmt and control, and eth1 for data plane.

(Note: Each PC has dual 1gigE integrated ethernet ports: eth0 and eth1. We use separate physical interfaces for management+control and data plane. We have arbitrarily designated eth0 for management and control, and eth1 as the data plane interface. The standards do allow for control, management, and data to be provisioned over the same physical interface, but the DRAGON implementation does not yet support this feature.)

In a real network, network elements have a separate “management” address and “loopback” address assigned to each device. The management addresses are used by the network operations to access the devices for configuration and monitoring and are often deliberately not visible to general users. The loopback addresses are used to associate a publicly reachable address with each device that will always be UP. The GMPLS control channels are provisioned over GRE tunnels between network elements. In these exercises the GRE tunnels use the management addresses as the tunnel endpoints. In a real network, the loopback addresses would be more appropriate for tunnel endpoints as the management addresses may not be visible. We have not assigned loopbacks in this workshop in order to simplify the overall addressing, and so we use the management addresses instead.

For these exercises we use the 192.168.0.0/16 subnet for management addressing. We use the 10.0.0.0/8 subnet for control plane addressing. In a real network, the intra-domain control plane links do not strictly need to be public unless they are advertised as part of the inter-domain topology distribution between NARBs. In the black box scenario (minimum internal topology exposed), only the border links need be public addresses as they will be visible to the peering network.

The TE-addresses are largely a matter of convenience. TE-Addresses are not actual IP addresses in the normal IP sense – they are simply a means to associate specific data plane interfaces with their control links. We use the 11.0.0.0/8 net for this purpose in order to provide a numbering scheme congruent with the control plane addressing.

Create a list of Data Plane links. Number them  $D_1$ , to  $D_n$ . Identify the device and port/interface for each end of the data link.

<i>Cid</i>	<i>Device</i>	<i>Int/Port</i>	<i>Device</i>	<i>Int/Port</i>
<b>D1</b>	<b>ES1</b>	<b>eth1</b>	<b>VLSR1-SW</b>	<b>port3</b>
<b>D2</b>	<b>VLSR1-SW</b>	<b>port4</b>	<b>VLSR2-SW</b>	<b>port4</b>
<b>D3</b>				

## Step 2: Design the control plane architecture to support the GMPLS operations.

Since each of the Ethernet switches will act as a dynamically provisioned switching element, each switch will need a control PC associated with it to run the OSPF-TE and RSVP-TE protocols. These switch-PC pairs are VLSRs. The control PC is responsible for running the routing and signaling protocols and reconfiguring the associated Ethernet switch as necessary.

Every networked device participating in the dynamically switched network will require a “control link” to be established with neighboring such devices. In GMPLS, these control plane devices may not be physically adjacent to each other. So in order to allow for intervening network infrastructure, we establish GRE tunnels between the neighboring control plane speakers, and assign ip addresses to these tunnel interfaces for the control links.

With respect to the VLSRs, all control plane links are established with the VLSR-PC (not the VLSR-SW). The team should layout the control plane links that will cover the data plane designed in step 1. Each control plane link will require a GRE tunnel be established between logically adjacent network elements (VLSRs and/or ES’s). We use the management addresses as the endpoints for building the tunnels.

Each control link will require a /30 subnet be assigned from the appropriate CIDR blocks:

Red.....10.1.0.0/16

Blue.....10.2.0.0/16

Yellow...10.3.0.0/16

Green....10.4.0.0/16

Tip: In order to keep things manageable, number all the control link uniquely within the domain, and congruently with the data plane links. Then use this numbering scheme to identify which GRE interface to use for each control link. The GRE number can also be used as part of the control plane subnet as follows: 10.<asn>.<gre#>.h

Red GRE1: 10.1.1.0/30

Blue GRE5: 10.2.5.0/30

Green GRE14: 10.4.14.0/30

(Note: The GRExx interfaces defined at each end of the control link do \*not\* need to be the same. But doing so will reduce the confusion factor and help in debugging config problems...)

Create a list of control plane links. Each control link should include the associated data link, the GRE tunnel # to be used, the IP addresses of the tunnel endpoints, and the ctrl plane IP addresses to be assigned to the GRE interface. You may wish to build this list for each device in the network (rather than by link) in order to facilitate the set up process later:

Red Pod:

Cid	GRE	Local			Remote		
		Device	Ctrl Addr	Tun Addr	Device	Ctrl Addr	Tun addr
D1	GRE1	ES1	10.1.1.1	192.168.1.2	VLSR1-PC	10.1.1.2	192.168.1.4
D1	GRE1	VLSR1-PC	10.1.1.2	192.168.1.4	ES1	10.1.1.1	192.168.1.2
D2							
D3							
D2							
D4							

### Step 3: Assign TE-Addresses

TE Addresses should be assigned for each data plane connection. TE-addresses are not IP addresses for the interface...they simply represent a numbering scheme to uniquely identify data plane connections. If the user wished to use IP over a dynamically allocated circuit, the user will need to allocate and assign an IP address to the interface associated with the newly created link. The TE address should not be used for this purpose. For this exercise, and to keep the addressing scheme somewhat humanly intuitive, we use the 11/8 net in a similar fashion to the 10 net assignments used for control links :

Red data plane link D1: 11.1.1.0/30

Yellow data plane link D4: 11.3.4.0/30

Before continuing, go over the diagram with the workshop instructor and/or the other attendees. There are many ways to design a GMPLS network – the objective is to understand the key components and the process.

### Step 4: Configure the VLSRs:

Now we use a standard network configuration for these workshop exercises. See the “Pod Topology” at the end of Exercise #1. This diagram provides a physical layout in each pod and the excel sheets after the diagram provide detailed addressing scheme for a small GMPLS network. All pod PCs have been pre-loaded with Debian Linux, the software dependency packages required for the DRAGON software and/or systems support, and a recent release of the DRAGON GMPLS software suite.

Each attendee should have been assigned a user login.

We will configure the three VLSRs to reflect this network diagram. To configure a VLSR, there are several steps:

1. In the VLSR-PC, we must create the appropriate GRE tunnels and then assign control plane addresses to those GRE interfaces. We modified the Debian startup process to invoke and /etc/rc.local script.

All GRE tunnels in the pod are already up and running. However, it will be a good idea to get familiar with tunnel set up by following the steps as shown below:

Edit **/etc/rc.local** The following modprobe commands should be at the top of the file:

```
#!/bin/sh
# needed for GRE tunnel support, e.g. 'ip tunnel'
/sbin/modprobe ip_gre
# needed for tagged VLAN support, e.g. 'vconfig'
/sbin/modprobe 8021q
```

Note: In this exercise, all LSP setup will be in port-to-port/untagged mode, so the support for tagged VLAN is not necessary, but we will add the support here for exercises that come later.

For each GRE tunnel required for this network element, the following Linux commands should be modified and added to /etc/rc.local :

```
# RED_ES1
ip tunnel del gre1
ip tunnel add gre1 mode gre remote 192.168.1.2 local 192.168.1.4 ttl 255
ip link set gre1 up
ip addr add 10.1.1.2/30 dev gre1
ip route add 10.1.1.1/32 dev gre1
```

Alternatively, we also provide a script at /usr/local/dragon/etc/setup\_gre\_tunnels.sh on each machine to set up the GRE tunnels.

2. Then, we must edit the configuration files for the DRAGON software  
In the directory /usr/local/dragon/etc, copy the sample .conf files to working files:

```
> cd /usr/local/dragon/etc
> cp RSVPD.conf.sample RSVPD.conf
> cp zebra.conf.sample zebra.conf
> cp ospfd.conf.sample ospfd.conf
> cp dragon.conf.sample dragon.conf
```

Edit the dragon.conf file and the zebra.conf file. For now, these files only contain the hostname assignment for this VLSR and the login password. For simplicity sake, we set the password to “dragon”.

Next, we edit the **RSVPD.conf** file. In this file, for now, all we need to do is to define each GRE interface RSVP will be using for signaling. For example, red\_vlsr1 will have three GRE

interfaces: GRE1 (to ES1), GRE2 (to VLSR2) and GRE3 (to VLSR3). Add the following commands to **RSVPD.conf**:

```
interface gre1 tc none mpls
interface gre2 tc none mpls
interface gre3 tc none mpls
```

Next, we can edit the **dragon.conf** file. For now, all we want to do is to change the hostname to reflect the name of the VLSR.

```
hostname red-vlsr1-pc
password dragon
```

Finally, we edit the **ospfd.conf** file. Ospf must be told which control links to listen to and/or flood updates, and it must also be given switchport mappings in order to perform path computation and manage the link state associated with each link.

It is in the ospfd where we specify the switch ip address and capacity associated with each class of service priority. The following text is excerpted from one VLSR. The student should be able to review this configuration and make any appropriate changes for the other control links on this VLSR

```
! *- ospf *-
!
! OSPFd sample configuration file
!
hostname red-vlsr1-ospf
password dragon
enable password dragon
log stdout
! log file /var/log/ospfd.log
!
!
! *- sample ospf configuration for a dragon VLSR controlling a Layer2
Ethernet switch *-
!
interface gre1
description GRE tunnel between red-vlsr1 and red-es1
ip ospf network point-to-point

interface gre2
description GRE tunnel between red-vlsr1 and red-vlsr2
ip ospf network point-to-point

interface gre3
description GRE tunnel between red-vlsr1 and red-vlsr3
ip ospf network point-to-point

router ospf
ospf router-id 192.168.1.4
network 10.1.1.2/30 area 0.0.0.0
network 10.1.2.2/30 area 0.0.0.0
network 10.1.3.2/30 area 0.0.0.0
```

```

ospf-te router-address 192.168.1.4
ospf-te interface gre1
    level gmpls
    data-interface ip 11.1.1.2 protocol snmp switch-ip 192.168.1.3
switch-port 3
    swcap l2sc encoding ethernet
    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
ospf-te interface gre2
    level gmpls
    data-interface ip 11.1.2.1 protocol snmp switch-ip 192.168.1.3
switch-port 4
    swcap l2sc encoding ethernet
    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

```

3. Then login to the covered Ethernet switch and configure it appropriately

Now that the control plane software is configured, we need to configure the switch to match the features configured in the GMPLS \*.conf files. Every switch is different – in how they handle VLANs, spanning tree, mtus, even whether or not they support SNMP. For the workshop, we have provided Dell 5324 Ethernet switches. We need to setup the proper management IP address, set the SNMP read/write community string, initialize ports that will be under VLSR control. We want to disable spanning tree protocol and filter BPDU packets on all the VLSR ports in order to prevent SPT flooding. For these Dell switches in the workshop configuration, we want ports 1 and 2 to be in VLAN #1 and this will be the management interface. All other ports g3 thru g24 will be available for switching.

The following are the CLI commands used to initialize the Dell 5324 switches for their role in these workshop networks:

```
vlan database
vlan 100-200
exit
no spanning-tree
spanning-tree bpdu filtering
port jumbo-frame
interface range ethernet g(3-24)
  switchport mode general
  spanning-tree disable
  switchport general pvid 2
exit
interface range ethernet g(1-2)
switchport mode general
switchport general pvid 1
exit
logging console debugging
enable password level 15 admin
username admin password admin level 15
snmp-server community dragon rw
```

The teams should login to the VLSR switches to peruse the configuration and to display the port and vlan mappings:

To show the port and vlan assignments, do:

```
>show vlan
```

To show the entire switch configuration:

```
>show running-config
```

#### 4. Start the DRAGON software on the VLSR PC.

The final step is to start all of the DRAGON protocol agents. To start the VLSR daemons, use the following shell command on the VLSR PC:

```
#> /usr/local/dragon/bin/dragon.sh start-vlsr
#> ps auxwww | grep dragon
```

This command starts **zebrad**, **ospfd**, **RSVPd**, and **dragond** and places them in the background. Note: **dragon.sh** is used to start, restart, stop, status all of the protocol daemons. Invoke it without any command line parameters to get a list of options.

After starting the daemons, we want to verify that the protocols are talking properly. For example, when **ospfd** starts up, it will try to issue HELLO exchange on the configured GRE links. Adjacencies should be formed and link state announcements (LSAs) flooded across to neighbors to build the link state database, i.e.the intra-domain topology of the network. You can query the status of these ospf adjacencies by telneting to the ospfd CLI port (2604): (note: typical password for the dragon software agents is “dragon” ☺)

```
red_vlsr1_pc#> telnet localhost 2604
password> *****
red_vlsr1-ospf> sh ip ospf interface
red_vlsr1-ospf> sh ip ospf-te database detail
red_vlsr1-ospf> sh ip ospf neighbor
```

The protocol daemons will write to log files in /var/log. You can inspect these files to get a more detailed understanding of the state of the protocols or interfaces.

```
less /var/log/ospfd.log
less /var/log/RSVPD.log
```

## 5. Set up the End System

Since we will be working this first exercise without the NARB, we want to configure the end systems to be able to source and sink LSPs. For “peer” mode, the End Systems will participate in both the routing and signaling protocols, i.e. the end systems will run both ospfd and rsvpd. In large part, this makes an end system look and feel like a VLSR – but without a switching fabric to be responsible for. (Note: A “UNI” mode of operation – where the end systems only run RSVPD - is implemented in the current version of the software, but requires additional configuration. UNI mode operation will be the subject of a later lab.)

Configuring the End Systems for peer mode operation is similar to configuring a VLSR. The difference is that there is no switch information specified in the ospfd.conf.

```
data-interface ip 11.1.11.1 protocol snmp switch ip 192.168.1.7 switch port 6
```

The teams should edit the ES1 and ES2 file sets to reflect this, and then run:

```
Red_vlsr1_pc#> /usr/local/dragon/bin/dragon.sh restart-vlsr
```

## 6. Set up an LSP

Finally...It is time to actually configure and provision the LSP. For these exercises, we will be using the Dragon CLI to instantiate the LSP. This is a two step process: first we’ll define and edit the LSP and all of its parameters, and second, we’ll “commit” the LSP to place it into service. Afterward, we’ll explore some of the ways to inspect the status of an LSP or to debug issues in the control plane protocols.

Login to the dragon CLI on port 2611 and configure the LSP:

```
Password: *****
```

```
red-es1-dragon> edit lsp test1
red-es1-dragon(edit-lsp-test1)# set source ip-address 192.168.1.2 lsp-id 1000
destination ip-address 192.168.1.9 tunnel-id 1000
red-es1-dragon(edit-lsp-test1)# set bandwidth eth100M swcap 12sc encoding ethernet
gpip ethernet
red-es1-dragon(edit-lsp-test1)# exit
```

```
red-es1-dragon> sh lsp
```

```
**LSP status summary**
```

Name	Status	Dir	Source (IP/LSP ID)	Destination (IP/Tunnel ID)
test1	Edi t	=>	192.168.1.2 1000	192.168.1.9 1000

Now, commit the LSP to put it into service:

```
red-es1-dragon> commi t lsp test1
```

You can check the status of the LSP with:

```
red-es1-dragon> show lsp test1
```

```
**LSP status summary**
```

Name	Status	Dir	Source (IP/LSP ID)	Destination (IP/Tunnel ID)
test1	In service	=>	192.168.1.2 1000	192.168.1.9 1000

It is important to understand that while the LSP has been set up and is operational, the interface on the End System(s) still needs some additional work. The easiest way to take advantage of the point to point layer2 path that has been established is to configure it as an IP interface. **The LSP is in port-to-port/untagged mode, so the data plane interface is eth1.** The teams can choose any available IP subnet for the link, ifconfig it, and then ping across it.

Login to red\_es2:

```
red_es2_pc: ~# ifconfig eth1 10.0.0.2 netmask 255.255.255.252
```

Login to red\_es1:

```
red_es1_pc: ~# ifconfig eth1 10.0.0.1 netmask 255.255.255.252
```

```
red_es1_pc: ~# ping 10.0.0.2
```

```
PING 10.0.0.2 (10.0.0.2) 56(84) bytes of data.
```

```
64 bytes from 10.0.0.2: icmp_seq=1 ttl=64 time=0.070 ms
```

```
64 bytes from 10.0.0.2: icmp_seq=2 ttl=64 time=0.050 ms
```

```
...
```

As we stated before, the TE address associated with the interface is NOT useable for IP packet forwarding. It is not actually configured on any interface. It is only configured in the control plane protocols as a means of identifying the paired interfaces/ports of the data plane topology.

## 7. Delete an LSP

You can delete the LSP at DRAGON CLI:

```
red-es1-dragon> delete lsp test1
```

## Exercise #2: The NARB and Local-ID mode

Objective #1: Incorporate the NARB in the network control plane.

Objective #2: Set up L2SC path from a port group on one switch to a port group on another switch.

### 2.1 The NARB

The NARB provides both intra-domain and inter-domain service routing functions. The path computation element (PCE) is used to create an Explicit Route Object (ERO) for the RSVP PATH message. This ERO specifies the path that the LSP is to take across administrative domains. The NARB interacts with RSVP and OSPF to create “loose hop” inter-domain EROs for local service requests, and is responsible for expanding loose hops across the local domain from transit requests, into a set of strict hops thru the core of the local domain.

The NARB utilizes OSPF messages between domains to disseminate topology information. The “public” topology information for the local domain is provided in the `narb.conf` file. This file must be edited to reflect the topology that the local domain wishes to advertise to peers. (Note: The current version of the NARB requires this public topology be expressly described manually rather than dynamically derived from the intra-domain link state database. Future versions should provide the ability to derive and/or selectively summarize the revealed internal topology.)

In this exercise, we will implement the NARB in order to provide advanced path computation capabilities on an intra-domain basis. While the intra-domain case will not take full advantage of the NARB’s capabilities, it’s presence will satisfy requirements to provide EROs for certain types of provisioning requests...most notably the “local ID” mode that we will implement as objective #2. And it will provide a good segue for the inter-domain configuration in exercise #3.

Procedure:

Setting up the NARB actually consists of configuring three functional entities: 1) an intra-domain OSPF listener, 2) an inter-domain OSPF speaker, and 3) the NARB itself. Traditionally, the NARB and all of its component daemons are run on their own host. While not strictly necessary, the path computation algorithms and link state processing in a large network could consume significant computational resources – at least more than a typical network element would normally have available.

In this exercise, we will run the NARB processes on a separate host. We must therefore allocate and set up appropriate GRE tunnels. One GRE tunnel must be established from the narb server to one of the intra-domain VLSRs. Any VLSR will work – this link will be configured in `ospfd` to flood LSAs to the NARB from inside the domain. Additional GRE tunnels will be required for inter-domain, but we’ll address those in Ex3.

2.1.1) Edit the `ospfd-intra.conf` file. In this file we need to describe a passive adjacency between the intra-domain narb ospfd instance and any one of the VLSR ospf instances within the local domain. The following statements make up the red domain `ospf-intra.conf` file:

```

! Intra-domain ospfd configuration for narb
! 2006/03/07 16:32:10
!
hostname red_narb_intra
password dragon
log stdout
!
interface gre6
 description GRE tunnel between red-narb and red-vlsr2
 ip ospf network point-to-point
!
router ospf
 ospf router-id 192.168.1.10
 network 10.1.6.2/30 area 0.0.0.0
 ospf-te router-address 192.168.1.10

```

There is no TE LINK between the NARB and VLSR.

2.1.2) For now, we will not define any inter-domain topology, so the `ospfd-inter.conf` file will contain only hostname and password info.

2.1.3) Copy `zebra.conf.sample` to **zebra.conf**.

2.1.4) Edit the `narb.conf` file. We must define the local domain id – we use the management CIDR block, but any unsigned integer could be used (e.g. 1 for red, 3 for yellow, etc). We also need to specify location of both the inter- and intra-domain ospf instances. (Even though we will not be implementing the inter-domain components yet, we still need to have an inter-domain ospfd active.)

```

!
domain-id {ip 192.168.1.0}
!
cli { host red-narb password dragon }
intra-domain-ospfd {address 192.168.1.10 port 2617
 originate-interface 10.1.6.2 area 0.0.0.0}

```

2.1.5) Before starting the NARB, please also login to the VLSR that's connected to the NARB, i.e. `vlsr2` in our case. Edit the file `ospfd.conf` so that the ospfd at `vlsr2` can pick up `gre6` to form adjacency to the intra-domain ospfd on the NARB. Restart `vlsr2`.

2.1.6) Make sure the VLSRs in the local domain are all up and active. Then start the NARB with:

```
#> /usr/local/dragon/bin/dragon.sh start-narb
```

The CLI ports associated with NARB elements are as follows:

- 2604 ospf-intra-domain
- 2614 ospf-inter-domain
- 2626 narb cli
- 2688 RCE
- 2611 dragond

You can telnet to these processes and inspect the ospf adjacencies and topology information.

To check intra-domain topologies at RCE CLI:

```
red_narb_pc:~# telnet localhost 2688
rce:cli>show topology intradomain
```

## 2.2 The Local-ID

The “Local ID” is a non-standard construct developed by DRAGON to facilitate switch to switch LSP creation. This feature can be used to allow a VLSR to proxy for an otherwise RSVP unaware device – e.g. a video camera, or computational cluster, etc. In the R&E networks, it is expected that Ethernet LSPs will often be used to link clusters in one location to clusters in another location – mapping a group of ports on one switch with a group of ports on some other switch. Since RSVP does not support VLANs as a termination point, the DRAGON software provides for the creation of a “local ID” on a local switch which can then be used as a sub-object for terminating a PATH request at the local switch. Via .conf file and/or CLI, a single [dumb] port, a group of ports, or a tagged group of ports can be set up and associated with a local-ID. This local ID can then be specified in the LSP configuration at the source and destination.

In order to use local-ID, there must be an active NARB available and configured in RSVPD.conf and the dragon.conf files at the source and/or destination VLSRs. As a general rule, pointers to the NARB should be configured on all VLSRs throughout the network.

While this is a non-standard feature incorporated into the DRAGON software, it provides a capability to link clustered systems on one switch to another set of clustered systems on another switch. The “local ID” is a terminating entity within a VLSR, allowing the signaling protocol to terminate the LSP to an internally defined VLAN/port group. In future releases, this feature should most likely be implemented as a point-to-multipoint LSP, but the mechanisms for path selection and routing for such LSPs is an advanced topic and not currently implemented.

Procedure:

2.2.1) Now that the NARB is configured and part of the local control plane, we need to tell the RSVPD and dragon daemons where to find the NARB. We must add the following statement to the red VLSR **RSVPD.conf**:

```
narb 192.168.1.10 2609
```

And we add the following statement to the red VLSR **dragon.conf** file in order for local-id to work:

```
configure narb intra-domain ip-address 192.168.1.10 port 2609
```

And, of course, restart the daemons in the usual way.

2.2.2) The local ID constructs must be defined before they can be used to establish an LSP. You can configure the local-id via the CLI, or you can define them in the \*.conf files.

Edit the dragon.conf file to define several local-ids to enable port, group, and tagged group LSPs to/from VLSR1 and VLSR3. The following statements will do so:

```
set local-id port 3
set local-id group 100 add 3
set local-id tagged-group 101 add 3
```

Once you have configured a number of local-ids for general use, or several specific local-ids, you are ready to login to the dragon CLI on VLSR1 (or VLSR3) and edit up an LSP. When using the tagged-group local-id, the vtag must match add both source and destination (this is fundamentally an issue associated with flat VLANs – the vtag is not actually swapped at each node and so must be maintained end-to-end. There are some emerging scenarios where this constraint will be reduced or eliminated, but the standards and generally available layer2 products do not support this capability yet.)

Login to the red-vlsr1 dragon CLI on port 2611 and configure the LSP:

```
Password: *****
```

```
red-vlsr1-dragon> edit lsp test1
red-vlsr1-dragon(edit-lsp-test1)# set source ip-address 192.168.1.4 port 3
destination ip-address 192.168.1.8 port 3
red-vlsr1-dragon(edit-lsp-test1)# set bandwidth eth100M swcap l2sc encoding
ethernet gpid ethernet
red-vlsr1-dragon(edit-lsp-test1)# exit
```

The LSP will be set between red-vlsr1-sw port 3 to red-vlsr1-sw3 port 3.

Once the LSP is in service, the end system can configure an IP address onto the appropriate interface and should be able to ping end-to-end. A traceroute should show no intervening hops.

### 2.2.3) narb\_test

narb\_test is a tool to query the NARB for an ERO. We use this tool often to test if a host is reachable to another host.

```
red_es1_pc:~# narb_test -H 192.168.1.10 -S 192.168.1.4 -D 192.168.1.8
NARB@[2007/05/10 16:33:05] : Request successful! ERO returned...
NARB@[2007/05/10 16:33:05] : HOP-TYPE [strict]: 11.1.2.1
NARB@[2007/05/10 16:33:05] : HOP-TYPE [strict]: 11.1.2.2
NARB@[2007/05/10 16:33:05] : HOP-TYPE [strict]: 11.1.4.1
NARB@[2007/05/10 16:33:05] : HOP-TYPE [strict]: 11.1.4.2
```

## **Exercise #3 Inter-Domain Routing and Signaling**

Objective: Configure and test a multi-domain GMPLS control and data plane.

Discussion:

From the previous exercises, we have constructed several independent networks each capable of dynamically provisioning LSPs across their respective domains. In this exercise, we will link these domains together at both the data plane and control plane to create one large service routing area.

The NARB will provide the key components for the inter-domain topology distribution and path computation. The NARB will also provide a rudimentary but important capability to manage the topology that each network advertises to its peer networks. In this exercise we will have two domains expose their full internal topology to their neighbors, and two domains which will reveal only a summarized view.

In this exercise, we will interconnect the four pod domains in a ring: Red connects to Blue which connects to Yellow which connects to Green which connects back to Red. Each pod will have two inter-domain links.

Inter-domain TE-links function slightly differently than intra-domain TE-Links: While OSPF will be aware of the inter-domain links and flood LSAs internally about the border link, these links themselves are passive, so ospf will not flood LSAs across the border to the neighbor domain. The control links associated with the inter-domain TE-link is only used by RSVP to signal up the link. Inter-domain routing will take place across control links between the NARB servers only.

Procedure:

3.1) First, as usual, we must make a plan: Lay out the inter-domain control and data plane links. Refer to diagram 3.1a. Add the appropriate data circuit IDs to your list of data links created in Exercise #1. Next, define the appropriate GRE tunnels for the corresponding control links. Assign IP addressing for both the TE links and the control links. Since this requires coordination with your colleagues in the neighboring pod, make sure you do so.

In addition to the GRE tunnels associated with the TE links, we will need to define several additional GRE tunnels that will be used by the NARB inter-domain ospfd.to peer with its neighboring NARBs. We need to define a GRE tunnel/control link for each NARB-NARB peering.

Edit the appropriate files on both the border VLSR nodes that service the inter-domain TE links and on the NARB server to create the GRE tunnels.

Go to the DRAGON Lab and install the appropriate Ethernet cabling to effect the desired data plane connectivity.

3.2) On each border VLSR we need to add the inter-domain link to the ospfd.conf. Add the “**interface gre<nn>**” configuration block. Under the “router ospf” block, you need to add “**passive-interface gre<nn>**” and the “**network <ctrl\_ip\_addr> area 0.0.0.0**” statement. And don’t forget to add an “**ospf-te interface gre<nn>**” block (looks like the other TE links).

Since RSVP will use this control link, we need to add the GRE interface to the RSVPD.conf file. You can look at the other interfaces to get the format.

Restart the VLSR.

3.3) We now move to the NARB.

On the NARB we need to do two things: first is to add the narb control links to the **ospfd-inter.conf** file, and second, we need to define the abstract topology in **narb.conf** that we want to advertise to our peers.

3.3.1) Edit the ospfd-inter.conf file to add the “**interface gre<nn>**” block, and the associated “**network...**” block.

3.3.2) Edit the narb.conf file. Add the following statement to locate the ospfd-inter daemon:

```
inter-domain-ospfd { address localhost port 2607 originate-interface 10.1.21.1 area 0.0.0.1 }
```

If there are more than one inter-domain-ospfd interfaces, you only need to pick one of the link to specify at narb.conf.

The following statements describe the **abstract topology** that NARB will distribute to its peers. Each router block specifies an LSR we wish to advertise as part of our topology. Within each router block are a set of “link” blocks that describe the links incident on the particular LSR. Note: The concept behind this topo specification is that we can explicitly control the topology info that gets exposed to the outside. So the router nodes and links so defined may or may not define a topology that resembles the real network. In the example below, red-narb only expose the red-vlsr3 which is the border VLSR and the two end systems.

```
!  
! red-vlsr3-pc  
!  
router {id 192.168.1.8  
!  
    intra-domain link to red-es1  
    link {id 192.168.1.2 type 1  
        max_bw 1000.0 max_rsv_bw 1000.0  
        unrsv_bw0 1000.0 unrsv_bw1 1000.0 unrsv_bw2 1000.0 unrsv_bw3 1000.0  
unrsv_bw4 1000.0 unrsv_bw5 1000.0 unrsv_bw6 1000.0 unrsv_bw7 1000.0  
        enc_type 2 sw_type 51  
        metric 50  
        local_if 11.1.4.2 remote_if 11.1.1.1  
        vlan_tags(100:100)
```

```

    }
!   inter-domain link to green-vlsr3-pc (D14/GRE14)
    link {id 192.168.4.8 type 1
        max_bw 1000.0 max_rsv_bw 1000.0
        unrsv_bw0 1000.0 unrsv_bw1 1000.0 unrsv_bw2 1000.0 unrsv_bw3 1000.0
unrsv_bw4 1000.0 unrsv_bw5 1000.0 unrsv_bw6 1000.0 unrsv_bw7 1000.0
        enc_type 2 sw_type 51
        metric 50
        local_if 11.4.14.2 remote_if 11.4.14.1
        vlan_tags(100:100)
    }
!   inter-domain link to blue-vlsr1-pc (D11/GRE11)
    link {id 192.168.2.4 type 1
        max_bw 1000.0 max_rsv_bw 1000.0
        unrsv_bw0 1000.0 unrsv_bw1 1000.0 unrsv_bw2 1000.0 unrsv_bw3 1000.0
unrsv_bw4 1000.0 unrsv_bw5 1000.0 unrsv_bw6 1000.0 unrsv_bw7 1000.0
        enc_type 2 sw_type 51
        metric 50
        local_if 11.1.11.1 remote_if 11.1.11.2
        vlan_tags(100:100)
    }
!   intra-domain link to red-es2 (D5/GRE5)
    link {id 192.168.1.9 type 1
        max_bw 1000.0 max_rsv_bw 1000.0
        unrsv_bw0 1000.0 unrsv_bw1 1000.0 unrsv_bw2 1000.0 unrsv_bw3 1000.0
unrsv_bw4 1000.0 unrsv_bw5 1000.0 unrsv_bw6 1000.0 unrsv_bw7 1000.0
        enc_type 2 sw_type 51
        metric 50
        local_if 11.1.5.1 remote_if 11.1.5.2
        vlan_tags(100:100)
    }
}
}
!
! red-es1
!
router {id 192.168.1.2
!   abstracted intra-domain link to red-vlsr3-pc
    link {id 192.168.1.8 type 1
        max_bw 1000.0 max_rsv_bw 1000.0
        unrsv_bw0 1000.0 unrsv_bw1 1000.0 unrsv_bw2 1000.0 unrsv_bw3 1000.0
unrsv_bw4 1000.0 unrsv_bw5 1000.0 unrsv_bw6 1000.0 unrsv_bw7 1000.0
        enc_type 2 sw_type 51
        metric 50
        local_if 11.1.1.1 remote_if 11.1.4.2
        vlan_tags(100:100)
    }
}
!
! red-es2
!
router {id 192.168.1.9
!   intra-domain link to red-vlsr3-pc
    link {id 192.168.1.8 type 1
        max_bw 1000.0 max_rsv_bw 1000.0
        unrsv_bw0 1000.0 unrsv_bw1 1000.0 unrsv_bw2 1000.0 unrsv_bw3 1000.0
unrsv_bw4 1000.0 unrsv_bw5 1000.0 unrsv_bw6 1000.0 unrsv_bw7 1000.0
        enc_type 2 sw_type 51
    }
}

```

```

metric 50
local_if 11.1.5.2 remote_if 11.1.5.1
vlan_tags(100:100)
}
}

```

Finally, we need to tell NARB which TE links are the inter-domain links. The following statement should be added following the topology blocks to identify the inter-domain links and the domain to which they lead:

```
inter-domain-te-link { id 11.1.11.2 narb-peer 10.1.21.2 port 2609 }
```

The id specified is the TE address of the \*far\* end of the TE link (not the local end) and its associated narb server IP address and port in the neighbor domain.

Restart the NARB daemons.

At NARB CLI,

To check the topology advertised by NARB:

```

red_narb_pc:~# telnet localhost 2626
password:
narb:cli>show topology

```

To check details on a particular TE-link:

```
narb:cli>show link local_if_addr local_te_addr remote_if_addr remote_te_addr
```

To check related daemons and peering NARBs:

```
narb:cli>show module
```

At RCE CLI,

To check inter- and intra-domain topologies:

```

red_narb_pc:~# telnet localhost 2688
rce:cli>show topology interdomain
...

```

3.3.3) Login to the End System and edit the file **dragon.conf**. Add the line “configure narb intra-domain ip-address <narb management ip> port 2609”. Restart the End System.

3.3.4) Go to an End System and login to dragon and set up an LSP across domains.

```

red-es1> edit lsp inter1
red-es1(edit-lsp-inter1)# set source ip-address 192.168.1.2 lsp-id 92 destination
ip 192.168.4.2 tunnel-id 298
red-es1(edit-lsp-inter1)# set bandwidth eth100M swcap 12sc encoding ethernet gpid
ethernet
red-es1(edit-lsp-inter1)# set vtag any

```

When using “set vtag any”, the LSP will be in tagged mode from ES to ES, and a vlan will be assigned to the LSP by the NARB.

```
red-es1> show lsp inter1
Src 192.168.1.2/92, dest 192.168.4.2/298
Generic TSPEC R=eth100M, B=eth100M, P=eth100M, m=100, M=1500
Encoding ethernet, Switching l2sc, G-Pid ethernet
Ingress Local ID Type: none id, Value: 92
Egress Local ID Type: none id, Value: 298.
E2E LSP VLAN Tag: 100.
Status: In service
```

The layer 3 IP address should be assigned to *eth1.<vlan>* instead of *eth1*.

## Exercise #4: Using Application Specific Topology (AST)

Objective: In this exercise, we will use the Application Specific Topology Description Language (ASTDL) to describe and instantiate a network.

Discussion:

In the previous exercises, we have created individual LSPs thru the CLI in dragon. Dragon software also includes Application Specific Topology (AST) tools for you to dynamically set up and tear down an entire network topology, i.e. a topology consisting of one or more LSPs.

ASTDL is an XML based specification for describing a distributed application. It provides the ability to specify all the nodes and links in the application, and to explicitly define the characteristics of each. The current version of ASTDL is a proof of concept implementation. Future versions will provide more sophisticated capabilities such as dynamic ASTs that can evolve and morph with the needs of the application community, and object oriented ASTs that allow embedding or nesting of ASTs inside other ASTs. The result is a very powerful means of creating highly customized dedicated resource environments for a broad range of applications.

In order to instantiate an AST, there is an AST protocol that processes the XML, locates and reserves the necessary resources, and sets up the network links, and passes control to user applications at the nodes. For our purposes in this workshop, we will focus on the ASTs as a means of using XML constructs for LSP creation and deletion.

A simple AST XML document is shown below:

```
<topology action="SET UP_REQ">
<resource type="pc" name="red-es1">
  <ip>192.168.1.2</ip>
  <router_id>192.168.1.2</router_id>
</resource>
<resource type="pc" name="red-es2">
  <ip>192.168.1.9</ip>
  <router_id>192.168.1.9</router_id>
</resource>
<resource type="non_uni" name="red-es1-es2">
  <src>
    <es>red-es1</es>
    <assign_ip>14.14.14.1/30</assign_ip>
  </src>
  <dest>
    <es>red-es2</es>
    <assign_ip>14.14.14.2/30</assign_ip>
  </dest>
  <te_params>
    <bandwidth>eth100M</bandwidth>
    <swcap>l2sc</swcap>
    <encoding>ethernet</encoding>
    <gp_id>ethernet</gp_id>
  </te_params>
</resource>
```

```
</topology>
```

This XML describes an application consisting of two end systems with a dedicated light path between them. In essence, the XML describes a graph where both computational nodes and network links are considered application “resources” of different types and with different characteristics.

In brief, the AST protocol for instantiating a topology employs a “master – minion” model. The AST XML is processed by the AST\_Master agent, which identifies and parcels out tasks to appropriate “minions” (i.e. dragon on the end systems) to set up the distributed environment. When the minions have all completed their assigned tasks, the master notifies all the minions to pass control to the application.

This exercise will create the “barbell” topology described above using the AST method. And if time allows, we will try to create a more complex topology such as a triangle – three end systems each connected by LSP to the other two.

Procedure:

#### 4.1) Set up AST server (ast\_master)

The AST server (ast\_master) is located at /usr/local/dragon/bin/ast\_master. We are going to put these lines to /usr/local/dragon/etc/ast\_master.conf to specify the ast\_master hostname and password used in AST CLI.

```
hostname ast_master  
password dragon
```

To run ast\_master:

```
red_es1#> ast_master -d
```

You can run ast\_master on **any machine** in the workshop. However, once the AST is established, that host will maintain state information necessary for monitoring and managing the AST. (Note: The AST may cross many administrative boundaries, so the master may be the only entity that really has a full view of the resources.)

You can login to ast\_master CLI to query AST status.

```
red_es1_pc:~# telnet localhost 2612
```

```
Password: *****  
ast_master> show ast  
No active AST on the list  
ast_master>
```

“show ast” will only return a list of “active” AST(s) that haven’t been released. If you want to query an already released AST, you need to provide the specified <ast\_id>.

```
ast_master> show ast red_es1_pc_1176840542
```

```
AST red_es1_pc_1176840542 status
```

```
-----  
overall status: AST_SUCCESS  
action status: RELEASE_RESP  
xml_file: blue-any.xml  
Total number of nodes: 2  
  NODE (red-es1) PC:  
...
```

## 4.2) Setting up node\_agent at ES

node\_agent is a minion running on the end systems that is responsible for initialization of the nodal components of the AST. Dragond contains the complementary minion for the network components of the application. For instance, dragond will be responsible for establishing the LSP, but node\_agent will be responsible for mapping those links to specific application functions.

The configuration file for node\_agent should be at /usr/local/dragon/etc/node\_agent.conf. And it should contain one line

```
set interface eth1
```

To run node\_agent,

```
node_agent -d
```

4.3) A sample AST XML file is located at /usr/local/dragon/etc/ast.xml.sample. You can copy this file and use the editor of your choice to create a simple text file containing the following XML specification. This is an example to set up an LSP between red\_es1 and red\_es2.

```
<topology action="SET UP_REQ">  
<resource type="pc" name="red-es1">  
  <ip>192.168.1.2</ip>  
  <router_id>192.168.1.2</router_id>  
</resource>  
<resource type="pc" name="red-es2">  
  <ip>192.168.1.9</ip>  
  <router_id>192.168.1.9</router_id>  
</resource>  
<resource type="non_uni" name="red-es1-es2">  
  <src>  
    <es>red-es1</es>  
    <assign_ip>14.14.14.1/30</assign_ip>  
  </src>  
  <dest>  
    <es>red-es2</es>  
    <assign_ip>14.14.14.2/30</assign_ip>  
  </dest>  
  <te_params>  
    <bandwidth>eth100M</bandwidth>  
    <swcap>l2sc</swcap>  
    <encoding>ethernet</encoding>
```

```
<gpId>ethernet</gpId>
  </te_params>
</resource>
</topology>
```

The xml file contains 3 resources: 2 ES as type “pc” and 1 link as type non\_uni. When defining an ES, `<ip_addr>` is the management IP and `<router_id>` is the ospfd router\_id (in our case, they are the same) on the ES. `<src>` and `<dest>` should refer to nodes defined within the xml file.

If we want vlan on this LSP from end to end, we would put `<vtag>... </vtag>` under the link `<resource>`. In this workshop, we would use vlan 100 – 200.

#### 4.4) Commit the AST

We provide **astb** as a client to send file to and receive response from the ast\_master. You can run

```
astb -v <filename>
```

to verify if xml file is syntactically correct.

```
astb -f <filename>
```

to actually commit the xml file to ast\_master and let it be provisioned!!

Note: astb will send the file to the ast\_master running at the same host. Please make sure ast\_master is also running.

ast\_master will return an xml message to astb to tell the user if the topology is successfully set up. This output will contain the xml tag `<ast_id>` which is a globally unique identifier that can be used to reference the AST later. Please copy the `<ast_id>` somewhere so that you can have access to it when you want to delete the topology.

```
<topology ast_id="red_es1_1173550824" action="SET UP_RESP">
<status>AST_SUCCESS</status>
...
<resource type="non_uni" name="red-es1-es2">
  <status>AST_SUCCESS</status>
  <link_status>IN-SERVICE</link_status>
  <lsp_name>AST-93194784</lsp_name>
...
```

To check the status of the LSP, login to the dragon CLI (i.e. in this case, it’s red\_es1 and red\_es2) and do a “show lsp `lsp_name`”.

To debug:

1. /var/log/ast\_master.log is the log file for ast\_master
2. /usr/local/ast/<ast\_id>/ directory contains all xml files exchanged in the life time of the ast.
3. ast\_master provides CLI at port 2612. We can telnet to that port and do “show ast” to list all active ASTs that’s being managed or “show ast <ast\_id>” to check the status of a particular AST

Assuming the AST was successfully created, you can now login to the End Systems specified in the AST description, and to ping each other over the LSP if you have provided <assign\_ip> at the SET UP\_REQ XML file.

#### 4.5) Releasing all resources associated with an AST

The following shell command will release all the resources associated with the AST:

```
astb -r ast_id
```

The ast\_master will return a result in an xml file. The node\_agent will also automatically remove the vlan that it created on the end systems for this AST.

Or, you can login to AST CLI.

To release a particular AST, enter command

```
ast_master> release ast ast_id
```

To release ALL active AST(s) in ast\_master, enter command

```
ast_master> release ast all
```

#### 4.6) Building a triangle AST

The following AST XML will create a triangle topology:

```
<topology action="SET UP_REQ">
<resource type="pc" name="red-es1">
  <ip>192.168.1.2</ip>
  <router_id>192.168.1.2</router_id>
</resource>
<resource type="pc" name="blue-es2">
  <ip>192.168.2.9</ip>
  <router_id>192.168.2.9</router_id>
</resource>
<resource type="pc" name="green-es1">
  <ip>192.168.4.2</ip>
  <router_id>192.168.4.2</router_id>
</resource>
<resource type="non_uni" name="red-blue">
  <src>
    <es>red-es1</es>    <assign_ip>110.0.0.1/30</assign_ip>
  </src>
  <dest>
    <es>blue-es2</es>  <assign_ip>110.0.0.2/30</assign_ip>
  </dest>
  <te_params>
    <bandwidth>eth100M</bandwidth>  <swcap>l2sc</swcap>
    <encoding>ethernet</encoding>    <gpip>ethernet</gpip>
    <vtag>l10</vtag>
  </te_params>
</resource>
```

```

<resource type="non_uni" name="blue-green">
  <src>
    <es>blue-es2</es> <assign_ip>111.0.0.1/30</assign_ip>
  </src>
  <dest>
    <es>green-es1</es> <assign_ip>111.0.0.2/30</assign_ip>
  </dest>
  <te_params>
    <bandwidth>eth100M</bandwidth> <swcap>l2sc</swcap>
    <encoding>ethernet</encoding> <gp_id>ethernet</gp_id>
    <vtag>l11</vtag>
  </te_params>
</resource>
<resource type="non_uni" name="green-red">
  <src>
    <es>green-es1</es> <assign_ip>112.0.0.1/30</assign_ip>
  </src>
  <dest>
    <es>red-es1</es> <assign_ip>112.0.0.2/30</assign_ip>
  </dest>
  <te_params>
    <bandwidth>eth100M</bandwidth> <swcap>l2sc</swcap>
    <encoding>ethernet</encoding> <gp_id>ethernet</gp_id>
    <vtag>l12</vtag>
  </te_params>
</resource>
</topology>

```

This file will set up a triangle topology between blue-es2, red-es1 and green-es1. If you logon to dragoon CLI on any of the machines, you will see two LSPs in service:

```

red-es1> show lsp
**LSP status summary**
Name      Status  Dir  Source (IP/LSP ID)  Destination (IP/Tunnel ID)
-----
AST-011433760
  In service => 192.168.1.2/1318 192.168.4.2/1004
AST-366811584
  In service => 192.168.2.9/1909 192.168.1.2/2

```

red-vlsr1 switch will show

```
RED_VLSR1_SW# show vlan
```

Vlan	Name	Ports	Type	Authorization
1	1	g(1-2), ch(1-8)	other	Required
100	100	g(3, 5)	permanent	Required
101	101		permanent	Required
102	102	g(3, 5)	permanent	Required