Multi-region GMPLS control and data plane integration

PONTUS SKÖLDSTRÖM

KTH Information and Communication Technology

Master of Science Thesis
Stockholm, Sweden 2008

COS/CCS 2008-16

Multi-region GMPLS control and data plane integration

in Partial Fulfillment of the Requirements for the Degree Master of Science in Engineering

PONTUS SKÖLDSTRÖM

Master's Thesis at the School of Information and Communication Technology
Supervisor: Annikki Welin
Examiner: Prof. Gerald Q. Maguire Jr.

Abstract

GMPLS is a still developing protocol family which is indented to assume the role of a control plane in transport networks. GMPLS is designed to provide traffic engineering in transport networks composed of different network technologies such as wavelength switched optical networks, Ethernet networks, point-to-point microwave links, etc. Integrating the different network technologies while using label switched paths to provide traffic engineering poses a challenge.

The purpose of integrating multiple technologies under a single GMPLS control plane is to enable rapid service provisioning and efficient traffic engineering. Traffic engineering in networks provides two primary advantages, network resource utilization optimization and the ability to provide Quality of Service. Utilizing network resources more efficiently translates to lower expenditures for the network provider. Quality of Service can be used to provide the customer with for example guaranteed minimum bandwidth packet services.

Specifically this thesis focused on the problems of signaling and establishing Forward Adjacency Label Switched Paths (FA-LSPs), and on a experimental method of connecting different network technologies. A testbed integrating an Ethernet network and a wave length division multiplexing network was used to show that the proposed solutions can work in practice.

Sammanfattning

GMPLS består av en samling protokoll under utveckling, de är tänkta att anta rollen som kontrollplan i transportnätverk. GMPLS är designat för att tillhandahålla trafikplanering i transportnätverk bestående av flera olika nätverksteknologier såsom Ethernet, våglängds switchande nätverk m.fl. Integration av dessa olika nätverksteknologier under ett gemensamt kontrollplan och uppsättning av ”label switched paths” i dataplanet är en utmaning.

Syftet med att integrera multipla teknologier under ett ensamt GMPLS kontroll plan är att snabbt kunna tillhandahålla tjänster över nätverket samt möjliggöra advancerad trafikplanering. Trafikplanering i nätverk ger två stora fördelar, optimering av utnyttjandet av nätverksresurser samt ökade möjligheter att erbjuda ”Quality of Service” till kunder. Bättre utnyttjande av nätverksresurser innebär lägre kostnader för nätverksleverantören medans ”Quality of Service” kan ge kunden t.ex. en garanterad bandbredd.

Specifikt fokuserar denna avhandling på problemen med att signalera och etablera ”Forwarding Adjaceny Label Switched Paths” samt en experimentell metod som båda sammankopplar olika typer av nätverk. En testbed bestående av ett Ethernet nätverk samt ett optiskt våglängdsswitchande nätverk användes för att visa att lösningarna kan fungera i praktiken.

Acknowledgments

I would like to thank the all people that has helped and guided me during the completion of this thesis. These include not only my supervisor and examiner but several others at Ericsson and Acreo.

Many thanks to Annikki Welin (Ericsson), András Kern (Ericsson), Anders Gavler (Acreo), Roland Elverljung (Acreo), and Lars P. Fink (Dataunit). These friendly persons answered a lot of questions, gave feedback and ideas, provided literature, taught interesting Bash and Emacs tricks, discussed implementation issues, and shared lots of cups of coffee with me. Without all your help, support, and guidance this thesis would never have been completed. Again, many thanks to all of you!

I am also very thankful for the amazingly quick and thorough reviews, ideas and suggestions done by Prof. Maguire.

And last but not least, thanks to my family and friends for your support and encouragement.

Contents

List of Figures

List of Tables

Nomenclature

AS	Autonomous System
BGP	Border Gateway Protocol
CWDM	Coarse Wavelength Division Multiplexing
DWDM	Dense Wavelength Division Multiplexing
EGP	Exterior Gateway Protocol
ELC	Explicit Label Control
ERO	Explicit Route Object
FTN	FEC-to-NHLFE
GRE	Generic Routing Encapsulation
HDLC	High-Level Data Link Control
IEEE	Institute of Electrical and Electronics Engineers
IETF	Internet Engineering Task Force
IGP	Interior Gateway Protocol
ILM	Incoming Label Map
IP-IP	IP-within-IP Encapsulation Protocol
IS-IS	Intermediate System to Intermediate System
LER	Label Edge Router
LMP	Link Management Protocol
LSA	Link State Advertisement
MAC	Media Access Control
MLCP	Multi-Layer Control Plane

NHLFE	Next Hop Label Forwarding Entry
OXC	Optical Cross-connect
ROADM	Reconfigurable Optical Add/Drop Multiplexer
RSVP	Resource ReserVation Protocol
SDH	Synchronous Digital Hierarchy
SONET	Synchronous Optical Networking
SPF	Shortest Path First
STP	Spanning Tree Protocol
TLV	Type-Length-Value
VID	VLAN Identifier

Chapter 1

Introduction

One of the "growing pains" of the Internet in the early and mid-1990s was the computational cost of performing route look-ups for Internet Protocol (IP) packets. This look-up was becoming a problem when trying to keep up with the increasing speed of data links. For each packet entering an IP router a Longest Matching Prefix look-up based on this packet's destination address has to be performed in order to calculate its next hop. In an effort to remedy this Multi-Protocol Label Switching (MPLS) was developed. In MPLS networks only the ingress router (i.e. the edge router in the network) has to perform such a look-up. It then assigns the packet to a Forwarding Equivalence Class based on its destination and other characteristics. This Forwarding Equivalence Class is represented by a label which is added to the packet and subsequently used by Label Switching Routers (LSRs) in the network to forward the packet. A LSR that receives a labeled packet needs only to perform a table look-up, not a longest prefix match.

Independent development in techniques for IP forwarding increased the performance of the forwarding look-up so that it could cope with the increasing data link speeds and the primary motivation behind the development of MPLS was no longer relevant. However, MPLS still offered other features such as its ability to be used for Traffic Engineering in which more complicated routing schemes than Shortest Path First can be used to guarantee Quality of Service for some traffic, to optimize the utilization of network equipment, and to prevent congestion for prioritized traffic.

The desire to perform automated service provisioning and traffic engineering on network technologies other than packet switched networks has led to the development of Generalized MPLS (GMPLS). GMPLS extends the concept of label switching to the link layer (ISO OSI layer 2), Time division multiplexing (TDM), Wavelength Division Multiplexing (WDM), and Fiber switching network technologies. GMPLS is a relativity young protocol which still has many technical problems that have to be solved in order to produce a functional solution. Many of these problems relate to the implementation of a Multi-Layer Control Plane and integrating different network technologies (called "regions" in GMPLS terminology). A multi-layered control plane allows signaling and routing decisions to be made for a network consisting of multiple (potentially different technology) regions, allowing for more efficient and automated traffic engineering than would be possible if each region did its own traffic engineering. The hope is that the ability to do efficient traffic engineering over *multiple regions* would lower costs for constructing, extending, and operating transport networks.

This thesis will focus on the problems of signaling and establishing of Forward Adjacency Label Switched Paths (FA-LSPs). A testbed integrating an Ethernet and WDM region will be used as an example to show that the suggested solution can work in practice.

1.1 Objectives

The goal of this thesis is to investigate the possibilities of creating a Multi-region GMPLS network containing at least two regions of networking technology (in this case we will use specifically Ethernet as the Layer 2 network and a optical WDM network as Layer 1). In order to achieve this the primary objectives are:

- Implement and verify that an RSVP-TE implementation has the ability to create and remove regular LSPs and FA-LSPs (see sections 2.1.1 and 3.3.2 for a clarification of these terms).
- Implement and verify conversion of LSPs at border nodes of the Ethernet tagged GMPLS traffic to and from WDM GMPLS traffic.

These can be divided into several sub-objectives:

- Implement a virtual test bed with nodes acting as Ethernet, WDM, and border nodes.
- Verify that the RSVP-TE implementation has the ability to create and remove LSPs within each region.
- Implement a region border node that can use triggered signaling to create (and remove) a FA-LSP if appropriate.

This work should result in an operational multi-region GMPLS network with the ability to create and remove LSPs over a combined Ethernet and WDM network. This work is part of a project called ”Multi-Layer Control Plane” (MLCP) which is a joint research project between Ericsson[1] and Acreo[2].

1.2 Thesis Outline

Chapter 2 gives an introduction to GMPLS networks and discusses the reasoning behind the development of multi-region control planes. Chapter 3 introduces

[1]http://www.ericsson.com
[2]http://www.acreo.se

the network technologies that are considered in this thesis and the proposed solutions to handle the control of region border nodes. In Chapter 4 the software implementation is described. Chapter 5 describes the testbed used and verification of the functionality of the implementation is performed. In Chapter 6 the solution is evaluated. In Chapter 7.1 conclusions drawn and ideas for future work are presented.

Chapter 2

Introduction to GMPLS

Generalized Multi-Protocol Label Switching (GMPLS) is a rapidly developing collection of protocols, of which some parts are still being standardized. The Internet Engineering Task Force (IETF) is further extending the routing and signaling protocols associated with MPLS-TE in order to support the specific needs of the technologies that are desirable to support GMPLS on.

2.1 Multi-Protocol Label Switching (MPLS)

To understand some of the ideas underlying GMPLS it is useful to know the basics of MPLS. The primary motivation for developing MPLS was to reduce the cost of making forwarding decisions in Layer 3 (primarily IP) networks, which must be made by every router on a path between the source and destination. In MPLS the destination of the incoming IP packet is only examined at the ingress router, which is the first router on the border of the MPLS network. This is known as a Label Edge Router (LER). The LER assigns the packet to a particular Forward Equivalence Class based not only on the Layer 3 header but additional information such as which port it has arrived on may be taken into account. If a path does not already exist for this Forward Equivalence Class, the router signals all the MPLS routers (known as Label Switching Routers, LSRs) on the path to the egress router (the other LER in the path to the destination). The signaling creates a fixed path for all traffic assigned to this particular Forward Equivalence Class. For a simple example of an MPLS network see figure 2.1.

2.1.1 Pushing, Popping, Swapping, and Stacking Labels

Attaching a label to a packet is called “pushing” a label, the reverse operation which removes the label is called “popping” the label. Labels can be ”swapped” by first popping a label - examining this label, making a forwarding decision, then pushing a new label on the packet. It is also possible to push more than one label onto a packet, known as “stacking” labels. Each label, or entry in the label stack, consists

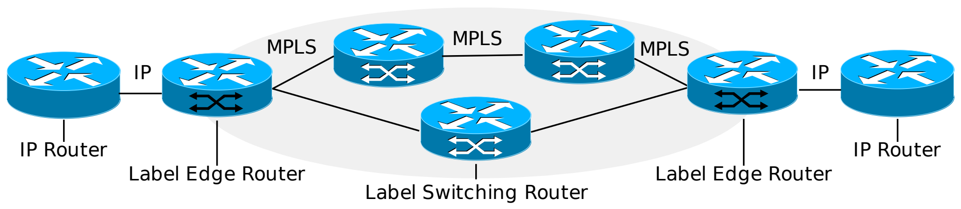

Figure 2.1: Example of an MPLS network showing IP routers, Label Edge Routers, and Label Switching Routers. Traffic between the IP and Edge routers are forwarded based on the IP header, within the MPLS network packets are switched on labels.

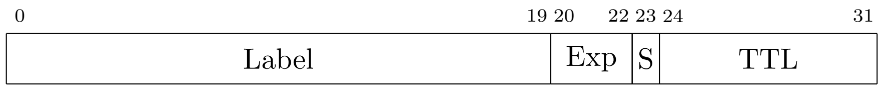

Figure 2.2: MPLS label stack entry. Exp is a 3-bit field reserved for experimental use. S is the Bottom of Stack bit which is set on the first label pushed to the label stack [2].

of four fields; the 20-bit Label field, a 3 bit Experimental field, a 1-bit Bottom of Stack field, and a 8-bit Time-to-live field (see figure 2.2)[2].

Using a signaling protocol each pair of routers agrees on a label to use on the link between them for a particular Forward Equivalence Class. The mapping of an incoming label to an outgoing label creates a Label Switched Path (LSP) defined by the label used at the ingress. It should be noted that the labels can be link-local or node-local, which is controlled by a label space which is assigned to the different network interfaces. If all interfaces on a router share the same label space the labels are node-local, a labeled packet will be forwarded to the same destination regardless of which interface it enters. If on the other hand all interfaces are assigned to different label spaces, the labels are link-local. Interfaces may also be grouped into label spaces, which may be useful if for example two nodes have several parallel links to each other and wish to perform load-balancing over these links.

Conceptually each label space has a three different maps for keeping track of the labels and how to forward packets. These are the Next Hop Label Forwarding Entry (NHLFE) map, the Incoming Label Map (ILM), and the FEC-to-NHLFE (FTN) map [3]. The NHLFE tables primary content is the next hop of the packet and what kind of operation should be performed on the label stack of the packet, for example a label swap operation. The ILM is a mapping between incoming labels and NHLFE entries and is used when a packet enters the LSR with a label already attached. The FTN performs the same mapping as the ILM, but for unlabeled packets, the mapping can be between for example an IPv4 destination address and a NHLFE, see figure 2.3 for an example of how these tables might look. Based on this mapping the LSRs pushes, pops, and swaps labels and forwards a packet to its next hop. Based upon this hop-to-hop forwarding and swapping of labels; packets

Input Label Map	
Incoming Label	NHLFE
544	1
325	2

(a)

FEC-to-NHLFE	
Destination Address	NHLFE
172.16.100.1	3
172.16.200.1	4

(b)

NHLFE	
Operation	Next Hop
Swap(123)	172.16.0.1
Swap(44)	172.16.0.2
Push(321)	172.16.0.1
Push(77)	172.16.0.2

(c)

Figure 2.3: Example of MPLS tables for one label space with IPv4 addresses. ILM with two entries (a). FTN with two entries (b). NHLFE with four entries (c).

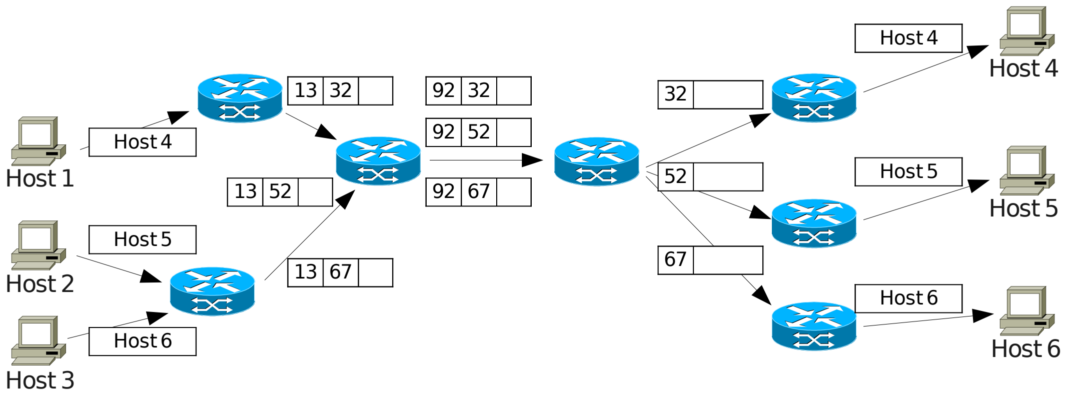

Figure 2.4: Label swapping and stacking.

are sent through the network on the path.

By stacking several labels on a packet LSPs may be tunneled through other LSPs. Stacking improves scaling and path setup time since fewer new paths have to be signaled and maintained (see figure 2.4 for an example of MPLS forwarding with stacking).

2.1.2 MPLS Routing and Signaling

In order to set up a LSP a path between the ingress and egress LERs must somehow be calculated and label mappings signalled. This can be done by using a

0 3	4 7	8 15	16 31
Version	Flags	Msg Type	RSVP Checksum
Send_TTL		(Reserved)	RSVP Length

Figure 2.5: Common RSVP header [4].

routing protocol such as Open Shortest Path First (OSPF) or Intermediate System to Intermediate System (IS-IS) to distribute information about network topology throughout the network. The topology information can then be used to create a routing database from which appropriate paths can be calculated. After the ingress LER has calculated a path the label mapping setup is handled by the signaling protocol. The IETF does not mandate a specific signaling protocol for MPLS. This has led to the use of two different protocols: Resource ReSerVation Protocol (RSVP) and the Constraint based Label Distribution Protocol (CR-LDP). In this thesis only RSVP and more specifically the extended version called the Resource ReSerVation Protocol - Traffic Engineering (RSVP-TE) will be considered.

RSVP was originally designed to allocate resources for the Integrated Service extension to the Internet Protocol [4]. In order to allocate resources on the path between two nodes the RSVP Path packet would be sent to IP destination address of the end node. This would allow the packet to be routed using regular routing protocols and pass through routers that do not support RSVP. Those routers that support RSVP would inspect every RSVP packet it forwards, create a "soft state" with information about for example the previous RSVP capable router (this information is carried in the Previous Hop object), make alterations to the packet, and forward to the next hop. Referring to the state information as "soft" means that the state will be removed after a specific time unless it is refreshed (and that loss of this state will not result in the loss of the ability to correctly forward packet, but rather simply the loss of the RSVP based path forwarding.). Upon reaching the end node a RSVP Resv packet will be propagated back, hop-by-hop, using the same path as stored (the soft state information). When each RSVP capable router receives a Resv packet the reservation is confirmed and installed. If any errors occur when processing either the Path or Resv packet a PathErr or ResvErr packet is transmitted in order to alert the sender.

All RSVP packets share a common header, followed by a number of Type-Length-Value (TLV) fields called objects which may have additional TLV field within (these are called sub-objects). The TLV fields are actually in the Length-Type-Value order[1] with the Type field split into two separate field, the Class field and the C-Type field. The common header carries the type of the message, total length of the message, and information for defragmentation (see figure 2.5). Any additional information is carried in objects or sub-objects (see figure 2.6).

[1]I will refer to them as TLVs or objects despite this order of the fields.

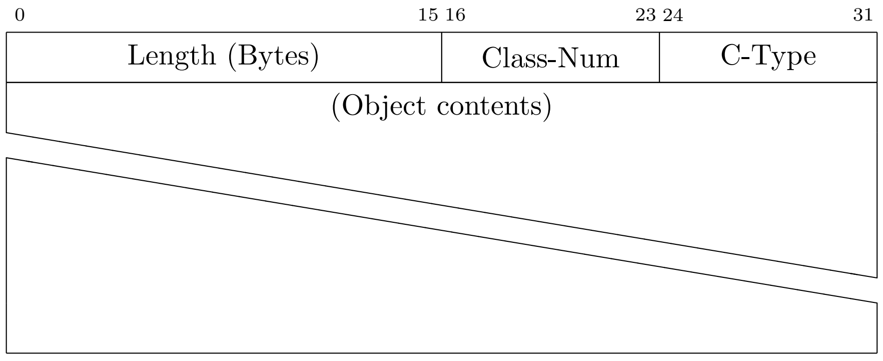

Figure 2.6: The RSVP Object/Sub-Object header [4].

To allocate labels using RSVP the Path message is sent to the egress LER and each router on the way checks that requested resources are available. The Path message does not create a label mapping, rather labels are allocated when the LSR receives a Resv message. The downstream LSR, i.e. the one closer to the egress LSR, allocates an available label and sends this label in the Resv message in a LABEL object. If any LSRs on the path do not have sufficient resources for the LSP, then this LSR must send a PathErr message which propagates to the ingress LSR, which then aborts the path setup, removing any previously reserved states (belonging to this specific LSP) on the way. Similarly, errors that occur when processing Resv packets are reported with a ResvError. Note that this procedure only establishes labels for uni-directional communication, for bi-directional communication in MPLS another LSP must be created from the egress to the ingress LSR. The actual path traversed by the Path message may be explicit or hop by hop routed. A hop by hop routed Path message travels through the network according to the Layer 3 forwarding tables, like any other Layer 3 packet. Explicitly routed Path messages travels across the network following a path that has been specified by the ingress LER.

2.1.3 Traffic engineering

Once regular IP routing had sufficient speed to handle the maximum rate of the underlying communication technology the main reason for developing MPLS was lost as label switching no longer offered lower switching delay. However, MPLS turned out to be easily extended to support traffic engineering in packet networks. Traffic engineering is an familiar concept for those who plan road construction, they are concerned with getting the best flow of traffic through congested roads while minimizing the risk of collisions. For example, what effects do junctions have, how many lanes should the new road have in order to avoid congestion? How long should the traffic light be red and is there any way to allow prioritized traffic such as ambulances to get priority at intersections without causing problems for the regular traffic?

If one imagines data packets as cars, network links as roads, and switches and routers as road junctions the same issues more or less apply to traffic engineering in packet networks. Congestion on roads causes delays and accidents, in packet networks it causes delays and packet loss. IP networks are generally Shortest Path First routed – which is a great scheme as long as the network is unused, packets reach their destination quickly while using the minimum amount of network resources possible (assuming that all links have basically the same cost – thus the goal is to minimize the number of hops). However, if the network has got a lot of traffic going through it the shortest path may become congested and routers will start to drop the packets they cannot forward. Even if there are routes that avoid the congested routers, traffic will still be routed over the shortest path leaving alternative paths underutilized. There are ways to manage congestion, for example DiffServ or IntServ can be used to prioritize traffic so that it is not dropped and TCP has mechanisms for lowering transmission rates when congestion is detected [5][6][7]. These are good for improving the reliability and quality of traffic delivery, but do not increase the amount of traffic which can go from one point to another in an Shortest Path First network [8].

In order to increase the amount of traffic the network can deliver some parts of the traffic has to be delivered via these alternate paths. Equal-cost multi-path routing is another routing scheme which under some circumstances is able to make use of these alternate paths. When an Shortest Path First router calculates routes and is confronted with more than one path to a destination with the same length or cost it chooses one of them, equal-cost multi-path instead stores both paths. It can then choose to either use an alternative path when the first one becomes congested or to load balance the different paths. The road analogy for this would be two parallel roads of the same lengths to a destination. The second road would either be opened when the first becomes congested or one would direct every other car down the second road. However, if these equal-cost paths converge at one point and the congestion occurs on a link further down the parallelism might not make a difference. The problem of not using alternate paths still exists as well, a path which costs one unit more would not be used with equal-cost multi-path.

Better traffic engineering can be achieved if one knows all the paths and links of the network and the current utilization of them *and* then somehow directs traffic based on that information. And here is where MPLS has found a niche, if one extends it to support an explicit route instead of only a Shortest Path First route it can be used to direct traffic over predetermined links chosen for traffic engineering reasons. By extending the MPLS routing protocols to carry information such as the available bandwidth of links and extending signaling protocols to create an LSP on an explicit path one can support advanced traffic engineering. These extensions of the original MPLS protocols are called MPLS TE [9].

2.2 Generalized MPLS

The origin of GMPLS was a wish to automate previously manual systems for provisioning services in transport networks. The manual planning and configuration of the services could take a long time and negatively affect other services [10]. As networks grew in size the demand for an automated way to deal with this grew. As optical networks became more popular, ways to deal specifically with lambda switching networks (see section 3.1.2) was developed. It was realized that distributing label mappings {incoming label, outgoing label} and lambda mappings {incoming lambda, outgoing lambda} was essentially the same problem and could utilize a similar solution. One proposal which borrowed heavily from MPLS was called Multi-protocol Lambda Switching (MPλS), however it never became an official standard [11].

The generalization of MPLS was expanded to include switching time slots in TDM systems, switching between optical fibers, and switching layer 2 frames – as these are more or less the same thing. Why not create a unified standard for managing all these technologies since they were essentially the same problem? Another advantage of creating a general protocol it that networks of different types can now interoperate easily. That means that end-to-end services could be set up and maintained over networks composed of different networking technology without any complicated interoperability layers. The ability to distribute traffic engineering information between technologies may also lead to more efficient provisioning of services, for example if there is a problem within one segment this can be taken into account before any signaling is performed. Larger networks also lead to a larger amount of routing traffic, hence GMPLS can be deployed in several models which balance control plane overhead against TE efficiency (see section 2.2.6 for more on this).

The effort needed to calculate paths grows with the number of nodes involved and is likely to be slower when performed for many different kinds of technologies at once – due to different constraints on each of the different technologies. Path calculation is only necessary when initially provisioning the service so this may not be a large problem, however if many paths fail simultaneously this could be a problem since many new routes would be requested at once. This could be mitigated by pre-calculating failure paths, perhaps completely disjoint from the original path. If the original path fails one of the pre-calculated alternatives can be signalled without any need for additional calculation. The task of performing route calculation, which may be computationally expensive, can either be centralized or distributed or somewhere in between, this is discussed in section 2.2.10.

2.2.1 Control and Data plane

The nature of some of the technologies intended to be used with GMPLS requires that the control plane and data plane must be separated. In packet switching systems control data can be sent over the same channels as traffic, as routers and

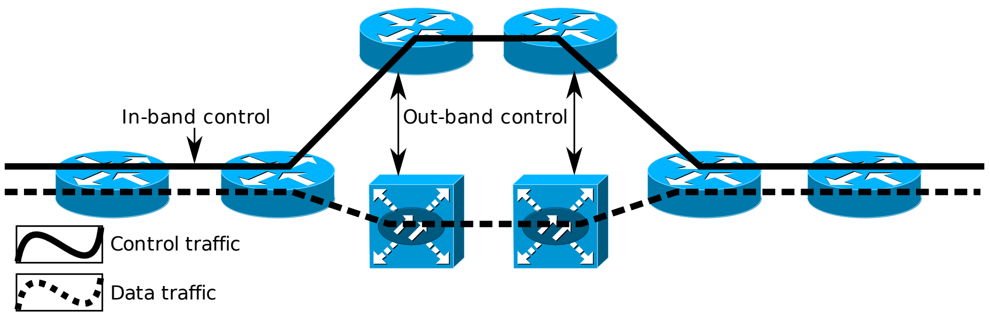

Figure 2.7: Example of in-band and out-band control. Data traffic (shown as a dashed line) and control traffic (shown as a filled line - above the data traffic) going through packet routers and splitting up when going over optical network components connected to packet routers.

switches read the header of each packet and if a packet is found to be destined to itself it can pass it along to the control system within, this is called in-band control. This is not possible for some devices (e.g. optical switches, see section 3.1.2) since they do not examine the data that they transport, therefore control data must be communicated through a separate control channel (see figure 2.7), this is called out-band control. In WDM systems this can be done either by having a separate connection, for example an Ethernet link connected to the switch or, if the switch supports it, a data channel reserved for control data.

Separating control and data plane also adds resilience to the network, the data plane can continue to function even if the control plane is broken (although no new connections can be made) and vice versa. However, detecting errors in the data plane may be harder since you cannot detect errors by loss of control traffic as in networks with in-band control data. Errors such as “loss of light” in some optical networks can only be detected at the end of the light path and thus finding the faulty link or device may not be a trivial task. To aid this process the Link Management Protocol (LMP) and LMP-WDM for optical systems may be used which are covered briefly in section 2.2.9. The Link Management Protocol helps the switch to discover its links and their capabilities, as well as assisting in the detection and isolation of errors on the links.

2.2.2 Interface Switching Type

Five kinds of interface switching types have been defined: Packet switching capable (PSC), Layer 2 switching capable (L2SC), Time division switching capable (TDM), Lambda switching capable (LSC), and Fiber switching capable (FSC). The values used to represent these switching types in the signaling and routing protocols can be seen in table 2.1, how they are used is described in sections 3.3.4 and 2.2.8. A network interface on a node is said to have a Interface Switching Capability (ISC)

Table 2.1: Interface Switching Types and their corresponding values

Switching type	Value
PSC-1	1
PSC-2	2
PSC-3	3
PSC-4	4
L2SC	51
TDM	100
LSC	150
FSC	200

depending on what information the node can use to switch data that arrives on that interface. For example if data arriving on an interface can be switched based on a Layer 2 header, such as an Ethernet frame header, then that interface is L2SC capable. If the interface can separate different colors of light from an optical fiber and the node can direct them into different optical fibers, then the interface is of type LSC. If the interface can switch packets based on an MPLS label, it is of type PSC and so on. The four different PSC types can be used with MPLS label stacking to tunnel LSPs within LSPs (see section 2.1.1)[12]. All interfaces on a node do not have to have the same switching type. Additionally an interface may actually have several switching types. For example an interface may be able to switch on both an MPLS label and a Ethernet header, then it would then be both PSC and L2SC capable. Nodes that have interfaces which support multiple switching types (per interface) are called ”hybrid” nodes whereas those with only a single switching type per interface are called ”plain” or ”simplex” nodes [13]. Hybrid nodes are not considered in this thesis.

2.2.3 Regions and Layers

There is some confusion regarding the nomenclature surrounding GMPLS, the terms region and layer need to be clarified. In this thesis the terms will be used as they are defined in RFC 4397 which defines a layer as “a set of [data plane] resources of the same type that could be used for establishing a connection or used for connectionless data delivery.” [14]. Using this definition a multi-layer data plane device could for example be an Ethernet interface capable of sending in 100 Mbit/s as well as 1 Gbit/s. A “TE region” is defined as “a set of one or more layers that are associated with the same type of data plane technology”, such as MPLS, ATM, WDM, etc. The term “TE region”, “LSP region”, and just “region” can be used interchangeably. Multi-layer networks are not considered in this thesis; we will instead only focus on multi-region networks where each region consists of only a single layer. For an example of a multi-region GMPLS network see figure 2.8.

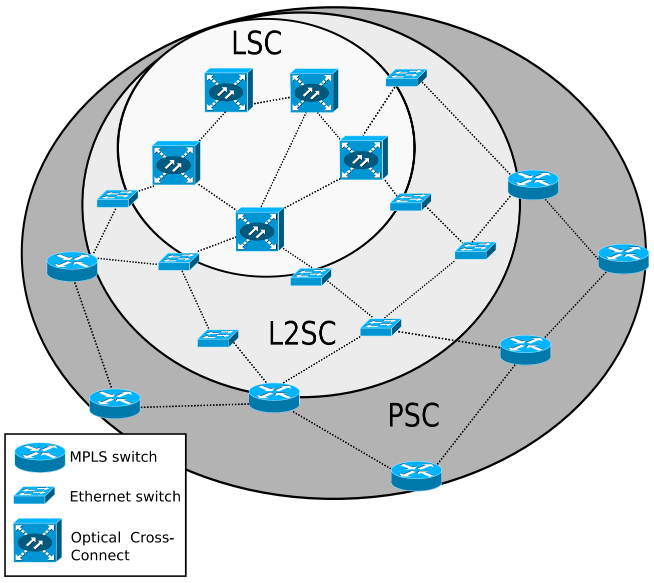

Figure 2.8: Multi-region network with three regions, LSC, L2SC, and PSC.

2.2.4 Labels

In MPLS labels describe a Forward Equivalence Class; this is also true in GMPLS, but a label may also describe a physical resource, for example a wavelength or a time slot. When the labels describe a physical resource the label is often not actually carried with the data (as for example an MPLS shim label or Ethernet label). Instead the label only exists in the control plane where it describes how the switching matrix of the particular switch should be set up. In that case the label may be referred to as a ”virtual” label. The need to support different switching techniques means that the label format has to be generalized in order to fit the switching capability and encodings used by different switches (e.g. ATM and Ethernet are both L2SC, but need different label formats since they switch on different kinds of headers). The collection of these specific label formats are called “generalized labels”, and each of them has to contain the information necessary for the specific technology to control its cross-connect in order to create a path. As in MPLS the labels may be link-local, but in some networks (e.g. WDM, Ethernet) the same label may have to be used through the whole network segment (so called end-to-end labels). The generalized label itself is has a variable length; in most cases it is a 32-bit value, but longer labels are supported. Shorter labels such as MPLS or Frame Relay labels are right justified within a 32-bit value [15].

The different generalized labels have no field to indicate what type of label it contains, therefore one cannot easily conclude how to interpret a label. Finding out how to interpret a generalized label requires information about the TE-link to which the label should be allocated. By examining the Switching Capability of the far- and near-end of the TE-link one can determine the label type in some cases. In other cases the Encoding taken either from the Traffic Engineering Database (see section 2.2.8) or the Label Request (see section 2.2.7) might be necessary. If for example the switching capabilities of the TE-link are [PSC, PSC] the label is an MPLS label; if the switching capabilities are [PSC, LSC], then the label indicates a lambda [12].

2.2.5 Interface Identification

The data plane consists of links between data interfaces, which may have different capabilities (such as a switching capability, adaption capability, and termination capability). These different capabilities and other constraints (such as available bandwidth) are taken into account when calculating a path. When signaling an LSP these interfaces have to be identified in some way in the Explicit Route Object (ERO) of the RSVP Path message (see section 2.2.7), this identification can be either numbered or unnumbered. A numbered interface has a public or private IP address assigned to it, this IP address can then be used to unambiguously identify the link in RSVP-TE and OSPF-TE. However, for other interfaces, such as Ethernet or WDM interfaces that cannot terminate traffic, assigning an IP address to identify the interface may be a waste of IP address space (if the addresses are public) and assigning this interface an IP address makes little sense if it cannot receive IP traffic. When assigning private IP addresses to you need to keep track of all the addresses since they have to be unique within the network, this is error prone and may requires a significant amount of work. Instead of assigning IP addresses one can use so called unnumbered identifiers. The unnumbered identifier consists of the LSR's Router ID (which usually is a network unique loop back IPv4 or IPv6 address, in order to be reachable through any of its interfaces) combined with a node-unique 32-bit identifier. This combination is a network unique identifier that can be generated by any node without the need for some kind of address management scheme. One can imagine other identifiers that could be used, for example MAC addresses which are already used by some networking technologies. However, adding technology specific identifiers complicates the protocols and implementations, especially when there already are generic identifiers defined.

2.2.6 Network Architecture

GMPLS can be deployed in three distinct architectural models:

- The *peer* or *unified service* model
- The *overlay* or *domain service* model
- The *hybrid* or *augmented* model

The models primarily differ in how much information is distributed between domains and how services are set up. In the peer model all LSRs in the network

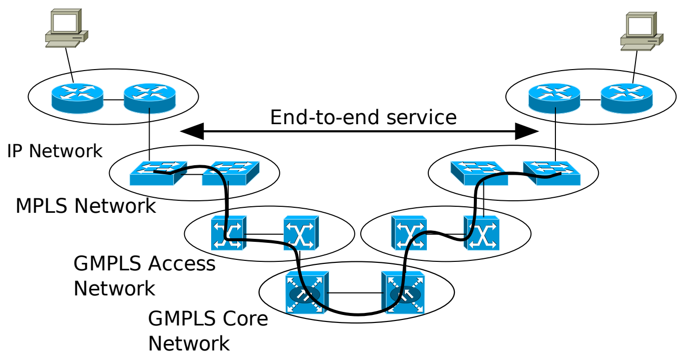

Figure 2.9: The peer model. A common control plane is used to route and signal a path across a network of different technologies and realize an end-to-end service.

have full visibility of the entire GMPLS network, each node knows the complete topology of the network and what resources are available (see figure 2.9). A single common signaling protocol is used through the network which allows clients at the border to set up end-to-end services. However, this model has scaling problems as the number of nodes grow, but it is also the most flexible model in terms of service establishment since a full view of the network is kept and paths are fully traffic engineered end-to-end.
In the overlay model (see figure 2.10) each LSR has a limited visibility – either to the network it is a part of (i.e. with the same switching capability) or to its administrative domain. This means that no TE information is shared with a neighbor of either a different technology (e.g. between MPLS and Ethernet switched networks) or of a different administrator (e.g. between two Internet Service Providers). Instead a User-to-Network Interface (UNI) is placed at the border of network domains, there the client network asks the interface for service and the ”server” network is free to deliver the service in its own fashion. This allows for greater separation of networks which are free to operate independently of each other. In this approach service providers can operate their own network without allowing any routing information or signaling from partners or competitors into their own control plane and similarly not permitting any of their routing or signaling information to leave their network. Several protocols for communicating via a UNI have been developed, for example the Optical Internetworking Forum’s UNI implementation which sends the service request as a new RSVP-TE object. This model scales better but obviously paths cannot be fully traffic engineered since each network performs setup on its own.

The hybrid model is a mixture of the above models (see figure 2.11). It allows for some trust between network domains and a bit of leakage of routing and signaling information between domains, but not the full topology. This removes the need for extra protocols such as UNI, as all domains can utilize GMPLS. Depending on how much information is shared over domain borders path setup may be fully or almost

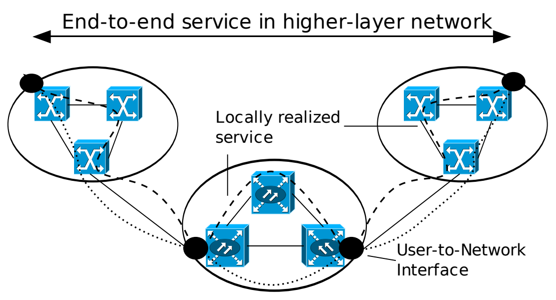
End-to-end service in higher-layer network
Locally realized service
User-to-Network Interface

Figure 2.10: The overlay model. ”Client” or ”higher layer” networks request services from another network in order to create an end-to-end service. No signaling or routing information is shared between the networks.

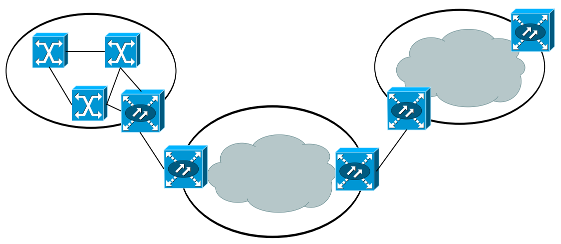

Figure 2.11: View of the network from nodes in the leftmost network segment in the hybrid model. Signaling is handled in the same fashion as in the peer model.

fully traffic engineered.

This model is similar to the way the Internet is designed, typically each autonomous system runs an inter gateway routing protocol (IGP) and a exterior gateway routing protocol (EGP). The IGP handles routing internally in the autonomous system, but exchanges some information with the EGP that is responsible for routing between the autonomous systems.

In the rest of this thesis only the peer model will be considered since it is the simplest model, it avoids the need for UNI protocols, and does not need to restrict control plane information at borders.

2.2.7 Signaling Extensions

RSVP-TE is the primary signaling protocol used in GMPLS. (The IETF stopped development of CR-LDP, thus it is unlikely that it will be extended to support GMPLS.) Many of the features of RSVP are reused in RSVP-TE, but some extensions have been made to support label distribution, explicit routing, different data plane network technologies, separation of control and data plane, and more. A number of these signaling extensions were introduced when creating MPLS TE, these are kept or updated by GMPLS which also introduces several extensions of

0 7	8 15	16 23	24 31
Length		Class-Num (19)	C-Type (4)
LSP Enc. Type	Switching Type	G-PID	

Figure 2.12: The Generalized Label Request object, request a label for specific type of LSP while informing the egress what kind of traffic it will carry [15]

.

its own. Some of the more important extensions are presented here in the form in which they appear within GMPLS.

- **Label management**
  - Generalized Label Request
  - Suggested Label
  - Upstream Label
- **Explicit routing**
  - Explicit Route Object
  - Record Route Object
- **Control and data plane separation**
  - Extended HOP Object

Label Management

By adding a Label Request object to a RSVP Path message the sending node can request that the downstream node return a label which can be used to send traffic to the downstream node. The Label Request object carries three fields: encoding, switching type, and Generalized Payload Identifier (G-PID) (see figure 2.12). The encoding field describes how data traffic will be presented to the transport medium and determines what type of LSP is created. For example an encoding of ”Packet” would request an MPLS label (and MPLS data plane reservation). The switching capability field is useful if the LSR is a so called hybrid node, i.e. it can switch on multiple switching type levels. For example an optical switch can be both LSC and FSC at the same time – if it is capable to not only switch individual wavelengths but also switch between different fibers. In that case it may be necessary to specify on what level the switching should take place. The G-PID indicates to the egress node what type of payload is carried within the LSP. The G-PID can use regular EtherType values, but some additional values have been specifically defined for GMPLS (such as SONET/SDH and HDLC traffic) [15].

In MPLS labels are returned in the RSVP Resv message which makes the LSPs uni-directional, but GMPLS uses by default bi-directional LSPs. Instead of having to setup two separate LSPs to achieve bi-directional communication (as in MPLS) GMPLS can send labels downstream in the RSVP Path message in order to establish a two-way label mapping during a single Path setup. This label is carried in the Upstream Label object which has the same properties as the regular Resv Label object, i.e. it can be swapped at each hop, carry the same types of labels, etc. The use of the Upstream Label object reduces both the signaling overhead and the time needed to create LSPs, since only one reservation request is necessary. Another label carrying object is the Suggested label object. This object can be sent within a Path message to indicate what label the upstream node would like to use. Using the suggested label object can also reduce LSP setup time as it can be used with a pre-configured data plane (i.e., the configuration is done before the actual reservation is made) when a Resv message is received. In many switches the time to create a switching rule is negligible, but in (for example) some optical switches which need to move physical components this many increase the overall LSP setup time significantly.

Explicit Routing

To support explicitly routed LSPs the Explicit Route Object (ERO) has been added. This object holds a list of ”abstract nodes” which should be traversed by the LSP (see figure 2.13). There are three different abstract node types defined: IPv4, IPv6 prefix, and Autonomous System (AS) number subobject. Each of these abstract nodes can contain several real nodes. For example the AS subobject contains all the GMPLS nodes within an Autonomous System, an IPv4 subobject contains all the GMPLS nodes within a specific subnet (for example a /24 subnet or a single host if the subnet prefix is /32). By creating a list of these objects an explicit route can be specified with large flexibility in granularity.

Each abstract node is defined as either ”strict” or ”loose”. This ”strictness” and ”looseness” is always relative to the previous abstract node, if a node is strict the path traversed **must** consist of nodes which are part of the previous and the strict abstract node. When traversing the previous abstract node in conjunction with a loose hop the path may traverse nodes which are **not** part of either abstract node. Note that this does not mean that **all** the nodes specified in a strict abstract node has to be traversed, it is not necessary to traverse all the nodes in an AS for example, only the ones needed to reach the following abstract node.

While traversing the nodes specified in the ERO in the Path message the nodes are successively removed from the ERO. When the egress node is reached it has no way to know what path was used to reach it (except the previous hop). In order to collect this information the Record Route Object contains a list of abstract nodes. The abstract nodes used in the Record Route Object is identical to the ones used in the ERO with the exception of that the AS number subobject is not allowed while a Label subobject can be used to record what label was used on a specific link. By adding abstract nodes (and optionally the label used) to this list as the Path message traverses the network both ingress and egress node may find out the

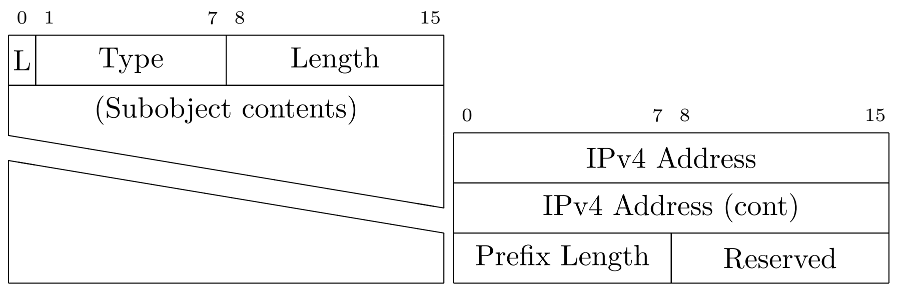

Figure 2.13: Abstract node (left) and IPv4 subobjects (right). The L flag indicates of the hop is loose or strict. The Type field holds the type of the subobject (IPv4 or IPv6 prefix, AS number). The IPv4 subobject contains an IPv4 address and netmask (prefix length) [16]. (Note that the widths of these figures are 16 bits.)

exact path used (as the egress is allowed to send the Record Route Object back to the ingress in the Resv message)[16].

Control and data plane separation

When an LSR receives an Path message it stores some information about it in a so called Path State Block. The Path State Block contains, for example, information about the previous hop. This information is taken from the RSVP_HOP object carried in the Path message. Before GMPLS the RSVP_HOP contained only an IPv4 or IPv6 address and a Logical Interface Handle (a node local value identifying the interface used to send the RSVP message) which was used by the LSR to find out whom to send replies to. The previously stored RSVP_HOP object is returned within the Resv message to make it easier for the previous hop to find out which interface it used to send the Path message. The previous node cannot assume that the Path message was sent on the same interface that receives the Resv message, since the message may pass through a network between these two nodes (which is why this information is sent along with the messages). Since GMPLS supports out-of-band signaling – and multiple data plane links per control plane link – the RSVP_HOP object has been extended to indicate which data plane interface the message refers to as well as the control plane interface. There are two new RSVP_HOP objects which contains either an IPv4 or IPv6 control plane address and a Logical Interface Handle and a TLV identifying the data plane interface (see figure 2.14). There are five types of data plane interface identifier TLVs: IPv4, IPv6, unnumbered interface, and two component interface identifiers which are identical to the unnumbered interface TLV except for the Type field.

2.2.8 IGP Extensions

As previously stated GMPLS can use both ISIS-TE and OSPF-TE for routing of the control plane and dissemination of TE information in the network. I will however

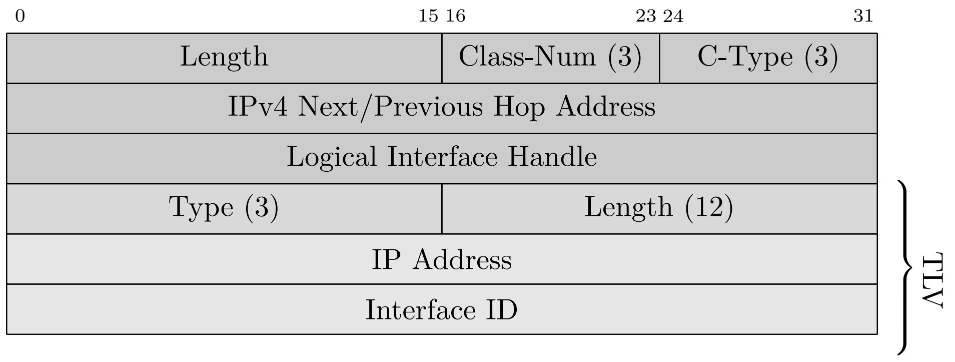

Figure 2.14: An IF_ID RSVP_HOP object with an IPv4 control plane identifier and a unnumbered interface data plane identifier [15] [17].

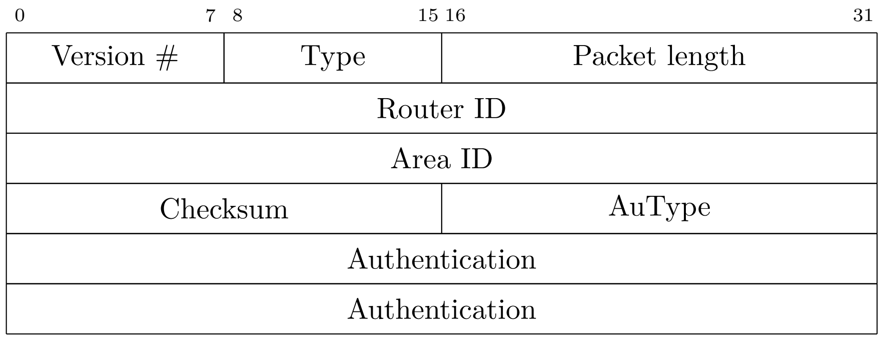

Figure 2.15: The OSPF packet header. The version field indicates what OSPF version is used, the Type field differentiates the different messages types, and the Packet length is the length of the entire packet in bytes. Router ID is the ID of the source of the packet. The AuType and Authentication fields can be used to authentication the packet [19].

focus on the extensions to OSPF which enables this.

OSPF was designed to be used within an autonomous system. The networks in an AS can be subdivided into *areas* making routing more efficient [18]. The areas are connected by *area border routers* which summarize the routing information about an area and send it into other areas. A OSPF network contains at least one area, the backbone area. OSPF has five different kinds of messages, all with a common header (see figure 2.15) [19]. These five messages are Hello, Database Description, Link State Request, Link State Update, and Link State Acknowledge (see table 2.2 for details).

Some of these message can carry a list of Link State Advertisements (LSAs) or a list of LSA headers. Most important is the Link State Update message which performs the actual flooding of link state information in the network by distributing LSAs. In OSPFv2 there are five kinds of LSAs which describe the local topology

0 … 15	16 …	… 31
LS age	Options	LS Type
Link State ID		
Advertising Router		
LS sequence number		
LS checksum	length	

Figure 2.16: The common LSA header [19].

of the router transmitting them (see table 2.3). For example, a router originates a router-LSA which contains a list of all its links, whether the link is a point-to-point link to another router, a link to a network without any other routers, or any of the other four types of links that exist in OSPFv2. If a router is the Designated Router for a network segment (either by being the only router on the segment or by election) it transmits a network-LSA which describes all routers attached to that particular network segment. The other LSA types announce links across areas (Summary Link), links that reach AS border nodes (Summary Link to AS Boundary Router), and links received from other routing processes such as BGP (External LSA). All LSAs share a common header and are distinguished from each other by a 8-bit value (see figure 2.16). The header contains several fields, the LS age field is a timer which holds the age of the LSA in seconds, updated by each router. The Options field contains some flags, for example it can indicate if the area is a stub area or not. LS type is the type field which is shown in table 2.3. The Link State ID field holds an identifier for the link which this LSA concerns, for example the router IP address in a router LSA. The Advertising router field holds the Router ID of the originator of the message. The LS sequence number field is a sequence number assigned to each LS Update message. The length field is the length of the whole packet in bytes.

All routers within an area receive and store the router- and networks-LSAs from that area in a Link State Database. After the Link State Database has been constructed it is used to calculate Shortest Path First routing tables.

The "OSPF Opaque LSA Option" is an additional three LSA types which makes it easy to extend OSPF for application specific purposes [20]. These LSAs are flooded like the other LSAs, but are not used for regular OSPF routing, instead they carry application specific data. Of importance to us, these are the LSAs[2] used to carry TE information in GMPLS [21]. The TE LSAs have a standard LSA header followed by a Type-Length-Value field to distinguish between the different TE objects. The TE LSA TLV contains a number of sub-TLVs which hold the actual TE link information, such as Link ID, remote and local addresses, reservable bandwidth, etc. An actual LS Update message may look like the one in figure 2.17 where the OSPF header can be seen along with three different LSA headers, the

[2]Actually only the area local opaque (type 10) LSA is used

Table 2.2: OSPF message types.

Type	Name	Description
1	Hello	Sent periodically on all interfaces to create and maintain routing adjacencies
2	Database Description	Sent when an adjacency is created in order to exchange information about the LSAs in the router's database
3	Link State Request	Used to request fresh information if part of the router's database is out-of-date
4	Link State Update	Performs the actual flooding of LSAs to neighbors, multicasted if the network supports it
5	Link State Acknowledge	Sent to acknowledge Link State Updates in order to make the flooding reliable

Table 2.3: Different LSAs, their use and flooding scope.

Type	Name	Scope	Description
1	Router	Area local	Describes all links of a particular router
2	Network	Area local	Describes all routers in a network
3	Summary	Inter-area	Summarizes the links in one area and informs the other attached areas
4	ASBR-Summary	AS (except stubs)	Informs areas which networks are attached to AS border nodes
5	External	AS (except stubs)	Informs areas of routes received from external routing processes, such as BGP
9	Link local opaque	Link local	Carries opaque information within one network
10	Area local opaque	Area local	Carries opaque information within one area
11	External opaque	AS (except stubs)	Carries opaque information within one AS

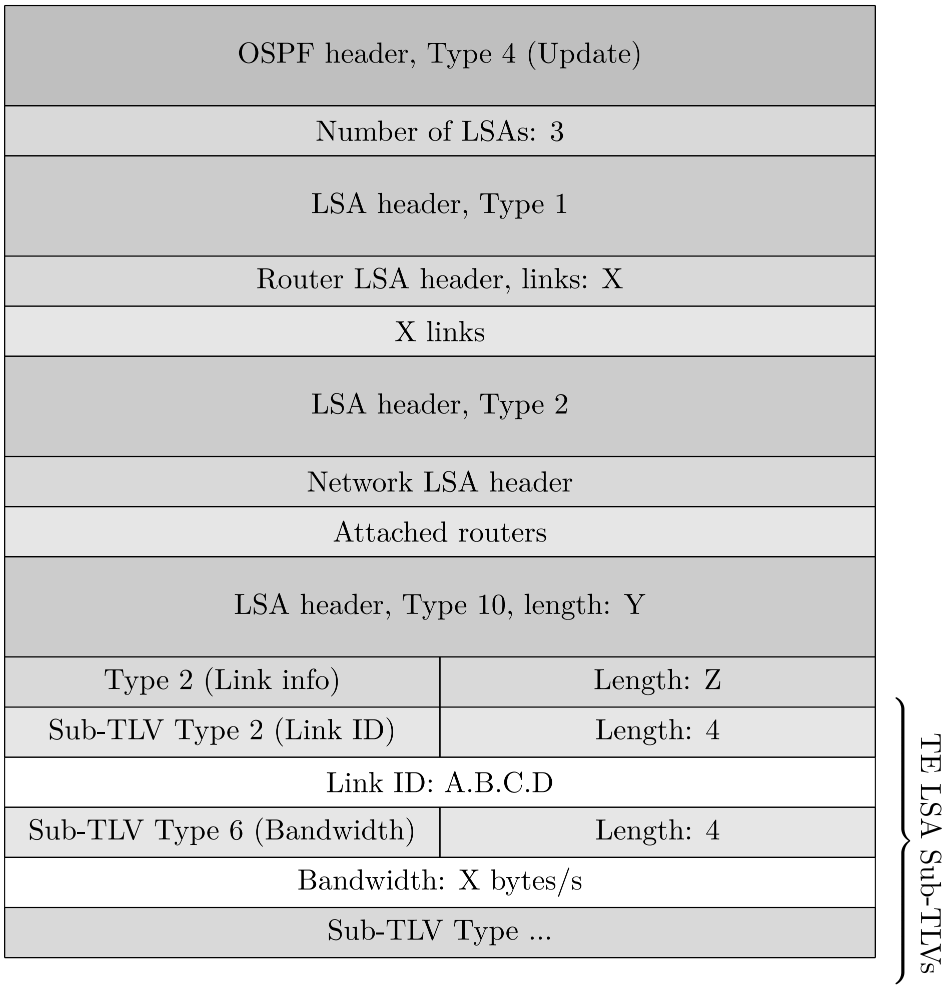

Figure 2.17: Simplified illustration of how an OSPF LS Update message may be constructed. This one carries three LSAs, one router LSA, one network LSA, and a TE LSA. The TE LSA is in turn composed of one TLV consisting of several sub-TLVs which holds information such as Link ID, Maximum reservable bandwidth, Interface Switching Capability Descriptor and so on.

TE LSA TLV and sub-TLVs are also shown. Unlike the regular LSAs the TE LSAs are not stored in the Link State Database, but in a separate Traffic Engineering Database.

2.2.9 Link Management Protocol

The Link Management Protocol (LMP) is responsible for data plane link discovery, configuration, and fault isolation. This could partly be done manually, at least the configuration of data plane links could be done, but with a large number of links this becomes error prone and can be quite a lot of work to both initially configure and keep up to date. The LMP protocol is UDP based and runs on port 701 between GMPLS nodes. The messages are similar to the RSVP-TE message with a common message header followed by objects and sub-objects (TLVs) distinguished by Class and C-Type. The main functions of LMP are:

Control channel management Initialization and management of control channels. Initially data plane neighbors establish each others identity and capabilities, after this is done they enter a management phase which periodically sends Hello messages to make sure there is still connectivity.

Data plane link discovery Discovery of and connectivity check are needed on each of the data plane links to and from neighbors. Before this is done the node only knows the local identifier of each link, but not the state of or the remote identifier of each link.

Data plane link capability exchange Depending on data plane technology an exchange of link information may follow link discovery. This may also be necessary if multiple types of links are supported.

Data plane link verification Verification can be conducted at any time to check connectivity and the status of a data plane link. The procedures followed to verify a link are identical to the link discovery procedures.

Fault isolation One of the most important parts of LMP is fault isolation. It is initialized by a downstream node whenever a fault such as signal degradation is noticed. Since some data plane technologies are oblivious to the data they are transporting another protocol is sometimes necessary to find the faulty link in order to bypass it.

Authentication LMP has support for authentication, although it is not a requirement it may be a good idea to use authentication. Since the control link or channel can traverse a public IP network each message should be authenticated in order to prevent spoofing attacks.

An extended LMP protocol is LMP-WDM which runs between a OXC (see section 3.1.2) and optical components which may be external to the OXC, such as optical amplifiers and multiplexers. These external components are known as an Optical Line System.

2.2.10 Path Computation Element

The Path Computation Element is responsible for performing path calculation within the GMPLS network. Since path calculation is usually only necessary at the edge of the network, there is no requirement that each node in the network can perform path calculation, unlike for example IP networks (where each routes is expected to be able to perform route calculation). Instead the LSRs implement a protocol that can send requests to the Path Calculation Element which calculates a route and returns it to the LSR. This client-server path calculation architecture makes it possible to make path calculation fully decentralized, centralized, or various degrees thereof. This is useful since constraint-based path calculation may be a computationally expensive operation, if it is centralized the LSRs can be ”dumber” and cheaper to produce. Multiple Path Calculation Elements can be used to make the path calculation more resilient and route request can be distributed between the Path Calculation Elements in order to balance load.

Chapter 3

Multi-region GMPLS Networks

Many complicated and interesting issues in GMPLS networks arise when going from one network region to another. Path calculation has to take into account constraints specific to each region; additionally extra signaling may be necessary to tunnel switching types and a correct “inter-technology” mapping must be established. The focus of this thesis is the switch between L2SC and LSC regions, specifically between Ethernet and a Wavelength Division Multiplexing system. How these different switching systems operate is discussed below.

3.1 Network Technologies Considered in this Thesis

3.1.1 Ethernet

Ethernet has been a very popular network technology since it was invented in the mid-1970s. It is standardized by the Institute of Electrical and Electronics Engineers (IEEE) in the IEEE 802.3 family of standards [22]. Traditionally Ethernet has been used in Local Area Networks (LANs). In recent years Ethernet has also been used to implement Metropolitan Area Networks and carrier backbone networks, displacing legacy time division multiplexing systems such as synchronous optical networks (SONET/SDH). Metropolitan and carrier networks frequently use 1 or 10 Gigabit Ethernet; as the physical layer supports both short twisted-pair cables and longer optical fibers. A major reason for the popularity of Ethernet is the relative low price of Ethernet equipment due to the large production volume [23].

The Ethernet frame header contains two addresses, a source and destination Media Access Control (MAC) address. When a switch receives a Ethernet frame the destination MAC address is examined and a table look-up indicates on which port the frame should be forwarded; this is based upon an earlier procedure called ”learning” which has discovered the MAC addresses of Ethernet interfaces attached to the different ports. If the destination MAC address is not found in the forwarding table, i.e. the destination address is unknown, the frame is transmitted on all ports except for the one it arrived on. Each port of a Ethernet switch belongs to a different collision domain, every node in a collision domain can intercept frames

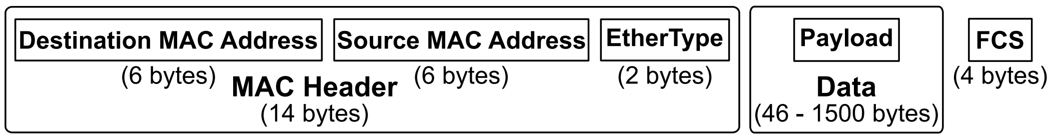

Figure 3.1: Ethernet Type II frame format. The header contains source and destination address and a field indicating what type of data is carried in the payload. The frame check sequence (FCS) is a checksum of the frame inserted after the payload.

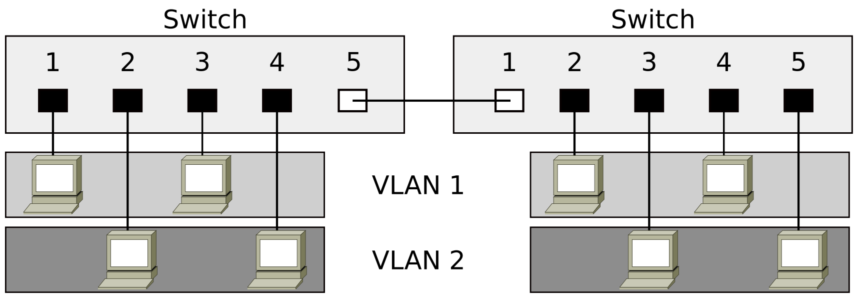

Figure 3.2: Example of two connected Ethernet switches, each with 2 VLANs. Equipment on a VLAN can only communicate with equipment on the same VLAN.

sent on that domain and if they try to send frames at the same time the frames will collide. Another important domain is the broadcast domain, this is the set of collision domains reached by sending a local broadcast frame (a frame with the destination MAC address ff:ff:ff:ff:ff:ff). The broadcast domain generally covers all collision domains on a bridged network, but does not extend past network layer routers. Several slightly different Ethernet frame formats exists, see figure 3.1 for the format of the common Ethernet Type II frame. The two byte EtherType field indicates what type of data is carried in the Payload (for example the value 0800_{16} represents IP traffic, 0806_{16} ARP, etc). The four byte Frame Check Sequence (FCS) is a checksum used to detect corrupted frames.

Sometimes it can be useful to create different broadcast domains on top of a physical network in order to create logical network topologies different from that of the physical network. This can be done on Ethernet networks by using Virtual LANs (VLANs). Using VLANs one can partition different ports on a switch to be part of different networks.

The example shown in figure 3.2 shows two connected switches with two VLANs each. Interfaces on VLAN 1 can communicate with interfaces on the same VLAN on both switches, but not with interfaces connected to VLAN 2.

VLANs were implemented on Ethernet following the IEEE 802.1Q standard which adds a tag after the source MAC address in the Ethernet header. In figure 3.3 an Ethernet Type II frame with a 802.1Q VLAN tag is shown in which one can

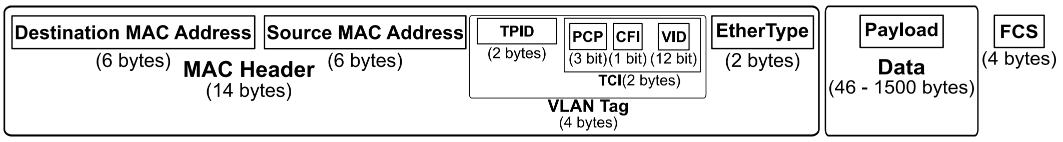

Figure 3.3: Ethernet Type II frame tagged with 802.1Q VLAN Tag.

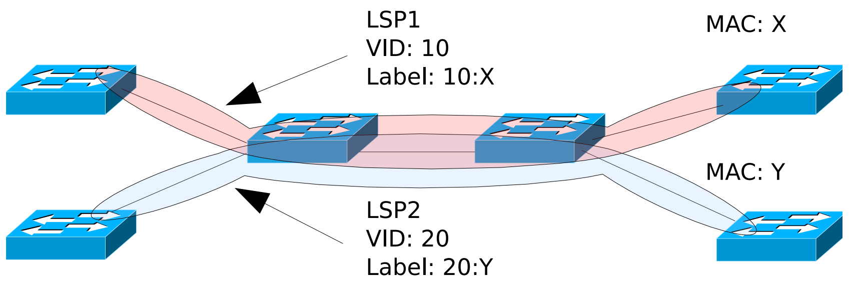

Figure 3.4: End-to-end Ethernet label using VID and destination MAC address.

see the VLAN tag inserted between the source address and the EtherType field. The VLAN tag consists of two parts: the Tag Protocol Identifier (TPID) and the Tag Control Information (TCI). The TPID takes the previous position of the EtherType field and is set to the value 8100_{16} and informs the switch that the frame is tagged, the actual EtherType is placed after the TCI. The 2 byte TCI is divided into a 3-bit field for priority, a 1-bit field called Canonical Format Indicator (CFI), and a 12-bit field for the VLAN identifier (VID) [24]. The priority field can be used to with IEEE 802.1p (part of IEEE 802.1D since 1998) to provide link layer Quality of Service. The CFI field is used to provide interoperability with Token Ring networks, on Ethernet networks it is always set to zero. The 12-bit VID identifies what VLAN the frame belongs to. The VID could be used as a GMPLS label in the same way MPLS labels are used, with swapping at each LSR. Unfortunately, this violates the 802.1Q standard and is not supported by Ethernet switches. If the VID is used as a label we are forced to use it as an end-to-end label, meaning that the same label is used for each hop along the entire Ethernet segment of the path. This constraint limits the number of LSPs to 4093 (12 bits minus 3 reserved labels) per link and Ethernet segment. By using the VID in combination with the destination address of the egress as the label; LSPs can share VIDs and the limitation changes to 4093 LSPs per destination address, quite an improvement (see figure 3.4).

Other possibilities for a GMPLS Ethernet labels can be found in the IEEE 802.1ad (Provider bridge [25]) and IEEE 802.1ah (Provider Backbone Bridges [26]) standards. These two standards allow provider networks to transport VLAN tagged Ethernet traffic, which makes it possible to create for example link layer private networks (virtual private LAN services). IEEE 802.1ad (also known as Q-in-Q) is an amendment to the 802.1Q VLAN tags which adds another almost identical VLAN tag in addition to the original one. These so called Service Tags may be a better choice for GMPLS labels since it would allow the GMPLS network to transport Ethernet frames which are already tagged (perhaps for use in a customer's network). The other possibility is using IEEE 802.1ah which adds additional MAC address fields as well as an additional VID. At the time of writing neither the GMPLS label format or which of these available tags to use has been standardized. Work on this is being done in the IETF Common Control and Measurement Plane (CCAMP) work group [27].

3.1.2 Optical Networks

Wavelength Division Multiplexing (WDM) is very similar to the Frequency Division Multiplexing (FDM) that is used in systems operating in the lower frequencies of the electromagnetic domain. The bandwidth of a channel is split into multiple discrete channels, in WDM the sub-channels are commonly referred to as lambdas, from the Greek letter λ used in physics to denote wavelength. In WDM the media typically is an optical fiber instead of open air. The bandwidth frequently ranges from the infra-red to ultraviolet - as this frequency range has low attenuation within that fiber, due to the materials used and the design of the fiber. Depending on the difference between the frequencies of the different lambdas (the so called channel spacing) the technique is called either Coarse Wavelength Division Multiplexing (CWDM) or Dense Wavelength Division Multiplexing (DWDM). Typical channel spacing for CDWM signals is 2500 GHz and in DWDM between 12.5 and 100 GHz (see table 3.1).

Table 3.1: Values used in the Grid field and in the Channel Spacing (C.S.) field.

Grid	Value
ITU-T DWDM	1
ITU-T CWDM	2
Future use	3 - 7

C.S.(GHz)	Value
12.5	1
25	2
50	3
100	4
Future use	5 - 15

Components of Optical Networks

In order to switch optical signals there are two basic approaches: optical-electronic-optical and the optical-optical-optical. In optical-electrical-optical the optical signal is demuxed and converted into an electric signal. An advantage of this approach is that an electrical signal can be switched easily and regenerated at perhaps a different wavelength than the incoming signal and emitted into a different fiber. A disadvantage of optical-electrical-optical is that the optical signal has to be decoded and sent through an electronic switching fabric, limiting the device to specific signal encodings and data rates. In the optical-optical-optical case the incoming signal is

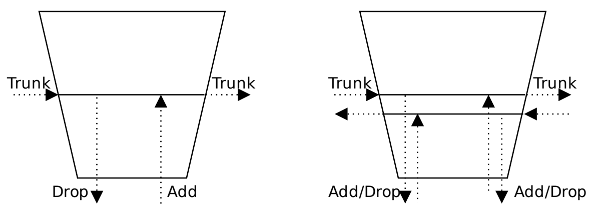

Figure 3.5: 1st and 2nd degree OADM.

demuxed (if necessary) and sent through a optical switching fabric. Such an optical switching fabric may be constructed by an array of micro-electromechanical system mirrors. These mirrors reflect the signal into a fiber or multiplexer that is a function by the configurable orientation of the mirrors.

The optical-optical-optical approach is generally cheaper since it does not involve any transceivers which are necessary in optical-electrical-optical to decode and regenerate the signal. Since the signal is transparent to the optical-optical-optical system it more easily supports upgrades to the systems; while an optical-electrical-optical switch would need to be replaced since each signal has to be decoded and regenerated. However, a disadvantage of the optical-optical-optical approach is that it does not even know if there is a signal present - so failures of switches or fibers might not be easily detected.

An optical add-drop multiplexer (OADM) is a device that has two or more so called “trunk” ports, and one or more pairs of add and drop ports (sometimes called tributary ports). If you have 1 tributary port, the device is called a 1-degree OADM, and so on. The add and drop ports are used to add and remove lambdas from the fibers connected to the trunk ports. This is usually used to connect networks into a fiber optic ring topology, but the trunk ports do not necessarily have to be connected in a ring. Which lambda that is added and dropped is statically configured in a traditional OADM, however more modern devices can be remotely reconfigured and are therefore called a reconfigurable optical add-drop multiplexers (ROADMs) [28].

The simplest, 1-degree ROADM has one input, from which you can drop a single lambda, and one output to which you can add a lambda (left side of figure 3.5). If you want bidirectional communication this forces you to have a ring topology. A common design is to have an input and output on both trunk ports and two or more add-drop ports (i.e. a 2- or N degree ROADM). The fibers used are either two bidirectional fibers for each port, or a single bidirectional fiber. As can be seen in the righthand portion of figure 3.5 the left trunk can only be added from or dropped to the left port, and vice versa on the right side. This light path constraint must be taken into account when calculating LSP paths.

Another switching device is the optical cross-connect (OXC). These devices allow arbitrary switching between input and output ports, which may be single lambdas (wavelength cross-connect), several lambdas (waveband cross-connect), or a whole

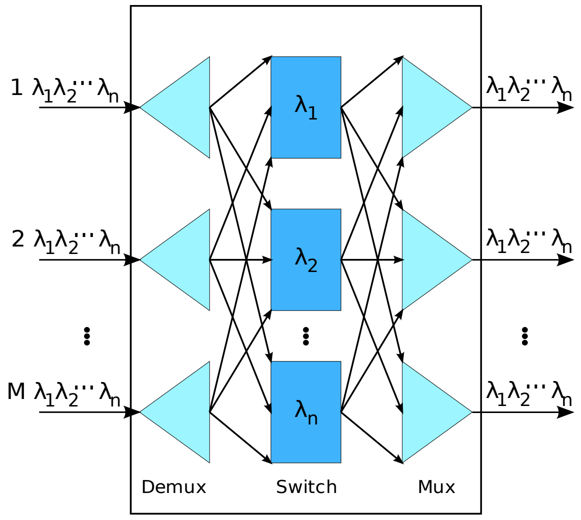
1 $\lambda_1\lambda_2\cdots\lambda_n$
2 $\lambda_1\lambda_2\cdots\lambda_n$
M $\lambda_1\lambda_2\cdots\lambda_n$
λ_1
λ_2
λ_n
$\lambda_1\lambda_2\cdots\lambda_n$
$\lambda_1\lambda_2\cdots\lambda_n$
$\lambda_1\lambda_2\cdots\lambda_n$
Demux
Switch
Mux

Figure 3.6: Architecture of an optical cross-connect [1].

fiber (fiber cross-connect)[1] [1]. In figure 3.6 the architecture of a optical-optical-optical wavelength cross-connect (WXC) can be seen. These devices are commonly used to join two or more rings together or in mesh topologies. Since they allow arbitrary switching there are no light path constraints other then that you cannot switch two or more incoming lambdas of the same wavelength into an outgoing fiber. In connect an incoming lambda to a different outgoing lambda the cross-connect must be able to perform wavelength conversion, which is available in some products. The optical signal is generated by a laser in a transponder, for example by a Gigabit Ethernet network interface. These usually operate on a static wavelength, but tunable lasers are under development. Such a transceiver usually only utilizes a single wavelength, it is therefore not WDM per se. The optical signal is turned into a WDM signal by connecting it to a OADM, WXC, or a WDM multiplexer. A WDM network containing these devices may look like the one illustrated in figure 3.7.

WDM Labels

In order to advertise available wavelengths or signal paths the nodes involved must agree upon a generalized label format to use. Currently one format is standardized in RFC 3471 and another is progressing as an Internet draft [15] [30]. The format in RFC 3471 is a 32-bit value used to identify a FSC port or a LSC wavelength. This value is either manually configured or locally agreed upon via LMP. The label format proposed in the Internet draft supports both coarse and dense WDM. It is based on wavelength grids (lists of wavelengths and distances) published by the

[1]For simplicity I will refer to all these devices as OXCs.

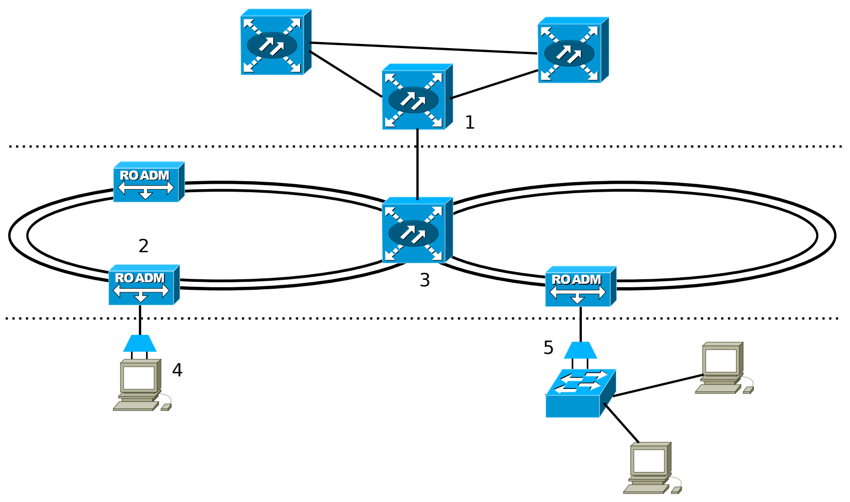

Figure 3.7: A possible design of an optical network. The backbone consists of a mesh of optical cross-connects (1) connected to a fiber ring. Rings may be concatenated by an optical cross-connect (3), on the rings ROADMs are used to drop specific wavelengths into customer networks. At the customer the optical signal is passed through a WDM system (4 and 5) and is converted into electrical signals [29].

ITU-T as G.694.1 and G.694.2 [31][32]. The CWDM label specifies the grid and a wavelength, the wavelength in the label is added to a reference wavelength to calculate the final wavelength of the signal (equation 3.1 and figure 3.8). In the DWDM label the frequency is calculated by multiplying the Channel Spacing, the Sign (positive or negative), the channel number (n), and adding the value to a reference frequency (equation 3.2 and figure 3.8). However, these labels are only usable on so called ”colored” links, on which the optical signal is well defined and conforms to the ITU-T grids. Unfortunately, there are optical links which do not conform to the ITU-T grids, therefore some other means must be used to specify labels for such links. Such non-conforming links are outside of the scope of this thesis.

$$\text{Wavelength (nm)} = 1470(\text{nm}) + n \times 20(\text{nm}) \tag{3.1}$$

$$\text{Frequency (THz)} = 193.1(\text{THz}) \pm n \times \text{Channel Spacing (THz)} \tag{3.2}$$

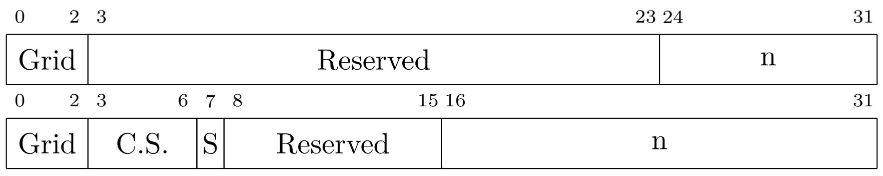

Figure 3.8: Proposed CWDM (top) and DWDM (bottom) label. Grid values can be found in table 3.1 along with values for Channel Spacing (C.S.). The Sign (S) value in the DWDM label indicates if the value should be added or removed from the reference frequency (see equation 3.2) [30].

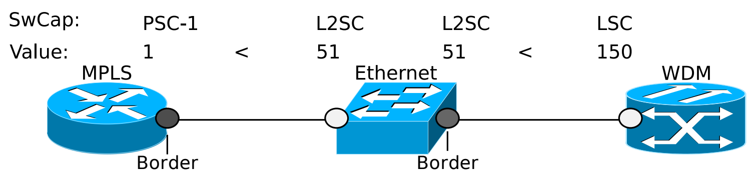

Figure 3.9: Three nodes of different switching capability. Nodes are (from left) MPLS, Ethernet, and WDM nodes. Above are the switching capabilities and their values of the interfaces on the links. Since $1 < 51$ the MPLS node is the border on the leftmost link. Since $51 < 150$ the Ethernet node is the border on the rightmost link.

3.2 Region Boundaries

When two nodes with different switching capabilities are connected a region border is created, but which of the two nodes should be responsible for handling the problems of connecting the two different regions? In RFC 4206 a procedure for determining which LSR is the responsible border node is defined as follows:

> ”Define an ordering among interface switching capabilities as follows: PSC-1 < PSC-2 < PSC-3 < PSC-4 < TDM < LSC < FSC. Given two interfaces if-1 and if-2 with interface switching capabilities isc-1 and isc-2 respectively, say that if-1 < if-2 iff isc-1 < isc-2 or isc-1 == isc-2 == TDM, and if-1's max LSP bandwidth is less than if-2's max LSP bandwidth.
>
> Suppose an LSP's path is as follows: node-0, link-1, node-1, link-2, node-2, ..., link-n, node-n. Moreover, for link-i denote by [link-i, node-(i-1)] the interface that connects link-i to node-(i-1), and by [link-i, node-i] the interface that connects link-i to node-i.
>
> If [link-(i+1), node-i)] < [link-(i+1), node-(i+1)], we say that the LSP has crossed a region boundary at node-i; with respect to that LSP path, the LSR at node-i is an edge LSR.”

In the case of simple nodes, nodes with only one switching capability per TE-link, this procedure boils down to the node with the lower ISC is the border node. In figure 3.9 three nodes are shown together with their ISCs (see table 2.1 for the values assigned to different switching capabilities). One can see that the middle node is the border on the Ethernet to WDM link since $51 < 150$; however it is not the border on the MPLS to Ethernet link. On the MPLS to Ethernet link the MPLS node is the border since $1 < 51$. This way border nodes are defined together with region boundaries. Note that a node is a border node with respect to a LSP. A node with for example two L2SC interfaces and one PSC interface is not a border node (relative the LSP) if an LSP enters and leaves the node through the L2SC links. However, if the LSP enters through the PSC interface and leaves through a L2SC interface the node is a border node in respect to that particular LSP.

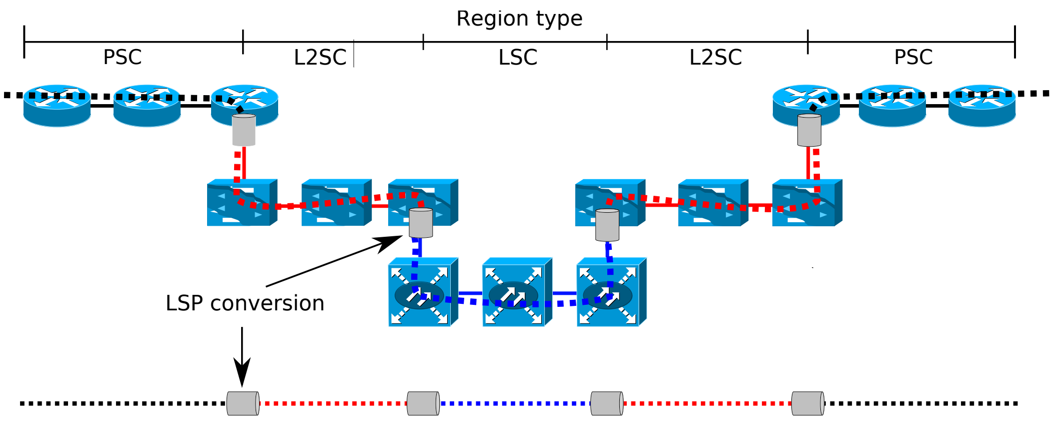

Figure 3.10: Example of LSP conversion. An LSP is created through three regions, at each region border it is converted to the lower region.

3.3 Cross Region LSP setup

Cross region hops cannot be treated like hops within regions. Two ways of dealing with cross region LSP setup are presented below.

3.3.1 LSP Conversion

The ”naïve” or simple way of handling cross region LSP setup would be to treat it more or less as a intra-region TE-link hop. This would mean that in for example a Path message one would replace the switching capability in the LABEL_REQUEST object and changes the labels in any UPSTREAM_LABEL and/or SUGGESTED_LABEL object (see figure 3.10 for an example). If its possible to simply convert the labels (for example an MPLS label with value 22 converted to a Ethernet label with VID 22) this could be done or completely new labels could be allocated.

This simple way of dealing with cross-region LSPs however has some problems. The most obvious one is the issue of granularity and label space. If the minimum reservable bandwidth in one region is significantly larger than in the other region, then a large amount of bandwidth may be reserved but **not** utilized. For example an LSP with 100 Mbit/s can be reserved on a Gigabit Ethernet without any wasted bandwidth, but on a optical WDM network the only reservable bandwidth might be 2.5 Gigabits/s. Reserving 2.5 Gigabit/s when only 100 Mbit/s is used is of course a waste of resources (at least if there is other traffic which could have used the bandwidth).

However, if label space and bandwidth granularity are similar in both regions, then LSP conversion is a fast and simple way to do cross-region LSPs.

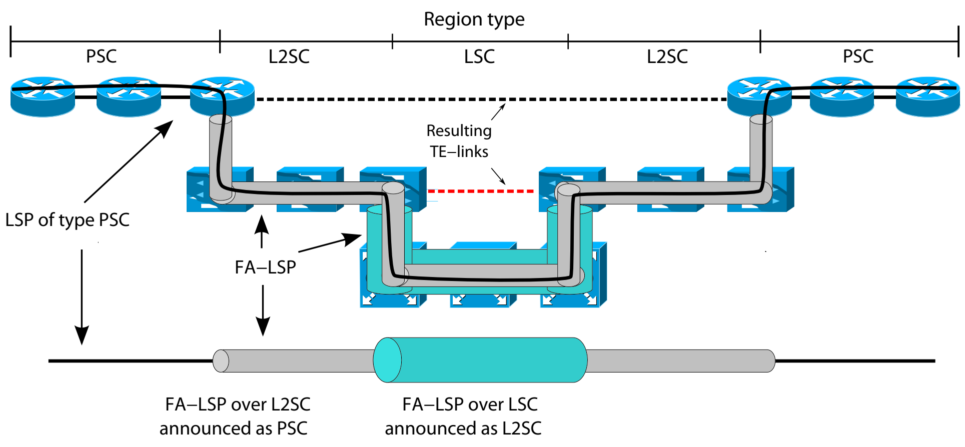

Figure 3.11: Example of how an FA-LSP is used to cross regions. At each region border the creation of a FA-LSP is triggered.

3.3.2 Forwarding Adjacencies

A more efficient way to handle cross region LSP setup is the concept of a Forwarding Adjacency LSP (FA-LSP), which is similar to the Label Stacking technique discussed in section 2.1.1. However, in GMPLS one cannot stack labels in the same sense since some labels (implicitly) refer to quantified physical resources and these cannot be stacked within each other. This can be solved by creating an LSP over the LSRs in the path that belong to the same continuous region. This LSP can then be used as if it was a link of the same switching type as the bordering region – but with the characteristics of such a link. The FA-LSP may also be advertised by the routing plane like a regular TE-link so that it may be used by other nodes for path computation, it is actually only when announced that the link becomes an FA-LSP, otherwise it is called an Hierarchical LSP (H-LSP). In figure 3.11 an example of cross-region LSP setup with FA-LSPs can bee seen.

The TE-link ”produced” by the FA-LSP inherits most of its TE-link information from the underlying LSP, for example bandwidth. Other information such as TE-link identifiers are generated while the Interface Switching Capabilty Descriptor is copied from the ingress node [33]. By using a LSP as a logical link the two border LSRs become neighbors in the incoming switching type region and thus they have a forwarding adjacency.

For an example of how the exchange of RSVP signaling messages look when two FA-LSPs are created via triggered setup see figure 3.12. In this example the FA-LSP setup is triggered, the LSRs decide to create a FA-LSP themselves. An FA-LSP may also be commissioned by other means, for example by a management plane. When triggered the Traffic Engineering Database must be used to find the other end of the FA-LSP.

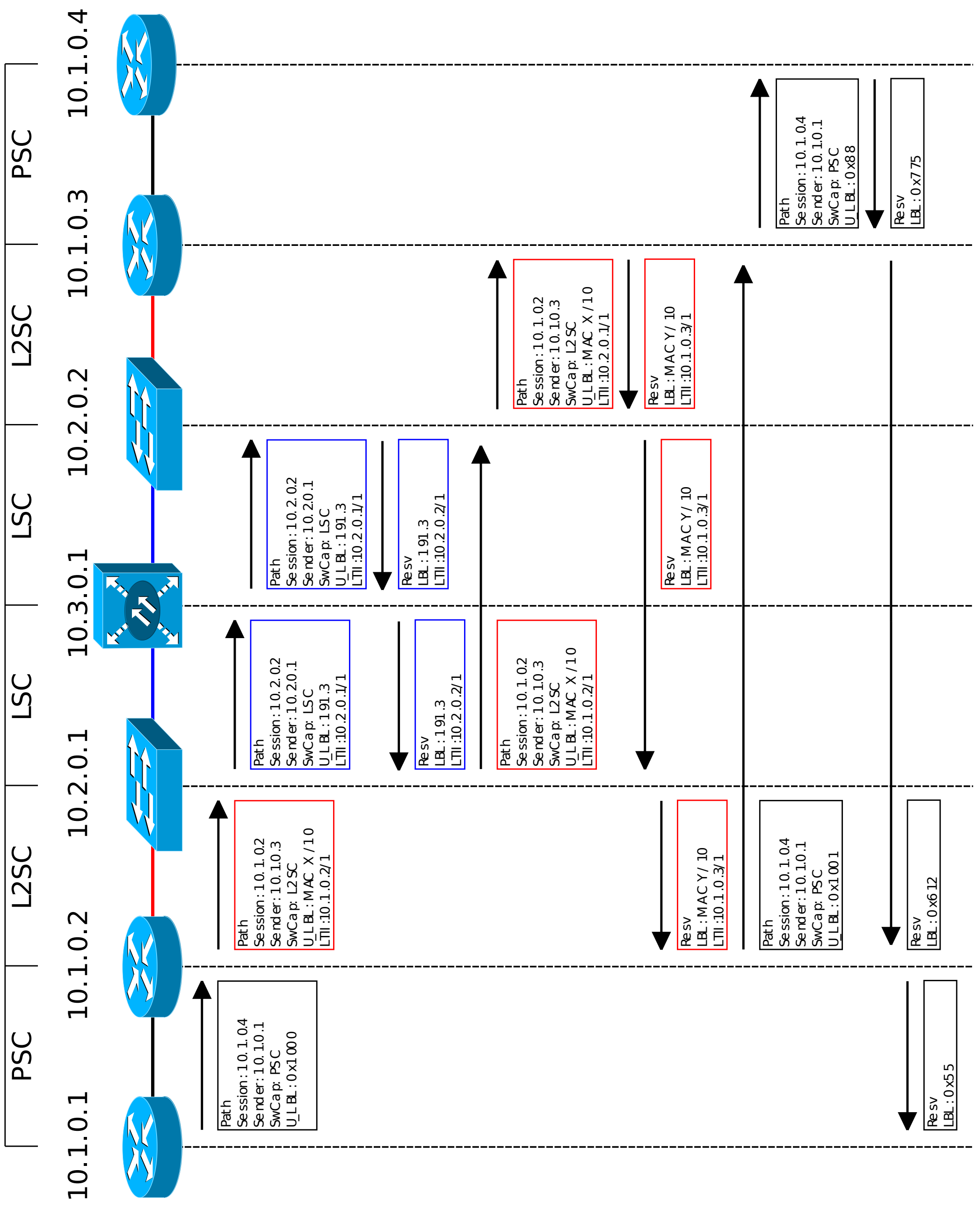
PSC
L2SC
LSC
LSC
L2SC
PSC
10.1.0.1
10.1.0.2
10.2.0.1
10.3.0.1
10.2.0.2
10.1.0.3
10.1.0.4
Path Session: 10.1.0.4 Sender: 10.1.0.1 SwCap: PSC U_LBL: 0x1000
Path Session: 10.1.0.2 Sender: 10.1.0.3 SwCap: L2SC U_LBL: MAC X / 10 LTII: 10.1.0.2/1
Path Session: 10.2.0.2 Sender: 10.2.0.1 SwCap: LSC U_LBL: 191.3 LTII: 10.2.0.1/1
Path Session: 10.2.0.2 Sender: 10.2.0.1 SwCap: LSC U_LBL: 191.3 LTII: 10.2.0.1/1
Resv LBL: 191.3 LTII: 10.2.0.2/1
Resv LBL: 191.3 LTII: 10.2.0.2/1
Path Session: 10.1.0.2 Sender: 10.1.0.3 SwCap: L2SC U_LBL: MAC X / 10 LTII: 10.1.0.2/1
Path Session: 10.1.0.2 Sender: 10.1.0.3 SwCap: L2SC U_LBL: MAC X / 10 LTII: 10.2.0.1/1
Resv LBL: MAC Y / 10 LTII: 10.1.0.3/1
Resv LBL: MAC Y / 10 LTII: 10.1.0.3/1
Resv LBL: MAC Y / 10 LTII: 10.1.0.3/1
Path Session: 10.1.0.4 Sender: 10.1.0.1 SwCap: PSC U_LBL: 0x1001
Path Session: 10.1.0.4 Sender: 10.1.0.1 SwCap: PSC U_LBL: 0x88
Resv LBL: 0x775
Resv LBL: 0x612
Resv LBL: 0x55

Figure 3.12: Simplified view of RSVP signaling triggering the creation of two FA-LSPs in order to bridge LSC and L2SC regions.

3.3.3 Keeping State

As previously mentioned RSVP is a soft state protocol. This means that unless periodically updated the reserved resources will be released and Tear-down messages sent to remove the state in other routers. In the original RSVP specification state was maintained by retransmitting the original Path and Resv messages [4]. This method has some problems, for example it has scaling issues, and an improved way of keeping state was introduced.

The RSVP Refresh Overhead Reduction extensions contain a number of new RSVP sub-objects and new RSVP message types [34]. A new sub-object is the MESSAGE_ID object which can be carried in the Path and Resv messages that triggers the creation of a reservation. The MESSAGE_ID carries a 32 bit node-local identifier which is used to identify the reservation. This identifier can later be used to refresh the state for that reservation either by piggybacking it within another RSVP messages or by using the RSVP Summary Refresh (Srefresh) message. The Srefresh message can carry a list of identifiers received in MESSAGE_ID objects which when received refreshes all the reservations identified in the list. This allows a node to periodically send a single message to refresh all reservations to a neighbor node instead of keeping track of individual timeouts for each reservation and sending one message per reservation. The RSVP Refresh Overhead Reduction extension has many other uses, for example it also supports reliable RSVP message delivery.

3.3.4 Signaling extensions for FAs

There are two different FA-LSP signaling procedures that have been standardized, one for numbered FA-LSPs and one for unnumbered [33] [35]. The procedure for signaling numbered FA-LSPs is:

1. The ingress node allocates a /31 IPv4 address and sets it as the tunnel source address in the Sender Template object of a RSVP Path message.

2. Upon receiving a Path message the egress node sets up the LSP as usual and allocates the other address in the /31 space (this is possible because the /31 netmask contains exactly two addresses).

3. When reservation is complete the ingress node announces the LSP as a TE-link in the IGP using the local /31 address.

4. The egress node examines every TE-link advertisement looking for links with a Link-ID matching its own Router-ID, if so – then it tries to match the interface addresses with the /31 address which was the source in the Sender Template object.

5. If a matching link is found the egress node also announces the LSP as a TE-link using the other /31 address as local interface address.

This procedure has several drawbacks; since the ingress node cannot indicate that the LSP should be announced as an FA-LSP and the ingress node's Router-ID is not carried to the egress node, therefore the egress node must wait for a routing advertisement to be able to realize that the LSP should be announced as an FA-LSP. It may potentially also have to examine many local LSPs for each new TE-link advertisement it receives which consumes processing power. The procedure used for unnumbered FA-LSPs is faster and cheaper in terms of processing power. The unnumbered FA-LSP signaling procedure defined in RFC 3477 is [35]:

1. The ingress node allocates a local unnumbered interface identifier. This identifier along with the ingress node's Router-ID is sent in a LSP_TUNNEL_INTERFACE_ID[2] object (see figure 3.13). The LSP_TUNNEL_INTERFACE_ID object is inserted in the Path message.

2. The egress node proceeds with LSP setup as usual and processes the LSP_TUNNEL_INTERFACE_ID object which indicates that this LSP should be announced as an FA-LSP. It stores the ingress node's Router-ID and interface identifier and allocates its own local interface identifier. The new identifier and the egress node's Router-ID are inserted in a new LSP_TUNNEL_INTERFACE_ID object which is sent back to the ingress node in the Resv message.

3. The egress node announces the LSP as an unnumbered FA-LSP in the IGP. Since it knows the ingress node's Router-ID and local interface identifier it can assign a Link-ID to the link and local/remote interface ids.

4. When LSP setup is complete the ingress node announces the LSP as an unnumbered TE-link.

This method is both faster and computationally cheaper than the numbered approach. Both ingress and egress nodes can announce the FA as soon as reservation is made and there is no need to examine newly announced TE-links for potential FA-LSPs. A current IETF draft suggests an extension of the LSP_TUNNEL_INTERFACE_ID object. This extension adds (among other things) numbered interface support and deprecates the previous numbered signaling method [36].

3.3.5 End-to-End Label Allocation

If the data plane does not support the MPLS notion of swapping labels, as is the case in for example optical networks without wavelength conversion, then a single label has to be used over the entire LSP. A label used over the entire LSP is called an end-to-end label, GMPLS offers several ways to allocate such a label:

[2]Not to be confused with the LSP_TUNNEL_IPv4 and LSP_TUNNEL_IPv6 objects from RFC 3209 [16]

0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31
LSR's Router ID
Interface ID (32 bits)

Figure 3.13: The LSP_TUNNEL_INTERFACE_ID object which indicates that an LSP should be announced as an FA-LSP. It contains the senders Router-ID and a unnumbered interface identifier. The LSP_TUNNEL_INTERFACE_ID object is allowed in Path and Resv messages [35].

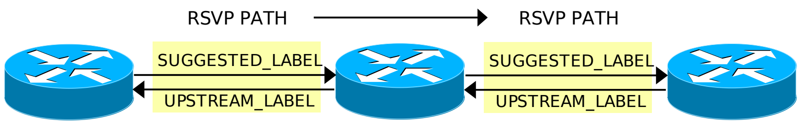

Figure 3.14: End-to-end label setup using UPSTREAM_LABEL and SUGGESTED_LABEL.

- The UPSTREAM_LABEL and SUGGESTED_LABEL objects
- The LABEL_SET object
- The Explicit Label Control ERO extension

Using the UPSTREAM_LABEL, and using the same label, across the entire LSP one can force the label to be used in the upstream direction. Since the UPSTREAM_LABEL must be accepted or an error transmitted – if the downstream node cannot use it (i.e., is not possible for the downstream node to use another label in the upstream direction than the received one). In the same way the SUGGESTED_LABEL, *suggests* what label should be used to send data to the downstream node (see figure 3.14). However, to force a label allocation using the SUGGESTED_LABEL object would go against standards, as RFC 3473 states that "Errors in received Suggested_Label objects **must** be ignored. This includes any received inconsistent or *unacceptable* values." (emphasis mine). If one wants to use this approach either the standards should be changed or a new object introduced (DOWNSTREAM_LABEL).

Another option to limit label selection is by using the LABEL_SET object (this is actually the standard way [37]). This object (see figure 3.15) can be used to define inclusive or exclusive labels or label ranges. Several of these objects can be sent in order to define the permitted label space from which the downstream node **must** choose a label to send upstream in the Resv message. The LABEL_SET object can be used instead of the hypothetical DOWNSTREAM_LABEL object without breaking the standards; however it is more complicated to process.

A third option is the Explicit Label Control extension to the Explicit Route Object. This extension enables the ingress node to decide upon the labels for each hop in the Explicit Route Object, by adding one or two Label ERO subobjects

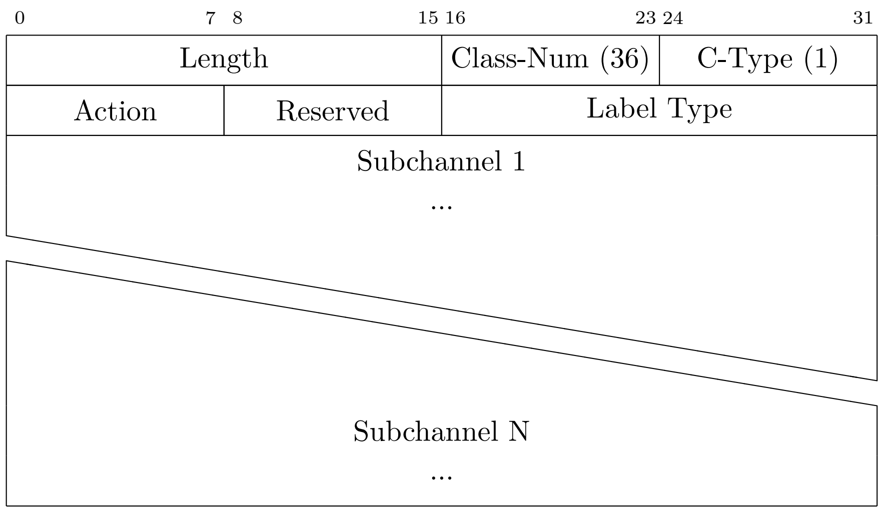

Figure 3.15: The LABEL_SET object, can be carried in Path messages. By setting different values in the "Action" field the object can specify inclusive or exclusive labels or label ranges. The label range or list is specified in the "Subchannel" fields [17].

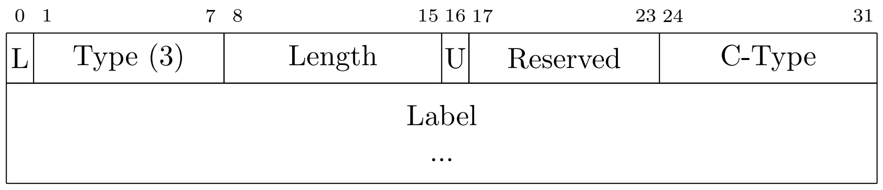

Figure 3.16: The Label ERO sub object. The L-bit must be set to 0, the Type field is 3 and the C-Type copied from the label. The U-bit indicates if the label is an upstream or downstream label [17].

following a IP address or interface identifier in the ERO. The Label ERO subobject (see figure 3.16) can carry both upstream and downstream labels.

3.4 Literature Study Conclusions

Based on a literature study some conclusions of what is appropriate and necessary to implement in order to reach the goal of cross region LSP setup over Ethernet and Optical regions can be drawn. Due to the differences in bandwidth granularity between these different regions the integration should be done by using FA-LSPs, preferably triggered, since this seems to be the most suitable solution for this scenario. One of the methods for enforcing end-to-end labels should be used, the Suggested label is the simplest one (however, it breaks standards). Since the standardized procedure for creating numbered FA-LSPs requires both an IP address allocation scheme (since the addresses are global) and there are interactions between OSPF-TE and RSVP-TE, the simpler unnumbered FA-LSPs procedure is preferred. Creating an FA-LSP consists of a few steps:

1. **Creating an LSP in the LSC region** The LSP should be able to traverse both OXCs and ROADMs and it should support end-to-end labels since the nodes may not be able to perform wavelength conversion.

2. **Identifier allocation** Unnumbered interface identifiers should be allocated and included in the LSP setup using the LSP_TUNNEL_INTERFACE_ID object.

3. **TE-link announcement** When LSP setup is completed the OSPF-TE daemon should announce a new TE-link. This requires communication between the RSVP-TE and OSPF-TE daemon as well as a method for dynamically creating TE-links in the OSPF-TE daemon.

Once the FA-LSP has been created the next step is to use it – just as any other TE-link in the network. This needs additional support, specifically:

1. **Control plane traffic forwarding** Control plane traffic utilizing the FA-LSP has to be forwarded to the tail-end LSR of the LSPs. Information about the FA-LSP's TE-link identifiers and the tail-end of the LSP needs to be stored so that traffic can be forwarded to the correct LSR.

2. **Data plane binding** When creating switching rules information mapping the FA-LSP TE-link to the lambda reserved by the LSP is necessary in order to forward traffic into the correct LSP.

Both the control and data plane mappings should be created when the FA-LSP has been setup.

Chapter 4

Software Implementation

4.1 DRAGON Software Suite

The Dynamic Resource Allocation via GMPLS Optical Networks (DRAGON) software suite is an open source GMPLS suite developed by a number of universities and others in the U.S. with support of the U.S. National Science Foundation [38]. It consists of a number of UNIX daemons for RSVP-TE, OSPF-TE, and Path Computation Element, and other functions.

The DRAGON version used was the release made in June 2006. This release is available at the DRAGON homepage: `http://dragon.east.isi.edu/twiki/bin/view/Main/WebHome`. The following DRAGON components used in the Multi-Layer Control Plane project:

OSPF-TE The OSPF daemon is an extended version of the OSPF daemon provided by the Zebra routing daemon project. It has been extended to support distribution of Opaque LSAs with TE information. It also plays the role of part of the LMP protocol which is not implemented in this suite. It is written in C.

RSVP-TE The RSVP daemon is based on an open source RSVP daemon called KOM-RSVP. This daemon is written in C++.

PCE This is provided by the NARB daemon created by the DRAGON project. It can among other things be used to perform path calculation. It is written in C++.

Dragon A daemon which acts as the endpoint of and LSP, this daemon can be used to initiate RSVP signaling. It is written in C.

In figure 4.1 the relationships between these daemons is shown. The different daemons communicate with each other using Inter-Process Communication (IPC). The communication between the NARB and the RSVP-TE daemon is IP based. The RSVP-TE daemon has a number of built-in Data Plane Controller modules which is

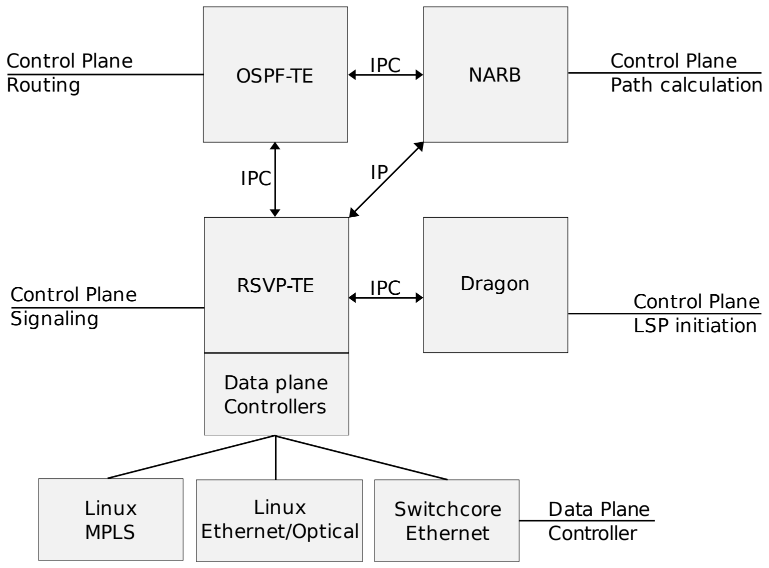

Figure 4.1: The different DRAGON daemons, their use and the relationship between them.

used to communicate with data plane hardware using Telnet, SSH or Sredir (allows a telnet session via a serial port [39]).

4.2 Emulating an Optical region

Creating LSPs in the optical region as well as implementing border node functionality requires some sort of programmable optical switches, either OXCs or ROADMs. However, none of the OXCs or ROADMs available in the MLCP project are programmable. To overcome this problem an emulated optical data plane has been developed. The emulated data plane uses common PC hardware and standard Linux kernel features.

4.2.1 Model

In order to emulating a WDM system using common hardware (such as Ethernet components and standard PCs) we make a a number of assumptions. For example it is assumed that the optical inputs and outputs are ”colored”, i.e. the optical signal is well defined in a specific frequency when arriving at or leaving the WDM component. Specifically we assume that they conform to ITU-T G.694.1 and G.694.2.

The developed WDM emulator uses the Linux kernel and common network tools. The emulation is rather superficial in that it does not try to emulate physical characteristics of WDM systems such as interference or errors at the optical level.

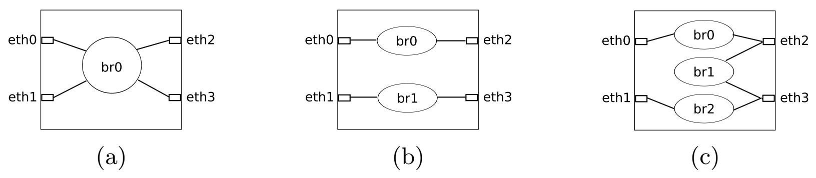

Figure 4.2: Some possible configurations of the Linux Bridge Controller. In figure (a) interfaces eth0-4 are bridged by a single bridge. In figure (b) interfaces eth0 and eth2 are bridged by one bridge and interfaces eth1 and eth3 by another bridge. In figure (c) the same connections as in (b) are made with the addition of another bridge (br1) that connects eth2 and eth3.

However, it simply allows for traffic to be switched by using GMPLS optical labels. Details on how the emulation operates is described in the following sections.

4.2.2 Linux Bridge Controller

The Linux kernel has support for connecting Ethernet segments by acting as an Ethernet bridge based upon an implementation of a subset of the ANSI/IEEE 802.1D MAC Bridges standard [40]. By enabling these features in the kernel at compile time, or loading them as kernel modules into a running system, one can create and control "virtual" bridges using the command line interface "brctl". Using brctl one can create several bridges, connect interfaces to them, turn on or off the Spanning Tree Protocol (STP) on individual bridges, and adjust STP parameters (see appendix A for details). Interfaces can be connected to several bridges at once, enabling all the configurations shown in figure 4.2. Since lambdas are emulated as regular Ethernet point-to-point links we can emulate bi-directional lambda switching by creating a bridge and adding two interfaces to it, thereby switching traffic from interface A (lambda X) into interface B (lambda Y) and vice versa.

4.2.3 Ethernet bridge tables

While the Linux Bridge Controller is sufficient for emulated bi-directional lambda switching it cannot perform uni-directional emulation and is cumbersome to automate since a new bridge has to be created for each switching decision. Ethernet bridge tables, or ebtables for short, are a firewalling tool that can filter traffic passing through a Linux Bridge [41]. Ebtables are similar to the standard Linux IP firewall tool "iptables", as they use a similar syntax. Ebtables can work together with iptables and other Linux filtering tools (such as arptables). Ebtables makes it possible to filter traffic based on Ethernet link layer information such as source/destination MAC address, incoming/outgoing interface, VLAN IDs, and more. Ebtables can also perform both source and destination Network Address

Translation (NAT) on the link layer. For more information on Ebtables see appendix B where a summary of ebtables commands can be found.

Ebtables are useful in the optical emulation model as it gives us the ability to limit traffic flows between interfaces attached to a Linux Bridge. If traffic enters a Linux Bridge and the destination MAC is unknown to the bridge, the bridge will send the data on all the other attached interfaces. Using ebtables this broadcast can be limited so that traffic entering on interface A is only allowed to leave through interface B. Filtering, NAT, and the other operations Ebtables support is done by inserting rules into ”chains” that are put into one of Ebtables’ three tables: “broute”, “nat”, and “filter”. The ”filter” table controls filtering operations, such as allowing traffic to enter from for example interface A and only leave through interface B. The filter table has three different chains: “INPUT”, “OUTPUT’, and “FORWARD”. The INPUT and OUTPUT chains apply rules to Ethernet frames destined *for* the node and frames *generated by the node* respectively. The FORWARD chain apply rules to frames forwarded by the node, by setting the default policy to DROP we can prevent any traffic from passing through the bridge. In order to switch traffic from interface A (lambda X) to interface B (lambda Y) uni-directionally, the following rule is inserted in the FORWARD chain of the filter table:

```
# ebtables -P FORWARD DROP
# ebtables -A FORWARD --in-if interfaceA --out-if interfaceB -j ACCEPT
```

The first command sets the default policy on the FORWARD chain to DROP, traffic not matching any of the inserted rules will be sent to the target DROP and thus not forwarded. The second command appends (-A) a rule to the chain FORWARD, if the incoming interface matches $interface_A$ and outgoing interface is $interface_B$, then the frame is sent to target (-j) ACCEPT and thus forwarded. To switch a bi-directional lambda the same rule is inserted again, but with the interface names are reversed (to establish forwarding in the reverse direction). By using ebtables to perform the switching in this manner all involved ”lambda interfaces” can be inserted into a single bridge and switching is done by adding and removing rules to ebtables, resulting in a cleaner solution that also allows for more advanced switching than simply using Linux Bridges alone. Since ebtables also can filter on the VLAN ID it is simple to handle the border node case where Ethernet frames on a particular VLAN and a particular destination MAC address enters from an Ethernet interface (called “eth0”) and should be forwarder to a specific lambda (here we assume that the outgoing interface is named “lambda”). The ebtables command for inserting such a filtering rule is:

```
# ebtables -A FORWARD --in-if eth0 --out-if lambda -p 802_1Q
         --vlan-id VID -j ACCEPT
```

4.2.4 Mapping between Control and Data plane

In order to identify which DWDM frequency or CWDM wavelength an interface represents, a mapping between each interface and a frequency or wavelength is necessary. Since the same frequency or wavelength may be emulated on several links each “lambda interface” must also be mapped to a control plane interface. These mappings are done by renaming the interfaces. The standard Linux “ip” command can be used to rename interfaces from their normal names (such as “eth0”, “tun0” etc.) to an ASCII string of a maximum of 15 characters. By naming control plane interfaces “control-ID” and lambda interfaces “ID-frequency” one can map control to data plane interfaces by the ID and frequencies or wavelengths to data plane interfaces. For example a control plane interface may be called “control-45ab”, the interface representing frequency 193.1 THz on the data plane would be named “45ab-1931000”. For DWDM the frequency part of the ”lambda interface” name is a 7 digit number, since the label format allows increments of 12.5 GHz the value must be in tenths of a GHz. For CWDM the frequency is the wavelength in nanometers, a 4 digit number.

4.2.5 VTund

Using a physical Ethernet interface for each lambda would require many interfaces if a WDM supporting a lot of lambdas is to be emulated. By using virtual interfaces the number of physical interfaces required can be limited. Linux supports a number of different IP tunnel interfaces for example Generic Routing Encapsulation (GRE), IP-in-IP, and IPsec tunnels [42] [43] [44]. However, these can not be used to carry VLAN tagged Ethernet frames since they only tunnel IP traffic. The ”Universal TUN/TAP” tunnel drivers included in Linux kernel version 2.6 and above allow the creation of several kinds of tunnels; IP tunnels, Ethernet tunnels, PPP tunnels, and others [45]. By using Ethernet tunnels as ”lambda interfaces”, 802.1Q VLAN tagged Ethernet frames can be sent through the emulated lambdas. Setup of tunnels is handled by the VTun userspace tool which is started in daemon mode on the server. A client can connect to the server and request the creation of a pre-configured tunnel. The tunnel is then created on both the client and server. In addition to several different tunnel types’ many other settings are supported, for example the tunnels can be compressed, have traffic shaping, use UDP or TCP for transporting data, be password protected, etc. More information about VTun can be found on the VTun homepage [46] and in appendix C where an example configuration can be found.

The VTun daemon can also execute arbitrary commands when setting up tunnels, this makes it easy to automatically add the ”lambda interfaces” to a bridge when the tunnels are created, and remove them from the bridge if the tunnel is torn down.

4.2.6 OXC and ROADM Emulation

Using the tools discussed above a simple emulation of OXCs and ROADMs can be realized. For example, if we want to emulate a OXC with two bi-directional fiber optic connections carrying ten lambdas (as in figure 4.3), each the following steps must be performed:

1. Create 2 control plane interfaces to neighbor nodes using for example GRE tunnels or physical Ethernet interfaces. Name them “control-oxc1” and “control-oxc2”.

2. Create 20 data plane lambda interfaces, 10 to each neighbor. Name the interfaces going to the same node as “control-oxc1” to “oxc1-1931000” ranging to “oxc1-1941000” (for emulating 193.1 THz to 194.1 THz lambdas). Apply the same naming to the data plane interfaces going to the node that the “control-oxc2” interface is attached to (but change “oxc1” to “oxc2” in the name).

3. Create a Linux Bridge and attach the lambda interfaces to it, set the default FORWARD policy to DROP.

With this configuration in place lambdas can be switched by inserting ebtables rules, if we for example want to switch lambda frequency 193.1 THz on interface oxc1 to lambda frequency 193.6 THz on interface "oxc2" two rules are inserted:

```
# ebtables -A FORWARD --in-if oxc1-1931000 --out-if oxc2-1936000 -j ACCEPT
# ebtables -A FORWARD --in-if oxc2-1936000 --out-if oxc1-1931000 -j ACCEPT
```

Traffic from interface oxc1-1931000 will now be forwarded into oxc2-1936000. This behavior emulates a OXC, if we want to enforce the reachability constraints imposed by ROADMs we can simply add more bridges. If we want to emulate a ROADM with two tributary ports and two trunk ports and enforce the reachability constraints that tributaries cannot communicate directly, but can only communicate with trunk ports, we create bridges in the layout used in figure 4.2c. In this case interfaces "eth0" and "eth1" would be tributary ports and "eth2" and "eth3" trunk ports. The ebtables rules used to switch lambdas look the same, but since the interfaces "eth0" and "eth1" are not connected to the same bridge data cannot flow between them.

4.2.7 Emulating Other Technologies

Since the emulation uses regular interfaces in Linux, this emulation could be extended with tools such as netem [47] [48]. Netem is a Linux tool for adding a specific queuing discipline for an interface. This can be used to provide programmable delays, packet loss, etc. Thus this could be used to emulate some types of optical error sources. Netem is implemented as a traffic control (tc) queuing

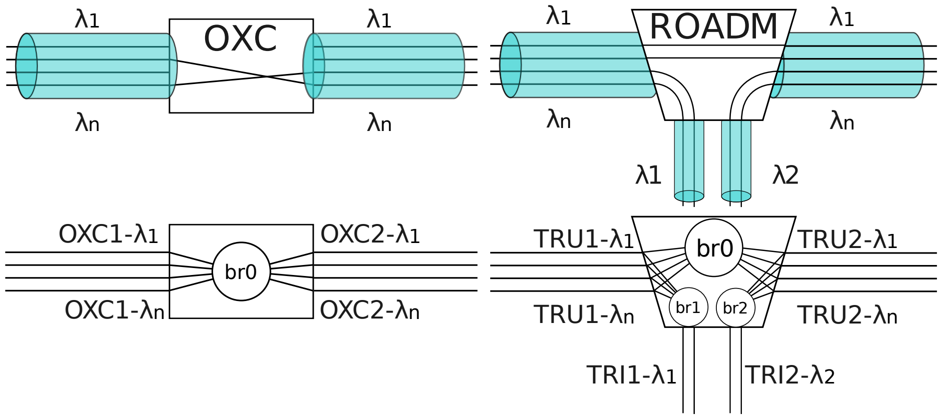

Figure 4.3: Models of OXCs and ROADMS and how they are emulated. The names of the OXC links have no special meaning, on the ROADM TRU refers to a trunk and TRI refers to a tributary.

discipline (qdisc). All interfaces in a Linux system are per default assigned a (simple) queueing discipline, however, this can be changed to more complicated disciplines. One can for example limit traffic to or from specific IP addresses to a certain bandwidth. These traffic control tools can be used to apply the bandwidth reservations made by RSVP-TE to the data plane (see relevant HOWTO documents for more information [49] [50]). However, no work on optical error emulation nor actual data plane resource reservation with tc has been done in this thesis.

The emulation model could also be extended to emulate other network technologies such as IEEE 802.1ad (Q-in-Q) Ethernet by using the interfaces to represent VLANs instead of lambdas (see section 6.3.

4.2.8 Extensions of OSPF-TE

A number of changes to the DRAGON OSPF-TE daemon were necessary in order to make it possible to dynamically announce new TE-links. Most of the changes made to the OSPF-TE daemon were required in order to circumvent the design of the daemon itself. A major limitation of the daemon is that each TE-link is coupled to a regular OSPF link in a 1:1 relation (see appendix D for details). This means that it is not possible to have several TE-links controlled by a single control channel using this daemon. After auditing the code it was concluded that a thorough rewrite of the code to separate control and data plane links was not possible within the assigned time. Therefore instead of separating these planes a configuration option called ”fa-lsp” was implemented. The rationale behind this modification are:

1. For an interface to be handled by the DRAGON OSPF daemon it must be a ”real” interface[1], it must exist in the operating system, have an IP address and

[1]So called aliased or secondary interfaces cannot be used. These are virtual interfaces that are

be in state UP. (Here we have assumed that the operating system is Linux.)

2. TE-link information is coupled to a regular OSPF interface, therefore adding a TE-link causes the regular OSPF interface to be distributed in the network and inserted in the routing tables on all participating nodes.

What the ”fa-lsp” option does is to turn off or on distribution of the regular OSPF LSAs when distributing TE LSAs. This makes it possible to add TE-links to the network without polluting the routing tables or transmitting any superfluous routing data. The ”fa-lsp” option can be set on a single OSPF-TE link in the configuration file or during operation via a command-line interface. The different arguments this option can take are:

OFF With ”fa-lsp” set to OFF the TE link will behave as usual, OSPF will emit link-state advertisements for the link and for the coupled TE-link.

ON With ”fa-lsp” set to ON the link will behave as a FA-LSP and only the TE-LSAs will be transmitted. **No** regular OSPF LSAs about this interface will be sent.

HOLD When set to HOLD, no LSAs will be sent at all, neither for the OSPF link nor the coupled TE-link.

When using this option to announce FA-LSPs (i.e. dynamically adding TE-links to the network) one needs a ”real” interface per FA-LSP, this can be provided by the ”dummy” Linux kernel module. Using this kernel module one can create ”dummy-interfaces”, i.e., fake network interfaces which do not correspond to an actual physical interface. These interfaces can be assigned IP addresses and set to state UP which allows OSPF to pick them up and transmit routing LSAs about them as if they were real network interfaces. By configuring a number of these (say 10) dummy-interfaces and setting their ”fa-lsp”option to HOLD in the configuration file one has 10 interfaces ready to be used as FA-LSP ”holders”. Assigning FA information to one of these interface and then setting the ”fa-lsp” option to ON creates a new TE-link in the network.

There is a simple communication protocol running between the OSPF-TE and RSVP-TE daemon. The protocol enables the RSVP-TE daemon to retrieve information from the Traffic Engineering Database about interfaces, routes, etc. The existing TLV protocol was extended to retrieve the TE-link information necessary to create a FA-LSP. When RSVP-TE sends the information the OSPF-TE daemon loops through all configured interfaces and tries to find an interface in the HOLD state. If one is found, then it sets the TE information of this interface and changes

used to receive traffic on several IP-addresses on a single physical interface. Their names are based on the physical interface combined with an additional number, for example ”eth0:1”. If the inability to use these types of interfaces is due to that the daemon is unable to parse interface names with ”:” or a more fundamental reason is not clear. Had the daemon been able to use aliased interfaces they would have been a good alternative to the ”dummy” interfaces discussed later.

it to the ON state and triggers a LSA update which to disseminates the TE information.

Additionally, a function for retrieving the switching type and encoding of the near and far ends of a TE-link was added to the protocol as well. This information is necessary when determining how to interpret labels and when determining whether or not the node is a domain border node.

For an example of a OSPF-TE configuration file containing among other things the ”fa-lsp” option and a summary of functions added to the OSPF-TE daemon see appendix D.

4.2.9 Extensions of RSVP-TE

While the changes to the OSPF-TE daemon were rather straightforward to implement, the RSVP-TE daemon changes took a lot more time and effort. This is probably mostly due to the fact that OSPF-TE daemon has been a part of the Zebra project and has been more thoroughly reviewed [51]. The RSVP-TE code was, at least for me, a lot harder to understand. Additions to the RSVP-TE daemon include:

- A data plane controller for the emulated optical OXC/ROADM
- Handling of the LSP_TUNNEL_INTERFACE_ID object (used to indicate that this LSP should be used as a FA-LSP)
- Handling of the SUGGESTED_LABEL RSVP-TE object (used for end-to-end label allocation)
- Two new GMPLS label formats (for Ethernet and WDM labels) based on Internet drafts [52] [30]
- IPC functions to announce and remove FA-LSPs in the OSPF-TE daemon

The Optical labels added are the DWDM and CWDM labels discussed in section 3.1.2. The Ethernet labels are based on a Internet draft from year 2005 which has expired (and has not been replaced by either a new draft or RFC). This label contains a VLAN ID and a destination MAC address (see figure 4.4). Both these labels are sent as a generalized label with a length of 4 and 8 bytes respectively. The functions for receiving, creating, and sending these types of labels was added to the RSVP-TE code; along with functions to decode these labels. By ”decoding” we mean the ability to convert for example a DWDM Optical label to the frequency it represents. While encoding means creating a DWDM label from a given frequency.

The LSP_TUNNEL_INTERFACE_ID object (discussed in section 3.3.4) was also implemented. This object requires more complicated handling (than the labels) since it should be treated differently depending on whether the node is ingress, egress, or a ”transit” node (i.e. neither ingress nor egress). In the ingress case this object should already be in the Path message received from the *dragon* daemon.

0 3	4 15	16 31
0000	VLAN ID	MAC (highest 2 bytes)
MAC Address		

Figure 4.4: Proposed Ethernet label format [52].

If it is present in the Path message, then the node chechs whether or not it is the ingress node, and if so sets a flag in the Path State Block (PSB). This flag is used by the data plane controller to see whether it should create a FA-LSP identifier → actual TE-link binding. When a Resv message has been received and the data plane configuration is complete, then the LSP should be announced as an FA-LSP – but only if the node is either an ingress or egress to the LSP and the LSP_TUNNEL_INTERFACE_ID object is present. If this is the case, then the daemon will try to announce the LSP via the IPC functions to OSPF-TE daemon. If the announcement is successful, then a reference to the LSP is inserted in a map along with the far-end and near-end TE link identifiers. This map is used when forwarding control plane messages; for example, if the next hop in the ERO of a Path is found in the mapping, then the message is forwarded to the far-end control plane address instead of to a neighbor.

The SUGGESTED_LABEL object (discussed in section 3.3.5) was implemented. The SUGGESTED_LABEL object is a generalized label with Class-Num 129 and the C-Type of the label it contains. This object is carried in RSVP Path messages and passed along to the specific data plane controller instantiated. If the controller cannot handle the label, then it may ignore it or abort the setup with a PathError message (in our implementation; the standards says it should only be ignored). Since the SUGGESTED_LABEL is used to enforce end-to-end labels an error would typically be generated if the resource is busy or does not exist on the particular data plane (i.e. only on optical or Ethernet data plane).

The largest addition to RSVP-TE is a data plane controller module able to act as a controller for the emulated optical plane, ”emulated” Ethernet, and as an emulated optical border node. There are several different data plane controllers implemented based on a abstract controller interface (i.e. a common superclass). Depending on the nodes configuration the daemon instantiates a suitable controller. The interface consists of a number of functions for operations which are common among all controllers (such as allocation of new labels, binding of {label, interface} pairs (i.e. create a switching rule), removing bindings, checking for free resources, etc.)
Typical calls to the controller during LSP setup are:

- PATH message received

1. allocateFreeInLabel() is called with a label and an interface identifier. It either allocates a new upstream label or checks that upstream label is

available (if the LSP is bidirectional). In the emulated optical case a label must be provided since end-to-end labels are required. The controller converts the given label and interface pair to an identifier (e.g. oxc1-1930000) and checks that the interface exists and is not already bound. If it is bound, then an error is returned with results in the transmission of a PathError message and further signaling is aborted. If the checks are successful, then the controller returns the label that it tested which is put into a new Path message and sent downstream.

- RESV message received

  1. allocateFreeInLabel() is called. This checks that the received (downstream) label is available, if the LSP is bi-directional then the upstream label is checked as well. The procedure is identical to the one described above.
  2. InstallInLabel() and InstallOutLabel() are called. These have serve no purpose in the LSC-Linux controller since the switching rules are created in a single step during bindLabels(). However, other controllers such as MPLS need to perform additional operations before binding.
  3. bindLabels() is called with two {data plane identifier, label} pairs which should have a uni-directional switching rule created between them. The interface identifiers and labels are converted to emulated identifiers and an ebtables rule is created, allowing traffic to flow – uni-directionally – from one interface to another.

However, the above is a very simplified description of what actually goes on during these calls. Since the controller supports both Ethernet and emulated optical links all operations first have to determine how to interpret any received label. This is done via an IPC call to the OSPF-TE daemon which requests the Interface Switching Capabilty Descriptor for the corresponding TE-link. By examining the Switching Capability and Encoding fields the controller can conclude that the label is for example a WDM label (Switching Capability LSC, encoding lambda). The controller then calls a corresponding internal function which handles that specific label type. In order to support FA-LSPs the controller cannot assume that the data plane interface identifier refers to an actual interface it might refer to an FA-LSP. When performing label binding the controller checks if the LSP is supposed to be announced as an FA and if the node is either the ingress or egress of the LSP – if so an entry in a map is created, pairing the FA-LSP unnumbered interface identifier with the actual data plane interface. Each function then looks for the data plane identifier in the map and calls a corresponding FA-LSP function (bindLabelsFA() for example). For more detailed information about the data plane controller see appendix E.

Chapter 5

Testing and Verification

The implementation has been tested in several ways. The parts of the code that could be tested independently (i.e. without a complete Traffic Engineering Database etc.) were tested with regression tests which calls the different functions and makes sure that they perform as expected. Unfortunately this was not possible for much of the implementation concerning signaling LSPs. These part were tested in a testbed by manually signaling different LSPs and verifying that the they were correctly set up. The final test was to create a multi-region LSP using both LSP conversion (MPLS to Ethernet) and FA-LSP (Ethernet to WDM).

5.1 Testbed

The testbed at Acreo has been used to test multi-region GMPLS. This testbed contains a number of different kinds of nodes. Roughly half the nodes in the network is virtualized using VMWare running on two servers. The physical part of the network consists of:

Juniper IP/MPLS routers In the network there are three different Juniper M-series IP/MPLS routers: M5, M7i, and M10i [53]. These are called JR1 to JR3 in figure 5.1.

Linux IP/MPLS routers There are three Linux (Ubuntu 6.06.1) machines running, using a kernel version 2.6.15.1 patched with MPLS support. They each run on a Dell Dimension 5150 with a 3 GHz CPU and 512 MB RAM. These machines are called LR1 to LR3 in figure 5.1.

Switchcore Ethernet switches There are three Switchcore Xpeedium2 RD1100 Ethernet switches in the network. Each of these are controlled by a computer identical to the Linux IP/MPLS routers, except that they do not have a patched kernel. The Ethernet switches are reference implementations, this combined with the acquisition of Switchcore by eSilicon makes it difficult to find documentation of these switches. The switches are called SwCE1 to SwCE3 in figure 5.1.

Lumentis optical network elements There are three optical cross-connects in the network, called T1 and T3 in figure 5.1. They consist of a Mentis 3000 chassi with several modules such as TP-MR2500 transponders, AD1AB add-drop multiplexers, and OXC8 cross-connect modules[54]. Lumentis has since these were bought been acquired by Transmode which is what I refer to these nodes as.

The virtual network is hosted by two computers, a HP xw8400 server with a quad-core Intel Xeon x5355 processor at 2.66 GHz and 4 GB RAM, and a Dell Optiplex GX620 dual-core Pentium processor at 3.00 GHz and 3 GB RAM. These computers are both running Ubuntu as their operating system (version 7.04 and 6.10 respectively) and VMWare server (version 1.0.3) to host virtual machines. The virtual machines are all running Ubuntu 6.06.1 as their operating system, the virtual optical nodes (vROADM1-3, vEC1-3, and vOXC1) are hosted by the HP computer while the others (vE1-3 and vR1-3) are hosted by the Dell computer.

The Transmode nodes were not operational during testing and are therefore missing from figure 5.2. This figure was generated from a dump of the Traffic Engineering Database (retrieved with an ”expect” script) using GraphViz and a simple C program which parsed the dump and generated a GraphViz source file [55].

All nodes except the Switchcore Ethernet switches consist of both control and data plane running on a single (sometimes virtual) machine. Each Switchcore Ethernet switch has a Linux PC acting as the control plane which is communicating with the data plane (the Switchcore switch) via serial cable. The control plane links shown in figure 5.1 consists of either VLAN-interfaces or GRE tunnels which are used to create a topology on top of the backbone used to connect all nodes.

Control plane IP addresses and data plane identifiers can be seen in figure 5.1 and 5.2 respectively. In table 5.1 the Router-ID and node type for all testbed nodes are summarized.

5.2 Multi-Region LSP Setup

In order to test multi-region LSP setup in the testbed two LSPs need to be setup and their functionality tested. First an FA-LSP over the optical layer and then a ”client” LSP that utilizes the FA-LSP.

The FA-LSP starts on node one of the emulated Ethernet/Optical border nodes and is add/dropped into one of the emulated optical ring networks. The emulated optical cross-connect switches the LSP onto the other ring network and from there it is add/dropped to another emulated Ethernet/Optical border node. When the setup is complete the border nodes announce the FA-LSP in the IGP and FA-LSP setup is complete.

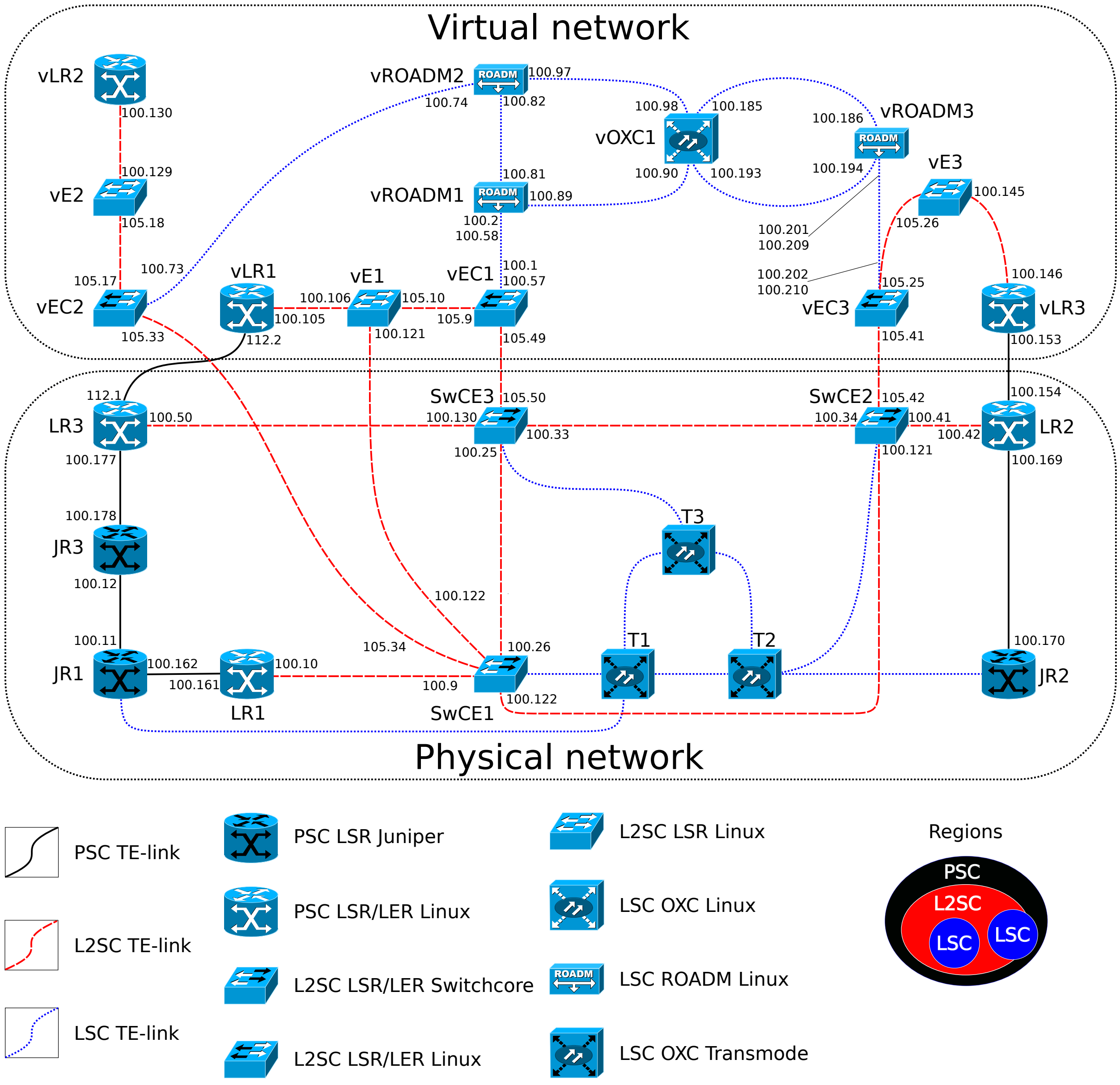

Figure 5.1: Overview of the testbed at Acreo. Different colored links indicate data plane switching capability, node icons indicate the type of node. The numbers next to each link is the least significant octets of the control plane interface IPv4 addresses, they have the prefix 172.16 which is not shown in order to save space. Links going to the Transmode physical optical OXCs lack IPv4 addresses since the nodes were not operational during testing. Some of the nodes that are connected to each other with multiple data plane links are shown with only one of the links.

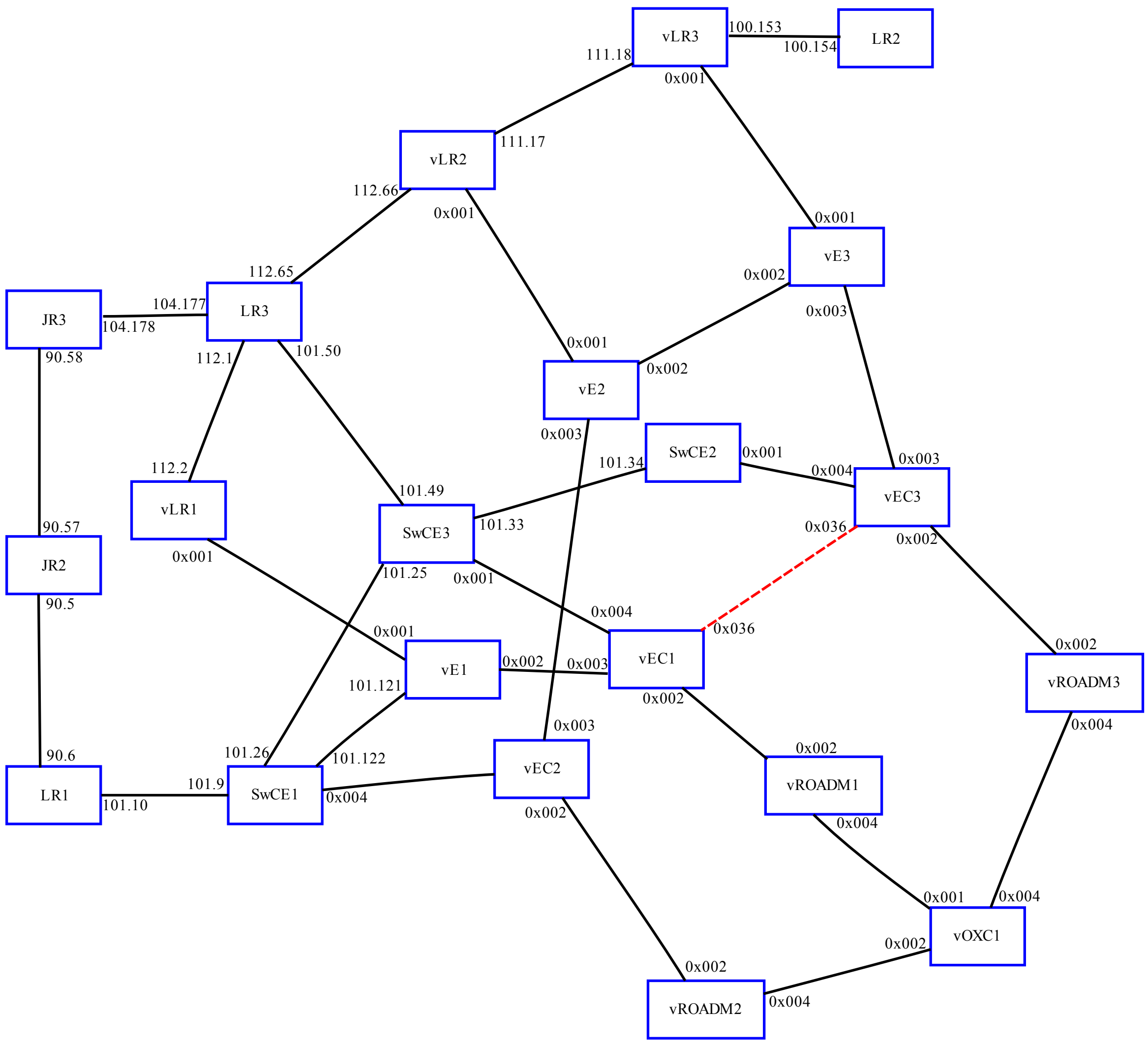

Figure 5.2: Overview of the data plane connections in the testbed. Link identifiers are shortened due to space. Adding 0x80000000 to unnumbered links (hexadecimal values) gives the actual value. Only the least significant two octets of numbered interfaces are show, they have the prefix 172.16. The dashed red link shows a FA-LSP between vEC1 and vEC3. In this figure the physical optical Transmode OXCs are missing since these nodes were not operational during testing. The nodes that are connected to each other with multiple data plane links are shown with only one of the links.

Table 5.1: Name, types, and Router-ID of the nodes in the testbed.

Node name	Type	Router-ID
vEC1	Virtual Ethernet/Optical border	172.16.100.230
vEC2	Virtual Ethernet/Optical border	172.16.100.231
vEC3	Virtual Ethernet/Optical border	172.16.100.232
vROADM1	Virtual ROADM	172.16.100.233
vROADM2	Virtual ROADM	172.16.100.234
vROADM3	Virtual ROADM	172.16.100.235
vOXC1	Virtual OXC	172.16.100.236
vE1	Virtual Ethernet	172.16.100.243
vE2	Virtual Ethernet	172.16.100.244
vE3	Virtual Ethernet	172.16.100.245
vLR1	Virtual Linux IP/MPLS	172.16.100.251
vLR2	Virtual Linux IP/MPLS	172.16.100.252
vLR3	Virtual Linux IP/MPLS	172.16.100.253
LR1	Physical Linux IP/MPLS	172.16.100.248
LR2	Physical Linux IP/MPLS	172.16.100.249
LR3	Physical Linux IP/MPLS	172.16.100.250
JR1	Physical Juniper IP/MPLS	192.168.1.1
JR2	Physical Juniper IP/MPLS	192.168.1.2
JR3	Physical Juniper IP/MPLS	192.168.1.3
T1	Physical Transmode OXC	192.168.80.1
T2	Physical Transmode OXC	192.168.80.17
T3	Physical Transmode OXC	192.168.80.25
SwCE1	Physical Switchcore Ethernet	172.16.100.240
SwCE2	Physical Switchcore Ethernet	172.16.100.241
SwCE3	Physical Switchcore Ethernet	172.16.100.242

When the FA-LSP has been created a client LSP will be created from a physical IP/MPLS node and then go through an MPLS/Ethernet border node and be converted from PSC to L2SC. Then the LSP will enter one of the Ethernet/Optical border nodes and utilize the FA-LSP (which is of switching type L2SC). After leaving the FA-LSP the path will pass through a L2SC node into another MPLS/Ethernet border node where the path will be converted back into type PSC and finally ends on a physical IP/MPLS node. When the client LSP is setup a IP/MPLS route is added on the ingress and egress nodes (a routing entry that forwards a /32 address into an MPLS tunnel). When the routing entries are setup an ICMP ping will be sent through the tunnel. If the data plane has been correctly configured and is functional through the whole path, then an ICMP ping reply should be received at the ingress node.

5.2.1 FA-LSP Setup

The FA-LSP path used is vEC1 ↔ vROADM1 ↔ vOXC1 ↔ vROADM3 ↔ vEC3. The FA-LSP is configured in the dragon daemon like:

```
edit lsp favec1vec3
 set source ip-address 172.16.100.230 lsp-id 32768 destination
            ip-address 172.16.100.232 tunnel-id 54
 set bandwidth gige_f swcap lsc encoding lambda gpid ethernet
 set direction bi
 set upstream dwdm label spacing 4 n 17 sign 1
 set suggested dwdm label spacing 4 n 17 sign 1
 set fa-lsp on
 add ero unnum 172.16.100.230 if_id 0x80000002
 add ero unnum 172.16.100.233 if_id 0x80000002
 add ero unnum 172.16.100.233 if_id 0x80000004
 add ero unnum 172.16.100.236 if_id 0x80000001
 add ero unnum 172.16.100.236 if_id 0x80000004
 add ero unnum 172.16.100.235 if_id 0x80000004
 add ero unnum 172.16.100.235 if_id 0x80000002
 add ero unnum 172.16.100.232 if_id 0x80000002
exit
```

This configuration first sets the ingress and egress Router-ID addresses as source and destination, with LSP-ID 32768 and Tunnel-ID 54 (the number 54 is just a random number, however the LSP-ID 32768 is necessary for TE-link dissemination). Then the correct LABEL_REQUEST object is created with a bandwidth request for 1 Gigabit, switching capability LSC, encoding lambda, and G-PID Ethernet. Upstream and suggested DWDM labels with a frequency of 191.4 THz are then added. The ”set fa-lsp on” option creates a LSP_TUNNEL_INTERFACE_ID based on the LSP-ID and Tunnel-ID. The last lines creates an eight hop Explicit Route Object, by combining table 5.1 and the link information from figure 5.2 each link can be identified.

This LSP is activated (by the Dragon command ”commit lsp favec1vec3”) and signaled. A packet capture of the signaling can be found in appendix G.1. When the signaling is complete the data plane has been configured. Examining the TE-link database one can see that a new link between vEC1 and vEC3 has been created. The new link can be seen in figure 5.2 as a dashed red line.

5.2.2 Client LSP Setup

The client LSP uses this path: LR3 ↔ vLR1 ⇌ vE1 ↔ vEC1 ⇔ vEC3 ↔ vE3 ⇌ vLR3 ↔ LR2, where the thick arrow (⇔) is the FA-LSP and the broken arrow (⇌) indicate LSP conversion. The dragon configuration for this LSP can be seen below:

```
edit lsp f32
 set source ip-address 172.16.100.250 lsp-id 32
     destination ip-address 172.16.100.249 tunnel-id 3032
 set bandwidth eth100M swcap psc1 encoding ethernet gpid ethernet
 set upstream mpls label 332
 set direction bi
 add ero ip 172.16.112.1
 add ero ip 172.16.112.2
 add ero unnum 172.16.100.246 if_id 0x80000001
 add ero unnum 172.16.100.243 if_id 0x80000001
 add ero unnum 172.16.100.243 if_id 0x80000002
 add ero unnum 172.16.100.230 if_id 0x80000003
 add ero unnum 172.16.100.230 if_id 0x80000036
 add ero unnum 172.16.100.232 if_id 0x80000036
 add ero unnum 172.16.100.232 if_id 0x80000003
 add ero unnum 172.16.100.245 if_id 0x80000003
 add ero unnum 172.16.100.245 if_id 0x80000001
 add ero unnum 172.16.100.253 if_id 0x80000001
 add ero ip 172.16.100.153
 add ero ip 172.16.100.154
exit
```

Note the FA-LSP in the ERO (underlined) is used like any other TE-link. This LSP is committed and signaled, a packet capture of the signaling messages can be found in appendix G.2.

When the reservation was complete an IP/MPLS route was added on the ingress and egress nodes. This is a simple host route which says that a IP address can be reached via a MPLS tunnel. A ping was then sent from the ingress node to the IP address routed via the MPLS tunnel. Ping replies were received and by capturing traffic along the path it was verified that the data packets were actually traveling in the LSP. Thus an operational multi-region LSP with LSP conversion and manual FA-LSP setup had been created.

Chapter 6

Evaluation

6.1 Performance

6.1.1 Data plane

By assigning IP addresses to the Linux Bridge Controller bridges at the ingress and egress of an ”optical LSP” IP packets can be sent across the LSP. To give a ballpark figure of the performance of the emulated optical data plane ICMP Ping packets can be sent to the other side and the time to receive a reply measured. This measurement will contain additional delays which is unrelated to the data plane, such as creating the ICMP Reply packet at the end node and so on. Four different data plane setups have been measured this way:

1. Only the two relevant rules are inserted in the Ebtables FORWARD chain.
2. The two relevant rules followed by a thousand additional nonsense rules.
3. Two relevant rules followed by ten thousand nonsense rules.
4. A thousand nonsense rules followed by the two relevant rules.
5. Ten thousand nonsense rules followed by the two relevant rules.

The optical LSP used during these tests was the same as mentioned in section 5.2.1, the rules were inserted on the machines vROADM1, vOXC1, and vROADM3, which are the only ones that forward traffic using ebtables in this LSP. The results of these tests can be seen in table 6.1. It is evident from the data that the position of the forwarding rules in the chain can have a significant impact on the time it takes to forward the traffic specified in the rule. As long as the rule is placed early in the chain the length of the chain does not make a big difference.

By performing the same test, using the same path, on the control plane gives an indication of the relative forwarding speed of the emulated data plane compared to regular IP routing on Linux. A number of ping packets was again sent from vEC1 to vEC3 but this time to and from the Router-ID instead of the IP address assigned

Case	Packets	Min (ms)	Mean (ms)	Max (ms)	Mean deviation (ms)
1	50	1.477	3.608	6.215	0.911
1	200	1.430	3.274	8.135	1.193
2	50	1.527	3.702	22.896	2.869
2	200	1.417	3.430	8.057	0.952
3	50	1.446	3.820	8.022	1.405
3	200	1.415	3.680	27.813	2.021
4	50	1.299	3.976	14.163	1.885
4	200	1.391	3.601	12.978	1.292
5	50	4.199	9.413	23.991	3.030
5	200	3.830	8.933	18.422	2.617

Table 6.1: Round-trip times for ICMP Ping in the data plane. The min, mean, and max round-trip times are specified in milliseconds, as is the mean deviation. The number of packets sent in each test case is shown in the packets column.

Packets	Min (ms)	Mean (ms)	Max (ms)	Mean deviation (ms)
50	0.588	1.499	4.532	0.753
200	0.673	1.499	7.594	0.786

Table 6.2: Round-trip times for ICMP Ping in the control plane. The different round-trip times are specified in milliseconds. Two tests were run, one with 50 packets and one with 200. The same path is used as in the data plane test.

to the bridge interfaces. That the packets followed the same path as on the control plane was verified by tracing the path in both directions using ”traceroute”.

As can be seen in table 6.2 the round-trip times are about half as long (compared to case 1 in the data plane) when using the control plane. This is not surprising since the data plane packets has to be packed in and out of additional tunnels, in and out of TCP packets as well as go through Ebtables.

6.1.2 Control plane

The performance of the control plane is for many reasons harder to measure. There are issues with synchronizing the clocks of the machines, and even getting the virtual machines’ clocks not to race ahead of time. Trying to get synchronized packet captures from multiple networks at the same time is another problem.

Despite these problems two packet captures has been performed to measure the processing time of Path and Resv messages on the different nodes as well as the total time for LSP setup. These two captures can be found in in appendix G.1 and G.2. The first capture consists of the FA-LSP setup discussed in section 5.2.1 and the second capture is of the client LSP discussed in section 5.2.2. The client LSP setup takes place in both the physical and virtual network and therefore consists of

Type	Mean P_{time} (s)	Standard deviation (s)
Path	0.03659	0.00278
Resv	0.51372	0.19615

Table 6.3: Mean ”processing time” and standard deviation of Path and Resv messages when setting up an FA-LSP.

merged data from two simultaneous Wireshark sessions.

The processing time (P_{time}) has been approximated as the time between capture of the previous packet and the current packet. This time is roughly the time spent in one node and contains the time spent in the operating system, transmission time on the link, etc. However, these external contributions should be negligible with large P_{time} values. When applying the same P_{time} calculation to ICMP ”Ping Request” packets the mean P_{time} for a sample of 50 packets was about 0.25 ms, a magnitude lower than the times measured for the RSVP-TE daemon. Since ICMP packet processing occurs in kernel space (and not user space as is the case with the RSVP-TE daemon) the ICMP data does not include time spent on for example context switching. However, it does give a hint about the amount of time contributed by external sources.

In table 6.3 the mean P_{time} for Path and Resv messages can be found. They show a small amount of time spent on Path messages while the Resv messages take about half a second to process. The long time spent during Resv is probably related to the data plane communication through Telnet/SSH which is not very optimised.

The packet capture of the client LSP is composed of two simultaneous captures which were not synchronized which makes it impossible to calculate correct P_{time} values for some of the packets. When possible the P_{time} has been calculated and it shows similar results to the FA-LSP setup, at least on the same hardware. The total time to complete FA-LSP setup with four hops was slightly more than two seconds while the client LSP with seven hops took approximately six seconds to complete.

6.2 Objectives

Most of the objectives set out in section 1.1 have been successfully met, it is possible to create FA-LSPs and set up LSPs from an Ethernet region over a WDM region (although this was done using an emulated WDM). However, not all objectives has been met, the ability for LSR to trigger FA-LSP setup on demand is a major objective that has not been met. The initial plan was to implement manual FA-LSP setup and following this to triggered setup if there was sufficient time, unfortunately there was not enough time to implement this automatic setup.

Other minor missing components is correct forwarding and receiving of **all** types of control plane message across FA-LSPs. The RSVP-TE daemon seems to have been designed with the assumption that messages **always** are sent to a neighbor node. Unfortunately, there was not sufficient time to find all the places where this assumption is made and to correct it taking FA-LSPs into account. Unfortunately, this causes Srefresh messages sent over FA-LSPs to be interpreted incorrectly, which in turn causes the client LSP to timeout after a certain amount of time and the LSP is torn down. However, if one disables the RSVP Refresh Overhead Reduction extension (which makes the daemon fall back to the original Path/Resv LSP refresh mechanism) processing is performed correctly and the LSP is not torn down.

A minor mistake was the use of the Suggested Label object to force end-to-end labels, this approach was implemented because it was the simplest available solution. However, when examining the standards carefully it turns out that this solution violates them. It should be replaced by the Label Set object – which can do the same job – although it requires more complicated processing.

Other parts of the implementation are a bit cumbersome to deal with, for example the way dynamically created TE-links were implemented. Overall the implementation was successful and performs its role in a proof-of-concept multi-region GMPLS network.

6.3 Emulation model

One interesting result of the work performed is the emulation model. The OXC/ROADM emulation shows a rather simple way to perform traffic engineering in packet networks without the need for MPLS. Traffic flows are separated with the use of IP based tunnels instead of labeling packets with MPLS labels. The IP based tunnels are set up between nodes (hop-by-hop) instead of end-to-end (which is the common way to use tunnels), this makes it possible to forward certain traffic separate from other traffic and independent of regular routing. On top of the tunnels one can create paths which can forward traffic based on an large array of headers. In this work only the link layer (VLAN ID, Destination MAC address) and physical layer (incoming interface) are used but ebtables is able to filter on network layer as well (some IP header fields). With the help of iptables and ip6tables forwarding can be based on IPv6 flow labels or transport layer headers, for example TCP destination port 80. These different modes of forwarding used in conjunction with Linux traffic queueing disciplines and GMPLS makes it possible to use cheap Linux computers to create flexible traffic engineered overlay networks. Such a network based on this kind of cheap devices may be useful for research where one wishes to separate production traffic from research traffic.

One idea on how to create such networks is called OpenFlow [56]. It proposes that networking equipment vendors add support for a special "flow table" in their regular networking equipment. An entry in the flow table match fields ranging from the physical layer to the transport layer and decides how to process the matching traffic. A controller (for example a regular PC) communicates with the networking device through a secure channel and adds entries to the flow table. In its basic form a flow entry either forwards a flow to specified port(s), drops the traffic, or sends it to the controller for further processing.

This idea is rather similar to the OXC/ROADM emulation but differs on some points, namely the forwarding is done entirely in hardware on devices which may have a large amount of ports. These devices are likely a better fit in regular network designs which allows for large scale deployment in (for example) campus networks. However, there is also an Linux implementation of OpenFlow on the OpenFlow homepage (`http://www.openflowswitch.org`).

Chapter 7

Conclusions and future work

7.1 Conclusions

The implementation shows that both FA-LSPs are a feasible way of handling cross region LSPs. The LSP conversion introduced by the MLCP project were tested as well and shown to work, although the conversion has a more limited use. The packet captured performed (shown in appendix G) indicate that the processing time of Path and Resv messages was not significantly affected by the introduction of FA-LSPs and LSP conversion.

The emulated WDM data plane based on Linux and a few common networking tools proved to function as intended and successfully emulated optical cross-connects and ROADMs. The initial model was successfully extended to function as an Ethernet bridge as well as an Ethernet-WDM region border node. However, the data plane delay introduced by the software based emulation could be a problem with a large amount of LSPs if the traffic is sensitive to delays.

The unnumbered method (with LSP_TUNNEL_INTERFACE_ID) proved to be a straightforward method of establishing FA-LSPs. Deprecating the numbered method and extending the unnumbered to take its place is a good idea.

Once should always be able to conclude how to interpret a label, at least if the links in the network are configured correctly. However, if the network is badly configured (and/or your implementation is bad or malicious) you may end up misinterpreting a label and install bogus switching rules. Adding explicit label type identifiers would lower the risk of this happening, at least in the case of badly behaving implementations and misconfigured nodes. The cost of adding explicit identifiers would be a small increase of signaling overhead and the risk of running out of identifiers.

7.2 Future Work

FA-LSPs have several scaling problems: the increased size of the Traffic Engineering Database when an FA-LSP is announced, the time required to set up the LSPs, and the time required for the IGP to distribute the new TE-link throughout the network. These scaling issues should be investigated further. The results of such an investigation should clarify how FA-LSP setup should be initiated, either triggered, manually or via a network management system.

The existing multi-region testbed could also be used to test different approaches to multi-region path calculation which also may have some scaling problems (primarily in the peer-model). To be able to calculate a path (which should have a high probability to actually be available) while using for example end-to-end labels some knowledge of what resources are available in the network is necessary. The availability of resources could be announced as TE information in the IGP along with available bandwidth etc, but this would dramatically increase the routing overhead. For example a bitmap of what VLANs are available on a L2SC link is about 512 bytes long. How to deal with such issues are not clear.

Another important part of providing traffic engineered services is protection (i.e. managing different kinds of failures). The emulated optical network could be extended to emulate some typical types of transmission problems and could be used to test procedures for failure recovery. The delay introduced by the emulation (which would not be present in actual OOO OXCs or ROADMs) could be a problem if data plane delay is considered as a failure which would trigger failure recovery.

References

[1] Rajiv Ramaswami and Kumar N. Sivarajan. *Optical networks: a practical perspective*. Morgan Kaufmann Publishers, Inc., San Fransisco, California, 1998.

[2] E. Rosen, D. Tappan, G. Fedorkow, Y. Rekhter, D. Farinacci, T. Li, and A. Conta. MPLS Label Stack Encoding. RFC 3032 (Proposed Standard), January 2001. Updated by RFCs 3443, 4182.

[3] E. Rosen, A. Viswanathan, and R. Callon. Multiprotocol Label Switching Architecture. RFC 3031 (Proposed Standard), January 2001.

[4] R. Braden, L. Zhang, S. Berson, S. Herzog, and S. Jamin. Resource ReSerVation Protocol (RSVP) – Version 1 Functional Specification. RFC 2205 (Proposed Standard), September 1997. Updated by RFCs 2750, 3936, 4495.

[5] S. Blake, D. Black, M. Carlson, E. Davies, Z. Wang, and W. Weiss. An Architecture for Differentiated Service. RFC 2475 (Informational), December 1998. Updated by RFC 3260.

[6] R. Braden, D. Clark, and S. Shenker. Integrated Services in the Internet Architecture: an Overview. RFC 1633 (Informational), June 1994.

[7] M. Allman, V. Paxson, and W. Stevens. TCP Congestion Control. RFC 2581 (Proposed Standard), April 1999. Updated by RFC 3390.

[8] Adrian Farrel. *The Internet and Its Protocols: A Comparative Approach*. Morgan Kaufmann, first edition, 2004.

[9] D. Awduche, J. Malcolm, J. Agogbua, M. O'Dell, and J. McManus. Requirements for Traffic Engineering Over MPLS. RFC 2702 (Informational), September 1999.

[10] Adrian Farrel and Igor Bryskin. *GMPLS: Architecture and Applications*. Elsevier, Amsterdam, 2006.

[11] D. Awduche, Y. Rekhter, J.Drake, and R. Coltun. Internet-Draft - Multi-Protocol Lambda Switching: Combining MPLS Traffic Engineering Control

With Optical Crossconnects, April 2001. URL: `http://tools.ietf.org/html/draft-awduche-mpls-te-optical-03` Expired. Accessed July 8, 2008.

[12] K. Kompella, Y. Rekhter, and Ed. Routing Extensions in Support of Generalized Multi-Protocol Label Switching (GMPLS). RFC 4202 (Proposed Standard), October 2005.

[13] K. Shiomoto, D. Papadimitriou, J.L. Le Roux, M. Vigoureux, and D. Brungard. Internet-Draft - Requirements for GMPLS-Based Multi-Region and Multi-Layer Networks (MRN/MLN), May 2008. URL: `http://tools.ietf.org/html/draft-ietf-ccamp-gmpls-mln-reqs-11` Work in progress. Accessed July 8, 2008.

[14] I. Bryskin and A. Farrel. A Lexicography for the Interpretation of Generalized Multiprotocol Label Switching (GMPLS) Terminology within the Context of the ITU-T's Automatically Switched Optical Network (ASON) Architecture. RFC 4397 (Informational), February 2006.

[15] L. Berger. Generalized Multi-Protocol Label Switching (GMPLS) Signaling Functional Description. RFC 3471 (Proposed Standard), January 2003. Updated by RFCs 4201, 4328, 4872.

[16] D. Awduche, L. Berger, D. Gan, T. Li, V. Srinivasan, and G. Swallow. RSVP-TE: Extensions to RSVP for LSP Tunnels. RFC 3209 (Proposed Standard), December 2001. Updated by RFCs 3936, 4420, 4874, 5151.

[17] L. Berger. Generalized Multi-Protocol Label Switching (GMPLS) Signaling Resource ReserVation Protocol-Traffic Engineering (RSVP-TE) Extensions. RFC 3473 (Proposed Standard), January 2003. Updated by RFCs 4003, 4201, 4420, 4783, 4874, 4873, 4974, 5063, 5151.

[18] Behrouz A. Forouzan. *TCP/IP Protocol Suite.* McGraw-Hill, 3 edition, 2006. ISBN 0-07-111583-8.

[19] J. Moy. OSPF Version 2. RFC 2328 (Standard), April 1998.

[20] R. Coltun. The OSPF Opaque LSA Option. RFC 2370 (Proposed Standard), July 1998. Updated by RFC 3630.

[21] D. Katz, K. Kompella, and D. Yeung. Traffic Engineering (TE) Extensions to OSPF Version 2. RFC 3630 (Proposed Standard), September 2003. Updated by RFC 4203.

[22] IEEE 802 LAN/MAN Standards Committee. IEEE 802.3 ETHERNET. URL: `http://www.ieee802.org/3/index.html` Accessed July 8, 2008. Last modified April 17, 2008.

[23] Sam Halabi. *Metro Ethernet.* Cisco Press, 1 edition, October 2003.

[24] IEEE 802 LAN/MAN Standards Committee. IEEE Standard for Local and metropolitan area networks - Virtual Bridged Local Area Networks, 2006. URL: `http://standards.ieee.org/getieee802/download/802.1Q-2005.pdf` Accessed July 8, 2008.

[25] IEEE 802 LAN/MAN Standards Committee. IEEE 802.1: 802.1ad - Provider Bridges. URL: `http://www.ieee802.org/1/pages/802.1ad.html` Accessed July 8, 2008.

[26] IEEE 802 LAN/MAN Standards Committee. IEEE 802.1: 802.1ah - Provider Backbone Bridges. URL: `http://www.ieee802.org/1/pages/802.1ah.html` Accessed July 8, 2008.

[27] IETF. Common Control and Measurement Plane (ccamp) Charter. URL: `http://www.ietf.org/html.charters/ccamp-charter.html` Accessed July 8, 2008. Last modified May 16, 2008.

[28] Biswanath Mukherjee. *Optical WDM Networks*. Optical Network Series. Springer Science+Business Media, Inc., 2006.

[29] Nasir Ghani. REGIONAL-METRO OPTICAL NETWORKS. In Krishna M. Sivalingam and Suresh Subramaniam, editors, *EMERGING OPTICAL NETWORK TECHNOLOGIES - Architectures, Protocols and Performance*, chapter 4, page 105. Springer Science + Business Media, Inc., 2005.

[30] T. Otani, H. Guo, K. Miyazaki, and D. Caviglia. Internet-Draft - Generalized Labels of Lambda-Switching Capable Label Switching Routers (LSR), February 2008. URL: `http://tools.ietf.org/html/draft-otani-ccamp-gmpls-lambda-labels-02` Work in progress. Updates RFC 3471. Accessed July 8, 2008.

[31] ITU-T Recommendation G.694.1. Spectral grids for WDM applications: DWDM frequency grid, June 2002. URL: `http://www.itu.int/rec/T-REC-G.694.1/en` Accessed July 8, 2008.

[32] ITU-T Recommendation G.694.2. Spectral grids for WDM applications: CWDM frequency grid, December 2003. URL: `http://www.itu.int/rec/T-REC-G.694.2/en`. Accessed July 8, 2008.

[33] K. Kompella and Y. Rekhter. Label Switched Paths (LSP) Hierarchy with Generalized Multi-Protocol Label Switching (GMPLS) Traffic Engineering (TE). RFC 4206 (Proposed Standard), October 2005.

[34] L. Berger, D. Gan, G. Swallow, P. Pan, F. Tommasi, and S. Molendini. RSVP Refresh Overhead Reduction Extensions. RFC 2961 (Proposed Standard), April 2001. Updated by RFC 5063.

[35] K. Kompella and Y. Rekhter. Signalling Unnumbered Links in Resource ReSerVation Protocol - Traffic Engineering (RSVP-TE). RFC 3477 (Proposed Standard), January 2003.

[36] K. Shiomoto, R. Rabbat, A. Ayyangar, A. Farrel, and Z. Ali. Internet Draft - Procedures for Dynamically Signaled Hierarchical Label Switched Paths, February 2008. URL: `http://tools.ietf.org/html/draft-ietf-ccamp-lsp-hierarchy-bis-03` Work in progress. Accessed July 8, 2008.

[37] E. Mannie. Generalized Multi-Protocol Label Switching (GMPLS) Architecture. RFC 3945 (Proposed Standard), October 2004.

[38] DRAGON project homepage. URL: `http://dragon.east.isi.edu/twiki/bin/view/Main/WebHome` Accessed July 8, 2008.

[39] G. Clark. Telnet Com Port Control Option. RFC 2217 (Experimental), October 1997.

[40] IEEE 802 LAN/MAN Standards Committee. IEEE 802.1: 802.1D - Media Access Control (MAC) Bridges. URL: `http://standards.ieee.org/getieee802/download/802.1D-2004.pdf` Accessed July 8, 2008.

[41] Jim Robson. Kernel Korner - Linux as an Ethernet Bridge. *Linux Journal*, May 2005. URL: `http://www.linuxjournal.com/article/8172` Accessed July 8, 2008.

[42] D. Farinacci, T. Li, S. Hanks, D. Meyer, and P. Traina. Generic Routing Encapsulation (GRE). RFC 2784 (Proposed Standard), March 2000. Updated by RFC 2890.

[43] C. Perkins. IP Encapsulation within IP. RFC 2003 (Proposed Standard), October 1996.

[44] S. Kent and K. Seo. Security Architecture for the Internet Protocol. RFC 4301 (Proposed Standard), December 2005.

[45] M. Krasnyansky and M. Yevmenkin. Universal TUN/TAP driver. URL: `http://vtun.sourceforge.net/tun` Accessed April 3, 2008.

[46] M. Krasnyansky et al. VTun - Virtual Tunnels over TCP/IP networks. URL: `http://vtun.sourceforge.net` Accessed April 3, 2008.

[47] Stephen Hemminger. Network Emulation with NetEm. Linux Conf Au, April 2005. URL: `http://developer.osdl.org/shemminger/LCA2005_paper.pdf` Accessed July 8, 2008.

[48] The Linux Foundation. Net:Netem Homepage. URL: `http://www.linux-foundation.org/en/Net:Netem` Accessed July 8, 2008.

[49] Martin A. Brown. Linux Traffic Control HOWTO. URL: http://tldp.org/HOWTO/Traffic-Control-HOWTO/index.html Accessed July 8, 2008. Last modified October 2006.

[50] B. Hubert et al. Linux Advanced Routing & Traffic Control HOWTO. URL: http://lartc.org/howto/ Accessed July 8, 2008.

[51] IP Infusion Inc. GNU Zebra homepage. URL: http://www.zebra.org/ Accessed July 8, 2008. Last modified August 8, 2005.

[52] D. Fedyk and D. Allan. Internet-Draft - GMPLS control of Ethernet IVL Switches , October 2005. URL: http://tools.ietf.org/html/draft-fedyk-gmpls-ethernet-ivl-00 Expired. Accessed July 8, 2008.

[53] Juniper M-series. URL: http://www.juniper.net/products_and_services/m_series_routing_portfolio/index.html Accessed August 10, 2008.

[54] Transmode TM-3000 fact sheet. URL: http://www.transmode.com/index.php?option=com_docman&task=doc_download&gid=12&Itemid=60 Accessed August 10,2008.

[55] Graphviz - Graph Visualization Software. URL: http://www.graphviz.org Accessed July 8, 2008.

[56] Nick McKeown, Tom Anderson, Hari Balakrishnan, Guru M. Parulkar, Larry L. Peterson, Jennifer Rexford, Scott Shenker, and Jonathan S. Turner. Openflow: enabling innovation in campus networks. *Computer Communication Review*, 38(2):69–74, 2008.

[57] B. De Schuymer et al. Ebtables. URL: http://ebtables.sourceforge.net Accessed April 3, 2008.

[58] Alexander Lindström. GMPLS multi-layer networking - Routing and constraint-based path computation in optical network segments. Master's thesis, Royal Institute of Technology (KTH), 2007.

Appendices

Appendix A

Brctl

The brctl command can be used to create virtual bridges, set STP parameters on a bridge, and add interfaces to a bridge. In table A.1 all arguments to the "brctl" command are shown. Creating a bridge, turning off STP, and adding interfaces to the bridge is done as follows:

```
# brctl addbr br0
# brctl stp br0 off
# brctl addif br0 eth0
# brctl addif br0 eth1
# brctl show
bridge name     bridge id               STP enabled     interfaces
br0             8000.00508d651fa1       no              eth0
                                                        eth1
```

This creates a bridge called "br0", turns of STP on the bridge and adds two interfaces: "eth0" and "eth1".

Table A.1: Arguments to the brctl command

addbr	<bridge>			add bridge
delbr	<bridge>			delete bridge
addif	<bridge>	<device>		add interface to bridge
delif	<bridge>	<device>		delete interface from bridge
setageing	<bridge>	<time>		set ageing time
setbridgeprio	<bridge>	<prio>		set bridge priority
setfd	<bridge>	<time>		set bridge forward delay
sethello	<bridge>	<time>		set hello time
setmaxage	<bridge>	<time>		set max message age
setpathcost	<bridge>	<port>	<cost>	set path cost
setportprio	<bridge>	<port>	<prio>	set port priority
show				show a list of bridges
showmacs	<bridge>			show a list of mac addrs
showstp	<bridge>			show bridge stp info
stp	<bridge>	{on|off}		turn stp on/off

Appendix B

Ebtables

Ebtables is used to create firewall rules on Linux virtual Ethernet bridges. By adding rules to chains one can for example drop unwanted frames, change fields in some frames etc. When receiving, forwarding or sending a frame one of the default chains is "traversed" by the frame until it matches one of the rules in the chain and the action defined in the rule is applied. If no specific rule is matched the default policy for that chain is executed, for example dropping all forwarding packets which do not match a rule.

Frames can be matched in many ways, for example by destination MAC address, incoming interface, payload protocol, VLAN ID etc. In the emulated optical and Ethernet nodes matching is done on interfaces, destination MAC addresses and VLAN IDs. Following commands sets the default forwarding policy to ”drop all traffic”, adds three forwarding rules, and displays the rules on the chain:

```
# ebtables -P FORWARD DROP
# ebtables -A FORWARD --in-if eth0 --out-if eth1 -j ACCEPT
# ebtables -A FORWARD -p 8021Q --vlan-id 10 -d 01:23:45:67:89:AB
          --out-if eth0 -j ACCEPT
# ebtables -L FORWARD
Bridge table: filter

Bridge chain: FORWARD, entries: 2, policy: DROP
-i eth0 -o eth1 -j ACCEPT
-p 802_1Q -d 1:23:45:67:89:ab -o eth0 --vlan-id 10 -j ACCEPT
```

The first line sets the default policy on the FORWARD chain. The second line matches traffic coming from interface ”eth0” and leaving through ”eth1”, this is set to the target ACCEPT which means that the traffic will be forwarded. The third line matches traffic on VLAN 10 with a specific destination MAC address and leaving on interface eth0, traffic matching this rule will also be forwarded. The fourth line lists the current rules in and the policy of the forwarding chain. More details about the many operations that can be performed with Ebtables can be found on the Ebtables homepage or in the Ebtables manual pages [57].

Appendix C

Vtun

Vtun is a simple way of creating tunnels. It supports several tunnel types such as IP, PPP, Ethernet and others. The host in one end of the tunnel runs in server mode while the other connect as a client. The client requests the creation of a pre-configured tunnel between the hosts. All of the emulated optical nodes have an identical configuration file and a script that depending on the hostname either starts Vtun in server mode or in client mode. By a command line argument the client can be made to attempt to connect until a tunnel is established, if the connection is dropped the client will attempt to reconnect and bring it back up. Below is part of the configuration file used in the testbed:

```
options {
 port 6000;
 syslog daemon;
 timeout 60;
 ppp /usr/sbin/pppd;
 ifconfig /sbin/ifconfig;
 route /sbin/route;
 firewall /sbin/iptables;
 ip /sbin/ip;
 firewall /usr/sbin/brctl;
}
default {
 type tun;
 proto tcp;
 compress no;
 speed 0;
 keepalive yes;
}
13ba-1913000{
 passwd abc123;
 keepalive yes;
 proto tcp;
 type ether;
 up {
  program "ip link set %% name 13ba-1913000" wait;
  ifconfig "13ba-1913000 up";
27   program "brctl addbr br0" wait;
28   program "brctl addif br0 13ba-1913000" wait;
29   ifconfig "br0 up";
30  };
31  down {
32   ifconfig "13ba-1913000 down";
33  };
34 }
```

The first eleven lines sets some default parameters for the server and client such as a default port and path to some programs used by Vtun. The next segment (line 12-18) sets default values for tunnels. This includes the type of tunnel ("tun"), what protocol to use to transport tunnel traffic ("tcp"), whether tunnel traffic should be compressed or limited to a certain bitrate, and if keepalive packets should be sent between the hosts to make sure firewalls do not close the ports used. The rest (lines 19-34) is the configuration for a specific tunnel called 13ba-1913000. It is configured as an Ethernet tunnel using TCP, which means that TCP is used to transport entire Ethernet frames. The "up" and "down" parameters defines calls that should be made when the tunnel has been brought up or down. The "up" command is used to rename the tunnel from the name it receives by default to the name of the tunnel, then it is brought up and added to a bridge.

Appendix D

OSPF-TE

The OSPF-TE daemon is responsible for routing the control plane and distributing the data plane Traffic Engineering Database. As the suite does not have any LMP support the OSPF-TE daemon has taken some of LMP's responsibilities such as data plane link configuration. It can be controlled either via a configuration file or by a command line interface reachable with Telnet. The configuration file for a node (in this case vEC1 which can be seen in figure 5.1) looks like this:

```
! host name and password
hostname vec1
password dragon
enable password dragon
log file /var/log/ospfd.log

! control plane interfaces
interface control-13ba
 description GRE tunnel between vec1 and vroadm1
 ip ospf network point-to-point
interface control-13cb
 description GRE tunnel between vec1 and vroadm1
 ip ospf network point-to-point
interface control-ve1
 description GRE tunnel between vec1 and ve1
 ip ospf network point-to-point
interface control-e3
 description GRE tunnel between vec1 and swce3
 ip ospf network point-to-point

! FA-LSP dummy interfaces
interface dummy0
 ip ospf network point-to-point
interface dummy1
 ip ospf network point-to-point
interface dummy2
 ip ospf network point-to-point
interface dummy3
 ip ospf network point-to-point
interface dummy4
```

```
 ip ospf network point-to-point

! OSPF settings
router ospf
 ospf router-id 172.16.100.230
! distribute route to router-id
 network 172.16.100.230/32 area 0.0.0.0
! connected networks
 network 172.16.100.0/29  area 0.0.0.0
 network 172.16.100.56/29 area 0.0.0.0
 network 172.16.105.8/29 area 0.0.0.0
 network 172.16.105.48/29 area 0.0.0.0

! dummy interfaces for FA-LSPs
 network 127.0.1.1/32 area 0.0.0.0
 network 127.0.1.2/32 area 0.0.0.0
 network 127.0.1.3/32 area 0.0.0.0
 network 127.0.1.4/32 area 0.0.0.0
 network 127.0.1.5/32 area 0.0.0.0

! OSPF-TE configuration
 ospf-te router-address 172.16.100.230
 ospf-te interface control-13ba
  level gmpls
  fa-lsp off
  metric 10
  link-type point-to-point
  data-interface unnumbered protocol snmp switch-ip 127.0.0.1 switch-port 8
  swcap l2sc encoding ethernet
  max-bw 12500000000
  max-rsv-bw 12500000000
  wavegrid new dwdm 100 191400 1 1250000000
  constraint add 0 0
  constraint add 1 1
  constraint add 2 5
 exit

 ospf-te interface control-13cb
  level gmpls
  fa-lsp off
  metric 10
  link-type point-to-point
  data-interface unnumbered protocol snmp switch-ip 127.0.0.1 switch-port 9
  swcap l2sc encoding ethernet
  max-bw 12500000000
  max-rsv-bw 12500000000
  wavegrid new dwdm 100 191300 1 1250000000
  constraint add 0 0
  constraint add 1 10
  constraint add 2 5
 exit

 ospf-te interface control-ve1
  level gmpls
 85    link-type point-to-point
 86    fa-lsp off
 87    metric 10
 88    link-type point-to-point
 89    data-interface unnumbered protocol snmp switch-ip 127.0.0.1 switch-port 2
 90    swcap l2sc encoding ethernet
 91    max-bw 12500000000
 92    max-rsv-bw 12500000000
 93   exit
 94
 95   ospf-te interface control-e3
 96    level gmpls
 97    link-type point-to-point
 98    fa-lsp off
 99    metric 10
100    link-type point-to-point
101    data-interface unnumbered protocol snmp switch-ip 127.0.0.1 switch-port 4
102    swcap l2sc encoding ethernet
103    max-bw 12500000000
104    max-rsv-bw 12500000000
105   exit
106
107  ! Dummy interface configuration
108  ospf-te interface dummy0
109   level gmpls
110   fa-lsp hold
111  exit
112  ospf-te interface dummy1
113   level gmpls
114   fa-lsp hold
115  exit
116  ospf-te interface dummy2
117   level gmpls
118   fa-lsp hold
119  exit
120  ospf-te interface dummy3
121   level gmpls
122   fa-lsp hold
123  exit
124  ospf-te interface dummy4
125   level gmpls
126   fa-lsp hold
127  exit
```

The first lines (1-5) sets the hostname, passwords and log file for the daemon. Then the interfaces which should be used by the OSPF-TE daemon is defined (lines 7-31), this is followed by the configuration of OSPF parameters such as the Router-ID and which networks should be included (lines 33-49). Note that the dummy interfaces have been assigned local IP addresses (127.0.1.1/32-127.0.1.5/32) and that they are included with the "network" command even though we do not want OSPF to announce these networks. After the OSPF parameters the OSPF-TE configuration follows (lines 51-127). First the regular OSPF-TE interfaces are configured then the

dummy interfaces. The regular interfaces are assigned their various TE information such as maximum reservable bandwidth, Interface Switching Capabilty Descriptor, metric etc. All the interfaces which are control links for data plane links have their ”fa-lsp” option set to off, since they should have both regular OSPF and TE LSAs distributed. The dummy interfaces (lines 107-127) are only configured with the fa-lsp option set to ”hold”. One of the lines that stand out in this configuration is the ”data-interface” (line 58, 73, . . .). This line indicates the method of control, IP address of and a port number for the data plane link this control plane link should be associated with. For example port number 4 on a Switchcore Ethernet switch with IP address A.B.C.D. Other odd lines are the ”wavegrid” and ”constraint” lines which add additional sub-TLVs used for path computation in the optical region, they were added by previous thesis workers at Ericsson [58].
Changes made to the OSPF-TE daemon are:

- The ”fa-lsp” option
- Interprocess Communication with the RSVP daemon

The FA-LSP option adds a flag to the structure holding information about interfaces. This structure holds both the regular OSPF information as well as TE information. When the daemon constructs the LSA packets this flag is checked and depending on its value the link is or is not added to the LSAs.
The interprocess communication consists of two functions, one that create a TE-link and one that removes a TE-link. When creating TE-link this function is called with all the TE-link information as arguments. It then tries to locate an interface with the FA-LSP option set to HOLD. If one is found the TE-link information is assigned to the interface and the FA-LSP option set to ON and a function to update LSA information is called. This causes the new link to be disseminated. The remove TE-link function is called with the local interface ID, it finds the interface with that ID and sets the fa-lsp flag to HOLD, removing it from any LSA.

Appendix E

RSVP-TE

Many changes to the RSVP-TE daemon has been made in order to support FA-LSPs and the emulated optical data plane.In this appendix more technical details about how these changes have been implemented is given. The RSVP-TE daemon is configured by a configuration file, unlike other daemons it has no command line interface. The configuration for the same node as the one shown in appendix D is:

```
1  interface control-13ba tc none mpls
2  interface control-13ba tc none mpls
3  interface control-13cb tc none mpls
4  interface control-ve1 tc none mpls
5  interface control-e3 tc none mpls
6  dataplane LSC-Linux
7  api 4000
```

The configuration is rather simple, the interfaces on which RSVP-TE message should be received are listed (line 1-5). The ”dataplane” option determines which data plane controller module which is instantiated by the daemon, in this case the ”LSC-Linux” daemon for emulated optical and emulated Ethernet. The data plane controller was implemented using a class structure created by the MLCP project. The structure generalizes the functions of a controller which allows the daemon to instantiate an appropriate controller object based on a configuration file. The controller object (called ”LSC-Linux”) can control emulated optical, Linux Ethernet and Linux Ethernet to emulated optical border nodes. During the MLCP project several data plane controllers were developed, all of these were implemented as sub-classes of a generic data plane controller class. In figure E the operations performed by one of the two major functions in the data plane controller is depicted. As can be seen there are some interprocess calls to the OSPFd daemon and some communication with the switch. The communication with the switch is performed using regular command line interface which is connected by Telnet or SSH. On the CLI regular unix commands are used, for example the operation ”Check for interface in bridge” executes a shell command which lists all interfaces in the bridge and uses a regular expression to match the output:

```
# sudo ebtables -L | egrep '\\-i[[:space:]]+INTERFACE_NAME'
```

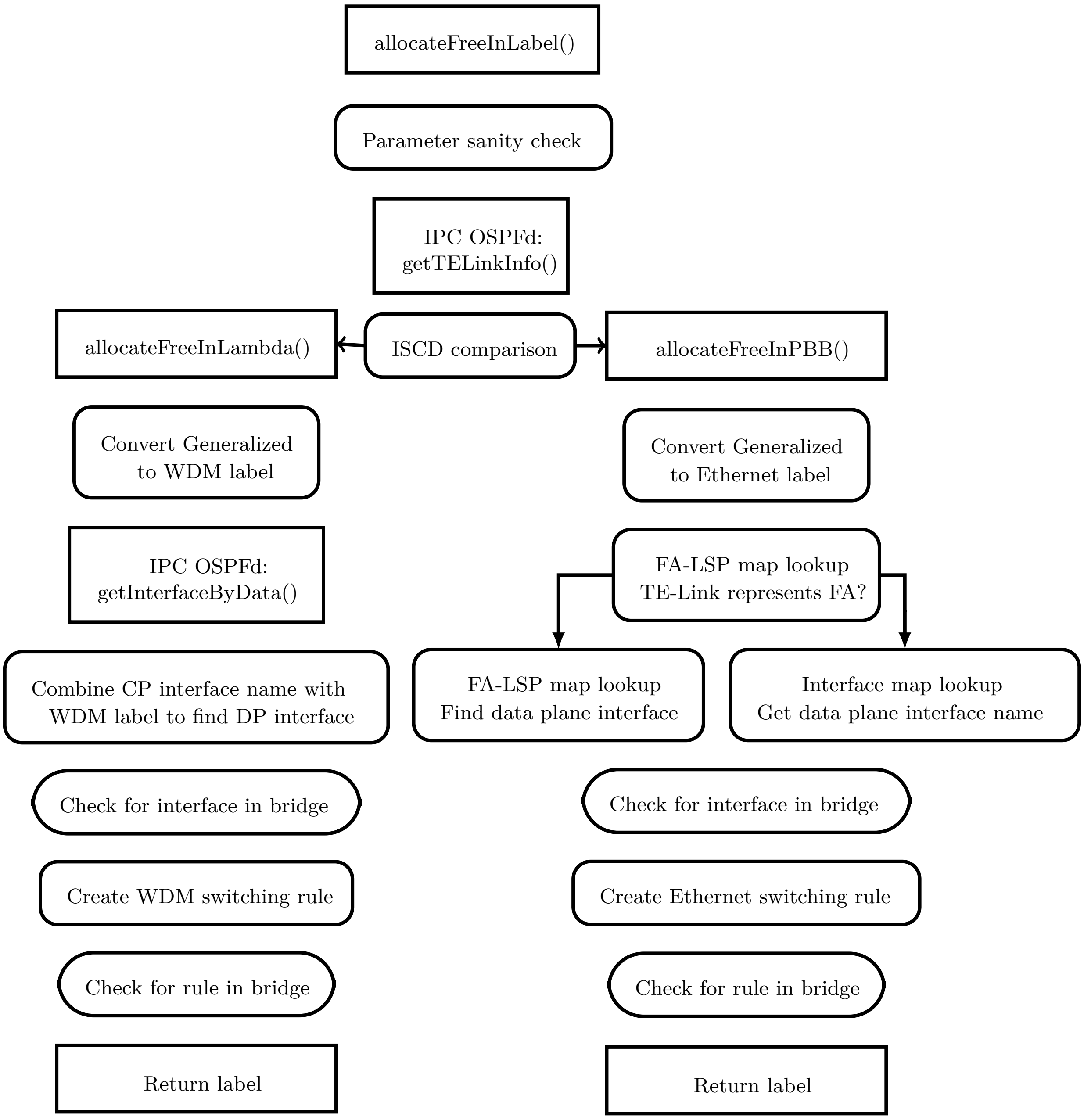
allocateFreeInLabel()
Parameter sanity check
IPC OSPFd: getTELinkInfo()
allocateFreeInLambda()
ISCD comparison
allocateFreeInPBB()
Convert Generalized to WDM label
Convert Generalized to Ethernet label
IPC OSPFd: getInterfaceByData()
FA-LSP map lookup TE-Link represents FA?
Combine CP interface name with WDM label to find DP interface
FA-LSP map lookup Find data plane interface
Interface map lookup Get data plane interface name
Check for interface in bridge
Check for interface in bridge
Create WDM switching rule
Create Ethernet switching rule
Check for rule in bridge
Check for rule in bridge
Return label
Return label

Figure E.1: Operations performed by the function allocateFreeInLabel() in the LSC-Linux data plane controller.

Appendix F

Dragon daemon

The dragon daemon is responsible for initiating and terminating LSPs. It is controlled similarly to the other daemons (i.e. Telnet command line interface and configuration file). A few configuration options has been added to this daemon, these are described here.

Added options are:

- FA-LSP option
- Upstream and Suggested WDM labels
- Manual ERO configuration

The FA-LSP option can be set to either on or off and is usable when editing an LSP:

```
set fa-lsp (on|off)
```

When set to "on" a LSP_TUNNEL_INTERFACE_ID object is created and added to the RSVP request sent to the RSVP-TE daemon. This option requires that LSP and Tunnel ID has already been set since it uses these two values to create the unnumbered interface identifier. The identifier is created by shifting LSP-ID 16-bit field 16 steps the left and adding the 16-bit Tunnel-ID. This results in a 32-bit unnumbered interface identifier, however since the OSPF-TE daemon requires unnumbered interface identifiers to be above 0x8000000 (2^{31}) the LSP-ID must be set to 32768 or larger if the LSP is to become an FA-LSP.

The Upstream and Suggested CWDM/DWDM label options can be used to add WDM labels to the RSVP request. Like the FA-LSP option it is usable when editing an LSP. These options take as arguments the different fields shown in figure 3.8 to construct the label objects, the commands are:

```
  set upstream dwdm label spacing <1-4> n <0-65535> sign <0-1>
  set suggested dwdm label spacing <1-4> n <0-65535> sign <0-1>
  set upstream cwdm label wavelength <0-65535>
  set suggested cwdm label wavelength <0-65535>
```

Where the numbers in brackets are possible values. When an WDM label is added the frequency or wavelength is calculated and printed so that it can be verified. The reason for havin multiple variables for the DWDM label selection and not just a frequency is that a single frequency can be represented in several ways since for example $2 \times 50.0GHz = 1 \times 100.0GHz$, this way there is no ambiguity about what label to create.

The third addition to the Dragon daemon is the commands for adding a manual ERO. These commands are usable when editing an LSP and support IPv4 and unnumbered addresses. The commands are:

```
add ero ip A.B.C.D
add ero unnum A.B.C.D if_id WORD
del ero <0-31>
show ero
```

The two ”add ero” commands add a single hop to the ERO, either an IPv4 address or an IPv4 address plus a hexadecimal number (the WORD parameter) which is the unnumbered interface ID. The hops are added in a ”downstream” order, i.e. the first hop added to the ERO is the next-hop. The ”del ero” command removes a single hop from the ERO, a number between 0 and 31 identify which hop should be removed. There is limit of 32 hops per ERO hard-coded into the daemon, however there is no significance to the number other than that. The ”show ero” command prints the current ERO hops along with a number that identifies each hop (used with ”del ero”). Note that there is no possibility to set a prefix length when adding ERO entries, it is hard-coded to /32 which is a mistake made during implementation.

Appendix G

Packet captures

G.1 FA-LSP setup

Packet capture from setup of the FA-LSP described in section 5.2.1, fields that are identical between packets has been removed to save space. Captured with Wireshark. The time indication are in seconds relative to the first packet. The P_{time} field is the time when the packet was captured minus the time since previous packet was captured, it is an estimation of how much time the sending node spent processing the incoming and outgoing packets.

```
No. Time (s) Ptime (s) Source       Destination
1   0.000000 0.000000  172.16.100.1 172.16.100.2
RSVP Header. PATH Message.
SESSION: IPv4-LSP, Destination 172.16.100.232, Tunnel ID 54, Ext ID e66410ac.
HOP: IPv4 IF-ID. Control IPv4: 172.16.100.1. Data If-Index: 172.16.100.230,
     -2147483647.
TIME VALUES: 30000 ms
LABEL REQUEST: Generalized: LSP Encoding=Lambda (photonic), Switching
      Type=Lambda-Switch Capable (LSC), G-PID=Ethernet (SDH, Lambda, Fiber)
SUGGESTED LABEL: Generalized: 0x29000011
UPSTREAM LABEL: Generalized: 0x29000011
LSP INTERFACE-ID: IPv4, Router-ID 172.16.100.230, Interface-ID -2147483594
EXPLICIT ROUTE: Unnum 172.16.100.233/-2147483647, Unnum
         172.16.100.233/-2147483644, Unnum 172.16.100.236/-2147483647, ...
SESSION ATTRIBUTE: SetupPrio 7, HoldPrio 7,  [favec1vec3\000\000]
SENDER TEMPLATE: IPv4-LSP, Tunnel Source: 172.16.100.1, LSP ID: 32768.
SENDER TSPEC: IntServ: Token Bucket, 156250000 bytes/sec.
ADSPEC

No. Time (s)  Ptime (s) Source        Destination
2   0.039219  0.039219  172.16.100.89 172.16.100.90
RSVP Header. PATH Message.
HOP: IPv4 IF-ID. Control IPv4: 172.16.100.89. Data If-Index: 172.16.100.233,
     -2147483644.
EXPLICIT ROUTE: Unnum 172.16.100.236/-2147483647, Unnum
         172.16.100.236/-2147483645, Unnum 172.16.100.235/-2147483645, ...
```

```
No. Time (s) Ptime (s) Source          Destination
3   0.072902 0.033683  172.16.100.185 172.16.100.186
RSVP Header. PATH Message.
HOP: IPv4 IF-ID. Control IPv4: 172.16.100.185. Data If-Index: 172.16.100.236,
     -2147483645.
EXPLICIT ROUTE: Unnum 172.16.100.235/-2147483645, Unnum
         172.16.100.235/-2147483647, Unnum 172.16.100.232/-2147483647

No. Time (s) Ptime (s) Source          Destination
4   0.109764 0.036862  172.16.100.201 172.16.100.202
RSVP Header. PATH Message.
HOP: IPv4 IF-ID. Control IPv4: 172.16.100.201. Data If-Index: 172.16.100.235,
     -2147483647.
EXPLICIT ROUTE: Unnum 172.16.100.232/-2147483647

No. Time (s) Ptime (s) Source          Destination
5   0.329423 0.219659  172.16.100.202 172.16.100.201
RSVP Header. RESV Message.
SESSION: IPv4-LSP, Destination 172.16.100.232, Tunnel ID 54, Ext ID e66410ac.
HOP: IPv4 IF-ID. Control IPv4: 172.16.100.202. Data If-Index: 172.16.100.235,
     -2147483647.
TIME VALUES: 30000 ms
LSP INTERFACE-ID: IPv4, Router-ID 172.16.100.232, Interface-ID -2147483594
STYLE: Fixed Filter (10)
FLOWSPEC: Controlled Load: Token Bucket, 156250000 bytes/sec.
FILTERSPEC: IPv4-LSP, Tunnel Source: 172.16.100.1, LSP ID: 32768.
LABEL: Generalized: 0x29000011

No. Time (s) Ptime (s) Source          Destination
6   0.932248 0.602825  172.16.100.186 172.16.100.185
RSVP Header. RESV Message.
HOP: IPv4 IF-ID. Control IPv4: 172.16.100.186. Data If-Index: 172.16.100.236,
-2147483645.

No. Time (s) Ptime (s) Source         Destination
7   1.547518 0.615270  172.16.100.90 172.16.100.89
HOP: IPv4 IF-ID. Control IPv4: 172.16.100.90. Data If-Index: 172.16.100.233,
     -2147483644.

No. Time (s) Ptime (s) Source        Destination
8   2.164658 0.617140  172.16.100.2 172.16.100.1
RSVP Header. RESV Message.
HOP: IPv4 IF-ID. Control IPv4: 172.16.100.2. Data If-Index: 172.16.100.230,
     -2147483647.
```

G.2 Client LSP setup

Packet capture from setup of the FA-LSP described in section 5.2.2, fields that are identical between packets has been removed to save space. Captured with Wireshark on two different networks then merged together. The merge of the two files introduced timing errors since the two clocks were not perfectly synced. A heading has been added to indicate where the following packets were captures, session 1 is from the physical network while session 2 is from the virtual network. The P_{time} field has been added for the same reasons as for the previous capture.

```
------ Wireshark session 1
No. Time (s) Ptime (s) Source       Destination
1   0.000000 0.000000  172.16.112.1 172.16.112.2
RSVP Header. PATH Message.
SESSION: IPv4-LSP, Destination 172.16.100.249, Tunnel ID 3784, Ext ID 101000a.
HOP: IPv4 IF-ID. Control IPv4: 172.16.112.1. Data IPv4: 172.16.112.1.
TIME VALUES: 30000 ms
LABEL REQUEST: Generalized: LSP Encoding=Packet, Switching Type=Packet-Switch
      Capable-1 (PSC-1), G-PID=IP
UPSTREAM LABEL: Generalized: 0x308
EXPLICIT ROUTE: IPv4 172.16.112.2, Unnum 172.16.100.246/-2147483647, Unnum
         172.16.100.243/-2147483647, ...
SESSION ATTRIBUTE: SetupPrio 7, HoldPrio 7,  [mefaem\000\000]
SENDER TEMPLATE: IPv4-LSP, Tunnel Source: 172.16.112.1, LSP ID: 99.
SENDER TSPEC: IntServ: Token Bucket, 12500000 bytes/sec.
ADSPEC

No. Time (s) Ptime (s) Source         Destination
2   0.147486 0.147486  172.16.100.105 172.16.100.106
RSVP Header. PATH Message.
HOP: IPv4 IF-ID. Control IPv4: 172.16.100.105. Data If-Index: 172.16.100.246,
     -2147483647.
LABEL REQUEST: Generalized: LSP Encoding=Ethernet v2/DIX, Switching
      Type=Layer-2 Switch Capable (L2SC), G-PID=IP
UPSTREAM LABEL: Generalized: 0x5e4000c, 0x29f833f5
EXPLICIT ROUTE: Unnum 172.16.100.243/-2147483647, Unnum
         172.16.100.243/-2147483646, Unnum 172.16.100.230/-2147483645, ...

No. Time (s) Ptime (s) Source        Destination
3   0.258414 0.110928  172.16.105.10 172.16.105.9
RSVP Header. PATH Message.
HOP: IPv4 IF-ID. Control IPv4: 172.16.105.10. Data If-Index: 172.16.100.243,
     -2147483646.
EXPLICIT ROUTE: Unnum 172.16.100.230/-2147483645, Unnum
         172.16.100.230/-2147483594, Unnum 172.16.100.232/-2147483594, ...

------ Wireshark session 2
No. Time (s) Ptime (s) Source         Destination
4   -------- 0.067305  172.16.100.230 172.16.100.232
RSVP Header. PATH Message.
HOP: IPv4 IF-ID. Control IPv4: 172.16.100.1. Data If-Index: 172.16.100.230,
     -2147483594.
EXPLICIT ROUTE: Unnum 172.16.100.232/-2147483594, Unnum
         172.16.100.232/-2147483645, Unnum 172.16.100.245/-2147483645, ...

No. Time (s) Ptime (s) Source        Destination
5   -------- 0.065828  172.16.105.25 172.16.105.26
RSVP Header. PATH Message.
HOP: IPv4 IF-ID. Control IPv4: 172.16.105.25. Data If-Index: 172.16.100.232,
     -2147483645.
```

```
EXPLICIT ROUTE: Unnum 172.16.100.245/-2147483645, Unnum
         172.16.100.245/-2147483647, Unnum 172.16.100.253/-2147483647, ...

------ Wireshark session 1
No. Time (s) Ptime (s) Source         Destination
6   0.496727 ---------  172.16.100.145 172.16.100.146
RSVP Header. PATH Message.
HOP: IPv4 IF-ID. Control IPv4: 172.16.100.145. Data If-Index: 172.16.100.245,
     -2147483647.
EXPLICIT ROUTE: Unnum 172.16.100.253/-2147483647, IPv4 172.16.100.153, IPv4
         172.16.100.154

No. Time (s) Ptime (s) Source         Destination
7   0.579304 0.082577  172.16.100.153 172.16.100.154
RSVP Header. PATH Message.
HOP: IPv4 IF-ID. Control IPv4: 172.16.100.153. Data IPv4: 172.16.100.153.
LABEL REQUEST: Generalized: LSP Encoding=Packet, Switching Type=Packet-Switch
      Capable-1 (PSC-1), G-PID=IP
UPSTREAM LABEL: Generalized: 0x14
EXPLICIT ROUTE: IPv4 172.16.100.154

No. Time (s) Ptime (s) Source         Destination
8   0.914794 0.335490  172.16.100.154 172.16.100.153
RSVP Header. RESV Message.
SESSION: IPv4-LSP, Destination 172.16.100.249, Tunnel ID 3784, Ext ID 101000a.
HOP: IPv4 IF-ID. Control IPv4: 172.16.100.154. Data IPv4: 172.16.100.153.
TIME VALUES: 30000 ms
STYLE: Fixed Filter (10)
FLOWSPEC: Controlled Load: Token Bucket, 12500000 bytes/sec.
FILTERSPEC: IPv4-LSP, Tunnel Source: 172.16.112.1, LSP ID: 99.
LABEL: Generalized: 0x14

No. Time (s) Ptime (s) Source         Destination
9   2.278155 1.363361  172.16.100.146 172.16.100.145
RSVP Header. RESV Message.
HOP: IPv4 IF-ID. Control IPv4: 172.16.100.146. Data If-Index: 172.16.100.245,
     -2147483647.
LABEL: Generalized: 0x6dd000c, 0x295a4742

No. Time (s) Ptime (s) Source        Destination
10   2.977480 0.699325  172.16.105.26 172.16.105.25
RSVP Header. RESV Message.
HOP: IPv4 IF-ID. Control IPv4: 172.16.105.26. Data If-Index: 172.16.100.232,
     -2147483645.

------ Wireshark session 2
No. Time (s) Ptime (s) Source         Destination
11   -------- 0.497052  172.16.100.232 172.16.100.230
RSVP Header. RESV Message.
HOP: IPv4 IF-ID. Control IPv4: 172.16.100.232. Data If-Index: 172.16.100.230,
     -2147483594.

------ Wireshark session 1
No. Time (s) Ptime (s) Source       Destination
```

```
12   4.094135 ---------  172.16.105.9 172.16.105.10
RSVP Header. RESV Message.
HOP: IPv4 IF-ID. Control IPv4: 172.16.105.9. Data If-Index: 172.16.100.243,
     -2147483646.

No. Time (s) Ptime (s) Source         Destination
13  4.739199 0.645064  172.16.100.106 172.16.100.105
RSVP Header. RESV Message.
HOP: IPv4 IF-ID. Control IPv4: 172.16.100.106. Data If-Index: 172.16.100.246,
     -2147483647.
LABEL: Generalized: 0x6dd000c, 0x295a4742

No. Time (s)  Ptime (s) Source       Destination
14  5.969861  1.230662  172.16.112.2 172.16.112.1
RSVP Header. RESV Message.
HOP: IPv4 IF-ID. Control IPv4: 172.16.112.2. Data IPv4: 172.16.112.1.
LABEL: Generalized: 0x14
```

COS/CCS 2008-16