Faculty of Engineering
Department of Information Technology
Chairman: Prof. Dr. Ir. P. LAGASSE

Development of an SNMP agent for Ethernet switches

by
Valentín Carela Español

Promoter: Mario Pickavet – Didier Colle
Advisor: Wouter Tavernier

Thesis submitted in partial fulfilment of the requirements for the degree of
MASTER IN INFORMATICS

Academic year 2006-2007

Acknowledgements

This thesis has been done with the help of a lot of people. I would like to especially thank following people for their help at the realisation of this thesis:

- Mario Pickavet, my promoter, for offering me this opportunity.
- Wouter Tavernier, my advisor, and Didier Colle, my other promoter, for helping me during the whole thesis and for your infinity patience.
- Fernando Solano, my colleague, for keeping me company during the long working days
- My family for taking care of me in the distance during these six months.
- To all the new friends I met in Gent, although they did not let me work, I will never forget you.
- Nuria Castell, my promoter in Spain, for letting me to make an Erasmus in this beautiful city.
- And thanks for everybody who were there and have not been named…

Overview

Development of an SNMP agent for Ethernet switches

by

Valentín Carela Español

Thesis submitted in partial fulfilment of the requirements for the degree of
MASTER IN INFORMATICS

Academic year 2006-2007

Promoter: Mario Pickavet – Didier Colle
Advisor: Wouter Tavernier

Faculty of Engineering
Department of Information Technology
Chairman: Prof. Dr. Ir. P. LAGASSE

Summary

To answer some of the present requests for the networks appears GMPLS. However the implantation of this technology is difficult because still there are a lot of non capable GMPLS elements in the network. DRAGON is a software that solves this problem in the Ethernet networks using SNMP to control this elements and making these equipments capable for working in a GMPLS network. The objective of this thesis is make the Click Router, a open source software for building routers, capable of working , with the help of DRAGON, in a GMPLS network.

Keywords: GMPLS, DRAGON, SNMP, Click Router

Development of an SNMP agent for Ethernet switches

Valentín Carela Español

Supervisor(s): Mario Pickavet, Didier Colle, Wouter Tavernier

h *Abstract*: To answer some of the present requests for the networks appears GMPLS. However the implantation of this technology is difficult because still there are a lot of non capable GMPLS elements in the network. DRAGON is a software that solves this problem in the Ethernet networks using SNMP to control this elements and making these equipments capable for working in a GMPLS network. The objective of this thesis is make the Click Router, a open source software for building routers, capable of working , with the help of DRAGON, in a GMPLS network.

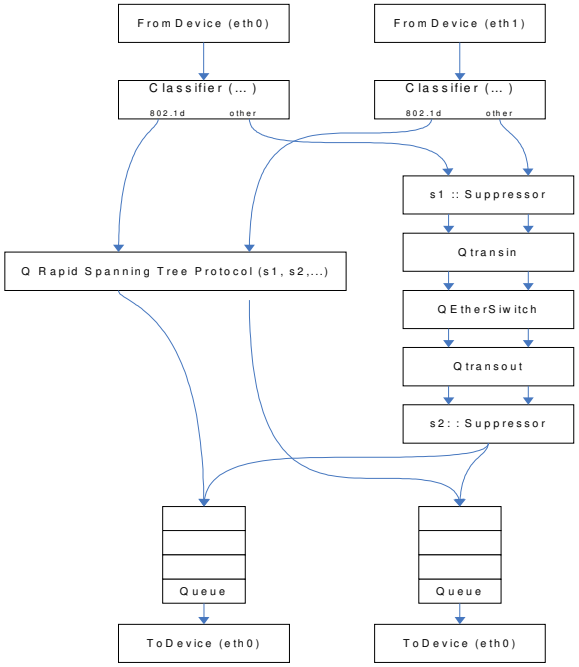
***Keywords*: GMPLS, DRAGON, SNMP, Click Router**

INTRODUCTION

First of all it is necessary to know some of the Ethernet networks because Click Router works over an Ethernet network. The more important points of the Ethernet networks for this thesis are the STP and the use of VLANs. The Spanning Tree Protocol (STP) creates a spanning tree within a mesh network of connected layer-2 bridges (typically Ethernet switches), and disables the links which are not part of that tree, leaving a single active path between any two network nodes. The VLAN is a method of creating independent logical networks within a physical network. To make this possible is necessary to add a new fragment to the packet, the VLAN tag.

In our case we use the Click Router that it is a new software for building flexible and configurable routers. The components of this router are packet processing modules called elements. Each element implements simple router functions like packet classification, queuing, scheduling, and interfacing with network devices. To build a router configuration, the user chooses a collection of elements and connects them into a directed graph.

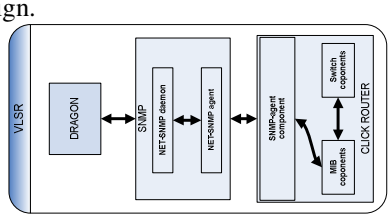
A typical configuration of Ethernet switch that support the characteristics before commented could be the example of the Figure 1. The main characteristics of these elements are that the *Q Rapid Spanning Tree Protocol* and the *Suppressors* are in charge of keeping the network without loops. The element *Qtransin* is in charge to tagging with a VLAN the incoming packets that are untagged. The *Qetherswitch* is in charge of making the forwarding decision. And the *Qtransout* is in charge of untagging the outgoing packets that have not to be tagged.



a Example of a Q Ethernet Switch by Click Router

The network that we commented before is not capable to guarantee a bandwidth for a client. In order to solve this problem appears the GMPLS that is an extension of Multiprotocol Label Switching. The Routing Protocol that we use is OSPF-TE and the Signaling Protocol that we use is RSVP-TE. OSPF-TE is a routing protocol that works as the OSPF but with Traffic Engineering that allow the routers to make forwarding decisions using QoS variables. The RSVP-TE is in charge of making the reservation of the LSP that is a path between two points with Traffic Engineering.

After this background it is possible to clarify that the objective of this thesis is to substitute one of the commercial switch that you can see in the above figure for the Click Router and below image show us their design.



b Design of the elements that we have to implement

DEVELOPMENT

A *SNMP*

Between the Click Router and the Dragon software we need a protocol to communicate both elements. The protocol in charge of this will be SNMP. The collection of information about the managed device that the agent keeps is called Management Information Base (MIB). The version that we are going to use is the NET-SNMP suite.

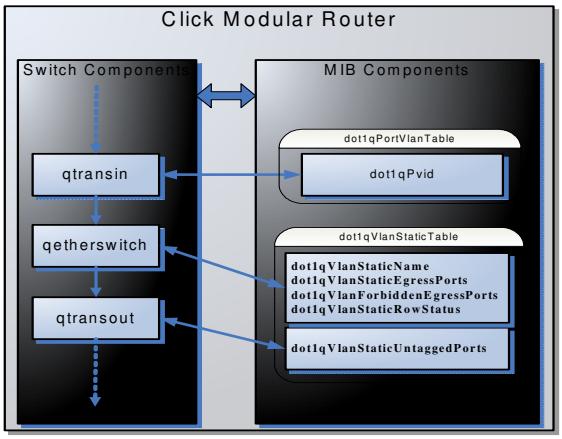
b *DRAGON*

The project DRAGON studies and develops (open source) software to enable dynamic provisioning of network resources on an interdomain basis across heterogeneous network technologies. The DRAGON software works with a switch with which it communicates through SNMP to create a VLSR. The VLSR enables Ethernet switches to act as a label switch in a GMPLS controlled network.

To know what information will be needed by the DRAGON software from the Click Router we made a test with a commercial switch and find out that the SNMP tables needed by DRAGON are the *dot1qVlanStaticTable* and the *dot1qPortVlanTable.*

d Implementation in Click Router

We implemented this tables as a new SNMP elements of the Click Router that share the information with the real switch elements.



c New structure of created and modified elements in Click Router

In the switch element *Qtransin* we implemented the table *dot1qPvid* that has the values of the tags that you have to add to the untagged frames. In the *Qetherswitch* element we implemented the table *dot1qVlanStaticEgressPorts* that has the values of VLANs configuration (which VLAN belongs to each port). We also implemented other tables in this element but there are not important for DRAGON. Finally in the switch element *Qtransout* we implemented the *dotq1VlanStaticUntaggedPorts* that has the values of the VLAN/ports that should be untagged before leaving the Click Router.

TESTS

We made four tests before the final test. Next I am going to explain you the objective of each tests. In all of the tests the Click Router works properly.

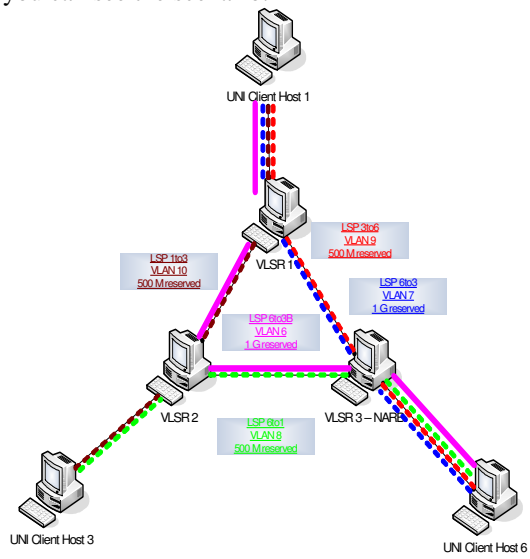
Test 1: Set up a LSP through a VLSR.

Test 2: Set up more than one LSP through a VLSR.

Test 3: Set up more than one LSP between two VLSR.

Test 4: Set up two LSP between two hosts.

At the end of these tests we made a final test with 3 VLSRs and 3 Host where in the first phase we used all the bandwidth available of a link between two VLSRs. In the second phase we tried to set up a LSP between those two VLSRs and the DRAGON software modify the theoretically path to make possible the reservation of the bandwidth through the other VLSR that still had bandwidth available. In the below picture you can see the scenario:



d Final Test - Phase II

CONCLUSIONS

GMLPS is an interesting technology but there are still many Ethernet switches that do not support GMPLS. However, DRAGON allows the non GMPLS Ethernet switches to work in a GMPLS network. On the other hand, Click Router is an open source software that is now capable of working with DRAGON. If we use these elements together the final result could be a very productive and cheap method to solve the network present petitions.

CONTENTS

INDEX OF FIGURES

TABLE OF ABBREVIATIONS

ASTB	Application Specific Topology Builder
CSA	Client System Agent
DRAGON	Dynamic Resource Allocation via GMPLS Optical Networks
GMPLS	Generalized MPLS
ICIR	ICSI Center for Internet Research
IETF	Internet Engineering Task Force
LER	Label Edge Router
LSA	Link-state advertisement
LSP	Label Switch Path
LSR	Label Switch Router
MIT	Massachusetts Institute of Technology
MPLS	Multiprotocol Label Switching
NARB	Network Aware Resource Broker
OSPF	Open Shortest Path First
OSPF-TE	OSPF Traffic Engineering
QoS	Quality of Service
RCE	Resource Computation Element
RFC	Request for Comments
RSVP	Resource Reservation Protocol
RSVP-TE	RSVP Traffic Engineering
SNMP	Simple Network Management Protocol
TED	Traffic Engineering Database
TLV	Type-Length-Value
UCLA	University of California, Los Angeles
UNI	User Network Interface
VLSR	Virtual Label Switch Router

Chapter 1
Introduction and objectives

In this chapter, an overview of the thesis' origin will be given, seeking to give the reader a broad perspective of the causes that moved my tutors to proposing it and of its primary objectives.

1.1 Introduction

Click Router is an open source router that was originally developed at MIT with subsequent development at Mazu Networks, ICIR and UCLA. The main characteristic of this router is that it is a modular router made by different elements that allows different configurations. Some new elements had been developed for this router with the goal of make it capable of working as a QinQ Ethernet Switch.

On the other hand new software called DRAGON appears in the panorama with the purpose to enable dynamic provisioning of network resources on an inter-domain basis across heterogeneous network technologies.

The purpose of my thesis would be to use and perhaps extend the new Click Router elements to make it capable of working with the DRAGON software.

1.2 Objectives

Finally I decided to choose this subject. It was interesting topic for me because it touches on many branches of the networks. Because of this the first objective was to learn the background of both elements.

Once I understood the background the next step would be to understand how the Click Router works and then how the DRAGON software works.

The penultimate objective would be to implement the necessary elements to make the Click Router capable of working with the DRAGON software.

Finally, to finish the project I would have to demonstrate that both elements together work properly with some tests.

1.3 Working phases

The first goal, as has been mentioned before, would be to understand the background of the different elements of the project. Because of this the first step of the background would be to refresh some subjects about Ethernet as it could be the operation of STP, the VLANs...

Also inside of the background the next step would be to study the GMPLS protocol that is the base of the DRAGON software. Another point of this background would be the study of the SNMP, a protocol that will be used to communicate the Click Router with the DRAGON software.

The next step would be to understand the operation of the Click Router, how to add new elements, how to communicate the different elements and study the different characteristics of this router.

Then it would be the turn of the DRAGON software. The understanding of this software will be more difficult than the Click Router because I would not have any similar software of DRAGON like could be a commercial router for the Click Router to use it like a reference. To understand the operation of the DRAGON software we will make some tests with a commercial switch.

Once I understood the operation of the different elements the next point would be to create or modify the necessary elements of the Click Router to make it capable of working with the DRAGON software. This step will have two points. The first one consists of the internal implementation of the elements and the second one consists of the SNMP communication between the Click Router elements and the Dragon software.

Finally the last step would be to demonstrate that all the work done works properly. To do it we will make some tests to show the different utilities of the combination of both elements.

Chapter 2
GMPLS

In this chapter is described the Generalized Multiprotocol Label Switching or GMPLS. All information found in this chapter is based on the *Sara Project* documentation produced by the RFC 3945 that describes this protocol and is complimented with the *From MPLS to GMPLS* presentation by Alcatel and other contributions.

2.1 Origins

It is easy to guess that GMPLS comes from MPLS. MPLS was introduced in the nineties and its best characteristics are that it could set up multiple tunnels and apply traffic engineering properties to them and also with MPLS had found a way to make two opposing Technologies coexist next to each-other and establish end-to-end paths in both packet-based and cell-based networks.

At the beginning of the new millennium appears GMPLS to put together all the current networking technologies. The GMPLS is an extension of MPLS that solves some problems and adds new features. GMPLS has a set of five interfaces such as a Time-Division Multiplex capable, Lambda Switch capable or Fiber Switched capable interfaces as well as the Packet switch capable and Layer-2 Switch capable interfaces inherited from MPLS. Furthermore, of the diversity of networking technologies the GMPLS supports, it eliminates the need of an operator, the entire network can be automated and no human interference will be required in the tunnelling process.

2.2 GMPLS Operation

To understand how the GMPLS works first it is necessary to understand how MPLS operates. MPLS uses a label which is added to the header of a packet. This label can represent a time-slot, physical space or even a single wavelength.

When a packet enters in a MPLS network a label is added to the packet by the Label Edge Router LER. With that label the packet travels over the Labeled Switched Path LSP. This packet will pass through each Label Switched Router LSR that belongs to the LSP. Each LSR will extract the label, make a forwarding decision and look up for the corresponding label for the specific interface in its Label Information Base and then will insert that label. When the packet arrives to the last hop of the MPLS network the LER will extract the label and commit the packet to the corresponding interface.

In order to make this process possible it is necessary a control mechanism. In MPLS this is done by a Label Distribution Protocol and one of the most popular that will be the one we are going to use is the Resource reservation Protocol with Traffic Engineering RSVP-TE. Also it is necessary to know what the data network looks like and this is responsible of a routing protocol. One of the possible that will be our option is the OSPF-TE protocol. Both protocols will be explained later.

GMPLS works in many ways identical to MPLS. One of the differences is that multiple LSPs can reside within other LSPs in GMPLS. However the LSPs should be set up between identical interfaces. Another difference is that due to GMPLS could work with different technologies the Data-plan and the control plane can be physically separated. There are two new features in GMPLS. First it is the usage of bi-directional LSPs. The other feature is the Suggested label that helps GMPLS to speed up the process.

2.3 GMPLS Routing

In GMPLS the routing has different responsibility than other Layer 3 networks. It is used to share information about the network topology and QoS configuration of the underlaying data plane. Because of this the GMPLS has a requirement that it is to be an IP based network due to the GMPLS routing protocols work only in these networks.

In our case we are going to use OSPF-TE that works similar than the OSPF but with some extensions. Next those extensions will be explained.

2.3.1 OSPF-TE

OSPF-TE is the Traffic Engineering version of OSPF that in many ways works similar to this. Using this version the LSR can know and transport all the TE necessary information. OSPF-TE uses a 10 LSA type which means that it is limited to the area in which it resides.

In order to hold the Traffic Engineering information each host which processes TE information will have a Traffic Engineering Database TED. The OSPF add dynamic properties to the route calculation algorithm like the Maximum Reservable bandwidth, the Unreserved bandwidth and the Available bandwidth. These fields are distributed between network nodes via the Type-Length-Value TLV fields.

2.4 GMPLS Signaling

The function of the GMPLS signaling is to set up and tear-down the LSPs. In our case we are going to use RSVP-TE and it will be explained next. But first the Link Management Protocol LMP objectives will be commented. The LMP is responsible for maintaining active knowledge of the LSP state and the connection state. Furthermore the LMP is also responsible of the fail-over options.

2.4.1 RSVP-TE

The RSVP-TE signaling has several objectives. The most well-known is the LSP Establishment. When an LSR wants to set up an LSP it sends an LSP setup message. In RSVP this message has the name of RSVP Path message and it is send to the next LSR between itself and the LSP destination. This LSP will be active when the origin LSR receives a LSP Accept message. This LSP Accept message will come from the destination LSR hop by hop to the origin LSR. In RSVP this message is called RSVP Resv message and it has the label to be used on the LSP and the TE properties of the LSP. From that moment the LSP becomes active and the data can flow.

In order to tear-down the LSP normally the LSR which requested the LSP will send an LSP Downstream release message telling every LSR between the origin and the destination that the LSP will be removed and the resources will be available again. This flow does not need

answer and this message is called RSVP Tear message. Also another LSR could send a RSVP PathError message to tear down the LSP.

Chapter 3
Click Router

This chapter will describe one of the components that we use in this project. Here you will find a brief explanation and summary of the more important points for this project of the documents *The Click Modular Router* that you can find in the bibliography.

Click is new software for building flexible and configurable routers. The components of this router are packet processing modules called elements. Each element implements simple router functions like packet classification, queuing, scheduling, and interfacing with network devices. To build a router configuration, the user chooses a collection of elements and connects them into a directed graph.

3.1 Architecture

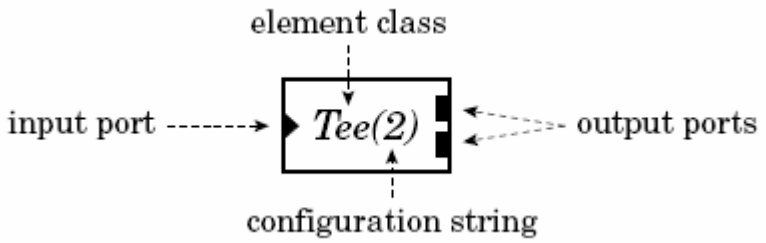
A Click element represents a unit of router processing. An element can represent a conceptually simple computation, such as decrementing an IP packets time-to-live field, or sometimes a large and complex computation, such an IP routing.

Inside a Click Router each element is a C++ object. The connections between these elements are represented as pointers to element objects, and passing a packet along a connection is implemented as a single virtual function call.

The most important parts of an element are:

- **Element class**. This specifies the code that should be executed when the element processes a packet, also the element's initialization procedure and the data layout.
- **Ports**. An element can have any number of input and output ports. Every connection goes from an output port to an input port.
- **Configuration string**. It is an optional configuration string that contains additional arguments that are passed to the element at router initialization time.

- **Method interfaces**. Each element supports one or more method interfaces. Every element supports the simple packet-transfer interface, but elements can create and export additional interfaces.



b Element example

3.1.1 Connections

Click supports two kinds of connections, push and pull. On a push connection, packets start at the source element and are passed downstream to the destination element. On a pull connection the destination element initiates packet transfer asking the source element to return a packet, or a null pointer if no packet is available.

Each port of the router is either push or pull; connections between two push ports are push, and connections between two pull ports are pull. Connections between a push port and a pull port are illegal. A port can also be an agnostic port, which behaves as push when connected to push ports and pull when connected to pull ports.

3.1.2 Packet Storage

Click elements do not have implicit queues on their input and output ports but there is implemented a separate Queue element. This element has a push input port and a pull output port; the input port responds to pushed packets by enqueueing them, and the output port responds to pull requests by dequeueing packets and returning them.

3.1.3 CPU Scheduling

Click runs in a single thread. It means that any packet transfer method must return to its caller before another task can begin. Therefore, the placement of Queues in a configuration graph determines how CPU scheduling may be performed.

3.1.4 Language

Click configurations are written in a simple language with two important constructs: declarations create elements, and connections say how they should be connected. The language contains a mechanism called compound elements that lets users define their own elements classes. A compound element is a router configuration fragment that behaves like an element class. At initialization time, each use of a compound element is compiled into the corresponding collection of simple elements. The language is declarative, it specifies what elements to create and how they should be connected, not how to process packets procedurally.

3.1.5 Installing Configurations

There are two drivers that can run Click router configurations, a Linux in-kernel driver and a user-level driver. The user-level driver is most useful for profiling and debugging, while the in-kernel driver is good for production work.

Installing a new configuration normally destroys any old configuration; for instance, any packets stored in old queues are dropped. This starts the new configuration from a predictable empty state. However, Click supports two techniques for changing a configuration without losing information:

- **Handlers**. They are access points for user interaction. A Click routing table element, for example, would likely provide add_route and del_route handlers as access points for user-level routing protocol implementations. Handlers are also useful for exporting statistics and other element information.

- **Hot swapping**. Some configuration changes, such as adding new elements, are more complex than handlers can support. In these cases, the user can write a new configuration file and install it with a hot-swapping option.

3.1.6 Element Implementation

Each Click element class corresponds to a subclass of the C++ class Element, which has about 20 virtual functions. Only three virtual functions are used during router operation, namely push, pull, and run_scheduled.

Before you start to develop a new element for Click is better to think about develop fine-grained elements, which have simple specifications, to coarse-grained elements with more complex specifications. However there are some functions that cannot be implemented in fine-grained elements.

Sometimes you could need to send information from an element to another element. Click uses annotations to carry such information along. An annotation is a piece of information attached to the packet header, but not part of the packet data; the Linux packet structure, sk_buff, provides 48 bytes of annotation space.

Chapter 4
SNMP

In this chapter you are going to find information about the SNMP extracted from the SNMP chapter of the *Cisco Documentation* and the *Net-SNMP* webpage and also some contributions of another DRAGON projects.

The Simple Network Management Protocol (SNMP) is an application layer control that is used by network management systems to exchange management information between network devices. SNMP enables network administrators to manage network performance, find and solve networks problems, and plan for network growth. SNMP exposes management data in the form of variables on the managed systems, which describe the system configuration. These variables can then be queried and sometimes set.

In our case the SNMP is going to be used to communicate the Click Router and the Dragon software. Next the different characteristics of the SNMP and the decision that we took will be explained.

4.1 SNMP Basic Components

There are three basic components:

4.1.1 Agents

An agent is a network-management software module that resides in a managed device. An agent has local knowledge of management information and translates that information into a form compatible with SNMP

4.1.2 Network-management systems (NMSs)

An NMS executes applications that monitor and control managed devices. It could be more than one NMS in the same managed network.

4.1.3 Managed devices

A managed device is a network node that contains a SNMP agent and that resides on a managed network. Managed devices collect and store management information and make this information available to NMSs using SNMP.

4.2 SNMP Commands

There are three types of commands, to read, to write and the traps. The commands to read are the GET, GETNEXT and GETBULK (WALK). These commands get the information from the SNMP agent. To write we have the command SET and finally the TRAP command is used by the agent to inform to the managing system in some special cases. An example of one SNMP could be this:

```
% snmpset –v 2c –c dragon localhost Q-BRIDGE-MIB::dot1qVlanStaticName s
"IBBT"
Q-BRIDGE-MIB::dot1qVlanStaticName = STRING: IBBT
```

Where the –v means the version, -c the community, after the IP, then the variable that we want to set, after the type of the value and finally the value. The possible types of the SNMP values are showed in the next example:

```
% snmpset –h |& tail –4
   type – one of i, u, t, a, o, s, x, d, n
     i: INTEGER, u: unsigned INTEGER, t: TIMETICKS, a: IPADDRESS
     o: OBJID, s: STRING, x: HEX STRING, d: DECIMAL STRING
     U: unsigned int64, I: signed int64, F: float, D: double
```

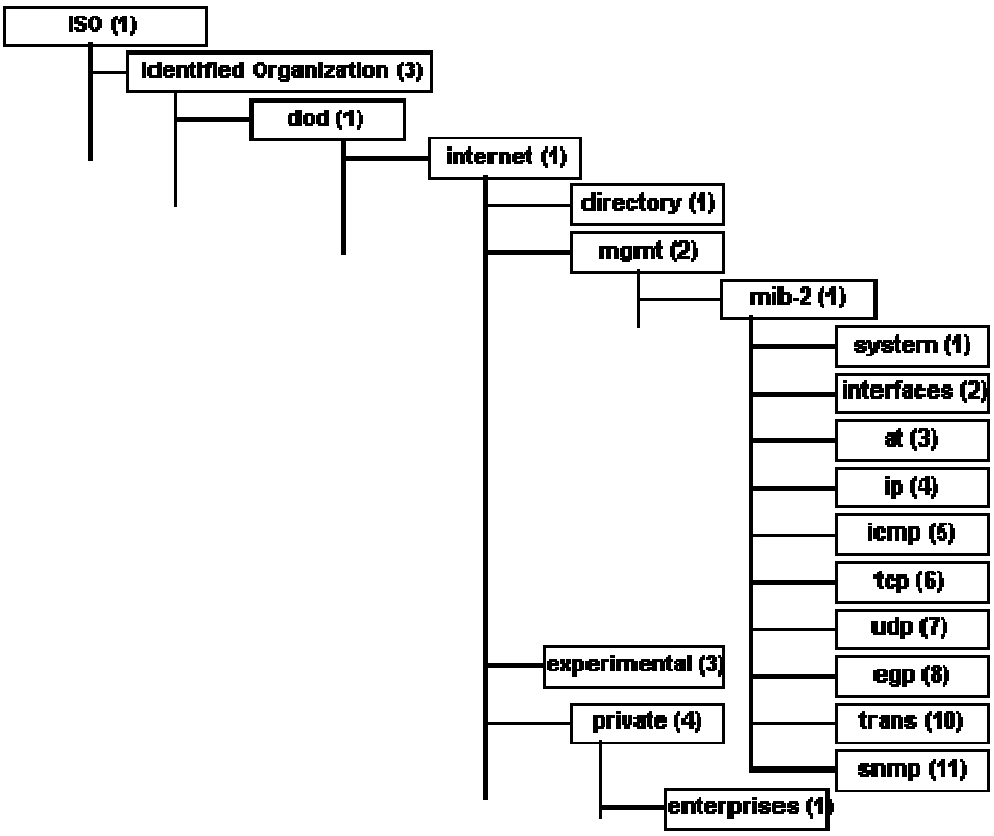
The variables accessible via SNMP are organized in hierarchies. These hierarchies, and other metadata, are described by *Management Information Bases* (MIBs).

4.3 Management Information Bases

The collection of information about the managed device that the agent keeps is called Management Information Base (MIB) and it can be accessed using a network-management protocol such as SNMP.

There are two types of managed objects:

1. Scalar objects define a single object instance.
2. Tabular objects define multiple related object instances that are grouped in MIB tables



c The MIB hierarchy shows the organization of the information

To get these values you can access using the name of the nodes of the tree or via the OID (Object ID) that is the sequence of numbers from the root to the variable. For example to get the information of the table system in the mib-2 we should use the OID 1.3.1.1.2.1.1.

4.4 Different versions

There are three versions of SNMP. Each new version appears for deficiency of their ancestors.

4.4.1 Version 1

SNMP version 1 (SNMPv1) is the initial implementation of the SNMP protocol. Version 1 has been criticized for its poor security. Authentication of clients is performed only by a

"community string", in effect a type of password, which is transmitted in cleartext. This version is the most extended in Internet.

4.4.2 Version 2

Version 2 was not widely adopted due to serious disagreements over the security framework in the standard.

SNMPv2 (RFC 1441–RFC 1452), revises version 1 and includes improvements in the areas of performance, security, confidentiality, and manager-to-manager communications. However, the new party-based security system in SNMP v2, viewed by many as overly complex, was not widely accepted.

Community-Based Simple Network Management Protocol version 2, or *SNMPv2c*, is defined in RFC 1901–RFC 1908. SNMP v2c comprises SNMP v2 *without* the controversial new SNMP v2 security model, using instead the simple community-based security scheme of SNMP v1. *SNMP v2c* is defined in RFC 1909–RFC 1910. This is a compromise that attempts to offer greater security than SNMP v1, but without incurring the high complexity of SNMP v2.

4.4.3 Version 3

The IETF recognizes *Simple Network Management Protocol version 3* as defined by RFC 3411–RFC 3418 as the current standard version of SNMP as of 2004. In practice, SNMP implementations often support multiple versions: typically SNMPv1, SNMPv2c, and SNMPv3. See RFC 3584 "Coexistence between Version 1, Version 2, and Version 3 of the Internet-standard Network Management Framework".

SNMPv3 provides three important services: authentication, privacy and access control.

After study the different characteristics we decided to use NET-SNMP that is a suite of applications used to implement SNMP v1, SNMP v2c and SNMP v3 using both IPv4 and IPv6. The version that we are going to use is the version 2c and the MIBs that we are going to implement will be studied in the Dragon test because it only will be necessary to implement a part of the MIB.

Chapter 5
DRAGON

DRAGON is an abbreviation of Dynamic Resource Allocation via GMPLS Optical Networks. The project DRAGON studies and develops (open source) software to enable dynamic provisioning of network resources on an interdomain basis across heterogeneous network technologies. The project tries to enable the communication between networks of different types through the GMPLS control suite.

5.1 DRAGON Control Plane Components

Software DRAGON is thought to work like control plane within a GMPLS network. The architecture of this control plane has three basic components.

5.1.1 CSA (Client System Agent)

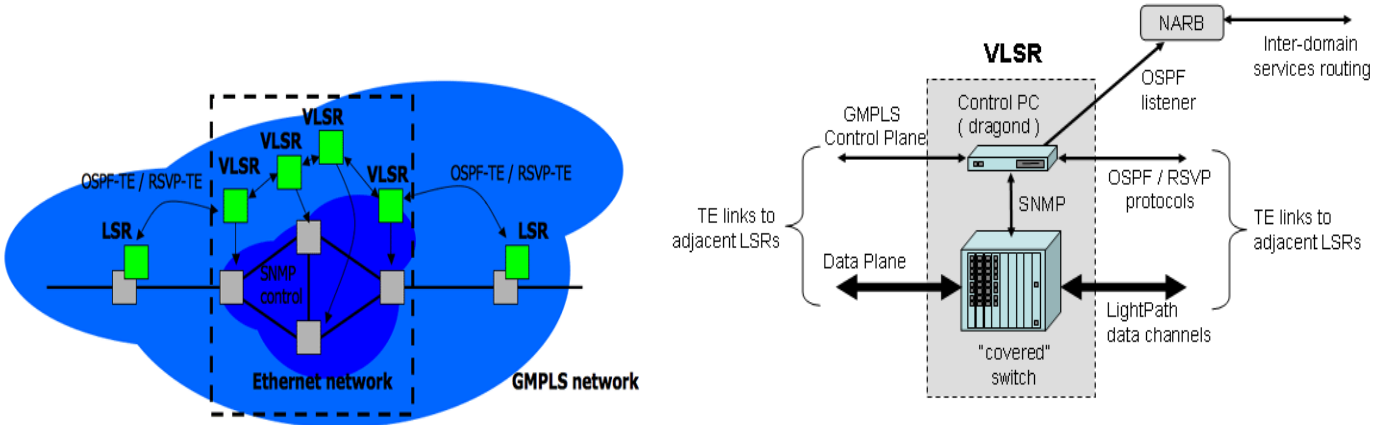
The CSA is software that runs on (or on behalf of) any system which terminates the data plane (traffic engineering) link of the provisioned service. This is the software that participates in the GMPLS protocols to allow for ondemand end-to-end provisioning from client system to client system. A CSA can be a host, a router, or any networked device.

5.1.2 VLSR (Virtual Label Switch Router)

Nowadays GMPLS technology has not extended on great scale because there are still many left elements in the network that do not support this technology. In order to save this difficulty the DRAGON project has developed the VLSR.
The VLSR enables Ethernet switches to act as a label switch in a GMPLS controlled network. This software implements the control part of a GMPLS node, and can connect to Ethernet switches through various interfaces.
VLSR software is prepared to work with different types of switches but we will focus in the Ethernet because it is the part my thesis is about. An important point of my thesis is the communication between DRAGON software and the Click Router due to this, we will pay a

specially attention to it. VLSR PC uses RFC 2674 SNMP to communicate with switch although it can also communicate through a command line (CLI) or other methods.

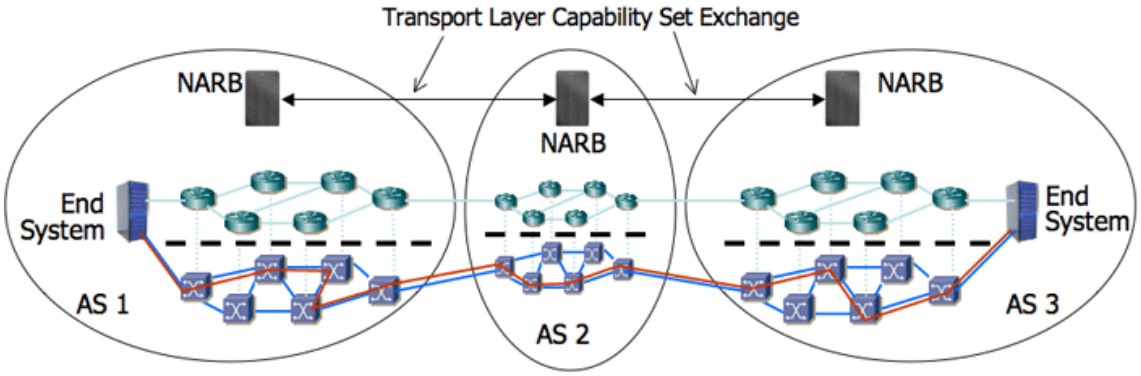
To communicate with other VLSRs and CSAs a VLSR uses the routing protocol OSPF-TE and the path signalling protocol RSVP-TE. These two protocols have been explained in the GMPLS chapter but both have been extended to support the DRAGON necessities.
Ethernet LSPs as being set up by DRAGON VLSRs, use VLAN-tags as a label in forwarding. In contrary to MPLS, at this moment the DRAGON suite only supports a constant VLAN-value per LSP, in other words no VLAN-label swap is performed along the data-path of the LSP.



d Overview of VLSR and overview of Ethernet network within GMPLS network

5.1.3 NARB and RCE (Network Aware Resource Broker and Resource Computation Element)

The NARB is an element of the DRAGON suite that represents an Autonomous System (AS). Its main function is the interdomain path computation and interdomain routing. A common topology would be the one found in the example where the NARB communicates the availability of traffic engineered paths between different Autonomous Systems.



e Overview of the interdomain NARB functions

Also the NARB acts as a protocol listener to the intradomain routing protocols. In our implementation, the intradomain protocol is OSPF-TE. In our experiment we have considered that we are always inside a unique AS and the interdomain functions from the NARB have not been used, however the intradomain functions are necessaries.
A subcomponent of NARB is RCE (Resource Computation Element) that is in charge of path computation task.

The NARB/RCE services are required for VLSR to be signaled with an Explicit Route Object (ERO) when the LSP is (a) a continuous end-to-end VLAN, (b) an inter-domain path, or (c) is created via the VLSR-to-VLSR signaling and/or DRAGON UNI methods.

5.2 DRAGON test case with a commercial Ethernet switch

In order to understand the operation of DRAGON software we have created a very simple infrastructure that will help us to understand its operation. From this test we will try to find out the data structures and SNMP commands that will be necessary to make Click Router capable of interacting with DRAGON software.

5.2.1 Scenario: Basic infrastructure

For our test we have followed the described steps of the VLSR Implementation guide which can be found in the DRAGON web. For this test 3 PCs at least 500MHz with 256 ram and 1GB free hard disk space with two Ethernet cards each were used. As commercial switch to interact with the DRAGON software, we made use of a DES-3526 of DLINK.

5.2.2 Installation and Configuration

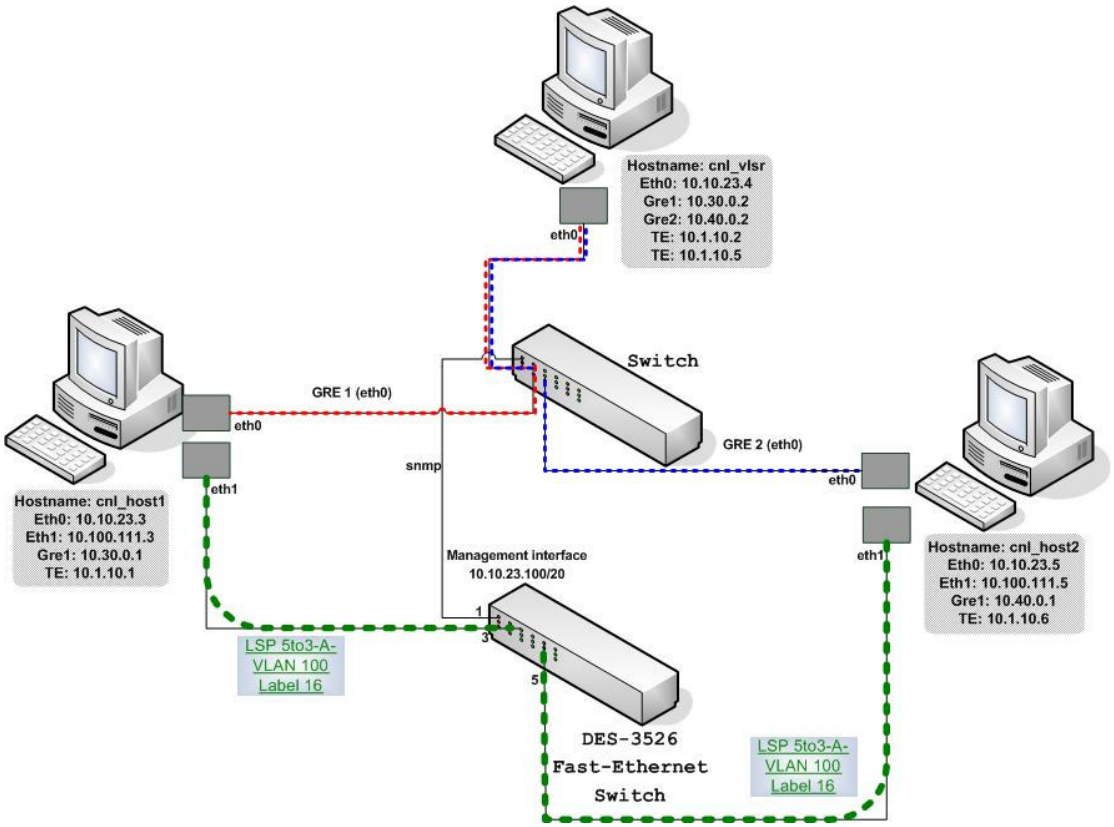
In order to install the DRAGON software the VLSR Implementation Guide has been followed. After installing all the dependent packages[1] it was necessary to configure the OSPF daemon, the RSVP daemon, the ZEBRA daemon and the DRAGON daemon. The configuration of these files is not very difficult but you have to be careful with the consistency of them.

Only one problem that has been found in this step, is that the Dragon package needs to know which type of switch is used to be able to start the SNMP communication. As this was not the case for the DLink switch, the only solution that was found consists of modifying the source code of DRAGON. The SwitchCtrl_Global.cc of the RSVP source files was modified adding a new case, the DLink case, in the possible switches allowed[2].

5.2.3 Testing

To understand how the DRAGON software works it is going to be explained step by step each point of a LSP creation. First of all it will be commented how to create a LSP via DRAGON software and then the internal message of this procedure will be explained.
You can see the scenario for this test in the image below, the configuration files and some logs of this experiment could be found in the CD. The GRE tunnels are used for the control plane, the Ethernet 1 devices are used for the data plane. Before start the test we set up two VLANs (100 and 200) in the Dlink switch.

[1] The following dependencies needed to be fulfilled: Bison, Flex, Net-snmp 5.4, libxml2, zlib1g. In order to compile the dragon package we needed to make a symbolic link from make to gmake.
[2] else if (String("DES-3526 Fast-Ethernet Switch") == venderSystemDescription)
vendor = RFC2674;



f Dragon test with a DLink switch

In order to create the LSP we entered the DRAGON daemon through port 2611 (port by default) and insert the following commands:

1 cnl_host2> edit lsp 5to3-A-

2 cnl_host2(edit-lsp-5to3-A-)# set source ip-address 10.10.23.5 lsp-id 150 destination ip-address 10.10.23.3 tunnel-id 250

3 cnl_host2(edit-lsp-5to3-A-)# set bandwidth gige swcap l2sc encoding ethernet gpid ethernet

4 cnl_host2(edit-lsp-5to3-A-)# exit

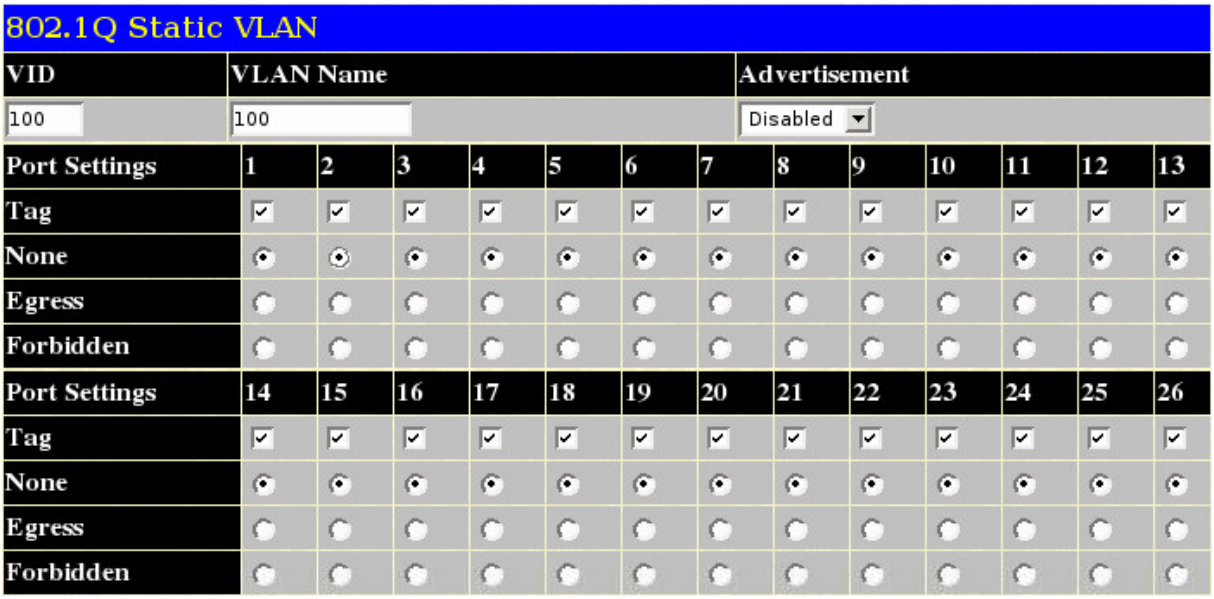
5 cnl_host2> show lsp

LSP status summary

Name	Status	Dir	Source (IP/LSP ID)	Destination (IP/Tunnel ID)
5to3-A-	Edit	=>	10.10.23.5	10.10.23.3
			150	250

The first line corresponds to the name of the LSP that we want to create. The line two is to specify the LSP, to do it is necessary to insert the origin and destination address, the lsp-id, which identifies the lsp for the source host, and the tunnel-id, which identifies the tunnel for the destination host. If the LSP is defined with the lsp-id and the tunnel-id the tunnel will be untagged, the creation of a tagged LSP will be explained in future chapters. The line three specifies the type and bandwidth of the requested LSP, in our case a gige of bandwidth, Layer 2 Switching and Ethernet. In our case we do not define the VLAN of the LSP and the DRAGON software will choose the first VLAN available (100).

Once finished defining the new LSP it is necessary to put it in operation with the commando commit and next the result is showed:



g 802.1Q Static VLAN before the LSP set up in the Dlink switch

cnl_host2> commit lsp 5to3-A-

cnl_host2> show lsp

LSP status summary

Name Status Dir Source (IP/LSP ID) Destination (IP/Tunnel ID)

5to3-A- In service => 10.10.23.5 10.10.23.3

100 200

802.1Q Static VLAN

VID	VLAN Name	Advertisement
100	100	Disabled

Port Settings	1	2	3	4	5	6	7	8	9	10	11	12	13
Tag	☑	☑	☐	☑	☐	☑	☑	☑	☑	☑	☑	☑	☑
None	◉	◉	○	◉	○	◉	◉	◉	◉	◉	◉	◉	◉
Egress	○	○	◉	○	◉	○	○	○	○	○	○	○	○
Forbidden	○	○	○	○	○	○	○	○	○	○	○	○	○

Port Settings	14	15	16	17	18	19	20	21	22	23	24	25	26
Tag	☑	☑	☑	☑	☑	☑	☑	☑	☑	☑	☑	☑	☑
None	◉	◉	◉	◉	◉	◉	◉	◉	◉	◉	◉	◉	◉
Egress	○	○	○	○	○	○	○	○	○	○	○	○	○
Forbidden	○	○	○	○	○	○	○	○	○	○	○	○	○

h 802.1Q Static VLAN after the LSP set up in the Dlink switch

GMPLS signalling

In another test we have created four LSPs to understand how the DRAGON software chooses the labels and the VLANs of the LSPs. Below a picture of the LSPs.

```
cnl_host1-dragon> show lsp
                          **LSP status summary**

Name         Status      Dir    Source (IP/LSP ID)   Destination (IP/Tunnel ID)
-------------------------------------------------------------------------------
3to5-A       In service   =>    10.10.23.3           10.10.23.5
                                 300                  400
3to5-B       In service   =>    10.10.23.3           10.10.23.5
                                 301                  401
5to3-A       In service   =>    10.10.23.5           10.10.23.3
                                 100                  200
5to3-B       In service   =>    10.10.23.5           10.10.23.3
                                 101                  201
```

i Created LSPs in the DRAGON and Dlink test

The order of execution of LSPs have been 5to3-A, 5to3-B, 3to5-A, 3to5-B.
Following the Ethereal dump from the VLSR I am going to make a brief explanation about how the Dragon software chose the labels.

o.	Time	Source	Destination	Protocol	Info
442	17:21:18.51	dia5.gre2	dia4.gre2	RSVP	PATH Message. SESSION: IPv4-LSP, Destination 10.10.23.3, Tunnel ID 200, Ext ID 5170a
482	17:21:18.71	dia4.gre1	dia3.gre1	RSVP	PATH Message. SESSION: IPv4-LSP, Destination 10.10.23.3, Tunnel ID 200, Ext ID 5170a
484	17:21:19.07	dia3.gre1	dia4.gre1	RSVP	RESV Message. SESSION: IPv4-LSP, Destination 10.10.23.3, Tunnel ID 200, Ext ID 5170a
500	17:21:19.23	dia4.gre2	dia5.gre2	RSVP	RESV Message. SESSION: IPv4-LSP, Destination 10.10.23.3, Tunnel ID 200, Ext ID 5170a
502	17:21:20.26	dia5.gre2	dia4.gre2	RSVP	PATH Message. SESSION: IPv4-LSP, Destination 10.10.23.3, Tunnel ID 201, Ext ID 5170a
519	17:21:20.30	dia4.gre1	dia3.gre1	RSVP	PATH Message. SESSION: IPv4-LSP, Destination 10.10.23.3, Tunnel ID 201, Ext ID 5170a
520	17:21:20.32	dia3.gre1	dia4.gre1	RSVP	RESV Message. SESSION: IPv4-LSP, Destination 10.10.23.3, Tunnel ID 201, Ext ID 5170a
543	17:21:20.47	dia4.gre2	dia5.gre2	RSVP	RESV Message. SESSION: IPv4-LSP, Destination 10.10.23.3, Tunnel ID 201, Ext ID 5170a
585	17:21:34.58	dia3.gre1	dia4.gre1	RSVP	PATH Message. SESSION: IPv4-LSP, Destination 10.10.23.5, Tunnel ID 400, Ext ID 3170a
602	17:21:34.61	dia4.gre2	dia5.gre2	RSVP	PATH Message. SESSION: IPv4-LSP, Destination 10.10.23.5, Tunnel ID 400, Ext ID 3170a
603	17:21:34.62	dia5.gre2	dia4.gre2	RSVP	RESV Message. SESSION: IPv4-LSP, Destination 10.10.23.5, Tunnel ID 400, Ext ID 3170a
626	17:21:34.74	dia4.gre1	dia3.gre1	RSVP	RESV Message. SESSION: IPv4-LSP, Destination 10.10.23.5, Tunnel ID 400, Ext ID 3170a
636	17:21:41.02	dia3.gre1	dia4.gre1	RSVP	PATH Message. SESSION: IPv4-LSP, Destination 10.10.23.5, Tunnel ID 401, Ext ID 3170a
653	17:21:41.07	dia4.gre2	dia5.gre2	RSVP	PATH Message. SESSION: IPv4-LSP, Destination 10.10.23.5, Tunnel ID 401, Ext ID 3170a
654	17:21:41.07	dia5.gre2	dia4.gre2	RSVP	RESV Message. SESSION: IPv4-LSP, Destination 10.10.23.5, Tunnel ID 401, Ext ID 3170a
677	17:21:41.20	dia4.gre1	dia3.gre1	RSVP	RESV Message. SESSION: IPv4-LSP, Destination 10.10.23.5, Tunnel ID 401, Ext ID 3170a
679	17:21:41.85	dia4.gre1	dia3.gre1	RSVP	PATH Message. SESSION: IPv4-LSP, Destination 10.10.23.3, Tunnel ID 200, Ext ID 5170a

j RSVP log of the DRAGON and Dlink test

The first LSP that I wanted to set up was the 5to3-A. In the first line we can see that dia5 sends a Path Message to the dia4 (VLSR) through the gre2. And in the second line the VLSR forwards the packet to dia3. Dia3 replies this request with a RESV Message that contains the label, in this case label 16, to the VLSR. The first labels (0-15) are reserved, if you want to know more about this you can find it in RFC 3032. Then, in the line 4, the VLSR (dia4) forwards the RESV Message to the dia5 with the label 16. In the following four lines the same procedure is repeated with the 5to3-B and the labels 17.

Therefore it is possible to say that usually the labels are different along a LSP. The value of the label is defined in chronological order of the input connections that the component have. In our case the VLSR receives 4 input connections and the VLSR assigns each one a label greater than 15. Other hand the value of the chosen VLAN is independent of the label and if it is not defined a VLAN in the creation of the LSP the DRAGON software takes the first one available.

VLSR-SNMP interface

In this section the relation between RSVP messages and SNMP commands from the VLSR to the switch is going to be explained. Two VLANs (100 and 200) have been created manually through the web-interface of the switch.

The MIB-structures that will be explained here will be implemented later in the Click Router. In one hand the SNMP MIB tables used and in the other hand the switch table and the correspondence between both.

The MIB-elements needed by SNMP in the set up of a LSP are shown in the next table:

1.3.6.1.2.1	**.1.1**	**.0**			sysDescr	String
	.17.7.1.4 *dot1qVlan*	**.3.1** *dot1qVlanStatic*	**.1**	**.x** x = *vlanId*	dot1qVlanStaticName	String
			.2		dot1qVlanStaticEgressPorts	PortList (1)
			.3		dot1qVlanForbiddenEgressPorts	
			.4		dot1qVlanStaticUntaggedPorts	
			.5		dot1qVlanStaticRowStatus	RowStatus(2)
		.5.1 *dot1qPortVlanTable*	**.1**		dot1qPvid	VlanIndex = value of assigned VLAN

1. "Each octet within this value specifies a set of eight ports, with the first octet specifying ports 1 through 8, the second octet specifying ports 9 through 16, etc. Within each octet, the most significant bit represents the lowest numbered port, and the least significant bit represents the highest numbered port. Thus, each port of the bridge is represented by a single bit within the value of this object. If that bit has a value of '1', then that port is included in the set of ports; the port is not included if its bit has a value of '0'." From http://tools.cisco.com/Support/SNMP/do/BrowseOID.do?local=en&translate=Translate&typeName=PortList

2. **1**:active
   2:notInService
   3:notReady
   4:createAndGo
   5:createAndWait
   6:destroy

k Table of the SNMP MIB elements necessary by the DRAGON software

Next a brief description of each field to understand the goal of each table.

sysDescr: String of the description of the Switch. This field is used to create the RSVP session that it depends of the type of the Switch.

dot1qVlanStaticName: String of the name of the VLAN. This field is not very important because is not used in the LSP set up.

dot1qVlanStaticEgressPorts: This table keeps for each VLAN the set of ports that are permanently assigned to the egress list.

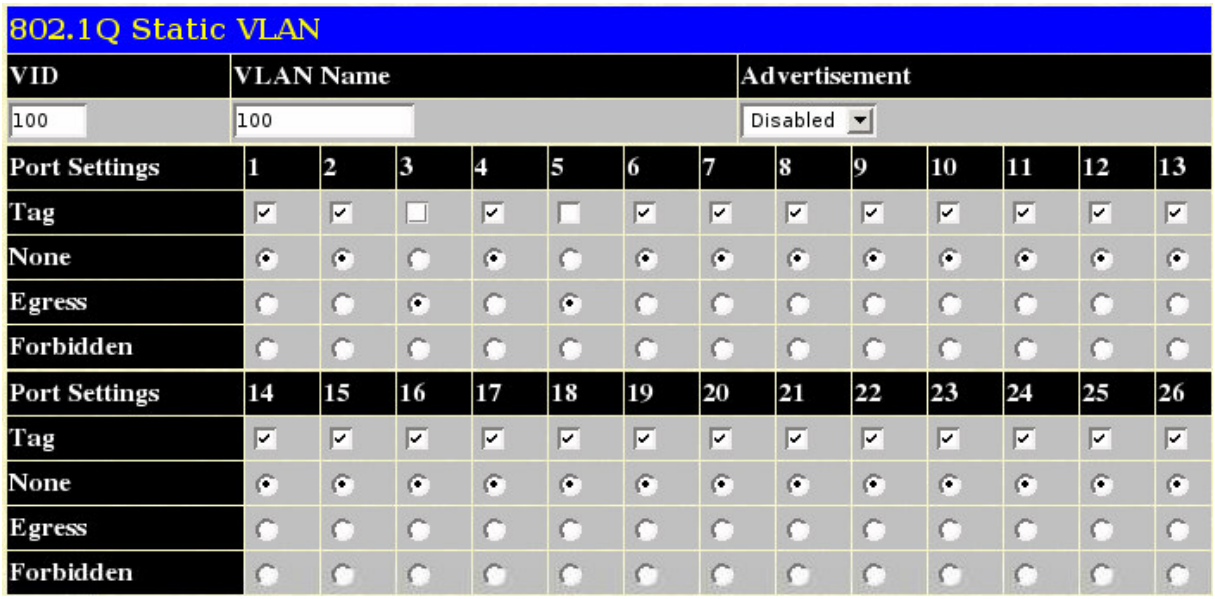
dot1qVlanForbiddenEgressPorts: This table keeps for each VLAN the set of ports that are not allowed to belong to this VLAN.

dot1qVlanStaticUntaggedPorts: This table keeps for each VLAN the set of ports that should transmit egress packets untagged “1” or tagged “0”. To tag or untag a port means that each egress packet from this port will be tagged or untagged with the VLAN label.

dot1qVlanStaticRowStatus: This object indicates the status of this VLAN. The possible values are explained in the table.

dot1qPvid: The PVID, the VLAN-ID assigned to untagged frames or Priority-Tagged frames received on this port.

In our test switch we have for each VLAN the next table:



l Structure of the Static VLAN tables in the Dlink switch

The content of the both tables is the same but their structure differs. In the MIB-table we first have to choose the characteristic of the port (egress, forbidden, untagged), and then the VLAN a hex value (configuration of all the ports). In the Switch table the column represents the VLAN and has different rows (one row for each type of port) and in each row different checkbox to choose if the port belong or not to the corresponding VLAN.

The Tag row refers to the dot1qVlanStaticUntaggedPorts (checked = tagged = default value). The Forbidden row refers to the dot1qVlanForbiddenEgressPorst. The None and Egress rows are complementary, a port can not be marked in both tables, and refer both to dot1qVlanStaticEgressPorts.

Other hand the structure of the dot1qPvid table and the Pvid table of the switch is exactly the same.

PATH Message

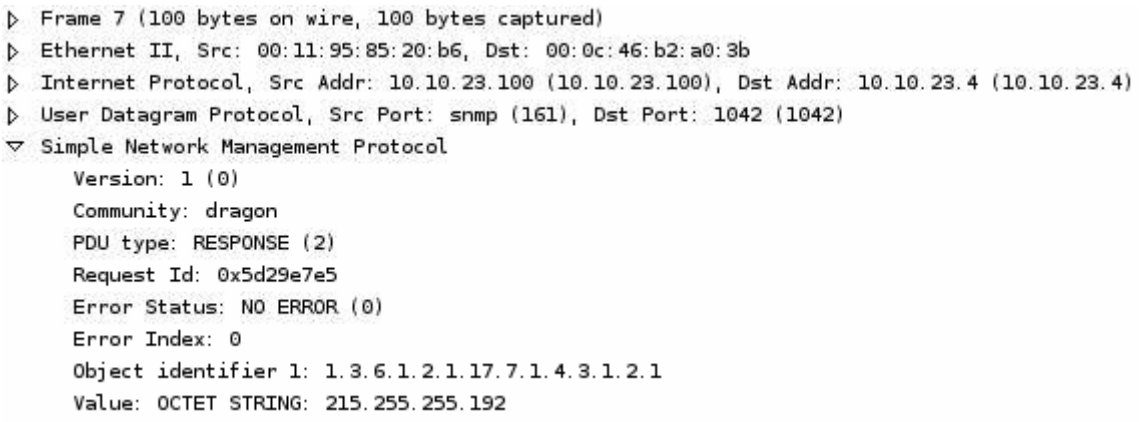
Below is a complete log of the SNMP-commands that the VLSR initiates when receives an RSVP PATH Message. A brief explanation of each SNMP call is given.

No.	Time	Source	Destination	Protocol	Info
1	17:21:18.510462	10.40.0.1	10.40.0.2	RSVP	PATH Message. SESSION: IPv4-LSP, Destination 10.10.23.3,
2	17:21:18.643847	10.10.23.4	10.10.23.100	SNMP	GET SNMPv2-MIB::sysDescr.0
3	17:21:18.646882	10.10.23.100	10.10.23.4	SNMP	RESPONSE SNMPv2-MIB::sysDescr.0
4	17:21:18.647638	10.10.23.4	10.10.23.100	SNMP	GET SNMPv2-MIB::sysDescr.0
5	17:21:18.650386	10.10.23.100	10.10.23.4	SNMP	RESPONSE SNMPv2-MIB::sysDescr.0
6	17:21:18.650670	10.10.23.4	10.10.23.100	SNMP	GET-NEXT SNMPv2-SMI::mib-2.17.7.1.4.3.1.2
7	17:21:18.654326	10.10.23.100	10.10.23.4	SNMP	RESPONSE SNMPv2-SMI::mib-2.17.7.1.4.3.1.2.1
8	17:21:18.654523	10.10.23.4	10.10.23.100	SNMP	GET-NEXT SNMPv2-SMI::mib-2.17.7.1.4.3.1.2.1
9	17:21:18.658442	10.10.23.100	10.10.23.4	SNMP	RESPONSE SNMPv2-SMI::mib-2.17.7.1.4.3.1.2.100
10	17:21:18.658688	10.10.23.4	10.10.23.100	SNMP	GET-NEXT SNMPv2-SMI::mib-2.17.7.1.4.3.1.2.100
11	17:21:18.662397	10.10.23.100	10.10.23.4	SNMP	RESPONSE SNMPv2-SMI::mib-2.17.7.1.4.3.1.2.200
12	17:21:18.662563	10.10.23.4	10.10.23.100	SNMP	GET-NEXT SNMPv2-SMI::mib-2.17.7.1.4.3.1.2.200
13	17:21:18.667740	10.10.23.100	10.10.23.4	SNMP	RESPONSE SNMPv2-SMI::mib-2.17.7.1.4.3.1.3.1
14	17:21:18.667910	10.10.23.4	10.10.23.100	SNMP	GET-NEXT SNMPv2-SMI::mib-2.17.7.1.4.3.1.4
15	17:21:18.671698	10.10.23.100	10.10.23.4	SNMP	RESPONSE SNMPv2-SMI::mib-2.17.7.1.4.3.1.4.1
16	17:21:18.671851	10.10.23.4	10.10.23.100	SNMP	GET-NEXT SNMPv2-SMI::mib-2.17.7.1.4.3.1.4.1
17	17:21:18.676465	10.10.23.100	10.10.23.4	SNMP	RESPONSE SNMPv2-SMI::mib-2.17.7.1.4.3.1.4.100
18	17:21:18.676696	10.10.23.4	10.10.23.100	SNMP	GET-NEXT SNMPv2-SMI::mib-2.17.7.1.4.3.1.4.100
19	17:21:18.680427	10.10.23.100	10.10.23.4	SNMP	RESPONSE SNMPv2-SMI::mib-2.17.7.1.4.3.1.4.200
20	17:21:18.680663	10.10.23.4	10.10.23.100	SNMP	GET-NEXT SNMPv2-SMI::mib-2.17.7.1.4.3.1.4.200
21	17:21:18.684600	10.10.23.100	10.10.23.4	SNMP	RESPONSE SNMPv2-SMI::mib-2.17.7.1.4.3.1.5.1
22	17:21:18.684780	10.10.23.4	10.10.23.100	SNMP	GET-NEXT SNMPv2-SMI::mib-2.17.7.1.4.3.1.2
23	17:21:18.688759	10.10.23.100	10.10.23.4	SNMP	RESPONSE SNMPv2-SMI::mib-2.17.7.1.4.3.1.2.1
24	17:21:18.688961	10.10.23.4	10.10.23.100	SNMP	GET-NEXT SNMPv2-SMI::mib-2.17.7.1.4.3.1.2.1
25	17:21:18.692877	10.10.23.100	10.10.23.4	SNMP	RESPONSE SNMPv2-SMI::mib-2.17.7.1.4.3.1.2.100
26	17:21:18.693037	10.10.23.4	10.10.23.100	SNMP	GET-NEXT SNMPv2-SMI::mib-2.17.7.1.4.3.1.2.100
27	17:21:18.696871	10.10.23.100	10.10.23.4	SNMP	RESPONSE SNMPv2-SMI::mib-2.17.7.1.4.3.1.2.200
28	17:21:18.697108	10.10.23.4	10.10.23.100	SNMP	GET-NEXT SNMPv2-SMI::mib-2.17.7.1.4.3.1.2.200
29	17:21:18.700975	10.10.23.100	10.10.23.4	SNMP	RESPONSE SNMPv2-SMI::mib-2.17.7.1.4.3.1.3.1
30	17:21:18.701149	10.10.23.4	10.10.23.100	SNMP	GET-NEXT SNMPv2-SMI::mib-2.17.7.1.4.3.1.4
31	17:21:18.704962	10.10.23.100	10.10.23.4	SNMP	RESPONSE SNMPv2-SMI::mib-2.17.7.1.4.3.1.4.1
32	17:21:18.705181	10.10.23.4	10.10.23.100	SNMP	GET-NEXT SNMPv2-SMI::mib-2.17.7.1.4.3.1.4.1
33	17:21:18.708919	10.10.23.100	10.10.23.4	SNMP	RESPONSE SNMPv2-SMI::mib-2.17.7.1.4.3.1.4.100
34	17:21:18.709073	10.10.23.4	10.10.23.100	SNMP	GET-NEXT SNMPv2-SMI::mib-2.17.7.1.4.3.1.4.100
35	17:21:18.712879	10.10.23.100	10.10.23.4	SNMP	RESPONSE SNMPv2-SMI::mib-2.17.7.1.4.3.1.4.200
36	17:21:18.713033	10.10.23.4	10.10.23.100	SNMP	GET-NEXT SNMPv2-SMI::mib-2.17.7.1.4.3.1.4.200
37	17:21:18.717102	10.10.23.100	10.10.23.4	SNMP	RESPONSE SNMPv2-SMI::mib-2.17.7.1.4.3.1.5.1
38	17:21:18.719131	10.30.0.2	10.30.0.1	RSVP	PATH Message. SESSION: IPv4-LSP, Destination 10.10.23.3,

m RSVP Path Message in the DRAGON and Dlink switch test

- If there is not yet a RSVP session created, the VLSR asks for the switch description. These messages correspond to packet number 2 and therefore have to access the MIB SNMPv2-MIB::sysDescr.0.
- Once the session is created, the VLSR starts to read the dot1qVlanStaticEgressPorts table for every available VLAN. The VLSR uses the SNMP command GET-NEXT to access every field of the table. In our case it has information in the VLAN 1, 100, 200, the default VLAN and the VLANs that have been created. The GET-NEXT command returns the next field that exists in our MIB below the OID received. The VLSR always starts with the OID = …17.7.1.4.3.1.2 which is the dot1qVlanStaticEgressPorts table.

The answer of these types of message is the value of the ports that belong to that VLAN. For example for the default VLAN (1) the ports that belong to this VLAN are shown in the next response (packet 7):



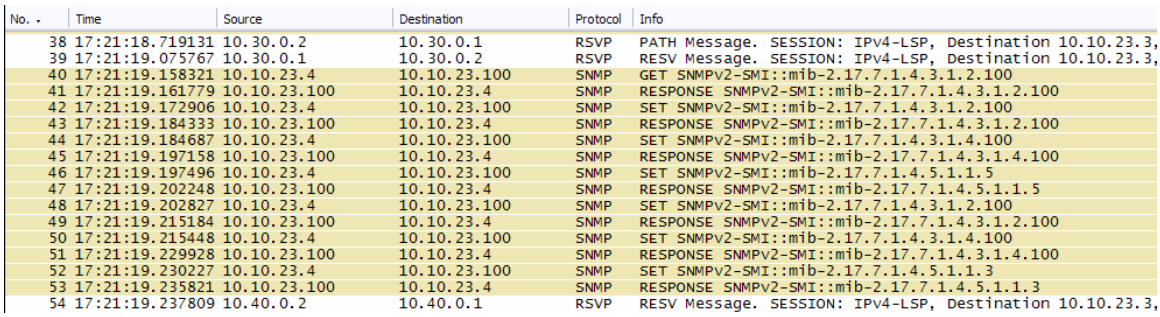
n SNMP response example

The value means that the ports 3 and 5 do not belong to this VLAN. Each byte-segment refers to 8 ports. The first segment, 215, in binary notation would be 11010111, where the positions of the 0s correspond to the port 3 and 5. The value of the two next segments means that all ports belong to the default VLAN. In the last segment the value is 192, the reason of this value is that with this method the switch can control up to 32 ports and in our case our switch has 26 ports. This 6 ports of different are the 0s of the last segment 1 1 0 0 0 0 0 0 .

- When the VLSR receives the RESPONSE with the OID = …17.7.1.4.3.1.3.1 (this OID correspond to the forbidden ports of the default VLAN, message 13) means that the end of dot1qVlanStaticEgressPorts has been reached. The next GET-NEXT message jumps the dot1qVlanForbiddenEgressPorts table and reads the dot1qVlanStaticUntaggedPorts table that it starts with the message 14 with the OID = …17.7.1.4.3.1.4. The structure of the scan of this table is similar than the other one and ends when the VLSR receives the RESPONSE with the OID = …17.7.1.4.3.1.5.1 (this OID correspond to the row status of the default VLAN, message 21).

RESV Message

Receiving a RESV message will generate the following:



No.	Time	Source	Destination	Protocol	Info
38	17:21:18.719131	10.30.0.2	10.30.0.1	RSVP	PATH Message. SESSION: IPv4-LSP, Destination 10.10.23.3,
39	17:21:19.075767	10.30.0.1	10.30.0.2	RSVP	RESV Message. SESSION: IPv4-LSP, Destination 10.10.23.3,
40	17:21:19.158321	10.10.23.4	10.10.23.100	SNMP	GET SNMPv2-SMI::mib-2.17.7.1.4.3.1.2.100
41	17:21:19.161779	10.10.23.100	10.10.23.4	SNMP	RESPONSE SNMPv2-SMI::mib-2.17.7.1.4.3.1.2.100
42	17:21:19.172906	10.10.23.4	10.10.23.100	SNMP	SET SNMPv2-SMI::mib-2.17.7.1.4.3.1.2.100
43	17:21:19.184333	10.10.23.100	10.10.23.4	SNMP	RESPONSE SNMPv2-SMI::mib-2.17.7.1.4.3.1.2.100
44	17:21:19.184687	10.10.23.4	10.10.23.100	SNMP	SET SNMPv2-SMI::mib-2.17.7.1.4.3.1.4.100
45	17:21:19.197158	10.10.23.100	10.10.23.4	SNMP	RESPONSE SNMPv2-SMI::mib-2.17.7.1.4.3.1.4.100
46	17:21:19.197496	10.10.23.4	10.10.23.100	SNMP	SET SNMPv2-SMI::mib-2.17.7.1.4.5.1.1.5
47	17:21:19.202248	10.10.23.100	10.10.23.4	SNMP	RESPONSE SNMPv2-SMI::mib-2.17.7.1.4.5.1.1.5
48	17:21:19.202827	10.10.23.4	10.10.23.100	SNMP	SET SNMPv2-SMI::mib-2.17.7.1.4.3.1.2.100
49	17:21:19.215184	10.10.23.100	10.10.23.4	SNMP	RESPONSE SNMPv2-SMI::mib-2.17.7.1.4.3.1.2.100
50	17:21:19.215448	10.10.23.4	10.10.23.100	SNMP	SET SNMPv2-SMI::mib-2.17.7.1.4.3.1.4.100
51	17:21:19.229928	10.10.23.100	10.10.23.4	SNMP	RESPONSE SNMPv2-SMI::mib-2.17.7.1.4.3.1.4.100
52	17:21:19.230227	10.10.23.4	10.10.23.100	SNMP	SET SNMPv2-SMI::mib-2.17.7.1.4.5.1.1.3
53	17:21:19.235821	10.10.23.100	10.10.23.4	SNMP	RESPONSE SNMPv2-SMI::mib-2.17.7.1.4.5.1.1.3
54	17:21:19.237809	10.40.0.2	10.40.0.1	RSVP	RESV Message. SESSION: IPv4-LSP, Destination 10.10.23.3,

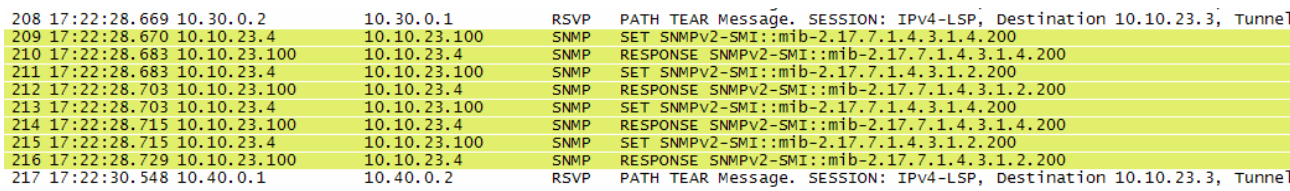
o RSVP Resv Message in the DRAGON and Dlink switch test

Following the log, after receiving and forwarding the PATH message the VLSR receives the reply. In this point the VLSR is configured to set new values in the dot1qVlanStaticEgressPorts and dot1qVlanStaticUntaggedPorts of the first VLAN that is not used.

- The first mentioned call that the VLSR makes when receive a RESV message is to set a new value in the dot1qVlanStaticEgressPorts table (OID = …17.7.1.4.3.1.2) of the first VLAN that is available, in this case the VLAN 100. This call corresponds to the packet number 42 and it puts the port 5 in to this VLAN.
- Then the VLSR sets the same value in the dot1qVlanStaticUntaggedPorts table (OID = …17.7.1.4.3.1.4). This call corresponds to the packet number 44.
- Finally the VLSR adds the VLAN to the port 5 in the dot1qPvid table (OID = …17.7.1.4.5.1.1). This call corresponds to the packet number 46.
- The same procedure is carried out for the other port of the LSP.

PATH TEAR Message

The other message that the VLSR could receive is the PATH TEAR message, intended for the destruction of the LSP. In this case the VLSR follow the same steps that in the RESV message but in the reverse way.



208	17:22:28.669	10.30.0.2	10.30.0.1	RSVP	PATH TEAR Message. SESSION: IPv4-LSP, Destination 10.10.23.3, Tunnel
209	17:22:28.670	10.10.23.4	10.10.23.100	SNMP	SET SNMPv2-SMI::mib-2.17.7.1.4.3.1.4.200
210	17:22:28.683	10.10.23.100	10.10.23.4	SNMP	RESPONSE SNMPv2-SMI::mib-2.17.7.1.4.3.1.4.200
211	17:22:28.683	10.10.23.4	10.10.23.100	SNMP	SET SNMPv2-SMI::mib-2.17.7.1.4.3.1.2.200
212	17:22:28.703	10.10.23.100	10.10.23.4	SNMP	RESPONSE SNMPv2-SMI::mib-2.17.7.1.4.3.1.2.200
213	17:22:28.703	10.10.23.4	10.10.23.100	SNMP	SET SNMPv2-SMI::mib-2.17.7.1.4.3.1.4.200
214	17:22:28.715	10.10.23.100	10.10.23.4	SNMP	RESPONSE SNMPv2-SMI::mib-2.17.7.1.4.3.1.4.200
215	17:22:28.715	10.10.23.4	10.10.23.100	SNMP	SET SNMPv2-SMI::mib-2.17.7.1.4.3.1.2.200
216	17:22:28.729	10.10.23.100	10.10.23.4	SNMP	RESPONSE SNMPv2-SMI::mib-2.17.7.1.4.3.1.2.200
217	17:22:30.548	10.40.0.1	10.40.0.2	RSVP	PATH TEAR Message. SESSION: IPv4-LSP, Destination 10.10.23.3, Tunnel

p RSVP Path Tear Message in the DRAGON and Dlink switch test

- The first call that the VLSR makes when receive a TEAR message is to set a new value in the dot1qVlanStaticUntaggedPorts table (OID = …17.7.1.4.3.1.4) of the VLAN of the LSP, in this case the VLAN 200. This call corresponds to the packet number 209 and the value of this packet means that the port 5 does not belong to this VLAN.
- Then the VLSR sets the same value in the dot1qVlanStaticEgressPorts table (OID = …17.7.1.4.3.1.2). This call corresponds to the packet number 211.
- The same procedure is done for the other port of the LSP.

Creation of a new VLANs

In this test the DLink works with a RFC2674 RSVP session. This type of sessions does not make use of the creation of VLANs. But there are other types of sessions where the creation of VLANs is needed like for example the Raptor sessions. In the next example of a Raptor session the VLAN 2 is created. Below, it is showed the procedure of a RESV message when in the PATH message the DRAGON software only finds the default VLAN.

227	55.529803	10.20.0.1	10.20.0.2	RSVP	RESV Message. SESSION: IPv4-LSP, Destination A.B.C.41, Tunnel ID 2000, Ext ID 277ca9c3. FILTERSPEC: IPv4-LSP, Tunnel Source: A.B.C.39, LSP ID: 1000.
228	55.558129	A.B.C.40	A.B.C.54	SNMP	GET SNMPv2-SMI::mib-2.17.7.1.4.3.1.2.2
229	55.559048	A.B.C.54	A.B.C.40	SNMP	RESPONSE SNMPv2-SMI::mib-2.17.7.1.4.3.1.2.2
230	55.559276	A.B.C.40	A.B.C.54	SNMP	SET SNMPv2-SMI::mib-2.17.7.1.4.3.1.5.2
231	55.578444	A.B.C.54	A.B.C.40	SNMP	RESPONSE SNMPv2-SMI::mib-2.17.7.1.4.3.1.5.2
232	55.578603	A.B.C.40	A.B.C.54	SNMP	SET SNMPv2-SMI::mib-2.17.7.1.4.3.1.2.2
233	55.593738	A.B.C.54	A.B.C.40	SNMP	RESPONSE SNMPv2-SMI::mib-2.17.7.1.4.3.1.2.2
234	55.593817	A.B.C.40	A.B.C.54	SNMP	SET SNMPv2-SMI::mib-2.17.7.1.4.5.1.1.9
235	55.597948	A.B.C.54	A.B.C.40	SNMP	RESPONSE SNMPv2-SMI::mib-2.17.7.1.4.5.1.1.9
236	55.598077	A.B.C.40	A.B.C.54	SNMP	SET SNMPv2-SMI::mib-2.17.7.1.4.3.1.2.2
237	55.603808	A.B.C.54	A.B.C.40	SNMP	RESPONSE SNMPv2-SMI::mib-2.17.7.1.4.3.1.2.2
238	55.603876	A.B.C.40	A.B.C.54	SNMP	SET SNMPv2-SMI::mib-2.17.7.1.4.5.1.1.1
239	55.607250	A.B.C.54	A.B.C.40	SNMP	RESPONSE SNMPv2-SMI::mib-2.17.7.1.4.5.1.1.1

q Creation of a new VLAN in a example from the SARA project

- The first call that the VLSR makes is a GET in the dot1qVlanStaticEgressPorts table (OID = …17.7.1.4.3.1.2) of the next VLAN after the default VLAN (packet number 228).
- The switch answers with an error because this VLAN does not exist.
- Then the VLSR puts the value 4 in the dot1qVlanStaticRowStatus (OID = …17.7.1.4.3.1.5) for this VLAN. The value 4 means the state "CreateAndGo". This state means that if there is not any inconsistency with the rest of VLANs available (for example same name for two different VLANs) this VLAN will pass to state 1 that means "Active".

Chapter 6
Implementation of SNMP in Click Router

Once it is known how DRAGON works, the next step consists of adding the necessary elements in the Click Router elements to make it capable of working with DRAGON. To achieve this purpose I had the help of a pre-how-to done by Didier Colle. This pre-how-to describes the first steps and here this first steps and the new ones are going to be explained.

6.1 Click Router and SNMP environment

The first step was to install the last version of the Click Router[3] and the last version of the NET-SNMP[4]. Also to make use of the linux kernel module I install the *linux-2.6.16.13-click1.5.0* version. Furthermore it was necessary to add the new elements to support the use of VLANs and other features that you can find in the clicksrc directory of the CD. To compile these new elements you should provide a link[5] between the Click local elements and our clicksrc directory. Also it was necessary to add the file *ether.h*[6] in the *clicknet* directory of the CLICK-SRC and to use this *configure*[7] command to compile the Click Router to use later all the possibilities of the new features.

Finally to feed the SNMP-agent with the descriptions for which you want to develop modules we had to copy the necessary MIB description files in the */usr/ local/share/snmp/mibs/* . In our case only the Q-BRIDGE-MIB.txt[8] was necessary because DRAGON uses this MIB.

[3] Click Router version 1.5.0
[4] Net-SNMP version 5.4
[5] In *CLICK-SRC/elements/local/: ln –s ~/clicksrc/ clickdev* (if you have the clicksrc in the home directory)
[6] You can find this file in *clicksrc/ether.h*
[7] *./configure --enable-local --enable-userlevel --enable-analysis –with-linux=/usr/src/linux-2.6.16.13-click1.5.0*
[8] *cp ~/clicksrc/QSwitch/MIBS/Q-BRIDGE-MIB.txt /usr/local/share/snmp/mibs/*

6.2 Generate template code for the MIB-modules

In order to generate the template code for the *dot1qVlanStaticTable* we make use of the *mib2c* tool that has been installed together with the Net-SNMP framework. Firstly it is necessary to create a new directory[9] and later to call the *mib2c*[10] from there. The location of the directory is important because the code generation generates some includes that refer to the other source files.

Regarding the questions that you will receive when you execute the *mib2c* you should answer always the default options except the 9th question where you have to answer the option "container-cached".

6.3 Compiling and building the new Click environment

With the new generated code we should re-compile the Click environment. To make it possible it is necessary to modify the *CLICK-SRC/userlevel/Makefile.in*. The lines that you should modify are :

```
CXXFLAGS = @CXXFLAGS@ `net-snmp-config --cflags`
LDFLAGS = @LDFLAGS@ -static
LIBS = @LIBS@ `net-snmp-config --agent-libs` `$(top_builddir)/click-compile --otherlibs`
$(ELEMENT_LIBS)
```

After you recompile the Click environment you should be able to use the new created element. Now you could use the new element in a new Click script like this:

```
theSwitch :: QEtherSwitch(512);
agentx :: SnmpSubagentElement();
vlantable :: dot1qVlanStaticTable_element(agentx, theSwitch);
```

[9] *mkdir ~/clicksrc/QSwitch/dot1qVlanStaticTable/*
[10] *mib2c -c ~/clicksrc/mib2c/mib2c.mfd.conf -I ~/clicksrc/mib2c/mib2c-data/ Q-BRIDGE-MIB::dot1qVlanStaticTable*

6.4 Preparing, executing and testing the Net-SNMP daemon

The new click-element SnmpSubagentElement is an AgentX subagent that needs to communicate with the standard Net-SNMP daemon. In order to make this possible you should add to the snmpd.conf the line:

```
master agentx
```

You should find the complete snmpd.conf in the appendix A. Furthermore, you should change the permission of the */var/agentx/master*[11] and the */var/net-snmp/* because the Net-SNMP daemon will reset the permissions when you execute the daemon.

To test that everything works properly you should execute the previously commented script and try to ask to the new element. For example with the next snmp call:

```
snmpwalk -v 2c -c rwcomm localhost Q-BRIDGE-MIB::dot1qVlanStaticTable
```

The answer should return some results but without sense. Now is when we have to modify the template code to return useful information.

6.5 Modifying the template code

In this chapter we are not going to explain each modification done in the template code because this would be very boring. We will try to explain only the most important points to make it more pleasant but also comprehensible.

Next we are going to describe the files modified and a brief description of the modification done in that file.

6.5.1 File: dot1qVlanStaticTable.h

In this file is defined the external click elements that are used in this element. First you have to make the includes of the external elements and after the call.

[11] *chmod a=xwr /var/agentx/master*

6.5.2 File: dot1qVlanStaticTable_data_access.cc

This file is in charge of creating the MIB structure. First you have to make a iteration that creates the indexes that in the dot1qVlanStaticTable are the VLANs and after it fill up the different fields of each VLAN. In this case the dot1qVlanStaticName, the dot1qVlanStaticEgressPorts, the dot1qVlanForbiddenEgressPorts, the dot1qVlanStaticUntaggedPorts and the dot1qVlanStaticRowStatus.

6.5.3 File: dot1qVlanStaticTable_data_set.cc

In this file you have to modify the function `dot1qVlanStaticTable_commit` commenting each `rc=-1`. Commenting this line you will allow the modification of the different fields of the tables. Then you can modify each `dot1qVlanStaticXXXXXX_set` adding the necessary code to access to the other Click elements. You can not allow all the sets. The template code takes care of the consistence of the incoming values. And also you can choose to do another kind of checks before a set. You can make the checks in this file or in the Click element that correspond to that field. In our case I did in the *qetherswitch.cc* and in the *qtransout.cc.*

6.5.4 File: dot1qVlanStaticTable_element.cc

This is the file where the *dot1qVlanStaticTable_element* is defined. Here you have to add the necessary parameters that you will need to create the element. In this case a *qtransout* click element and a *qeterswitch* click element.

Furthermore all the commented modifications you also have to add to every cc file this line[12] at the end of the file. The value of this line depends on the click elements that are needed in the creation of the *dot1qVlanStaticTable_element*.

[12] `ELEMENT_REQUIRES(userlevel QEtherSwitch QTransOut dot1qVlanStaticTable dot1qVlanStaticTable_interface)`

6.5.5 dot1qPvidTable

In order to make the Click Router capable of working with the DRAGON software we also need to create the *dot1qPvidTable*. The procedure for this table is similar as the dot1qVlanStaticTable. The main difference is that in the *dot1qVlanStaticTable* is used the click elements *qtransout* and *qetherswitch* and for the *dot1qPvidTable* is used the click element *qtransin*. Another difference is the dot1qPvidTable belongs to the *dot1qPortVlanTable* but we only need the *dot1qPvidTable*. To solve this problem we create in the *dot1qPortVlanTable_data_access.cc* only the *dot1qPvidTable* field and comment the rest of the fields.

6.6 Modifying qtransin, qtransout and qetherswitch

In each of those elements sets, gets, checks and another functions have been made. These functions are more or less simple, the only problematic thing that we had was the conversion of the data between these elements and the SNMP tables. In these elements we work with integers and in the SNMP tables we work with hex or binary values.

In each of those elements sets, gets, checks and other functions have been made. These functions are more or less simple, the only problematic thing that we had was the conversion of the data between these elements and the SNMP tables. In these elements we work with integers and in the SNMP tables we work with hex or binary values.

The correspondence between the SNMP tables and the Click element structures can be seen in the next table:

SNMP-MIB	CLICK ROUTER element
dot1qVlanStaticName	`_vlanNamesTable in qetherswitch element`
dot1qVlanStaticEgressPorts	`_booltable and _vectortable in qetherswitch element`
dot1qVlanForbiddenEgressPorts	`_forbiddenVectorTable in qetherswitch`
dot1qVlanStaticUntaggedPorts	`EncapTable in qtransout element`
dot1qVlanStaticRowStatus	`_vlanStatusTable in qetherswitch`

dot1qPvid	`EncapTable in qtransin element`

r Table of created structures to make Click Router capable of working with DRAGON

The *_vlanNamesTable* and the *_vlanStatusTable* are a String Vector and an int Vector. The position of the vector means the VLAN.

The _booltable and the *_forbiddenVectorTable* of the *qetherswitch* are a Hash of a bool array. The index of the Hash will correspond to the VLAN and each bool of the array correspond to the port. For example if the port 3 belongs to the VLAN 5 the `_booltable[5][2]` will be true. A port in the Click Router is the same port + 1 for the SNMP. The *_vectortable* keeps the same information than the *_booltable* but in a different structure, a Hash of a int Vector.

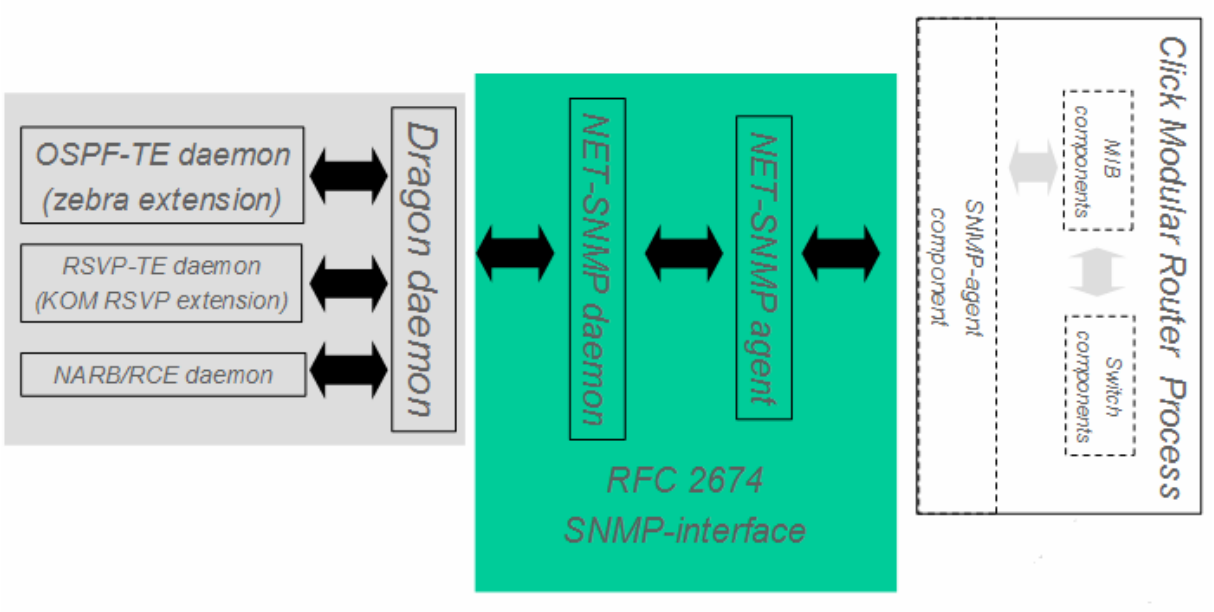
The EncapTables of the *qtransout* and the qtransin elements are a Vector of an int Vector. The first index corresponds to the port and the second index corresponds to the VLAN. In the *qtransin* the second index is not used and the value of the Pvid corresponds to the value of the EncapTable[port][4097]. We chose the 4097 value because is the first value available that cannot be a VLAN id.

Actually the *dot1VlanStaticName* and the *dot1qVlanForbiddenEgressPorts* are not needed to work with the DRAGON software but it could be handy for future extensions.

The description of the Click Router that it should answer to create a RSVP session is not associated to any Click Router element because is defined in the SNMP daemon configuration with the definition of the *sysdescr* that you can see in the Appendix A.

The most notable of this implementation is that the Click Router behaves like a commercial switch. It means for example that the VLAN 1 is the default VLAN and it behaves like that (you can modify the ports of this VLAN but you can not modify the name or status).

Below, a picture to see how is finally the relationship between the different elements of the configuration.



s Relationship between the different elements of the configuration.

Finally the Click Router is ready to be tested with the DRAGON software. The next chapter will test the new features added to the Click Router.

Chapter 7

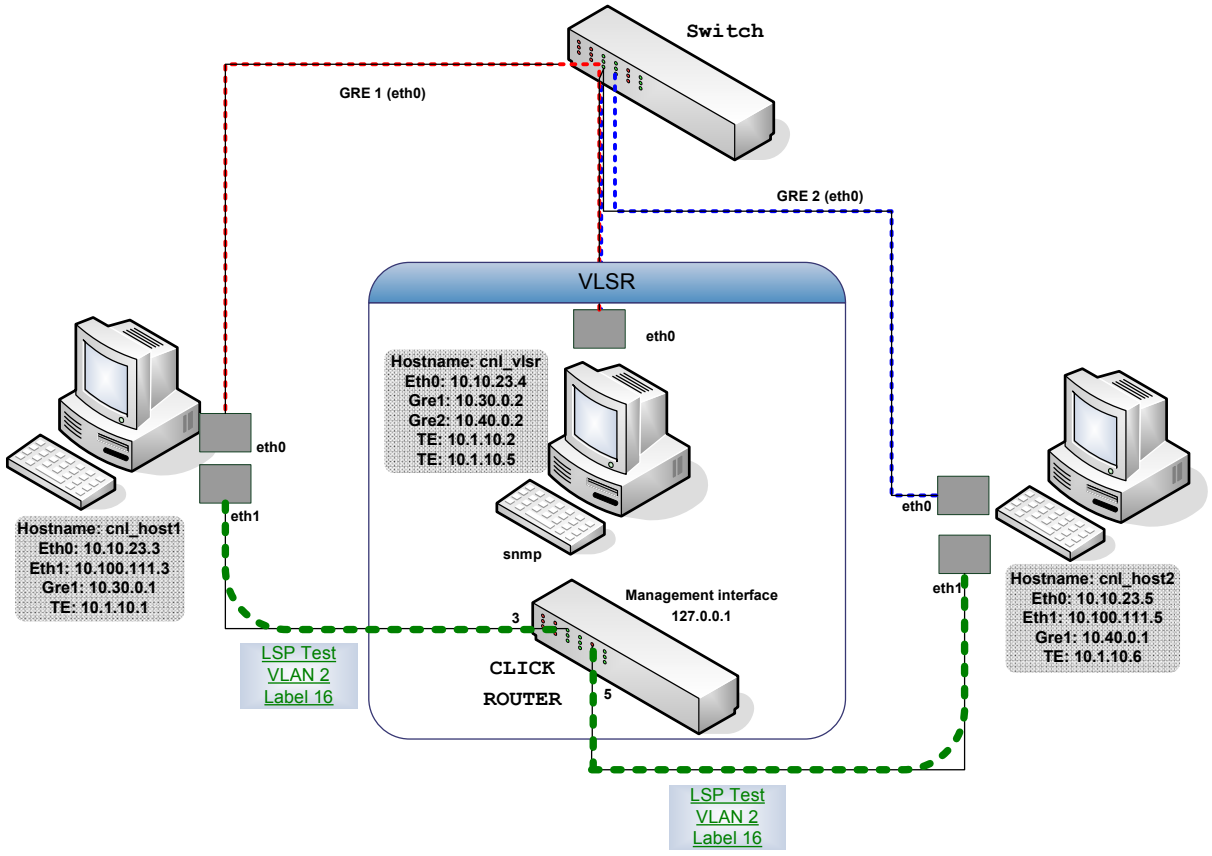
DRAGON & Click Router Tests

In this chapter the new CLICK elements are going to be tested with the DRAGON software. We start with a simple test and we are going increase the difficulty in each test.

For each test only the most notable things will be discussed but all the configuration files, and some dumps or logs can be found it on the CD. The Click Router script is the same for all the tests and consists in a switch of 8 ports capable of support Q encapsulation.

7.1 TEST 1 – 2 CSA and 1 VLSR

This is the same test as I did with the DLink switch. With this test I want to show you that the new elements of the CLICK ROUTER correctly work with the DRAGON software.



t Click Router and DRAGON test 1 - Scenario

In this test an LSP from host2 to host1 is created with this configuration:

```
cnl_host5-dragon> show lsp
                          **LSP status summary**

Name        Status     Dir   Source (IP/LSP ID)  Destination (IP/Tunnel ID)
------------------------------------------------------------------------
test        Edit        =>   10.10.23.5          10.10.23.3
                             100                 200
cnl_host5-dragon> commit lsp test
cnl_host5-dragon> show lsp
                          **LSP status summary**

Name        Status     Dir   Source (IP/LSP ID)  Destination (IP/Tunnel ID)
------------------------------------------------------------------------
test        In service  =>   10.10.23.5          10.10.23.3
                             100                 200
```

u Click Router and DRAGON test 1 - created LSP

At the beginning of the test the dot1qVlanStatic table was empty and the dot1qPvid table was filled for every port with PVID 1. After the commit of the LSP the DRAGON software made via SNMP some modifications in the CLICK ROUTER. This is a simplification of the final state of the tables:

```
dia4:~/dragon-sw# snmpwalk -v 2c -c dragon localhost Q-BRIDGE-MIB::dot1qVlanStaticTable
Q-BRIDGE-MIB::dot1qVlanStaticEgressPorts.1 = Hex-STRING: D7 00 00 00
Q-BRIDGE-MIB::dot1qVlanStaticEgressPorts.2 = Hex-STRING: 28 00 00 00
Q-BRIDGE-MIB::dot1qVlanStaticUntaggedPorts.1 = Hex-STRING: D7 00 00 00
Q-BRIDGE-MIB::dot1qVlanStaticUntaggedPorts.2 = Hex-STRING: 28 00 00 00
dia4:~/dragon-sw# snmpwalk -v 2c -c dragon localhost Q-BRIDGE-MIB::dot1qPvid
Q-BRIDGE-MIB::dot1qPvid.1 = Gauge32: 1
Q-BRIDGE-MIB::dot1qPvid.2 = Gauge32: 1
Q-BRIDGE-MIB::dot1qPvid.3 = Gauge32: 2
Q-BRIDGE-MIB::dot1qPvid.4 = Gauge32: 1
Q-BRIDGE-MIB::dot1qPvid.5 = Gauge32: 2
Q-BRIDGE-MIB::dot1qPvid.6 = Gauge32: 1
Q-BRIDGE-MIB::dot1qPvid.7 = Gauge32: 1
Q-BRIDGE-MIB::dot1qPvid.8 = Gauge32: 1
```

v Click Router and DRAGON test 1 - SNMP tables

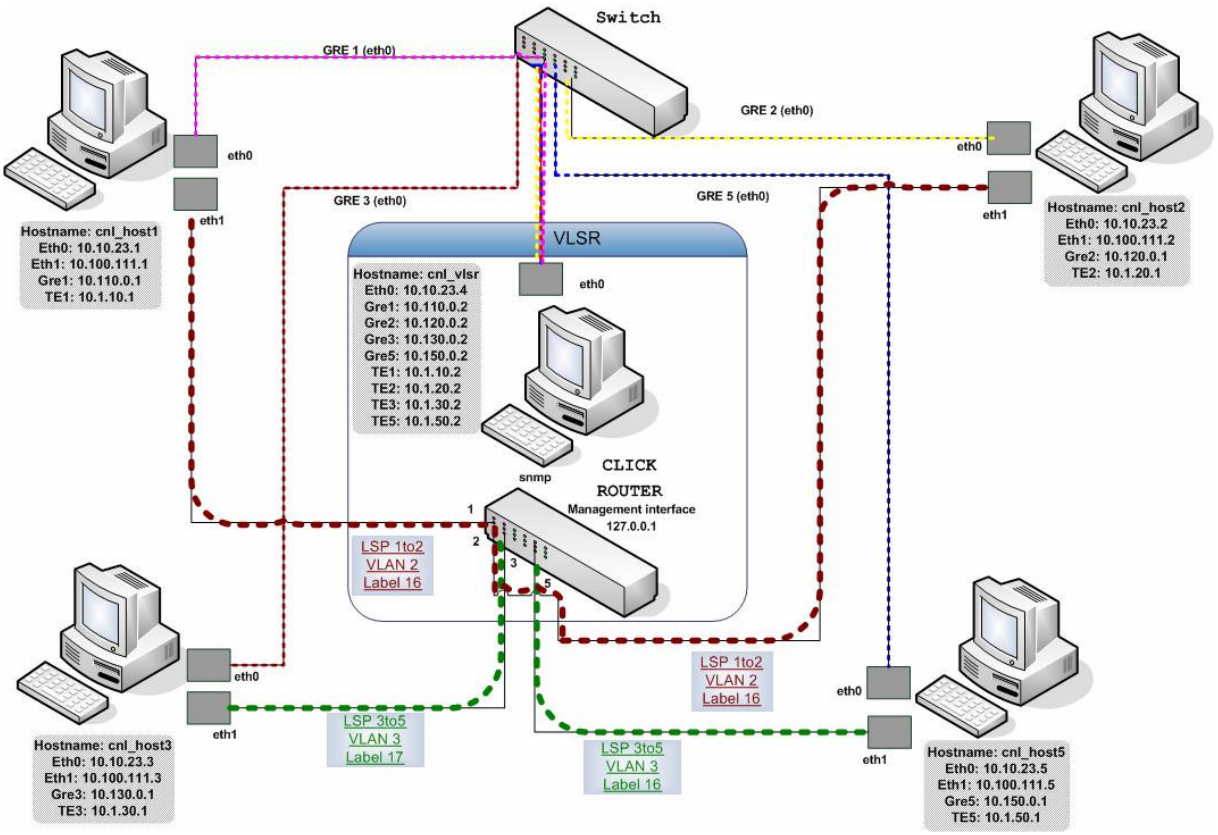
In this picture we can see that only the ports 3 and 5 belong to the VLAN 2 (hex value 28) and the rest of ports belong to the VLAN 1. Also the DRAGON puts those ports in the untagged table because the path that I set was an untagged path. Finally it modifies the dot1qPvid table putting the PVID 2 to the ports 3 and 5.

In order to test that the path works properly I made a ping from the host 5 to the other for through data-plane connection (eth1). The ping could not reach the host3 until the LSP is set. You can find these logs on the CD.

7.2 TEST 2 – 4 CSA and 1 VLSR

The goal of this test is to set up more than one path through the same VLSR but this test is not a multiple VLAN test yet because the path is set between different hosts. From this test to the future tests the Click script that is going to be used can be found in Appendix B.

The scenario of this test is this:



w Click Router and DRAGON test 2 - Scenario

In order to see how the tables are changing a simplified picture is going to be showed after each LSP set up. The LSPs are configured as in the last test. The first path to be committed is the LSP from host 1 to host 2 and this is the result of this action.

```
dia4:~/dragon-sw# snmpwalk -v 2c -c dragon localhost Q-BRIDGE-MIB::dot1qVlanStaticTable
Q-BRIDGE-MIB::dot1qVlanStaticEgressPorts.1 = Hex-STRING: 3F 00 00 00
Q-BRIDGE-MIB::dot1qVlanStaticEgressPorts.2 = Hex-STRING: C0 00 00 00
Q-BRIDGE-MIB::dot1qVlanStaticUntaggedPorts.1 = Hex-STRING: 3F 00 00 00
Q-BRIDGE-MIB::dot1qVlanStaticUntaggedPorts.2 = Hex-STRING: C0 00 00 00
dia4:~/dragon-sw# snmpwalk -v 2c -c dragon localhost Q-BRIDGE-MIB::dot1qPvid
Q-BRIDGE-MIB::dot1qPvid.1 = Gauge32: 2
Q-BRIDGE-MIB::dot1qPvid.2 = Gauge32: 2
Q-BRIDGE-MIB::dot1qPvid.3 = Gauge32: 1
Q-BRIDGE-MIB::dot1qPvid.4 = Gauge32: 1
Q-BRIDGE-MIB::dot1qPvid.5 = Gauge32: 1
Q-BRIDGE-MIB::dot1qPvid.6 = Gauge32: 1
Q-BRIDGE-MIB::dot1qPvid.7 = Gauge32: 1
Q-BRIDGE-MIB::dot1qPvid.8 = Gauge32: 1
```

x Click Router and DRAGON test 2 - SNMP tables with one created LSP

As in the last test we can see that only the ports 1 and 2 belong to the VLAN 2 (hex value C0) and the rest of ports belong to the VLAN 1. Also DRAGON puts those ports in the untagged table because the path that I set was an untagged path. Finally it modifies the dot1qPvid table putting the PVID 2 to the ports 1 and 2.

Once viewed that the first LSP was committed properly the second LSP was committed and next you can see the result where the DRAGON takes the port 3 and 4 of the VLAN 1 and puts in the VLAN 3.

```
dia4:~/dragon-sw# snmpwalk -v 2c -c dragon localhost Q-BRIDGE-MIB::dot1qVlanStaticTable
Q-BRIDGE-MIB::dot1qVlanStaticEgressPorts.1 = Hex-STRING: 17 00 00 00
Q-BRIDGE-MIB::dot1qVlanStaticEgressPorts.2 = Hex-STRING: C0 00 00 00
Q-BRIDGE-MIB::dot1qVlanStaticEgressPorts.3 = Hex-STRING: 28 00 00 00
Q-BRIDGE-MIB::dot1qVlanStaticUntaggedPorts.1 = Hex-STRING: 17 00 00 00
Q-BRIDGE-MIB::dot1qVlanStaticUntaggedPorts.2 = Hex-STRING: C0 00 00 00
Q-BRIDGE-MIB::dot1qVlanStaticUntaggedPorts.3 = Hex-STRING: 28 00 00 00
dia4:~# snmpwalk -v 2c -c dragon localhost Q-BRIDGE-MIB::dot1qPvid
Q-BRIDGE-MIB::dot1qPvid.1 = Gauge32: 2
Q-BRIDGE-MIB::dot1qPvid.2 = Gauge32: 2
Q-BRIDGE-MIB::dot1qPvid.3 = Gauge32: 3
Q-BRIDGE-MIB::dot1qPvid.4 = Gauge32: 1
Q-BRIDGE-MIB::dot1qPvid.5 = Gauge32: 3
Q-BRIDGE-MIB::dot1qPvid.6 = Gauge32: 1
Q-BRIDGE-MIB::dot1qPvid.7 = Gauge32: 1
Q-BRIDGE-MIB::dot1qPvid.8 = Gauge32: 1
```

y Click Router and DRAGON test 2 - SNMP tables with two created LSPs

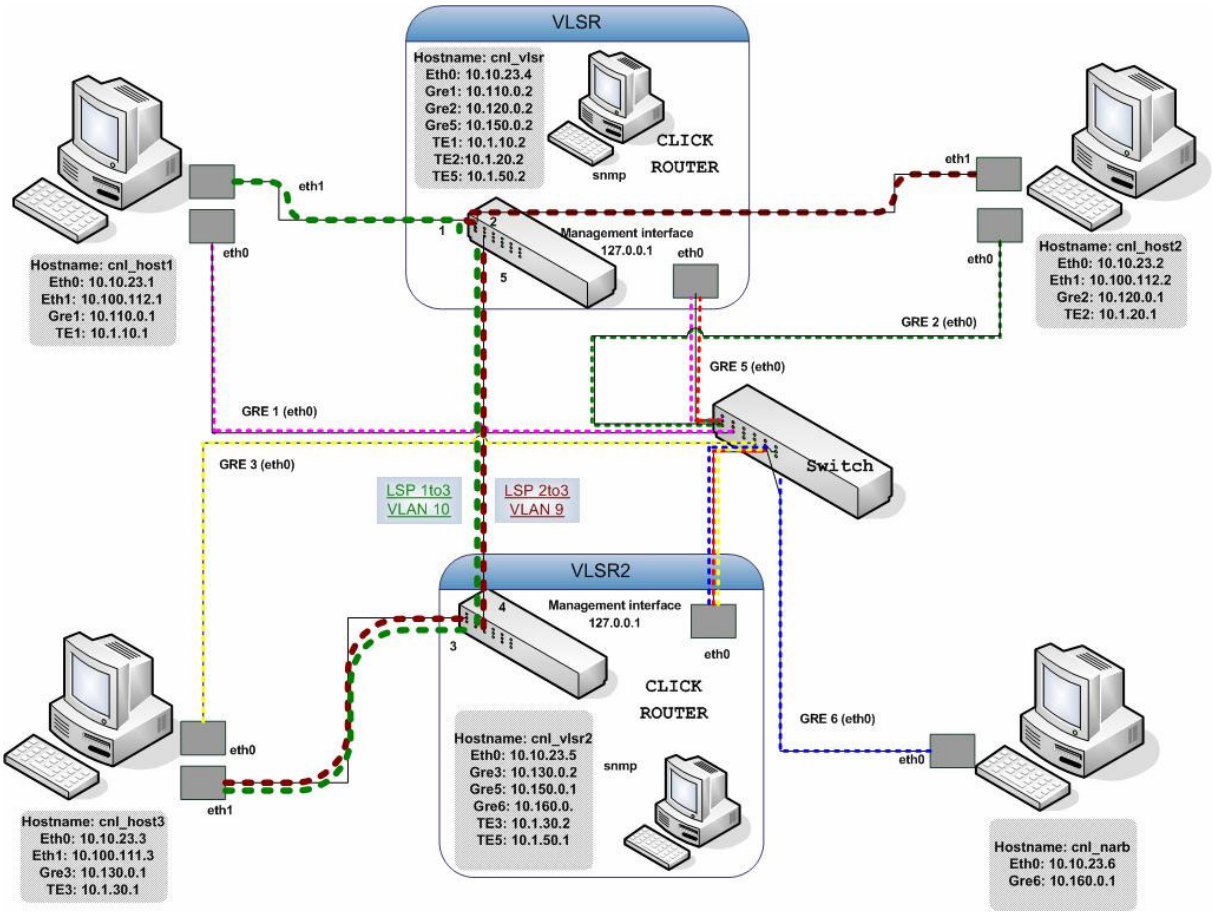
The ports 1 and 2 stay in the VLAN 2 (hex value C0) but now the ports 3 and 5 belong to the VLAN 3 (hex value 28), the reason that the DRAGON takes the VLAN 3 is that it was the first VLAN available because the VLAN 2 was used for another LSP.

After those tests it seems that communication between the DRAGON software, the SNMP and the CLICK ROUTER behave properly. The next step is to set up more than one LSP for the same connection but for this we need the NARB module from DRAGON software and also to tag the LSP.

7.3 TEST 3 – 3 UNI clients and 2 VLSRs

The goal of this test is to set up more than one path between two VLSRs. This is the first test with a multiple VLAN. To make a multiple VLAN test it is necessary to install the NARB/RCE module who is responsible of commitment the LSPs.

The scenario of this test is this:



z Click Router and DRAGON test 3 - Scenario

In this test we added the use of two new characteristics. The first one is that the hosts are not a CSAs. In this test the hosts are User Network Interface UNI clients. The difference between

the CSA and the UNI clients is mainly that the UNI client does not participate in the OSPF signaling. The second element that we added in this test is the use of the local-id and the tagged groups. Furthermore, of the NARB to create a tagged LSP for a multiple VLAN, we need to define the local-ids and/or the tagged groups. A local-id is an identifier and this identifier can or can not belong to a tagged or an untagged group. If it is a tagged group the name of this group will be the VLAN of the LSP.

In order to make this test we followed the instructions of the second test of the VLSR Implementation Guide. Next a creation of one of the LSPs is going to be commented.

```
cnl_host2-dragon(edit-lsp-2to3)# set uni client ingress gre2 egress gre3
cnl_host2-dragon(edit-lsp-2to3)# set source ip-address 10.10.23.4 port 2 destina
tion ip-address 10.10.23.5 tagged-group 9
cnl_host2-dragon(edit-lsp-2to3)# set bandwidth eth100M swcap l2sc encoding ether
net gpid ethernet
```

aa Click Router and DRAGON test 3 - Creation of LSP

The first instruction is to tell to the VLSRs and also the NARB from where and to where GRE tunnels are going the control plane. In this case we use the name of the tunnel but you also can use the value “implicit” that use the implicit GRE tunnel. The second instruction is similar to the previous tests but in this case we do not use the lsp-id and the tunnel-id because we need to tag the LSP to have more than one LSP for the same physical cable (the connection between the VLSRs). First of all the IP is the IP from the ingress VLSR not from the UNI client and then we put the port that it is defined in the configuration files. This port is the port from where the ingress VLSR is receiving the LSP request. After, the IP of the egress VLSR and finally the tagged group that also is defined in the configuration files. This tagged group means that these LSPs will run with the VLAN of the group, in this case 10. The last line is like in the other tests but here we requested only 100 M.

In order to use the port option in the set up of the LSPs you should add to the dragon.conf file in the UNI clients this:

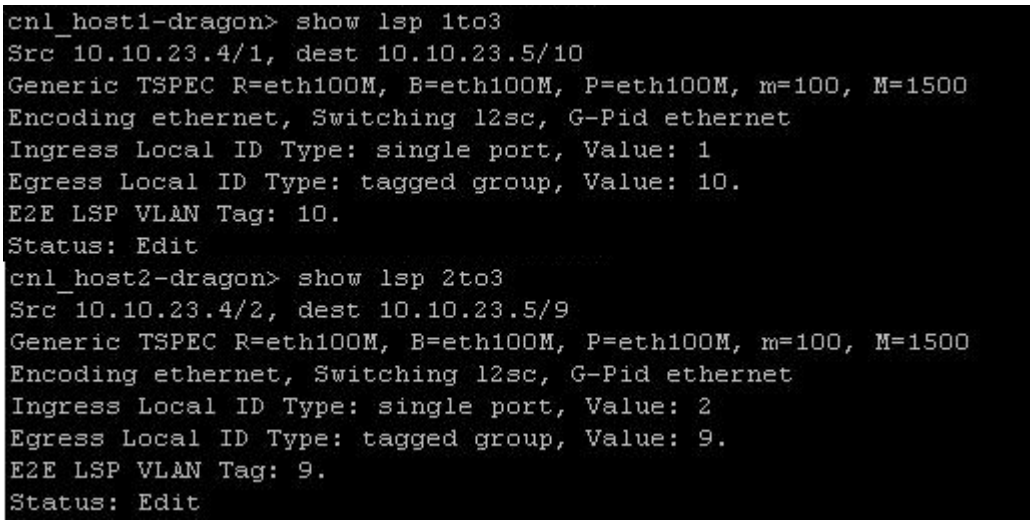
```
set local-id port x
```

Where x is depends on the UNI client. And to use the tagged-group option you should add to the dragon.conf file in the VLSRs this:

```
set local-id port 1
set local-id tagged-group 2 add 1
```

The first line is to define the port 1 and the second line is to define that the port 1 belongs to the tagged group 2. You can add the port to every group that you want. You also can see the complete configuration of the ports ant tagged-groups in the configuration files of the tests.

Below, the specification of the two created LSPs where you can see that the Ingress Local ID Types are single ports. It means that the VLSR knows for which interfaces comes the packets of this LSP and because of this the LSP for this port will be untagged. However the destination is the same for both LSP and it is because of this that the packet should be tagged to arrive to the destination.



bb Click Router and DRAGON test 3 - Created LSPs

Next, the final state of the SNMP tables in the VLSR 1 where you can see that the ports 2 and 5 (hex value 48) belong to the VLAN 9 and the ports 1 and 5 (hex value 88) belong to the VLAN 10. But if we look to the Unttaged information we see that the port 2 (hex value 40) and the port 1 (hex value 80) will be unttaged and therefore the port 5 will be tagged because the VLSR 2 will need to distinguish from which LSP (LSP with VLAN 10 or 9) come the packets.

About the dot1qPvid table the port 1 belongs to the VLAN 10 and the port 2 to the VLAN 9 and the port 5 belongs to the VLAN 9 because it was the last committed LSP. The value of

the Pvid in the ports 1 and 2 is important because the packets from this source will probably come untagged but the packets that come to the port 5 (from the VLSR 2) will come tagged because it is the only way to distinguish the LSP that is using.

```
dia4:~# snmpwalk -v 2c -c dragon localhost Q-BRIDGE-MIB::dot1qVlanStaticTable
Q-BRIDGE-MIB::dot1qVlanStaticEgressPorts.9 = Hex-STRING: 48 00 00 00
Q-BRIDGE-MIB::dot1qVlanStaticEgressPorts.10 = Hex-STRING: 88 00 00 00
Q-BRIDGE-MIB::dot1qVlanStaticUntaggedPorts.9 = Hex-STRING: 40 00 00 00
Q-BRIDGE-MIB::dot1qVlanStaticUntaggedPorts.10 = Hex-STRING: 80 00 00 00
dia4:~# snmpwalk -v 2c -c dragon localhost Q-BRIDGE-MIB::dot1qPvid
Q-BRIDGE-MIB::dot1qPvid.1 = Gauge32: 10
Q-BRIDGE-MIB::dot1qPvid.2 = Gauge32: 9
Q-BRIDGE-MIB::dot1qPvid.3 = Gauge32: 1
Q-BRIDGE-MIB::dot1qPvid.4 = Gauge32: 1
Q-BRIDGE-MIB::dot1qPvid.5 = Gauge32: 9
Q-BRIDGE-MIB::dot1qPvid.6 = Gauge32: 1
Q-BRIDGE-MIB::dot1qPvid.7 = Gauge32: 1
Q-BRIDGE-MIB::dot1qPvid.8 = Gauge32: 1
```

cc Click Router and DRAGON test 3 – Final state of the SNMP tables in the VLSR 1 (dia4)

In the next picture we can see the final state of the SNMP tables of the VLSR 2. The values of the dot1qPvid table are not very interesting because the packets from the ports 4 and 3 always will be tagged (with the VLAN 9 or 10). In the dot1qVlanStaticTable only is necessary to emphasize that port 3 and 4 (hex value 30) belongs to the VLANS 9 and 10 and both ports won't be untagged.

```
dia5:~# snmpwalk -v 2c -c dragon localhost Q-BRIDGE-MIB::dot1qVlanStaticTable
Q-BRIDGE-MIB::dot1qVlanStaticEgressPorts.9 = Hex-STRING: 30 00 00 00
Q-BRIDGE-MIB::dot1qVlanStaticEgressPorts.10 = Hex-STRING: 30 00 00 00
Q-BRIDGE-MIB::dot1qVlanStaticUntaggedPorts.9 = Hex-STRING: 00 00 00 00
Q-BRIDGE-MIB::dot1qVlanStaticUntaggedPorts.10 = Hex-STRING: 00 00 00 00
dia5:~# snmpwalk -v 2c -c dragon localhost Q-BRIDGE-MIB::dot1qPvid
Q-BRIDGE-MIB::dot1qPvid.1 = Gauge32: 1
Q-BRIDGE-MIB::dot1qPvid.2 = Gauge32: 1
Q-BRIDGE-MIB::dot1qPvid.3 = Gauge32: 9
Q-BRIDGE-MIB::dot1qPvid.4 = Gauge32: 9
Q-BRIDGE-MIB::dot1qPvid.5 = Gauge32: 1
Q-BRIDGE-MIB::dot1qPvid.6 = Gauge32: 1
Q-BRIDGE-MIB::dot1qPvid.7 = Gauge32: 1
Q-BRIDGE-MIB::dot1qPvid.8 = Gauge32: 1
```

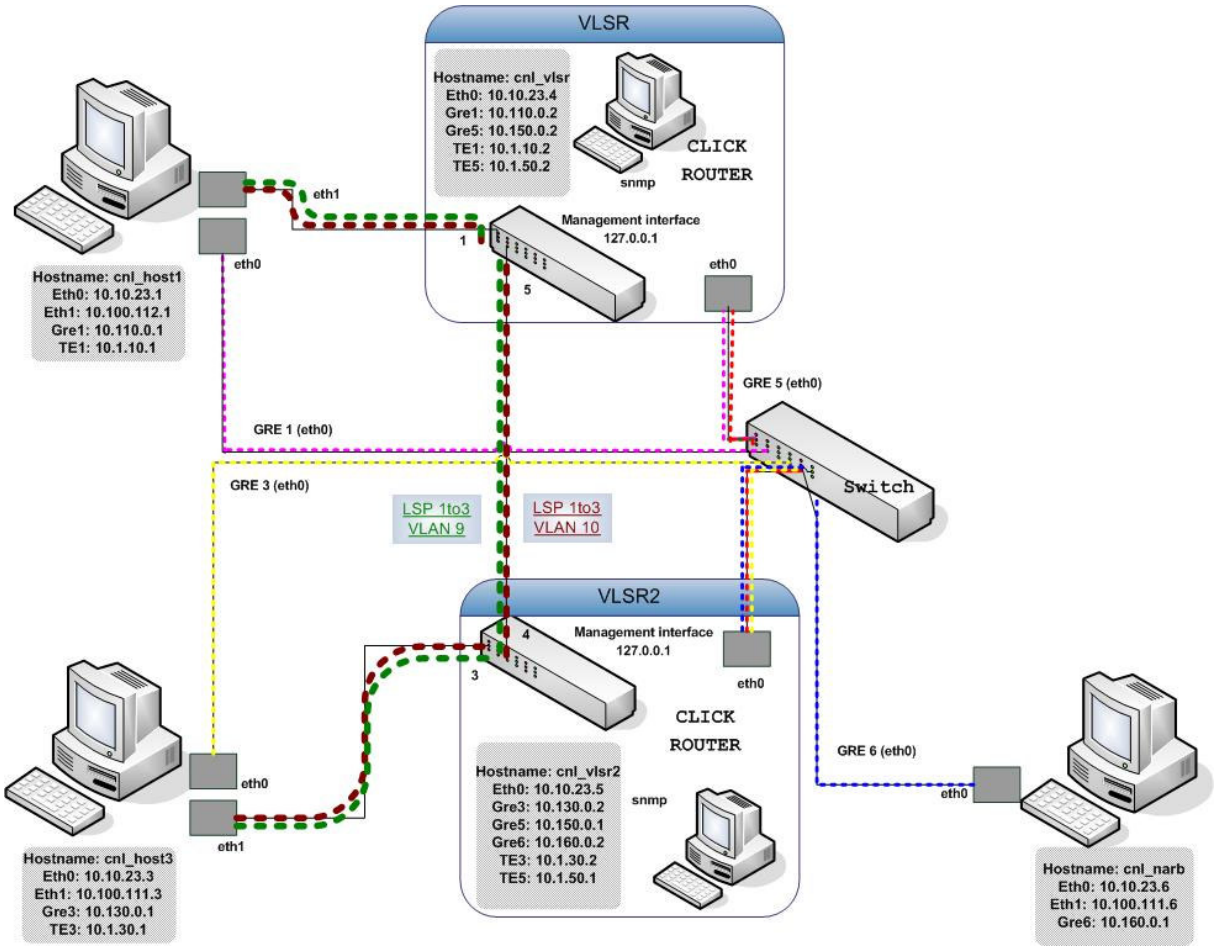
dd Click Router and DRAGON test 3 - Final state of the SNMP tables in the VLSR 2 (dia5)

After this test we can affirm that it is possible to set up more than one LSP between two VLSRs. In this case the destination was the same but the origin of the LSP was different. In the next test we will prove that it is possible to set up multiple VLAN between the same hosts.

7.4 TEST 4 – 2 UNI clients and 2 VLSRs

The goal of this test is to set up more than one LSP between the same couple of hosts. This test will be similar to the previous test but in this case the packets of the LSPs will be tagged during all the path because the origin and the destination host have to distinguish from which LSP comes the packets due that both LSPs come through the same physical cable.

The scenario of this test is this:



ee Click Router and DRAGON test 4 - Scenario

The scenario of this test is almost equal than the previous test and the way to set up the path is similar as we can see (with some difficulties) in the image of below. There is only a difference that is that in this case is not used the port definition and is used the tagged-group in the origin and the destination. This is because of we need to tag the packets from the

beginning to the end to make possible have more than one path between the same couple of hosts. The other LSP set up is equal than this but with the tagged-group 9.

```
cnl_host1-dragon> edit lsp 1to3-VLAN10
cnl_host1-dragon(edit-lsp-1to3-VLAN10)# set uni client ingress gre1 egress gre3
cnl_host1-dragon(edit-lsp-1to3-VLAN10)#  set source ip-address 10.10.23.4 tagged-group 10 destination ip-address 10.10.23.5 t
agged-group 10
cnl_host1-dragon(edit-lsp-1to3-VLAN10)# set bandwidth eth100M swcap l2sc encoding ethernet gpid ethernet
cnl_host1-dragon(edit-lsp-1to3-VLAN10)# exit
```

ff Click Router and DRAGON test 4 - Creation of LSP

Next, the SNMP tables after the creation of both LSPs are going to be showed. How is normal and we had seen in the other test the DRAGON software adds the ports to the VLANs that the tagged-groups order. In this test the ports 1 and 5 (hex value 88) in the VLSR 1 and the ports 3 and 4 (hex value 30) in the VLSR 2. The only data that is important to emphasize is in the VLSRs every port is not untagged. This makes possible the multiple VLAN between a couples of hosts.

```
dia4:~# snmpwalk -v 2c -c dragon localhost Q-BRIDGE-MIB::dot1qVlanStaticTable
Q-BRIDGE-MIB::dot1qVlanStaticEgressPorts.9 = Hex-STRING: 88 00 00 00
Q-BRIDGE-MIB::dot1qVlanStaticEgressPorts.10 = Hex-STRING: 88 00 00 00
Q-BRIDGE-MIB::dot1qVlanStaticUntaggedPorts.9 = Hex-STRING: 00 00 00 00
Q-BRIDGE-MIB::dot1qVlanStaticUntaggedPorts.10 = Hex-STRING: 00 00 00 00
dia4:~# snmpwalk -v 2c -c dragon localhost Q-BRIDGE-MIB::dot1qPvid
Q-BRIDGE-MIB::dot1qPvid.1 = Gauge32: 9
Q-BRIDGE-MIB::dot1qPvid.2 = Gauge32: 1
Q-BRIDGE-MIB::dot1qPvid.3 = Gauge32: 1
Q-BRIDGE-MIB::dot1qPvid.4 = Gauge32: 1
Q-BRIDGE-MIB::dot1qPvid.5 = Gauge32: 9
Q-BRIDGE-MIB::dot1qPvid.6 = Gauge32: 1
Q-BRIDGE-MIB::dot1qPvid.7 = Gauge32: 1
Q-BRIDGE-MIB::dot1qPvid.8 = Gauge32: 1
```

gg Click Router and DRAGON test 4 - Final state of the SNMP tables in the VLSR 1 (dia4)

```
dia5:~# snmpwalk -v 2c -c dragon localhost Q-BRIDGE-MIB::dot1qVlanStaticTable
Q-BRIDGE-MIB::dot1qVlanStaticEgressPorts.9 = Hex-STRING: 30 00 00 00
Q-BRIDGE-MIB::dot1qVlanStaticEgressPorts.10 = Hex-STRING: 30 00 00 00
Q-BRIDGE-MIB::dot1qVlanStaticUntaggedPorts.9 = Hex-STRING: 00 00 00 00
Q-BRIDGE-MIB::dot1qVlanStaticUntaggedPorts.10 = Hex-STRING: 00 00 00 00
dia5:~# snmpwalk -v 2c -c dragon localhost Q-BRIDGE-MIB::dot1qPvid
Q-BRIDGE-MIB::dot1qPvid.1 = Gauge32: 1
Q-BRIDGE-MIB::dot1qPvid.2 = Gauge32: 1
Q-BRIDGE-MIB::dot1qPvid.3 = Gauge32: 9
Q-BRIDGE-MIB::dot1qPvid.4 = Gauge32: 9
Q-BRIDGE-MIB::dot1qPvid.5 = Gauge32: 1
Q-BRIDGE-MIB::dot1qPvid.6 = Gauge32: 1
Q-BRIDGE-MIB::dot1qPvid.7 = Gauge32: 1
Q-BRIDGE-MIB::dot1qPvid.8 = Gauge32: 1
```

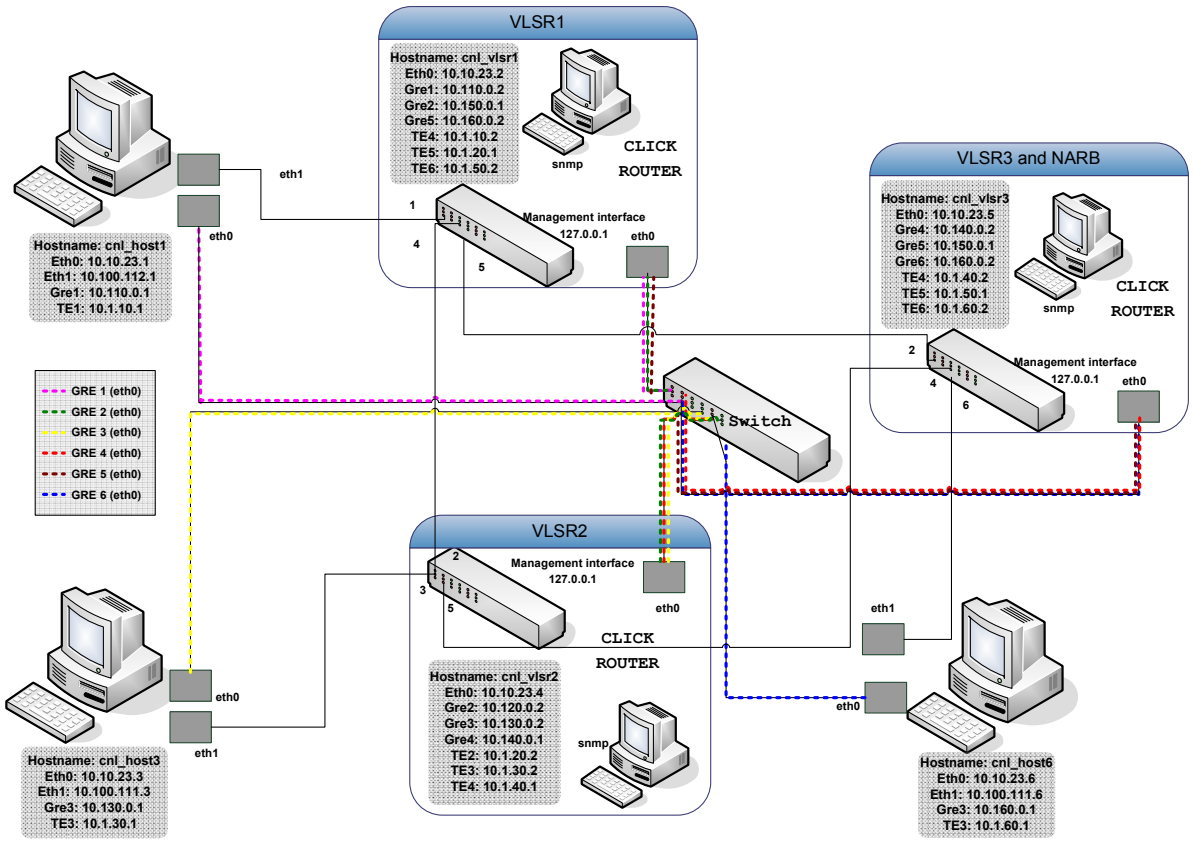
hh Click Router and DRAGON test 4 - Final state of the SNMP tables in the VLSR 2 (dia5)

After these four tests we can affirm that the DRAGON software and the Click Router with the new elements work together properly and with the new elements of the Click Router we could use all the possibilities of the DRAGON software like a commercial router can do it.

7.5 FINAL TEST

In order to finalize with the DRAGON and Click Router tests we are going to do a test to prove how DRAGON distribute the LSPs according to the bandwidth conditions of the links.

The scenario of the test will be this:



ii Click Router and DRAGON Final test - Scenario

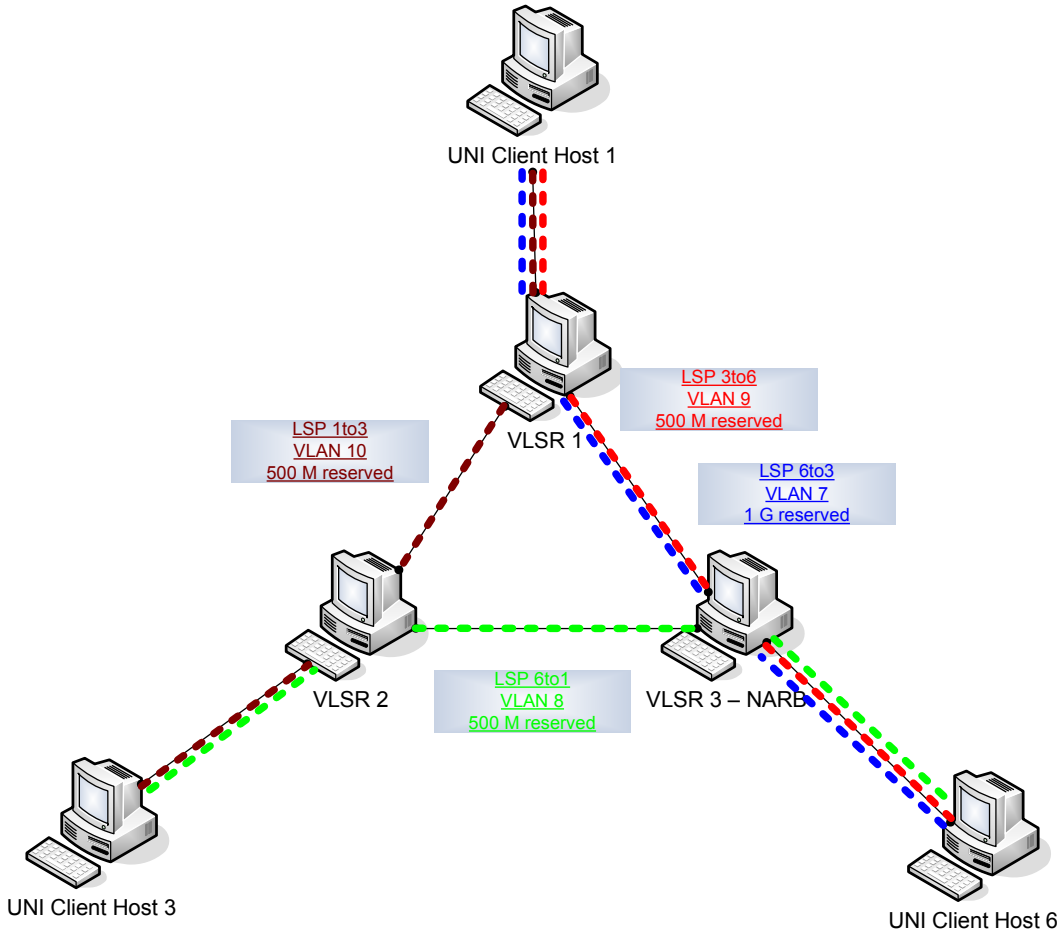
The scenario consist in a cluster of three VLSR connected between them. Furthermore, of the two connections from the other VLSRs, each VLSR have been connected to an UNI client, the host 1 is connected to the VLSR 1 (dia2), the host 3 is connected to the VLSR 2 (dia4)

and the host 6 is connected to the VLSR 3 (dia5). Another important detail in this test is that the NARB is in the same machine as the VLSR 3. In this machine the ospfd.conf file is used by the NARB like a ospf intra-domain configuration file and we use the command *./dragon.sh start-vlsr-narb* to start both elements.

In order to prove how the DRAGON distributes the LSPs depending on the bandwidth we have limited the bandwidth of the links between the VLSRs with a bandwidth of 1.5 Gigabytes. In this test we have two phases that we are going to explain.

7.5.1 First phase

In the first phase we have created an LSP between each UNI client with a bandwidth requested of 500M of the 1500M available. In order to create the LSPs we use the same method like in the previous test using the same tagged-group for origin and destination of the LSP. The tagged-group used for each LSP goes from 10 to 6 that means the VLANs will be from 10 to 6. Furthermore, of the three LSPs between the UNI clients, we have set up another LSP from the host 6 to the host 3 of 1 Giga of bandwidth. Therefore the link between the VLSR 2 and the VLSR 3 will be with all the available bandwidth reserved. Below, the configuration of the LSPs in the first phase:



jj Click Router and DRAGON Final test - Topology of the created LSPs in the first phase

In the next picture you can see the SNMP tables of the three VLSRs. In these tables you can see that each port is added to the corresponding VLAN and all the ports are not unttaged.

```
dia2:~# snmpwalk -v 2c -c dragon localhost Q-BRIDGE-MIB::dot1qVlanStaticTable
Q-BRIDGE-MIB::dot1qVlanStaticEgressPorts.6 = Hex-STRING: 00 00 00 00
Q-BRIDGE-MIB::dot1qVlanStaticEgressPorts.7 = Hex-STRING: 00 00 00 00
Q-BRIDGE-MIB::dot1qVlanStaticEgressPorts.8 = Hex-STRING: 98 00 00 00
Q-BRIDGE-MIB::dot1qVlanStaticEgressPorts.9 = Hex-STRING: 00 00 00 00
Q-BRIDGE-MIB::dot1qVlanStaticEgressPorts.10 = Hex-STRING: 98 00 00 00
Q-BRIDGE-MIB::dot1qVlanStaticUntaggedPorts.6 = Hex-STRING: 00 00 00 00
Q-BRIDGE-MIB::dot1qVlanStaticUntaggedPorts.7 = Hex-STRING: 00 00 00 00
Q-BRIDGE-MIB::dot1qVlanStaticUntaggedPorts.8 = Hex-STRING: 00 00 00 00
Q-BRIDGE-MIB::dot1qVlanStaticUntaggedPorts.9 = Hex-STRING: 00 00 00 00
Q-BRIDGE-MIB::dot1qVlanStaticUntaggedPorts.10 = Hex-STRING: 00 00 00 00
dia2:~# snmpwalk -v 2c -c dragon localhost Q-BRIDGE-MIB::dot1qPvid
Q-BRIDGE-MIB::dot1qPvid.1 = Gauge32: 8
Q-BRIDGE-MIB::dot1qPvid.2 = Gauge32: 1
Q-BRIDGE-MIB::dot1qPvid.3 = Gauge32: 1
Q-BRIDGE-MIB::dot1qPvid.4 = Gauge32: 8
Q-BRIDGE-MIB::dot1qPvid.5 = Gauge32: 8
Q-BRIDGE-MIB::dot1qPvid.6 = Gauge32: 1
Q-BRIDGE-MIB::dot1qPvid.7 = Gauge32: 1
Q-BRIDGE-MIB::dot1qPvid.8 = Gauge32: 1

dia4:~# snmpwalk -v 2c -c dragon localhost Q-BRIDGE-MIB::dot1qVlanStaticTable
Q-BRIDGE-MIB::dot1qVlanStaticEgressPorts.6 = Hex-STRING: 00 00 00 00
Q-BRIDGE-MIB::dot1qVlanStaticEgressPorts.7 = Hex-STRING: 68 00 00 00
Q-BRIDGE-MIB::dot1qVlanStaticEgressPorts.8 = Hex-STRING: 00 00 00 00
Q-BRIDGE-MIB::dot1qVlanStaticEgressPorts.9 = Hex-STRING: 68 00 00 00
Q-BRIDGE-MIB::dot1qVlanStaticEgressPorts.10 = Hex-STRING: 68 00 00 00
Q-BRIDGE-MIB::dot1qVlanStaticUntaggedPorts.6 = Hex-STRING: 00 00 00 00
Q-BRIDGE-MIB::dot1qVlanStaticUntaggedPorts.7 = Hex-STRING: 00 00 00 00
Q-BRIDGE-MIB::dot1qVlanStaticUntaggedPorts.8 = Hex-STRING: 00 00 00 00
Q-BRIDGE-MIB::dot1qVlanStaticUntaggedPorts.9 = Hex-STRING: 00 00 00 00
Q-BRIDGE-MIB::dot1qVlanStaticUntaggedPorts.10 = Hex-STRING: 00 00 00 00
dia4:~# snmpwalk -v 2c -c dragon localhost Q-BRIDGE-MIB::dot1qPvid
Q-BRIDGE-MIB::dot1qPvid.1 = Gauge32: 1
Q-BRIDGE-MIB::dot1qPvid.2 = Gauge32: 7
Q-BRIDGE-MIB::dot1qPvid.3 = Gauge32: 7
Q-BRIDGE-MIB::dot1qPvid.4 = Gauge32: 1
Q-BRIDGE-MIB::dot1qPvid.5 = Gauge32: 7
Q-BRIDGE-MIB::dot1qPvid.6 = Gauge32: 1
Q-BRIDGE-MIB::dot1qPvid.7 = Gauge32: 1
Q-BRIDGE-MIB::dot1qPvid.8 = Gauge32: 1

dia5:~# snmpwalk -v 2c -c dragon localhost Q-BRIDGE-MIB::dot1qVlanStaticTable
Q-BRIDGE-MIB::dot1qVlanStaticEgressPorts.6 = Hex-STRING: 00 00 00 00
Q-BRIDGE-MIB::dot1qVlanStaticEgressPorts.7 = Hex-STRING: 54 00 00 00
Q-BRIDGE-MIB::dot1qVlanStaticEgressPorts.8 = Hex-STRING: 54 00 00 00
Q-BRIDGE-MIB::dot1qVlanStaticEgressPorts.9 = Hex-STRING: 54 00 00 00
Q-BRIDGE-MIB::dot1qVlanStaticEgressPorts.10 = Hex-STRING: 00 00 00 00
Q-BRIDGE-MIB::dot1qVlanStaticUntaggedPorts.6 = Hex-STRING: 00 00 00 00
Q-BRIDGE-MIB::dot1qVlanStaticUntaggedPorts.7 = Hex-STRING: 00 00 00 00
Q-BRIDGE-MIB::dot1qVlanStaticUntaggedPorts.8 = Hex-STRING: 00 00 00 00
Q-BRIDGE-MIB::dot1qVlanStaticUntaggedPorts.9 = Hex-STRING: 00 00 00 00
Q-BRIDGE-MIB::dot1qVlanStaticUntaggedPorts.10 = Hex-STRING: 00 00 00 00
dia5:~# snmpwalk -v 2c -c dragon localhost Q-BRIDGE-MIB::dot1qPvid
Q-BRIDGE-MIB::dot1qPvid.1 = Gauge32: 1
Q-BRIDGE-MIB::dot1qPvid.2 = Gauge32: 7
Q-BRIDGE-MIB::dot1qPvid.3 = Gauge32: 1
Q-BRIDGE-MIB::dot1qPvid.4 = Gauge32: 7
Q-BRIDGE-MIB::dot1qPvid.5 = Gauge32: 1
Q-BRIDGE-MIB::dot1qPvid.6 = Gauge32: 7
Q-BRIDGE-MIB::dot1qPvid.7 = Gauge32: 1
Q-BRIDGE-MIB::dot1qPvid.8 = Gauge32: 1
```

kk Click Router and DRAGON Final test - SNMP tables of the VLSR 1 (dia2), VLSR 2 (dia4) and VLSR 3 (dia5) in the first phase

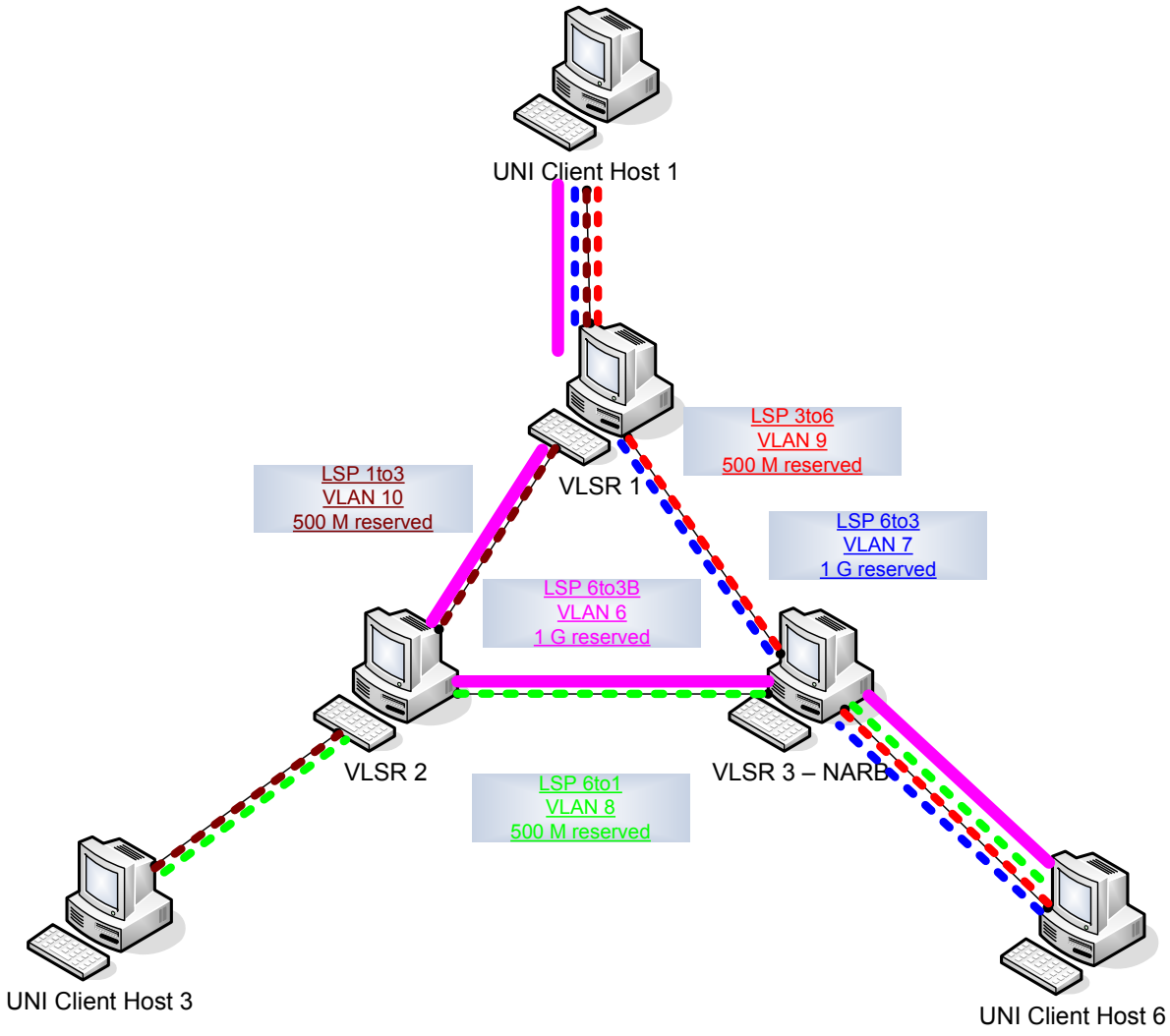
7.5.2 Second phase

In this second phase we are going to set up a new LSP of 1 gige between the host 6 and the host 3. But now DRAGON should change the typical path because the link between VLSR 2 and the VLSR 3 has not bandwidth available. In order to see how the DRAGON manages the request of this LSP a log of the NARB that you can find in the CD is going to be commented.

```
1 NARB: Accepted an LSPQ connection on socket(11)
2 NARB: LSP_Broker::MSG_APP_REQUEST: The LSPQ (ucid=0x5170a0a,
seqno=0x946d260).
3 NARB: RCE_APIClient::ReadMessage return ERO with 2 hops
4 NARB: HOP-TYPE [strict]: 10.1.40.2 [UnumIfId: 0(0,0): vtag:1792]
5 NARB: HOP-TYPE [strict]: 10.1.40.1 [UnumIfId: 0(0,0): vtag:1792]
6 NARB: HOP-TYPE [strict]: 10.10.23.5 [UnumIfId: 196615 (3,7)]
7 NARB: HOP-TYPE [strict]: 10.1.40.2 [UnumIfId: 262151 (4,7)]
8 NARB: HOP-TYPE [strict]: 10.1.40.1 [UnumIfId: 262151 (4,7)]
9 NARB: HOP-TYPE [strict]: 10.10.23.4 [UnumIfId: 196615 (3,7)]
10 NARB: HandleResvConfirm upating LSP link states: (ucid=0x5170a0a,
seqno=0x946d260).
11 NARB: Accepted an LSPQ connection on socket(12)
12 NARB: Connection closed for APIReader(12)
13 NARB: LSP_Broker::ReadMessage failed. Closing the socket 12.
14 NARB: Accepted an LSPQ connection on socket(12)
15 NARB: LSP_Broker::MSG_APP_REQUEST: The LSPQ (ucid=0x5170a0a,
seqno=0x946ee70).
16 NARB: RCE_APIClient::ReadMessage return ERO with 4 hops
17 NARB: HOP-TYPE [strict]: 10.1.50.1 [UnumIfId: 0(0,0): vtag:1536]
18 NARB: HOP-TYPE [strict]: 10.1.50.2 [UnumIfId: 0(0,0): vtag:1536]
19 NARB: HOP-TYPE [strict]: 10.1.20.1 [UnumIfId: 0(0,0): vtag:1536]
20 NARB: HOP-TYPE [strict]: 10.1.20.2 [UnumIfId: 0(0,0): vtag:1536]
21 NARB: HOP-TYPE [strict]: 10.10.23.5 [UnumIfId: 196614 (3,6)]
22 NARB: HOP-TYPE [strict]: 10.1.50.1 [UnumIfId: 262150 (4,6)]
23 NARB: HOP-TYPE [strict]: 10.1.50.2 [UnumIfId: 262150 (4,6)]
24 NARB: HOP-TYPE [strict]: 10.1.20.1 [UnumIfId: 262150 (4,6)]
25 NARB: HOP-TYPE [strict]: 10.1.20.2 [UnumIfId: 262150 (4,6)]
26 NARB: HOP-TYPE [strict]: 10.10.23.4 [UnumIfId: 196614 (3,6)]
27 NARB: HandleResvConfirm upating LSP link states: (ucid=0x5170a0a,
seqno=0x946ee70).
```

From the line 1 to the line 10 there is the reserve confirmation done by the NARB for a LSP between the host 3 and 6 when there is bandwidth available. You can see there are only 2 hops in the line 3 and these two hops are the 10.1.40.1 and the 10.1.40.2 that are the data-interfaces of the link between the VLSR 2 and VLSR 3.

The reserve confirmation done by the NARB for the LSP created in this phase (from the host 6 to the host 3) can be seen from the line 14 to the line 27. In this case there is not bandwidth available in the link between the VLSR 2 and the VLSR 3 and the NARB has to re-direct the path by the link between the VLSR 3 and the VLSR 1 (10.1.50.x data-interface) and the link between the VLSR 1 and the VLSR 2 (10.1.20.x data-interface) that have bandwidth available. In order to clarify by where the trajectory goes a new image of the LSPs it is going to be shown.



ll Click Router and DRAGON Final test - Topology of the created LSPs in the final phase

Below, the SNMP tables of the three VLSRs that correspond to the previously mentioned topology.

```
dia2:~# snmpwalk -v 2c -c dragon localhost Q-BRIDGE-MIB::dot1qVlanStaticTable
Q-BRIDGE-MIB::dot1qVlanStaticEgressPorts.6 = Hex-STRING: 18 00 00 00
Q-BRIDGE-MIB::dot1qVlanStaticEgressPorts.7 = Hex-STRING: 00 00 00 00
Q-BRIDGE-MIB::dot1qVlanStaticEgressPorts.8 = Hex-STRING: 98 00 00 00
Q-BRIDGE-MIB::dot1qVlanStaticEgressPorts.9 = Hex-STRING: 00 00 00 00
Q-BRIDGE-MIB::dot1qVlanStaticEgressPorts.10 = Hex-STRING: 98 00 00 00
Q-BRIDGE-MIB::dot1qVlanStaticUntaggedPorts.6 = Hex-STRING: 00 00 00 00
Q-BRIDGE-MIB::dot1qVlanStaticUntaggedPorts.7 = Hex-STRING: 00 00 00 00
Q-BRIDGE-MIB::dot1qVlanStaticUntaggedPorts.8 = Hex-STRING: 00 00 00 00
Q-BRIDGE-MIB::dot1qVlanStaticUntaggedPorts.9 = Hex-STRING: 00 00 00 00
Q-BRIDGE-MIB::dot1qVlanStaticUntaggedPorts.10 = Hex-STRING: 00 00 00 00
dia2:~# snmpwalk -v 2c -c dragon localhost Q-BRIDGE-MIB::dot1qPvid
Q-BRIDGE-MIB::dot1qPvid.1 = Gauge32: 8
Q-BRIDGE-MIB::dot1qPvid.2 = Gauge32: 1
Q-BRIDGE-MIB::dot1qPvid.3 = Gauge32: 1
Q-BRIDGE-MIB::dot1qPvid.4 = Gauge32: 6
Q-BRIDGE-MIB::dot1qPvid.5 = Gauge32: 6
Q-BRIDGE-MIB::dot1qPvid.6 = Gauge32: 1
Q-BRIDGE-MIB::dot1qPvid.7 = Gauge32: 1
Q-BRIDGE-MIB::dot1qPvid.8 = Gauge32: 1

dia4:~# snmpwalk -v 2c -c dragon localhost Q-BRIDGE-MIB::dot1qVlanStaticTable
Q-BRIDGE-MIB::dot1qVlanStaticEgressPorts.6 = Hex-STRING: 68 00 00 00
Q-BRIDGE-MIB::dot1qVlanStaticEgressPorts.7 = Hex-STRING: 68 00 00 00
Q-BRIDGE-MIB::dot1qVlanStaticEgressPorts.8 = Hex-STRING: 00 00 00 00
Q-BRIDGE-MIB::dot1qVlanStaticEgressPorts.9 = Hex-STRING: 68 00 00 00
Q-BRIDGE-MIB::dot1qVlanStaticEgressPorts.10 = Hex-STRING: 68 00 00 00
Q-BRIDGE-MIB::dot1qVlanStaticUntaggedPorts.6 = Hex-STRING: 00 00 00 00
Q-BRIDGE-MIB::dot1qVlanStaticUntaggedPorts.7 = Hex-STRING: 00 00 00 00
Q-BRIDGE-MIB::dot1qVlanStaticUntaggedPorts.8 = Hex-STRING: 00 00 00 00
Q-BRIDGE-MIB::dot1qVlanStaticUntaggedPorts.9 = Hex-STRING: 00 00 00 00
Q-BRIDGE-MIB::dot1qVlanStaticUntaggedPorts.10 = Hex-STRING: 00 00 00 00
dia4:~# snmpwalk -v 2c -c dragon localhost Q-BRIDGE-MIB::dot1qPvid
Q-BRIDGE-MIB::dot1qPvid.1 = Gauge32: 1
Q-BRIDGE-MIB::dot1qPvid.2 = Gauge32: 6
Q-BRIDGE-MIB::dot1qPvid.3 = Gauge32: 6
Q-BRIDGE-MIB::dot1qPvid.4 = Gauge32: 1
Q-BRIDGE-MIB::dot1qPvid.5 = Gauge32: 6
Q-BRIDGE-MIB::dot1qPvid.6 = Gauge32: 1
Q-BRIDGE-MIB::dot1qPvid.7 = Gauge32: 1
Q-BRIDGE-MIB::dot1qPvid.8 = Gauge32: 1

dia5:~# snmpwalk -v 2c -c dragon localhost Q-BRIDGE-MIB::dot1qVlanStaticTable
Q-BRIDGE-MIB::dot1qVlanStaticEgressPorts.6 = Hex-STRING: 54 00 00 00
Q-BRIDGE-MIB::dot1qVlanStaticEgressPorts.7 = Hex-STRING: 54 00 00 00
Q-BRIDGE-MIB::dot1qVlanStaticEgressPorts.8 = Hex-STRING: 54 00 00 00
Q-BRIDGE-MIB::dot1qVlanStaticEgressPorts.9 = Hex-STRING: 54 00 00 00
Q-BRIDGE-MIB::dot1qVlanStaticEgressPorts.10 = Hex-STRING: 00 00 00 00
Q-BRIDGE-MIB::dot1qVlanStaticUntaggedPorts.6 = Hex-STRING: 00 00 00 00
Q-BRIDGE-MIB::dot1qVlanStaticUntaggedPorts.7 = Hex-STRING: 00 00 00 00
Q-BRIDGE-MIB::dot1qVlanStaticUntaggedPorts.8 = Hex-STRING: 00 00 00 00
Q-BRIDGE-MIB::dot1qVlanStaticUntaggedPorts.9 = Hex-STRING: 00 00 00 00
Q-BRIDGE-MIB::dot1qVlanStaticUntaggedPorts.10 = Hex-STRING: 00 00 00 00
dia5:~# snmpwalk -v 2c -c dragon localhost Q-BRIDGE-MIB::dot1qPvid
Q-BRIDGE-MIB::dot1qPvid.1 = Gauge32: 1
Q-BRIDGE-MIB::dot1qPvid.2 = Gauge32: 6
Q-BRIDGE-MIB::dot1qPvid.3 = Gauge32: 1
Q-BRIDGE-MIB::dot1qPvid.4 = Gauge32: 6
Q-BRIDGE-MIB::dot1qPvid.5 = Gauge32: 1
Q-BRIDGE-MIB::dot1qPvid.6 = Gauge32: 6
Q-BRIDGE-MIB::dot1qPvid.7 = Gauge32: 1
Q-BRIDGE-MIB::dot1qPvid.8 = Gauge32: 1
```

mm Click Router and DRAGON Final test - SNMP tables of the VLSR 1 (dia2), VLSR 2 (dia4) and VLSR 3 (dia5) in the final phase

The result of the VLSR 2 and VLSR 3 tables were foreseeable. The new path adds to the VLAN 6 the ports 2, 4, 6 (hex value 54) for the VLSR 3 and the ports 2, 3, 5 (hex value 68) for the VLSR 2. Theoretically for the other LSPs would not be necessary to add the port 2 but it is the normal behaviour of the DRAGON software. For instance a packet in the VLSR 3 from the port 6 will be forwarded to the ports 2 and 4 but the port 2 belongs to the VLSR 1 and in its SNMP tables the corresponding VLAN of the packet has not ports and the VLSR 1 won't forward the packet.

But the new LSP has a different behaviour due to the bandwidth restrictions that it forces the LSP to go by another path. That is the reason because of the ports 4 and 5 (hex value 18) belongs to the VLAN 6 in the VLSR 1 (dia2).

After this test we can affirm that the DRAGON software is able to distribute the LSPs depending on the bandwidth restrictions of the network.

Chapter 8
Conclusions

Certain conclusions have been drawn from both the study made and the implementation of the study case, and most of them have already been stated in this document. In this last chapter, those conclusions will be re-grouped listed, serving as an indicator of whether the objectives suggested at the beginning of the thesis have been achieved.

8.1 Conclusions

First I spent some time to update my knowledge about some topics of the Ethernet networks. These topics like could be the VLANs help me later to understand and re-implement the code of the Click Router element that I had to modify.

I spent the most of time of this project studying the DRAGON software. The project DRAGON studies and develops (open source) software to enable dynamic provisioning of network resources on an interdomain basis across heterogeneous network technologies. The project tries to enable the communication between networks of different types through the GMPLS control suite.

In order to understand DRAGON firstly I had to study about the GMPLS that is an extension of MPLS that solves some problems and adds new features. I had also to study and understand the Routing Protocol, OSPF-TE, and the Signaling Protocol, RSVP-TE, used by GMPLS.

Regarding the studying of the DRAGON software, to comment that, even though there are some manuals of installation and testing, the installation and test of this software is a bit difficult and sometimes you can receive some answers or behaviours that you did not expect. Also to comment that I wrote some mails to the DRAGON team and I did not receive any answer.

The other element of this thesis is the Click Router, a new software for building flexible and configurable routers that possibly you know better than me. I say this because I have not studied deeply all the features of the Click Router. Furthermore the time restriction, the Click Router is a very wide subject and I only needed to know some points of the creation of a new elements and the communication between them.

To communicate the new Click Router elements and the DRAGON software I also had to study about the SNMP. The Simple Network Management Protocol is an application layer control that is used by network management systems to exchange management information between network devices. The understanding and use of this protocol did not take a lot of time because there are a lot of documents about it and the pre-how-to by Didier Colle was very useful.

Finally, to comment that the last part of the project, to implement and to test the Click Router with the DRAGON software, has been very interesting and to demonstrate doing different tests that the Click Router and the DRAGON software worked properly together has been very gratifying.

8.2 Final impressions

In essence, all objectives of this thesis have been accomplished. The DRAGON software has been studied deeply, including GMPLS and SNMP. Some Click Router elements have been re-implemented as the *qetherswitch*, the *qtransin* and the *qtransout* or created like all the SNMP necessary elements making Click Router capable of working with DRAGON software. The good performance of the DRAGON software and Click Router working together has been demonstrated and shown in the different tests.

However, there are some points that perhaps would have been interesting to explore it a bit more like could be the ASTB from DRAGON software and a new test to understand and prove the operation of the DRAGON software and Click Router with more than one AS and its corresponding NARBs making necessary the use of the OSPF inter domain features, but time restrictions made it impossible.

Personally I am very happy with the work done and the new learned elements because it is a project that includes many subjects. I thank the support done by the tutors, without their help this would have been impossible. On the other hand I have realized that I must improve my organization, perhaps my greater weak point.

Bibliography

The Click Modular Router

http://www.cs.berkeley.edu/~plishker/click.pdf

Massachusetts Institute of Technology

The Click Modular Router

http://www.cs.ucla.edu/~kohler/z/kohler00click-yapteaparprfotci.pdf

Massachusetts Institute of Technology

From MPLS to GMPLS

http://www.seeren.org/seeren1/content/news/docs/AdoptingAnEvolutionApproach-Mark%20Vanderhaegen.pdf

Alcatel

GMPLS: Generalized Multiprotocol Label Switching

http://www.javvin.com/protocolGMPLS.html

Javvin

Cisco Documentation - SNMP Chapter

http://www.cisco.com/univercd/cc/td/doc/cisintwk/ito_doc/snmp.pdf

Cisco Systems

Net-SNMP

http://net-snmp.sourceforge.net/

Net-SNMP

Sara Project

http://staff.science.uva.nl/~delaat/snb-2005-2006/p38/report.pdf

Universiteit van Amstrdam

Blueprint for VLSR in CHEETAH Network Deployment

http://www-ee.engr.ccny.cuny.edu/wwwb/web/ibrahim/design_documents/Reference%202-Blueprint-VLSR-Design-ccny-v4_3.pdf

Cheetah Project

Appendixes

8.3 Appendix A SNMP configuration file

```
###############################################################################
#
# snmpd.conf
#
#   - created by the snmpconf configuration program
#
###############################################################################
# SECTION: System Information Setup
#
#   This section defines some of the information reported in
#   the "system" mib group in the mibII tree.

syslocation  IBBT

sysdescr Ethernet Switch
#sysdescr DES-3526 Fast-Ethernet Switch #Description of the DLINK switch
#sysdescr Ether-Raptor #Description of the RAPTOR switch

syscontact  yourmail@gmail.com

###############################################################################
# SECTION: Access Control Setup
#
#   This section defines who is allowed to talk to your running
#   snmp agent.

# rwuser: a SNMPv3 read-write user
#   arguments:  user [noauth|auth|priv] [restriction_oid]

rwuser  "dragon" noauth

# rouser: a SNMPv3 read-only user
#   arguments:  user [noauth|auth|priv] [restriction_oid]

# rwcommunity: a SNMPv1/SNMPv2c read-write access community name
```

```
#   arguments:  community [default|hostname|network/bits] [oid]

rwcommunity  dragon

master agentx
```

8.4 Appendix B Click script for Test 2,3,4 and Final Test

```
// Valentin Carela 04/07/2007

elementclass EightPortVLANBridge { $a |
      // suppressors used for blocking incoming and outgoing ports
      in::Suppressor;
      out::Suppressor;

      // VLAN switch with support for 10 VLANs
      s :: QEtherSwitch(10);

      stp:: QRSTP($a,in,out, s,2);
      agentx :: SnmpSubagentElement();
      encap:: QTransIn(1);
      decap:: QTransOut(1);
      vlantable :: dot1qVlanStaticTable_element(agentx, s, decap);
      porttable :: dot1qPortVlanTable_element(agentx, encap);


      // Classifier for (spanning tree PDUs, normal traffic)
      //c1, c2, c3 :: Classifier(0/0180C2000000, -);
      c1, c2, c3, c4, c5, c6, c7, c8:: Classifier(0/0180C2000000, -);

      // output queues
      q1,q2,q3,q4,q5,q6,q7,q8::Queue(1000);

      // CONNECTIONS
      // inputs to classifiers
      input[0]->c1;
      input[1]->c2;
      input[2]->c3;
      input[3]->c4;
      input[4]->c5;
      input[5]->c6;
      input[6]->c7;
      input[7]->c8;


      // 1th classifier branch (rstp bpdu) to spanning tree module
```

```
	c1[0]->[0]stp;
	c2[0]->[1]stp;
	c3[0]->[2]stp;
	c4[0]->[3]stp;
	c5[0]->[4]stp;
	c6[0]->[5]stp;
	c7[0]->[6]stp;
	c8[0]->[7]stp;

	// 2th classifier branch (normal traffic) to suppressor for blocking
on incoming ports
	c1[1]->[0]in;
	c2[1]->[1]in;
	c3[1]->[2]in;
	c4[1]->[3]in;
	c5[1]->[4]in;
	c6[1]->[5]in;
	c7[1]->[6]in;
	c8[1]->[7]in;

	// bridging (switch mode) after blocking
	in[0]->[0]encap;
	in[1]->[1]encap;
	in[2]->[2]encap;
	in[3]->[3]encap;
	in[4]->[4]encap;
	in[5]->[5]encap;
	in[6]->[6]encap;
	in[7]->[7]encap;

	encap[0]->[0]s;
	encap[1]->[1]s;
	encap[2]->[2]s;
	encap[3]->[3]s;
	encap[4]->[4]s;
	encap[5]->[5]s;
	encap[6]->[6]s;
	encap[7]->[7]s;

	s[0]->[0]decap;
	s[1]->[1]decap;
```

```
s[2]->[2]decap;
s[3]->[3]decap;
s[4]->[4]decap;
s[5]->[5]decap;
s[6]->[6]decap;
s[7]->[7]decap;

// blocking after bridging
decap[0]->[0]out;
decap[1]->[1]out;
decap[2]->[2]out;
decap[3]->[3]out;
decap[4]->[4]out;
decap[5]->[5]out;
decap[6]->[6]out;
decap[7]->[7]out;

// stp messages (bpdus) queueing on output queues
stp[0]->q1;
stp[1]->q2;
stp[2]->q3;
stp[3]->q4;
stp[4]->q5;
stp[5]->q6;
stp[6]->q7;
stp[7]->q8;

// forwarding bridged traffic to output queues
out[0]->q1;
out[1]->q2;
out[2]->q3;
out[3]->q4;
out[4]->q5;
out[5]->q6;
out[6]->q7;
out[7]->q8;

// output queues to bridge output
q1->[0]output;
q2->[1]output;
q3->[2]output;
```

```
      q4->[3]output;
      q5->[4]output;
      q6->[5]output;
      q7->[6]output;
      q8->[7]output;

};

elementclass link {$b |
      input->Unqueue(10000)->ToDump($b)->output;
};

bridgeA::EightPortVLANBridge(00:90:aa:7c:51:9d);

Idle->[0]bridgeA;
Idle->[1]bridgeA;
Idle->[2]bridgeA;
Idle->[3]bridgeA;
Idle->[4]bridgeA;
Idle->[5]bridgeA;
Idle->[6]bridgeA;
Idle->[7]bridgeA;

bridgeA[0]->Idle;
bridgeA[1]->Idle;
bridgeA[2]->Idle;
bridgeA[3]->Idle;
bridgeA[4]->Idle;
bridgeA[5]->Idle;
bridgeA[6]->Idle;
bridgeA[7]->Idle;
```