GMPLS multi-layer networking

Routing and constraint-based path computation in optical network segments

ALEXANDER LINDSTRÖM

KTH Information and Communication Technology

Master of Science Thesis
Stockholm, Sweden 2007

COS/CCS 2007-25

Royal Institute of Technology (KTH)
Stockholm, Sweden
2007-11-26

Master of Science Thesis
Alexander Lindström
alindstr@kth.se

GMPLS multi-layer networking

Routing and constraint-based path computation in optical network segments

Alexander Lindström
alindstr@kth.se

2007-11-26

Stockholm, Sweden

Supervisor
Ericsson Research
Annikki Welin

Academic Supervisor / Examiner
Royal Institute of Technology (KTH)
Gerald Q. Maguire Jr.

Abstract

In recent years, IP based end-to-end services have grown in popularity. Efficiently meeting the user demand for such services, different techniques for traffic engineering transport networks have been developed. One such technique, currently being developed for multi-layered networks, is Generalized Multi-Protocol Label Switching (GMPLS). GMPLS is a necessary networking technique because provisioning end-to-end services will today, and in the foreseeable future, very likely require the co-operation of multiple network layers. Here, the readiness of GMPLS for optical networks is investigated by reviewing the current support for optical networking components in the GMPLS standard documents. Based on this investigation, a candidate solution for routing and constraint-based path computation in optical network segments has been derived. This candidate solution is shown to efficiently handle the additional attributes and constraints inherent in optical networking components.

Sammanfattning

De senaste åren har IP-baserade tjänster ökat i popularitet. För att effektivt möta de användarkrav som ställs på sådana tjänster har olika tekniker för att styra transportnätverk utvecklats. En sådan teknik, nu under utveckling för multi-lagrade nätverk, är GMPLS. GMPLS är en nödvändig nätverksteknik eftersom tillhandahållandet av sluttjänster mellan olika användare idag, och inom en överskådlig framtid, mycket sannolikt kommer att kräva samarbete mellan flera nätverkslager. Här undersöks GMPLS färdighet i optiska nätverk genom att se över det nuvarande stödet för optiska nätverkskomponenter i GMPLS standarddokument. Baserat på denna undersökning har en kandidatlösning för routing och begränsad vägberäkning i optiska nätverkssegment tagits fram. Denna kandidatlösning visas effektivt hantera de ytterligare attribut och restriktioner som existerar i optiska nätverkskomponenter.

Keywords: *GMPLS, multi-layer, traffic engineering, service provisioning, intra-domain routing, OSPF-TE, RSVP-TE, path computation, PCE, PCC, optical constraints, optical impairments, wavelength continuity, blocking switch architecture*

Acknowledgments

I extend my gratitude to the confidence shown in me and support given by my supervisor at Ericsson Research, Annikki Welin. Her many efforts to ease my daily work process have been greatly appreciated. Further, my academic supervisor and examiner at KTH, Gerald Q. Maguire Jr., has been a great help in shaping the contents and form of this thesis. His many comments and constructive feedback have significantly improved the quality of this report.

Additional appreciation is extended to the staff at Acreo AB, Stockholm, Sweden. Their valuable input on the GMPLS framework, the modified GMPLS control plane software implementation, and optical networking has been of great help. In addition, I would like to express my appreciation for everyone that in some way either directly, or indirectly, have helped or supported me during the period of time this work was accomplished.

Table of Contents

List of Figures

List of Tables

1 Introduction

As the Internet has experienced a near exponential increase in traffic since the mid 1990s, the need for controlling traffic flows in core transport networks has increased. Controlling traffic flows changes the operating model from the best-effort dynamic routing to what is commonly referred to as traffic engineering. To enable rapid service provisioning and assure that suitable Quality of Service (QoS) is experienced by end users, service providers need efficient traffic engineering mechanisms.

Meeting this demand, a standardization process for Multi-Protocol Label Switching (MPLS) began within the Internet Engineering Task Force (IETF). MPLS was one of the early IETF initiatives to enable Traffic Engineering (TE) and offered many interesting new features, such as automated management of virtual paths. Although originally thought to address performance issues associated with datagram forwarding, MPLS instead proved valuable for automated service provisioning of both packet and frame based networks. However, due to the switching limitations inherent in MPLS, traffic engineering in such networks is usually restricted to the network edges; where data is packet or frame switched.

As a direct consequence, Generalized Multi-Protocol Label Switching (GMPLS) has been invented to introduce MPLS on multiple layers. GMPLS is being deployed to enable automated traffic control in multi-layer transport networks. It defines architectural components as well as a protocol suite. In practice, what GMPLS really is, is a framework for software based interaction between network elements. Because GMPLS is being designed for multiple layers, traffic engineering of end-to-end services utilizing core transport networks that are not packet or frame switched can be enabled (e.g. time division or optical networks).

One of the most pressing issues for GMPLS today is to efficiently support optical network segments. The main motivation for this is clear; many currently deployed IP user networks are connected via optical backbone networks. Increasingly, optical equipment's obvious advantage when connecting user sites with each other is the achievable bandwidths far surpassing other transport media. Combining many wavelengths into a single optical fiber, using Wavelength Division Multiplexing (WDM), offers additional possibilities for traffic engineering a transport network.

GMPLS has currently, however, little standardized support for optical network segments. This lack of support is manifested in that such network segments still can not be efficiently traffic engineered. Inherently, this is because optical network segments are exposed to physical impairments and other constraints not visible to higher layers. As a result, the GMPLS control plane and the generic Path Computation Element (PCE) it utilizes for path computations must be extended to provide additional functionality. This is necessary to enable efficient provisioning of autonomous end-to-end services with optical core transport networks.

1.1 Objectives

This thesis will focus on GMPLS applicability in optical network segments. More specifically, the goal is to provide a candidate solution for enabling the GMPLS routing process and a PCE for such network segments. The candidate solution will be implemented as an extension to an open source software suite. Verification will later be done by deploying the software suite in a GMPLS network comprised of virtual machines or using a physical network test-bed.

The following specific objectives will be addressed throughout this thesis:

- Investigation of constraints imposed by optical network segments
- Determination of the need for GMPLS and PCE extensions
- Derivation of a candidate solution for optical path computation in GMPLS
- Software implementation of the candidate solution
- Functional verification and analysis of the software implementation.

After the thesis has been completed, one of the deliverables will be a solution for computing viable network paths in optical network segments. In this work, most optical impairments and constraints should have been addressed.

1.2 Thesis outline

The thesis begins with a general introduction to the area (chapter 1). It is then divided into two parts; a literature study (chapters 2, 3, and 4) and a description of the candidate solution, its implementation, and evaluation (chapters 5 and 6). The thesis will conclude by summarizing the major results and limitations and present some future work (chapter 7). Useful material which supports the thesis, but is outside the main flow of the thesis will be presented in the appendixes.

The literature study consists of three chapters. The first of these chapters introduces the reader to the GMPLS architecture and protocol suite. The second presents the PCE and different approaches to path computation in label switched networks. To close, the third of these chapters examines vital optical switching constraints. An excellent reference to the GMPLS architecture and its applications is the book written by Farrel and Bryskin [1].

Chapter 5 describes and explains the candidate solution and its implementation. In this chapter, a virtual test-bed design, derived GMPLS and PCE extensions, and the software implementation are detailed and motivated. Chapter 6 provides verification and analysis of the software implementation. Here, different aspects of the software implementation functionality and efficiency are evaluated.

2 Introduction to GMPLS

This chapter introduces the GMPLS framework and its building blocks. In addition, a brief background of traffic engineering in MPLS networks is given.

2.1 Background

Traffic Engineering (TE) is a set of scientific principles encompassing control, measurement, modeling, and characterization of Internet traffic. As described in RFC 2702 [2], the goal of traffic engineering is to optimize network performance by applying different networking techniques. In many large Autonomous Systems (ASs), traffic engineering has become an indispensable asset due to the high cost of networking components and the competitive nature of provisioning Internet services. For MPLS, the TE principles of most interest are control and measurement.

The primary TE performance objectives can be divided into those which are traffic oriented or resource oriented. Traffic oriented performance objectives concern enhancing the QoS of traffic streams. These performance objectives are visible to network users and might include minimization of end-to-end delay or packet loss. Less visible to network users, resource oriented objectives concern efficiently utilizing network resources. Resource oriented performance objectives aim to ensure that no network resource is either over utilized or underutilized. Meeting such performance objectives allows for efficient utilization of deployed networking equipment.

The MPLS architecture, as described in RFC 3031 [3], implements the TE principles by assigning network traffic to Forwarding Equivalence Classes (FECs). FECs classify network traffic (e.g. depending on destination addresses and desired QoS) that will be forwarded in the same manner. Because FECs are mapped to contiguous sequences of next hops, assigning network traffic to a specific FEC will deterministically establish a path through the MPLS network. Since all information needed to forward network traffic belonging to a specific FEC has also been installed into the MPLS network, subsequent hops need not analyze the network traffic further. This is a consequence of network traffic being assigned to a FEC, and forwarded accordingly, as it *enters* the MPLS network.

Using MPLS terminology, assignment to FECs is encoded using labels. The labels are link-local random 20-bit values inserted as “shim” headers. Based on these labels, the network traffic belonging to a specific FEC is forwarded throughout the MPLS network domain. More specifically, ingoing label-to-interface pairs are mapped to outgoing label-to-interface pairs to establish routes through MPLS enabled network routers; these are called Label Switching Routers (LSRs). By installing mappings into a set of contiguous LSRs, a switchable trail of labels is created, called a Label Switched Path (LSP). Given that a LSP has been established between two domain edge LSRs, network traffic can then be forwarded through the MPLS network by (1) inserting a label at the incoming edge LSR (2) switching the traffic based on that label within the MPLS domain, and (3) removing the label at the outgoing edge LSR and forwarding the network traffic as usual (see figure 2.1).

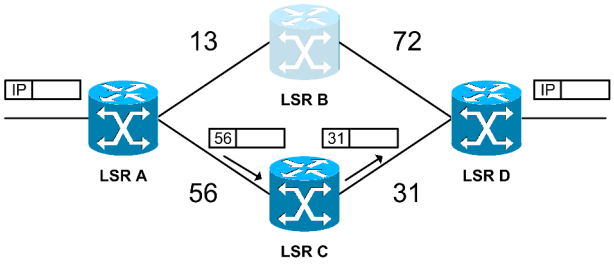

Figure 2.1: A LSP in an MPLS network. For this LSP, LSR B never participates in data forwarding.

In order to manage a traffic engineered MPLS network, LSRs implement a control plane. In this control plane, connected LSRs can exchange control information via extended signaling and routing protocols. More specifically, LSRs can request LSP establishment within the controlled network domain and distribute network topology information. Depending on how the MPLS network is managed, state might be altered either manually (e.g. via manual operation) or automatically. From now on, a LSR implementing the GMPLS control plane will be referred to as a Generalized LSR (GLSR).

2.2 Architectural components

This section will explain the main architectural components that comprise the GMPLS framework as described in RFC 3945 [4]. Because GMPLS is merely a set of MPLS extensions, only components specifically extended by GMPLS are presented.

2.2.1 Control plane extensions

Commonly referred to as the Multi-Layer Control Plane (MLCP), the GMPLS control plane is extended to support multiple switching layers. In GMPLS, there is a clear separation between the MLCP and the data plane (see figure 2.2). Unlike control signaling in MPLS, the MLCP can manage the GMPLS network out-of-band; hence control signaling need not follow the forwarded data. This means that the MLCP can continue to function although there is a disruption in the data plane and vice versa. What is more, this allows for separate control channels to be used for the MLCP. By deploying the MLCP on separate control channels, the other channels are completely dedicated to forwarding data.

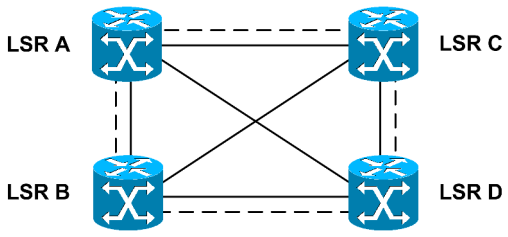

Figure 2.2: Separating the control and data planes. Dashed lines indicate the MLCP, while the solid lines identify data links.

In GMPLS, the MLCP utilizes routing and signaling protocols to traffic engineer the network in which it is based. These protocols are extensions to well-known protocols of the TCP/IP protocol suite. As such, the MLCP implements IPv4 or IPv6 addressing. This also applies to the data plane, but in cases where addressing is not feasible, or convenient, unnumbered links (i.e. links without network addresses) are supported. MLCP addresses are not required to be globally unique (however global uniqueness is required to allow for remote management). However, addressing in the MLCP is separated from that in the data plane. Essentially this is how the MLCP is separated from the data plane.

GMPLS supports deploying the MLCP according to the overlay, peer (integrated), or augmented (hybrid) models. In the overlay model, the network layers are clearly separated. This means that in order for a client layer to utilize some specific server layer it must request that service via a network interface. Using the peer model, all network layers are peers. As such, they have full visibility of each other and client layers can signal unhindered through serving layers. Thus, this model is very suitable for smoothly installing end-to-end services. The augmented model, in turn, is a hybrid model allowing for limited peering according to some implemented policy. By supporting these service models, GMPLS seems very suitable for independent control of multiple network layers.

2.2.2 Generalized labels

The labels in GMPLS have been generalized from those used in MPLS. Generalized labels are tightly coupled to network resources. In contrast to MPLS where labels merely represent network traffic, generalized labels represent network resources. For example, a generalized label on an optical link could identify a wavelength or fiber, while on a packet switched link it would simply identify network traffic, just as in MPLS. In the lower layers, generalized labels are “virtual” meaning that they are not inserted into the network traffic, but instead implied by the network resource being used (e.g. wavelength or fiber). This is necessary since neither packets nor frames are recognized at the lowest layers which GMPLS supports. Generalizing the label format, the conventional MPLS label has been extended to 32 bits (see figure 2.3).

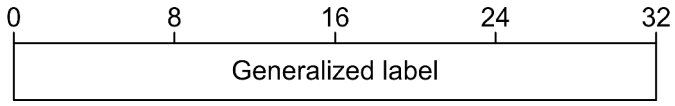

Figure 2.3: The generalized label. When smaller labels are represented they are right-justified within the label.

The interpretation of a generalized label is link-local and depends on the encoding of the interfaces (or resource) it labels. GMPLS defines labels for a specific set of interface types. More precisely, labeling of interfaces capable of fiber switching (FSC), lambda switching (LSC), time-division multiplexing (TDM), layer-2 switching (L2SC), and packet switching (PSC) are currently defined.

2.2.3 Bidirectional data paths

The core activity of GMPLS is to establish TE data paths in enabled networks. A data path between two GLSRs is abstracted by the LSP. Thus, a LSP consists of consecutive labels which, when swapped in a specific order, carries data from one point in a label switched network to another. In short, a LSP is represented by the distributed state needed to send data along a specifically traffic engineered route.

Because LSPs might differ in link composition, an entity requesting labels for a LSP needs to specify three major parameters: switching type, encoding type, and Generalized Payload-ID (G-PID). Switching type defines how an interface switches data. Because this is always expected to be known, a switching type needs only be specified for an interface with multiple switching capabilities. Encoding type is needed to specify the specific encoding of the data associated with the LSP. For example, data associated with an L2SC interface might be encoded as Ethernet. The G-PID finally defines the client layer of the LSP. This parameter is necessary to let the LSP ingress and egress identify what client layer utilizes the LSP.

In GMPLS, bidirectional LSPs are considered the default (see figure 2.4). Unlike MPLS, bidirectional data paths can be established without signaling for two unidirectional LSPs. Bidirectional LSPs are established through simultaneous label distribution in both directions. This halves the signaling overhead, albeit increasing the probability of race conditions for network resources. Such resource race conditions will occur when two bidirectional LSPs are simultaneously signaled in reverse directions; decreasing the likelihood of successful installation of traffic engineered data flows. How GLSRs signal bidirectional LSPs is detailed in later sections (see section 2.4).

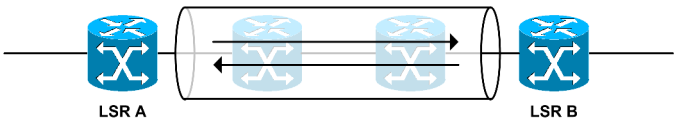

Figure 2.4: A bidirectional LSP tunnel.

2.2.4 Hierarchies

Since generalized labels are non-hierarchical, they do not stack. This is because some supported switching media can not stack. Given an optical link, for example, it is not possible to encapsulate a wavelength in another and then deterministically get it back again. In GMPLS, tunneling data through different layers is therefore based on LSP nesting (i.e. encapsulating LSPs within LSPs). LSPs can be nested either within or between network layers (i.e. switching types), but nesting is always based on some sort of LSP hierarchy. By exploiting LSP hierarchies, multiple layers can be connected and data plane scalability increased (e.g. by establishing forwarding adjacencies).

Ordering LSPs hierarchically within a network layer requires that LSP encodings themselves are hierarchical. When hierarchically ordering layers there is, however, a natural LSP hierarchy based on interface types. At the top of this hierarchy are FSC interfaces followed, in decreasing order, by LSC, TDM, L2SC, and PSC interfaces. This order is because wavelengths can be encapsulated within a fiber, time slots in wavelengths, data link layer frames in time slots, and finally network layer packets in data link layer frames. As such, an LSP starting and ending on PSC interfaces can be nested within higher ordered LSPs.

2.2.5 Protocol suite

The MLCP can make use of several signaling and routing protocols. These protocols can be divided into three distinct sets based on their functionality: routing, signaling, and link management.

Routing protocols must be implemented by the MLCP to disseminate the network topology and its TE attributes. For this purpose, Open Shortest Path First (OSPF) [5] with TE extensions [6] (OSPF-TE) or Intermediate System to Intermediate System (IS-IS) are currently defined. To account for multiple layers, however, GMPLS needs to add some minor extensions to these existing protocols. The GMPLS routing process using OSPF-TE is explained in the following sections (see section 2.3).

The signaling protocols are concerned with establishing, maintaining and removing network state (i.e. setting up and tearing down LSPs). For signaling, GMPLS can use either Resource ReSerVation Protocol (RSVP) [7] with TE extensions [8] (RSVP-TE) or Constraint-based Routing-Label Distribution Protocol (CR-LDP). Again, supporting multiple layers requires some extensions to existing protocols. Basic GMPLS signaling is explained in the following sections (see section 2.4).

For link management, a new protocol called the Link Management Protocol (LMP) has been defined. LMP can be used by GMPLS network elements to discover and monitor their network links (i.e. their connectivity). Since network links must always be advertised accurately, this is a vital part of GMPLS. To enable link discovery between an optical switch and an optical line system, LMP has been further extended, creating LMP-WDM. This extension can provide the MLCP with useful information about optical network segments. Nevertheless, since this is out of the scope of this thesis, neither LMP nor LMP-WDM will be considered further in this thesis.

2.3 Routing with OSPF-TE

This section describes the GMPLS routing process as defined in RFC 4202 [9]. Here, OSPF-TE and its GMPLS extensions [10] are considered in the context of routing.

2.3.1 Network topology dissemination

To enable automated configuration of the controlled network, GMPLS defines an intra-domain routing process. Via this routing process the network topology and its TE attributes are disseminated within the traffic engineered domain. This routing process is implemented using routing protocols specified for the MLCP. However, this routing process is not used for routing user traffic, but only for distributing information in the MLCP. Extensions to existing protocols necessary for this routing process were therefore created.

Essentially, with OSPF-TE, participating GLSRs first establish routing adjacencies by exchanging hello messages. After routing adjacencies have been established, the GLSRs then synchronize their link state databases. This is done by exchanging database description packets. The database description packets contain at least one database structure referred to as a Link State Advertisement (LSA). Different LSA types exist but all share a common 20-byte header (see figure 2.5) and have a payload describing the advertised links. For GMPLS routing purposes, the primary operation is to flood LSAs throughout the MLCP domain by appending them to "link state update" messages periodically sent between adjacent GLSRs. To avoid interference with any ordinary routing processes, a TE LSA is made opaque. Such an opaque LSA is a special type of LSA only processed by specific applications (e.g. the GMPLS routing process).

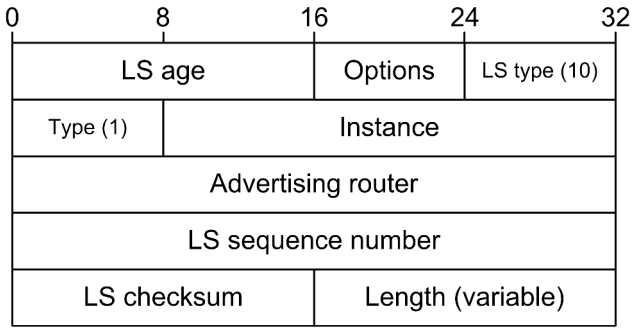

Figure 2.5: An opaque LSA header.

By extending the link state database with TE information, a Traffic Engineering Database (TED) is produced. From this TED, a network graph with traffic engineering content can be computed. Constructing a TE network graph is necessary to provide input for the constraint-based algorithms subsequently used to compute network paths. Different ways of computing network paths are presented in the following chapter (see chapter 3).

2.3.2 Type-Length-Value triplets

When flooding LSAs, each OSPF routing message contains a common 24-byte header which is used to forward it. This routing message header includes information about message type, addressing, and integrity. Within this header, LSAs are then encapsulated and specific payloads appended to each LSA. In order to enable advertisement of TE attributes in opaque LSAs, the LSA payload consists of Type-Length-Value (TLV) triplets (see figure 2.6). These TLV triplets contain arbitrary data structures defined by two 2-byte fields: the “type” and “length” fields.

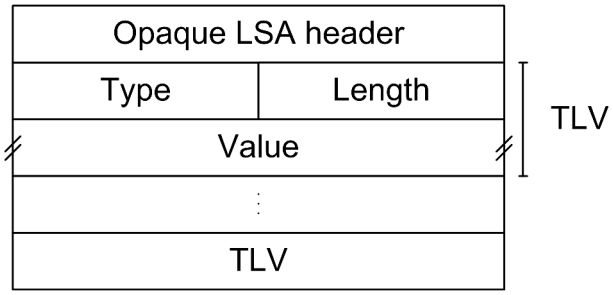

Figure 2.6: The format of opaque LSAs which are flooded by the GMPLS routing process using OSPF-TE.

Using TLVs, router addresses and TE links can be expressed. In GMPLS, if an advertising router is reachable, a “router address”-TLV can be used to describe a network address at which this router (i.e. GLSR) can always be reached. In turn, the “link”-TLV can be used to abstract advertised TE links. Because several sub-TLVs have already been defined for the “link”-TLV, multiple TE attributes can be represented on each link. In fact, new link sub-TLVs describing additional TE information (see table 2.1) are the *only* GMPLS extensions to OSPF-TE.

Sub-TLV name	**Type**	**Length**	**Value**
Link Local/Remote Identifiers	11	8 bytes	2 x 4 bytes local/remote link identifiers
Link Protection Type	14	4 bytes	1 byte for link protection (3 bytes reserved)
Interface Switching Capability Descriptor	15	variable	Minimum 36 bytes for ISCD information
Shared Risk Link Group	16	variable	*N* x 4 bytes for link SRLG identification

Table 2.1: GMPLS specific sub-TLVs. These are all appended to the ”link”-TLV. Length excludes “type” and “length” fields.

2.4 Signaling with RSVP-TE

This section describes the basics of GMPLS signaling as defined in RFC 3471 [11]. Here, RSVP-TE and its GMPLS extensions [12] are considered in the context of signaling.

Because RSVP-TE inherits its design from the RSVP protocol, it is based on distributing various signaling objects (see figure 2.7). These signaling objects, in turn, have been grouped. These groups contain mandatory and optional signaling objects (e.g. installing LSP state requires a mandatory set of signaling objects). Encapsulating the groups with a common header, distinct signaling messages are created. When a GLSR receives a signaling message, the resident objects are examined and interpreted based upon the message type indicated by the common header.

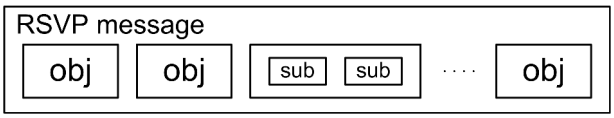

Figure 2.7: The conceptual RSVP message format. This RSVP message contains an arbitrary set of signaling objects.

Extending the RSVP-TE protocol for GMPLS was thus a matter of generalizing existing signaling objects, including some new objects (see table 2.2), and adding some minor signaling enhancements (e.g. signaling bidirectional LSPs and rapid notification). Considering the RSVP-TE protocol with GMPLS extensions for signaling, each signaling message contains a common 8-byte header. The common header defines the message type followed by the encapsulated objects. Encapsulated objects, in turn, are of variable length and contain a 4-byte header defining the object length, class, and type within class.

Object name	**Length**	**Message**	**Description**
Generalized Label Request	4 bytes	Path	Describes the requested LSP
IF_ID RSVP_HOP	variable	Path/Resv	Defines what interface to label
Generalized Label	variable (4 bytes)	Resv/ResvErr	Downstream label
Upstream Label	variable (4 bytes)	Path/PathErr	Upstream label
Label ERO	2 bits + label	Path/Resv	Explicit label control
Suggested Label	variable (4 bytes)	Path/PathErr	Label suggestion
Label Set	variable	Path	Label selection restriction

Table 2.2: Important GMPLS signaling objects. These are all specific to GMPLS. Length excludes the object header.

Furthermore, RSVP-TE implies downstream-on-demand label distribution (just as in RSVP). This means that upstream GLSRs request downstream GLSRs to select labels for the TE links connecting them. In this way, each GLSR acknowledges a request to install an LSP, forwards the request to the next downstream hop, and awaits the response. As a response is returned upstream, the GLSR can install a cross-connection (i.e. state describing ingoing and outgoing label-to-interface mappings and associated network resources) for this LSP. Here, downstream is defined as the direction in which data would flow on an unidirectional LSP properly installed (that is, the direction from LSP ingress to LSP egress).

2.4.1 Installing LSP state

Establishing bidirectional LSPs employing RSVP-TE for signaling requires full sets of *Path* and *Resv* messages to be exchanged between two GLSRs (see figure 2.8). Initially, a sender GLSR (LSP ingress) requests a LSP to be set up by sending a *Path* message downstream to the next hop. This *Path* message contains an UPSTREAM_LABEL object defining the label to use in the upstream direction, objects describing the data flow, and a GENERALIZED_LABEL_REQUEST object for requesting the LSP. If the *Path* message is successfully received, the next hop then reserves path state to enable correct signaling of returning *Resv* messages and saves the upstream label. The next hop then selects its own upstream label, creates state for the upstream direction, replaces the upstream label in the *Path* message and passes it on downstream to the next hop. This procedure is repeated until the next hop is the receiver GLSR (LSP egress). The LSP has now been established in the upstream direction, but no state has been saved in the downstream direction (i.e. label distribution is downstream-on-demand). Consequently, the receiver GLSR now selects a downstream label and returns a *Resv* message upstream. This *Resv* message mimics the *Path* message, but inserts a GENERALIZED_LABEL object defining the selected downstream label. If the *Resv* message is received successfully, the previous hop (the signaling direction has changed) then sets state for the downstream direction, replaces the downstream label with its own selected label and passes the *Resv* message further upstream. This procedure is repeated until the sender GLSR successfully receives the *Resv* message corresponding to a dispatched *Path* message. Now, the requested LSP has been fully established and is ready to tunnel data in both directions.

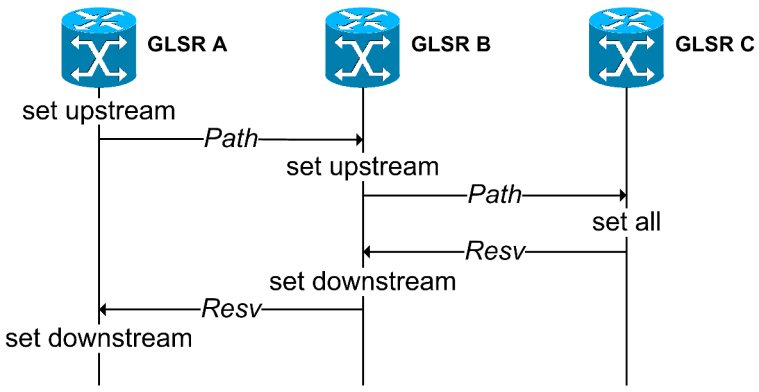

Figure 2.8: Installing state for a bidirectional LSP.

2.4.2 Removing LSP state

RSVP-TE is a soft-state protocol. This means that it continuously sends messages refreshing timers associated with installed state. Originally designed for MPLS, the "softness" is somewhat reduced in GMPLS, however timers are still implemented. LSP state removal can be triggered in two ways: when a timer expires in a GLSR or by some external mechanism (e.g. manual operator or management system).

To remove LSP state using RSVP-TE, *PathTear* or *PathErr* messages are dispatched (see figure 2.9). A *PathTear* message is dispatched downstream following the path of a *Path* message, while a *PathErr* message is sent upstream following the path of a *Resv* message. As these messages are processed by GLSRs they immediately clear, or partially clear, the LSP state. This enhancement is specific to GMPLS and enables the LSP egress and intermediate GLSRs to initiate LSP state removal. Using the *PathErr* message for clearing state, a flag introduced by GMPLS is set to indicate that path state is no longer valid (i.e. the Path_State_Removed flag). This means that GMPLS can tear down LSP state in both directions (both upstream and downstream). Additionally, GMPLS provides rapid error notification via the newly defined *Notify* message. The *Notify* message can be used to inform an LSP ingress or egress of errors, enabling them to initiate state removal in the place of an intermediate GLSR. Although the *PathErr* message is, strictly speaking, not needed, it can increase signaling efficiency by eliminating the need for notification.

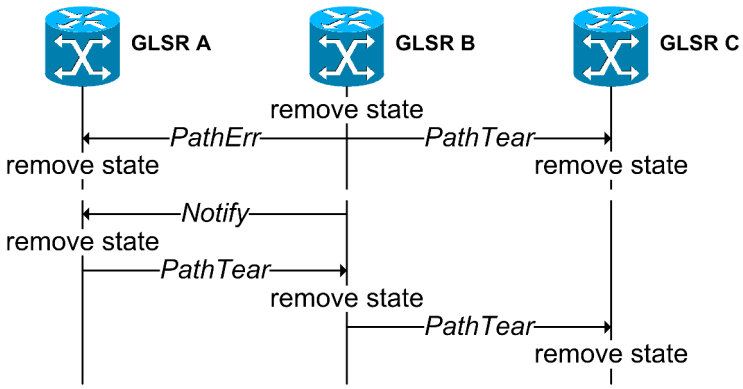

Figure 2.9: Two ways of removing LSP state.

2.4.3 Error handling

While the above description of the signaling procedures presumed that no errors occurred during signaling, this is unlikely to always be true. Thus, a need for error handling messages is implied. When errors occur, *PathErr* or *ResvErr* messages can therefore be signaled. A *PathErr* message indicates an error in processing a *Path* message and is sent upstream towards the LSP ingress. Similarly, a *ResvErr* message indicates an error in processing a *Resv* message and is sent downstream towards the LSP egress. A GLSR receiving an error message may try to correct the error itself, if minor, or pass it further on.

2.4.4 Explicit routes

To traffic engineer specific routes, the EXPLICIT_ROUTE object (ERO, see figure 2.10) must be included in the *Path* and *Resv* messages exchanged during LSP state installation. When used in *Path* messages, the ERO describes the next and previous hop for any GLSR along the explicit route. Thus, when signaling paths explicitly using an ERO, path state is not needed to indicate a reverse route, since returning *Resv* messages can instead be routed based upon the ERO. The ERO might define order dependent hops (i.e. strict hops) or hops that need only be visited regardless of order (i.e. loose hops). To deterministically install an LSP in a GMPLS network, an ERO must only define strict hops.

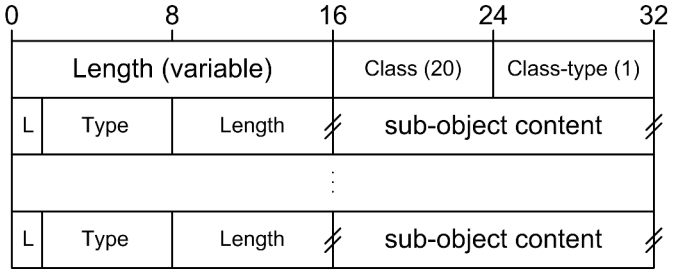

Figure 2.10: The EXPLICIT_ROUTE object (ERO). The figure includes the common 4-byte signaling object header.

While not specific to GMPLS, the ERO signaling object has been extended to support explicit label control. This is done via the label ERO sub-object (see figure 2.11), which defines what labels to install on specific interfaces along an explicit route. Expressing the labels to install on an interface, one or more label ERO sub-objects (both upstream and downstream labels may be specified) are inserted next to an ERO sub-object. This way, making use of the label ERO sub-object, a set of available GLSR interface labels could be selected and signaled. In GMPLS, signaling explicit routes with an ERO is considered the default way to signal the setup of an LSP.

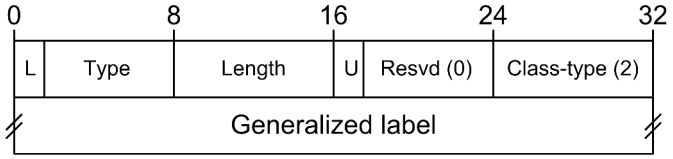

Figure 2.11: The label ERO sub-object. The figure excludes hierarchically higher objects.

3 Constraint-based path computation

This chapter will present the application of path computation in GMPLS networks. Here, architectural examples and some proposed algorithms are given.

3.1 Introduction to the PCE

A Path Computation Element (PCE), as defined in RFC 4655 [13], is a generic abstraction for computing TE paths in label switched networks. How a PCE implements its functions is not defined. However, the PCE framework defines several ways to implement path computation. The only responsibility of a PCE is to compute paths, not to signal them.

Path computation is requested when a Path Computation Client (PCC) actively sends requests to a PCE describing the path it wants to have computed. The PCC, embodied by any network element interested in computing a network path (e.g. an edge GLSR), then awaits the PCE response. When a response is returned, the PCC can signal the returned path with relatively high assurance of successful setup; however, due to path contention no guarantees of success can ever be given. As such, a PCC and PCE interact using a request-response model (see figure 3.1).

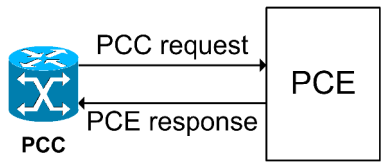

Figure 3.1: PCC to PCE interaction.

3.1.1 Architectural models

Several architectural models have been defined for a PCE. Given this, PCEs can be modeled as either distributed or centralized; in combination with being either composite or external (meaning that there is a total of four model types). Since each model has its own implications, they each also have their own uses.

When distributing several PCEs throughout the network domain, a PCE can be deployed in the network elements potentially needing to issue requests (e.g. edge network elements). This would balance the computational load between deployed PCEs, but increase the risk for path contention (e.g. if multiple paths are computed simultaneously). While in the centralized model, only a single PCE is deployed for the network domain resulting in a single point of failure possibly prone to computational bottlenecks (when many PCC requests are issued simultaneously).

A composite PCE is placed within a network element performing some other function (e.g. as a GLSR software upgrade). Conversely, an external PCE is implemented in a network element dedicated only to path computation. Implementing composite PCEs requires processing resources from hosting network elements. On the other hand, external PCEs may increase the network load and response latency since all PCCs are now remote, hence they must use network bandwidth to issue their requests (possibly resulting in a high network delay, see figure 3.2).

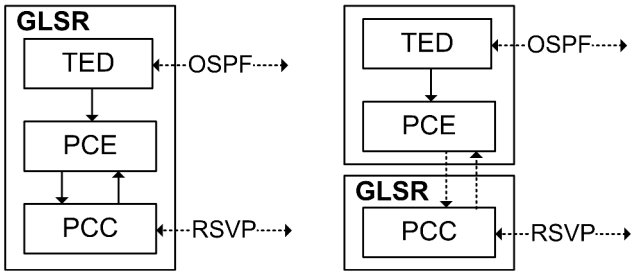

Figure 3.2: Composite (left) and external (right) PCEs. Dashed lines indicate external communication.

3.1.2 Operational modes

The PCE may also operate in different modes. However, the main operation is to apply a constraint-based algorithm to a network graph when computing a path. Which algorithm is applied is, as was earlier stated, not defined by the standard documents. Popular constraint-based algorithms are described in the following section (see section 3.2).

Furthermore, path computation can be performed by a single or multiple PCEs. Thus, several PCEs could distribute a PCC request between them, sharing the computational load. This need not be visible to a requesting PCC, but merely be an internal distribution of computational load. Nevertheless, if a single PCE is deployed (according to the centralized model), the use of a single PCE is naturally inferred (although it could be a multiprocessor node).

Finally, a PCE could be stateful or stateless. The stateful variant of these keeps track of all TE routes it has computed and returned. This means, in contrast to a stateless PCE, that not only the network state and available resources would be monitored, but also information about the allocated resources. Although being stateful would also increase computational overhead, compared to a stateless PCE, keeping state can potentially enable unsolicited PCE interaction. This is a very neat feature that, if the PCC to PCE communication would be extended, could mean that a PCE receiving a modified link advertisement could re-compute effected paths and inform any effected PCC. Effectively, a PCE could preempt future path computation requests generated by PCCs experiencing errors in the data plane.

3.2 Constraint-based algorithms

Farrel and Bryskin describe several popular constraint-based algorithms in [1]. The purpose of path computation with constraint-based algorithms is to find network paths that meet given requirements and constraints. In this section, such algorithms are presented.

3.2.1 Functional overview

Computing paths in a network domain, a conventional path computation algorithm tries to find the shortest path between single or multiple network elements. This is done by, in different ways, operating on a network graph built from a TED. Some input is processed and a single or a set of paths is returned. Here, the term "shortest" refers to some sort of minimum cost and is represented by a single metric; often bandwidth. Thus, such algorithms are often called Shortest Path First (SPF) algorithms (see figure 3.3).

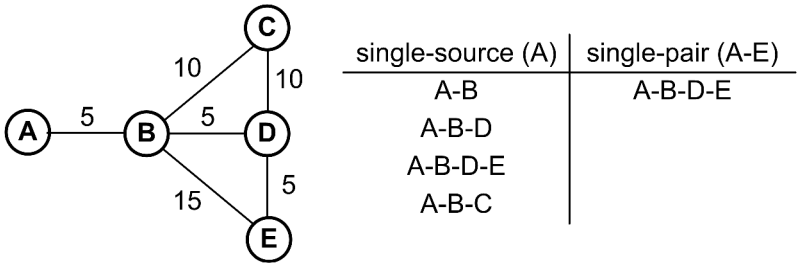

Figure 3.3: Single-source and single-pair algorithms. The table holds computed paths for the respective algorithms.

Sometimes considering only a single metric is not sufficient. This is especially true for optical network segments which impose multiple path constraints due to the low-layer nature of optical switches. Constraint-based algorithms take this into consideration; being capable of computing paths while resolving multiple constraints.

Implementing a constraint-based algorithm, it is important to distinguish between link-type (limited to links only, e.g. available bandwidth) and path-type (that apply to entire paths, e.g. end-to-end delay) constraints. Because these constraint types have different effects on path computation, they should be handled in different ways.

Link-type constraints are efficiently handled by grooming network graphs (see figure 3.4). This way, network graphs are merely pruned out of links not satisfying all specified constraints. For example, links with less available bandwidth than that requested could simply be removed from a network graph before it is operated on by a search algorithm. Consequently, link-type constraints can be handled with ease by only pre-processing network graphs.

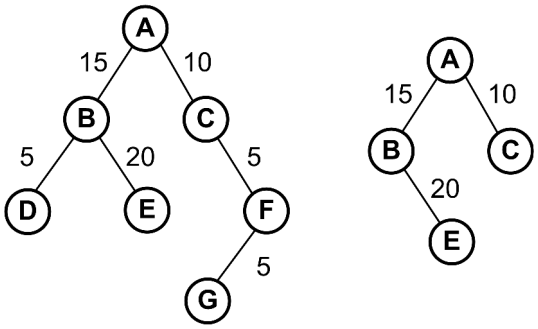

Figure 3.4: Network graph grooming. A network graph before (left) and after (right) pruning out all links with weights less than 10.

Path-type constraints can, on the other hand, not be handled when a network graph is pre-processed (because paths do not exist until they have been discovered). Instead, path-type constraints could be continuously evaluated using path-evaluation functions. By defining path-evaluation functions, entire paths can be continuously approved or discarded given specified constraints. For example, each time a candidate path is discovered (perhaps being an extension of an earlier found path), accumulated bandwidth could be compared to some maximum value by calling a path-evaluation function. If the called path-evaluation function would return true, then the evaluated path would be considered viable. This way, path-type constraints for entire paths can be evaluated.

3.2.2 Proposed algorithms

Given that a network graph has been groomed out of links not satisfying some specified link-type constraints, at least three different methods approaches to Constrained SPF (CSPF) algorithms exist: (1) computing paths using a conventional SPF algorithm after which the computed path is evaluated, (2) initializing an SPF algorithm to compute several paths and then sequentially request and evaluate the computed paths, and (3) concurrently compute all possible paths and immediately discard computed paths not satisfying some specific constraint.

When implementing a CSPF algorithm, that considers path-type constraints, the first method begins by first selecting a preferred SPF algorithm (see table 3.1). Then, the selected SPF algorithm must be modified to evaluate path-type constraints during path computation. This can be done by evaluating discovered sub-paths when additional hops are added (i.e. during arc relaxation). If a sub-path does not meet some specified path-type constraint when evaluated, then there is no point in considering this path further and this path is pruned from further consideration. Requesting network paths from such an SPF algorithm, only viable network paths fulfilling some specified constraints will be returned. In addition, accounting for path-type constraints that may be compensated for requires that the sub-path discarding optimization is never employed. This is because some future hop might modify a path to meet some earlier violated path-type constraint. However, note that some constraints can never be compensated for (e.g. end-to-end delay).

SPF Algorithm	Description	Run-time
Bellman-Ford	Iteratively traverses all arcs |V|-1 times, single-source, can detect negative loops	O(|V||A|)
Dijkstra	Uses a minimum priority queue, single-pair, can not account for negative weights	O(|V|lg|V|+|A|) *depends on queue
Modified Dijkstra	Uses a minimum priority queue, single-source, can account for negative weights	-
Breadth First Search	Does breadth first search, single-source, can be optimized for single-pair	O(|V|+|A|)

Table 3.1: Popular SPF algorithms. V is the set of vertices and A the set of arcs on a considered network graph.

Another approach would be to employ a K Shortest Paths (KSP) algorithm. Such an algorithm computes the *k* shortest paths between two network elements. This is analogous to iteratively calling an SPF algorithm while in between modifying the network graph. However, this type of algorithms are usually optimized for this type of task. Once a KSP algorithm has been initialized, paths can be sequentially requested and evaluated using path-evaluation functions. When a suitable path is then found, it can be returned by the path computing entity (e.g. PCE).

To close, all paths could be computed concurrently. Rather than sequentially evaluating computed paths, paths could be computed using an algorithm based on the Optimal Algorithm for Maximal Disjointness. Such an algorithm would grow all possible paths concurrently, immediately discarding those not meeting specific path-type constraints. Essentially, this can be done by iteratively initializing path candidates, evaluating constraints, and detecting loops at each hop until a single or several viable paths are found. This type of algorithm would be computationally more expensive, but be capable of handling both link-type and path-type constraints. Note that the above arguing for not discarding evaluated sub-paths still applies.

4 Optical switching constraints

Efficiently enabling GMPLS for optical network segments, several constraints must be considered. Primarily, these constraints are imposed by physical impairments, limited switching capabilities, and limited connectivity. Here, several such constraints inherent in optical components are examined.

4.1 Wavelength switching

The applicability of GMPLS and a PCE for wavelength switching has been discussed by Bernstein, et al. in a recent IETF Internet draft [14]. This Internet draft details additional wavelength specific information needs and the inability to do wavelength conversions.

4.1.1 Routing implications

To begin, additional wavelength-specific information needs to be disseminated in the GMPLS control plane. This is to increase the granularity of bandwidth allocation and allow for the wavelengths available on GLSR interfaces to be considered by a PCE. Thus, additions to the GMPLS routing process will be necessary.

First of all, the need for wavelength-specific bandwidth information is necessitated by the nature of WDM links. As of now, the MLCP disseminates information about maximum bandwidth, maximum reservable bandwidth, and unreserved bandwidth. However, since each wavelength (or a band of wavelengths) on a WDM link might have a different bandwidth, available bandwidth might not be uniformly distributed (see figure 4.1). This means that a tenth of the available bandwidth on a WDM link is not automatically reserved simply because a tenth of its available wavelengths has been reserved. To understand why this is, imagine a WDM link supporting wavelengths λ_1 – λ_{10} with an available bandwidth of 70 Gbit/s. In this case λ_1 – λ_4 might each have a bandwidth of 2.5 Gbit/s, while λ_5 – λ_{10} might have a bandwidth of 10 Gbit/s each. Thus, there is a need to distribute information about maximum bandwidth per wavelength on WDM links in the MLCP.

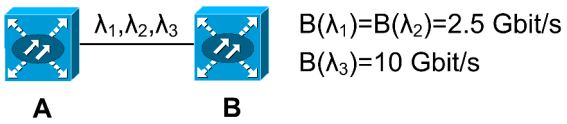

Figure 4.1: Wavelengths with different bandwidths on a WDM link.

In order to also know which set of wavelengths are available on any given link, the availability of wavelengths needs to be advertised in the GMPLS routing process. What this advertisement would look like is currently not defined in the GMPLS standard documents. One approach includes advertising a bitmask indicating available and occupied wavelengths via the link sub-TLV describing the interface switching capabilities. However, care should be taken not to make such a bitmask ambiguous or congest a control plane with this type of information.

In addition, the limited ability of an optical switch to receive a given wavelength and emit another may limit the connectivity in an optical network segment. Thus, a way to describe the conversion capabilities of an advertised interface would also be desired. At present, wavelength selective interfaces can be said to be LSC; however, no further specification of the level of supported wavelength conversion is (currently) possible. As a consequence, additional information describing convertible wavebands or the lack of conversion capability will be needed. Again, the link sub-TLV used to describe interface switching capability could be used for this purpose.

4.1.2 Full conversion capability

Full conversion capability exists when all optical switches in an optical region are able to convert all supported wavelengths on all their interfaces (see figure 4.2). This is typically the case when deploying opto-electronic-optic (OEO) switches that transform optical signals to electronic form during processing. OEO switches treat optical signals as bit streams. This enables compensation for optical impairments (by regenerating optical signals) and full freedom to select outgoing wavelengths (using tunable lasers). Processing bit streams also enables measurement of the Bit Error Rate (BER) induced by optical impairments, hence, such optical switches are often referred to as being “intelligent”.

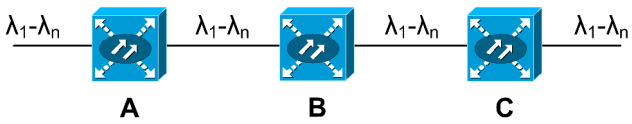

Figure 4.2: Full conversion capability of OEO switches.

Because OEO switches are capable of full conversion, wavelength assignment can be treated link-locally when establishing bidirectional LSPs in optical network segments comprised of such optical switches. Hence, the wavelength assignment problem need not be resolved by GMPLS for such cases.

The above implies that path computation need not be performed in this type of optical network segment. Hence, since a PCE will not be needed for link-local wavelength selection, a GLSR may instead simply suggest what link-local wavelengths to use on a specific link. Nevertheless, considering available wavelengths via a PCE could still prove to be meaningful (e.g. as wavelengths might represent different bandwidths, and this type of network segment might interface to network elements not capable of full wavelength conversion).

Nevertheless, full conversion capability is expensive. This is because of the opto-electronic transformation done in OEO switches, requiring the electronics to run at the maximum data rate of the optical media. Consequently, OEO switches will also impose constraints on bit rates because data is processed electronically (and suffer from electronical processing limitations). Hence, mechanisms to enable less expensive equipment to be deployed would be preferable. This is further described in the next section (see section 4.1.3).

4.1.3 Limited or no conversion capability

Limited or no conversion capability exists in optical network segments which are not capable of full wavelength conversion. Consequently, not all optical switches will be able to convert all wavelengths on all their interfaces (see figure 4.3). In the extreme case where no optical switch is able to convert any wavelength on any interface, no wavelength conversion will be possible.

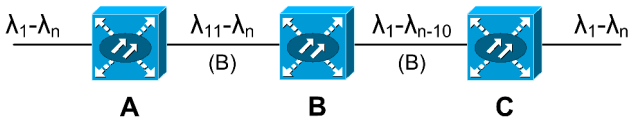

Figure 4.3: Limited conversion capability. B can not convert all wavelengths on its interfaces.

Optical network segments with limited conversion capabilities are referred to as transparent, and impose specific constraints. Data switching in such network segments is done by all-optical (OOO) switches. OOO switches do not transform the optical signal they process. Instead, they switch data using all-optical technologies (e.g. by adjusting micro mirrors to reflect specific wavelengths). Because there are limitations in all-optical switching, such switches can not, today, convert the wavelengths they process. However, because such devices are cost-efficient and have no restrictions on throughput (transparent switches are not at all aware of bit rates since they simply forward photons) enabling them for GMPLS (and vice versa) is imperative.

For signaling purposes, another IETF Internet draft standardizing the wavelength label has been written by Otani, et al. [15]. In this draft, a standardized label format is proposed for both coarse and dense WDM interfaces. The proposed label format specifies wavelengths according to the wavelength grids specified by the ITU-T [16] [17] (see figure 4.4). This eliminates ambiguity imposed by link-local wavelength perception and allows for signaling LSPs efficiently through optical network segments. Standardizing the wavelength label does not impose any constraints on signaling or routing, it merely enables the label abstraction to be significant at the control plane level.

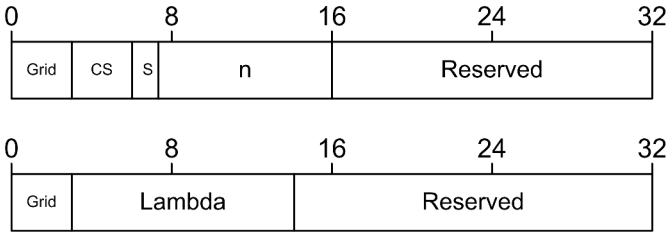

Figure 4.4: Proposed standardization of DWDM (top) and CWDM (bottom) lambda labels.

For the application of path computation, transparent optical segments translate into a "wavelength continuity constraint"; that is, all consecutive TE links connected by transparent switches must use the same wavelength (see figure 4.5). In order to solve the resulting problem, computed paths must be evaluated with the proper constraints. Thus, TE information describing wavelength conversion capabilities of advertising switches and available wavelengths on TE links are, as a minimum, needed as input to a path-evaluation function used by a constraint-based path computation algorithm.

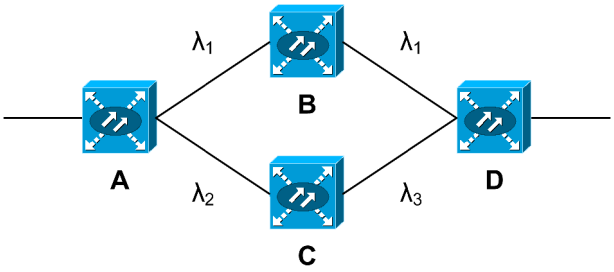

Figure 4.5: The wavelength continuity constraint. The only viable path between A and D is A-B-D.

4.2 Blocking switch architecture

The blocking switch architecture of Optical Add-Drop Multiplexers (OADMs) also imposes constraints on TE link advertisements. As described by Imajuku, et al. in a third recently released IETF Internet draft [18], this is because the OADM switch architecture results in a limited degree of connectivity (see figure 4.6). This limited connectivity occurs because OADMs connect to an optical network segment using only two ports. More specifically, west and east ports connect the OADM to the network. Using tributary ports internally connected to the west and east sides of the OADM, traffic can then be added onto or dropped off the network. In turn, this makes OADMs cost-effective and suitable for adding and dropping traffic to and from optical network segments. However, this limited degree of connectivity must be considered when enabling efficient installation of LSPs in this type of optical network segments.

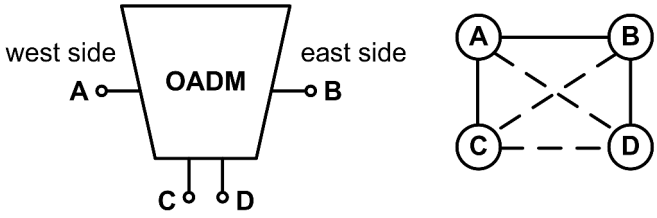

Figure 4.6: An OADM (left) and its network connectivity graph (right). C and D are west and east tributary ports respectively.

Addressing the limited degree of connectivity, a PCE must bypass the resulting blocking switch architecture. More specifically, some TE links advertised by an OADM will not be viable for use depending on specific sequences of adjacent TE links. For example, port selectivity for a network path entering the west side of an OADM is restrained not to consider the east side tributary ports. Nevertheless, as modern OADMs are becoming remotely reconfigurable (in software), the support for this type of networking component will become even more significant. Harnessing Reconfigurable OADMs (ROADMs) will, thus, be essential to allow service providers to build automated and cost-effective networks. However, in what way this should be treated has not yet been defined in the GMPLS standard documents. For simple network rings comprised of only ROADMs, wavelengths for traffic entering or leaving the network could be statically set. For more complex networks there is, on the other hand, a need to address these network elements. Doing so, selectable TE links could be announced within the TE link advertisements. Proposed by Imajuku, et al. in the above mentioned draft, it has been suggested that new link sub-TLVs will be defined for this purpose.

4.3 Impairments

Deployed optical equipment such as switches, amplifiers, multiplexers, and fibers might degrade optical signals due to impairments (see figure 4.7). If an optical network segment is carefully planned, such impairments should become minimal. However, preempting degraded performance in deployed equipment (or accounting for transparent network segments) such impairments could be considered path-type constraints. Accounting for impairments in general, however, is essential to guarantee that transmitted signals can be delivered with sufficient quality throughout an optical network segment. Several optical impairments are discussed in RFC 4054 [19].

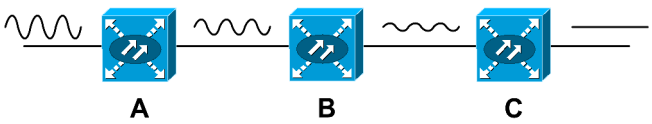

Figure 4.7: Attenuation resulting in signal loss at C.

Primarily, optical impairments can be classified into Optical Signal-to-Noise Ratios (OSNRs) and impulse widening (i.e. dispersion). Example impairments affecting OSNR are signal attenuation and Amplifier Spontaneous Emission (ASE) noise. Impulse widening can, in turn, be the result of Polarization Mode Dispersion (PMD) or chromatic dispersion. The above mentioned impairments are all linear and restricted to affecting only a single optical signal (see table 4.1). Non-linear impairments involve more than a single optical signal and are, thus, more difficult to predict. One such example is cross-talk which might introduce bit errors as optical signals in neighboring channels interfere with each other. This is most likely to occur in DWDM devices where many wavelengths compete for the same network resources. Because of the difficulties associated with handling such impairments, they will not be (explicitly) further considered.

Impairment	**Description**	**Effect**
Attenuation	As an optical signal goes through transparent network elements some of its energy, or power, is lost due to light absorption. Also known as power loss, the signal quality deteriorates.	Depending on the level of deterioration bit errors or signal loss at the end receiver might be introduced.
ASE noise	To prevent attenuation of optical signals, amplifiers are deployed to strengthen the signal. Amplifying signals, however, introduces random noise to the amplified signal.	This effects the OSNR and might introduce bit errors or signal loss at the end receiver.
Dispersion	Optical signals sent through fibers experience impulse widening. Specifically, chromatic dispersion is the result of light separation into several spectral components (i.e. colors). Similarly, PMD is the consequence of optical signals being randomly polarized in elliptic fibers. Nevertheless, common for both types of dispersion is that optical signals widen due to different propagation velocities.	Widened signals might interfere with each other and introduce bit errors.

Table 4.1: Linear optical impairments.

Considering relevant impairments as constraints, these must first be identified. Exposing GMPLS to *all* impairments could potentially create voluminous traffic in the control plane (depending on the implementation). On the contrary, some impairments might be valuable to disseminate. What is important, however, is to guarantee that computed network paths will be viable despite any optical impairments. Guaranteeing this, impairments from the links in a transparent network segment could be aggregated and evaluated as a path-type constraint. For example, the ASE noise on all links in a given network segment could be added together and compared to a minimum OSNR value for this impairment. Another method would be to use maximum link length as the *only* constraint. This way, a group of impairments are abstracted by assigning all TE links a “logical” maximum length. Then, a specific OSNR is guaranteed by simply limiting “logical” TE link lengths. Decreasing the number of constraints to consider, this can not account for impairments individually.

5 Implementation

This chapter presents a derived candidate solution (see section 1.1) based on the earlier literature study. Here, a virtual test-bed design, derived GMPLS and PCE extensions, and a software implementation of these are described.

5.1 Virtual test-bed design

In order to enable evaluation of the derived GMPLS and PCE extensions, a virtual test-bed has been jointly designed and implemented together with S. Reinhold [20]. In this test-bed, a number of virtual machines are hosted by a host computer. The virtual machines have been connected to the host computer via an internally emulated IP network (i.e. a ”host-only” network). The host computer configuration is specified in table 5.1.

Property	**Value**	**Notes**
Manufacturer and model	HP Workstation xw8400	-
Central Processing Unit (CPU)	Intel Xeon 5335 processor @ 2.66 GHz	Quad-core, 64-bit
Random Access Memory (RAM)	6 GB DDR2 (ECC) RAM @ 667 MHz	3.5 GB available (in 32-bit OS)
Hard Disk Drive (HDD)	1 TB 7200 RPM SATA-2 HDD	2 x 500 GB
Operating System (OS)	Ubuntu 7.04 (32-bit, Desktop Edition)	Linux kernel 2.6.20-16
Virtual machine software	VMware Server Console 1.0.3	Build-44356

Table 5.1: Host computer configuration.

Implementing an emulated network introduces some limitations. For example, the virtual interfaces employed by the emulated network need not exactly match the functionality or characteristics of corresponding physical interfaces. In addition, because *all* virtual machines must share hardware resources with the host computer (as they are running as host computer processes) software performance in the test-bed will be difficult to evaluate (and not likely match real case scenarios). The following sections further specify the virtual test-bed components and connectivity (see sections 5.1.1, 5.1.2, and 5.1.3).

5.1.1 Virtual machines

The virtual test-bed consists of eight VMware Server 1.0.3[1] virtual machines managed from a console on the host computer. These virtual machines have been connected using a ”host-only” network, assigning *all* virtual machines a virtual Ethernet interface connected to the host computer (see figure 5.1). For this purpose, the 192.168.112.0/24 network has been reserved within the host computer. It is via this network that the virtual machines will exchange information in a later deployed control plane.

1 See http://www.vmware.com/products/server for more information

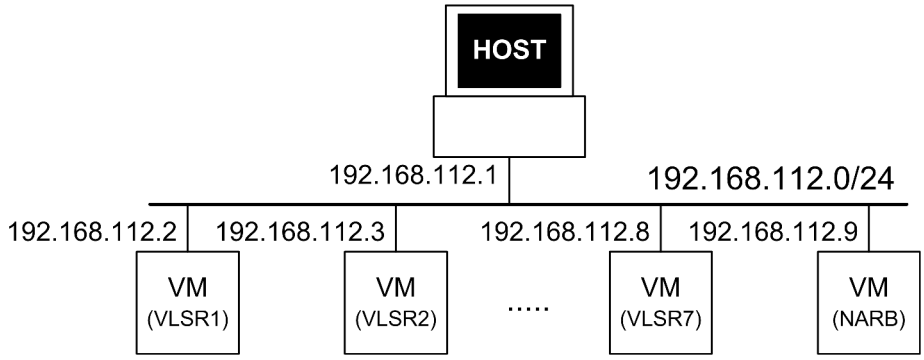

Figure 5.1: Virtual test-bed ”host-only” network.

Reserving hardware resources in the host computer, the virtual machines have each been given 20 GB HDD and 256 MB (logical nodes, VLSR1-VLSR7) or 512 MB (logical node, NARB) of RAM. This is consistent with the minimum hardware requirements needed to support both the operating system and the software components later deployed on a virtual machine (see section 5.3.1). Increasing performance, the virtual machines have been evenly distributed between the two host computer HDDs (expected to decrease HDD usage latencies).

In creating instances of the functional components, an operating system has then been deployed onto *all* virtual machines. After deploying an operating system onto a virtual machine, other software components can in turn be loaded into the operating system. By loading a specific software component (later described, see section 5.3.1) into the deployed operating system, a virtual machine finally becomes a specific network component. This way, as indicated by the names of the virtual machines, a virtual machine becomes either a Virtual LSR (VLSR, compare GLSR), or a Network Aware Resource Broker (NARB). In the latter case, here we only load a subset of the NARB functionality (i.e. that needed for path computation) into a NARB unit; namely, the stand-alone Resource Computation Engine (RCE, compare PCE). In order to avoid potential software conflicts, the same operating system has been deployed onto all virtual machines (i.e. Ubuntu 6.06, 32-bit, Desktop Edition, with Linux kernel version 2.6.15-16, proven compatible with the loadable software components).

5.1.2 Control plane configuration

Configuring a virtual control plane, the careful reader might have realized that deployed VLSRs *must* interface to multiple networks (to enable simulation of multiple control plane links). Recalling that the ”host-only” network is a *single* network, virtual control plane links must therefore be created. For this purpose, Generic Routing Encapsulation (GRE) tunnels [21] have been set up in the virtual test-bed. Using a GRE tunnel, a virtual machine is connected to another virtual machine via a logical point-to-point link. This way, a virtual topology consisting of point-to-point links has been placed on top of the ”host-only” network connecting the virtual machines (see figure 5.2). Automating the setup of this topology, start-up (bash) scripts have been installed (at the default runlevel) into the virtual machines. A summary of the control plane networks is given in table 5.2.

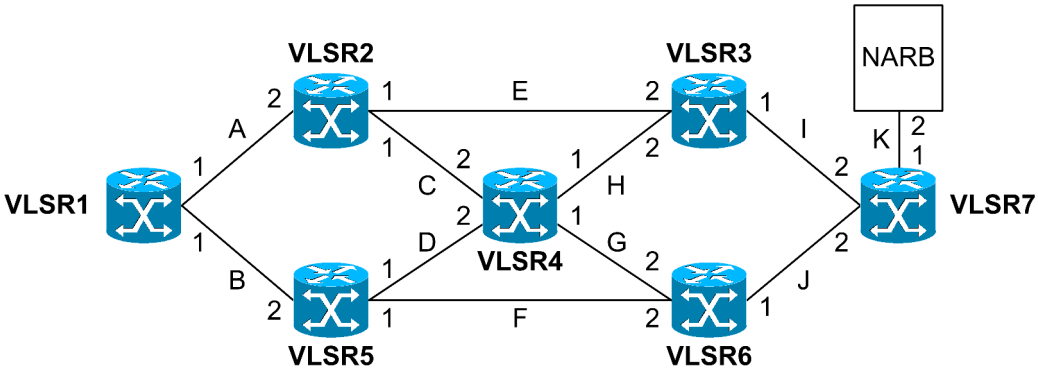

Figure 5.2: Configured control plane topology. Interface numbers represent network address suffixes.

Abbr.	Corresponding network	Description
A	192.168.0.0/24	Connecting VLSR1 and VLSR2
B	192.168.1.0/24	Connecting VLSR1 and VLSR5
C	192.168.2.0/24	Connecting VLSR2 and VLSR4
D	192.168.3.0/24	Connecting VLSR5 and VLSR4
E	192.168.4.0/24	Connecting VLSR2 and VLSR3
F	192.168.5.0/24	Connecting VLSR5 and VLSR6
G	192.168.6.0/24	Connecting VLSR4 and VLSR6
H	192.168.7.0/24	Connecting VLSR4 and VLSR3
I	192.168.8.0/24	Connecting VLSR3 and VLSR7
J	192.168.9.0/24	Connecting VLSR6 and VLSR7
K	192.168.10.0/24	Connecting VLSR7 and NARB

Table 5.2: A summary of the control plane networks.

5.1.3 Data plane configuration

Configuring a virtual data plane, data plane links have (as in the previous section, see section 5.1.2) been formed out of GRE tunnels. However, the data plane topology is *not* formed on top of the host computer ”host-only” network. Instead, GRE tunnels connect virtual interfaces that do not exist, constructing a distributable data plane topology. As a result, arbitrary data plane topologies can be advertised by the configured control plane without the need for an additional ”host-only” network. Using GRE tunnels, flexible configuration of the data plane topology is also made possible (however *existing* interfaces must be connected to enable data transport). Given the configured data plane topology (see figure 5.3), it is envisioned that VLSR2 and VLSR6 will represent ROADMs, adding or dropping traffic to or from the network consisting of VLSR3, VLSR4, and VLSR5 (providing a test-case for a blocking switch network architecture). Further, all data plane links have been given aliases (shown between parentheses in figure 5.3). A summary of the data plane networks is given in table 5.3.

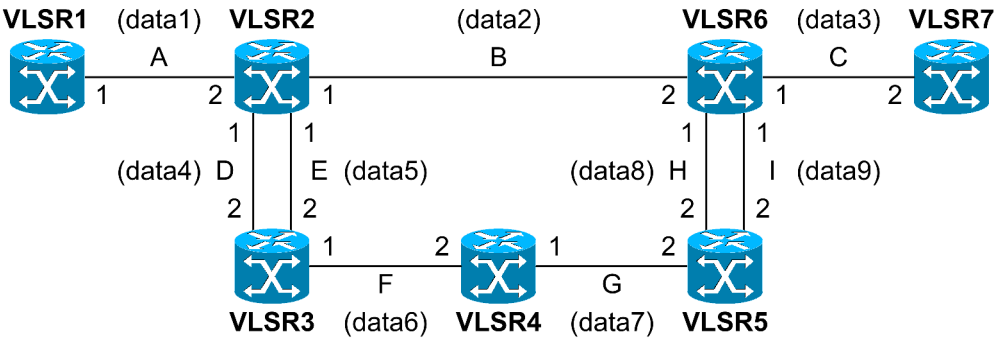

Figure 5.3: Configured data plane topology. Interface numbers represent network address suffixes.

Abbr.	Corresponding network	Description
A	10.0.0.0/24	Connecting VLSR1 and VLSR2
B	10.0.1.0/24	Connecting VLSR2 and VLSR6
C	10.0.2.0/24	Connecting VLSR6 and VLSR7
D	10.0.3.0/24	Connecting VLSR2 and VLSR3
E	10.0.4.0/24	Connecting VLSR2 and VLSR3
F	10.0.5.0/24	Connecting VLSR3 and VLSR4
G	10.0.6.0/24	Connecting VLSR4 and VLSR5
H	10.0.7.0/24	Connecting VLSR5 and VLSR6
I	10.0.8.0/24	Connecting VLSR5 and VLSR6

Table 5.3: A summary of the data plane networks.

5.2 GMPLS and PCE extensions

This section presents selected extensions to GMPLS and a PCE as motivated by the material presented in the earlier chapters. For this, three new link sub-TLVs are defined and a CSPF algorithm capable of dealing with these sub-TLVs introduced.

5.2.1 Wavelength availability

First of all, a new link sub-TLV for wavelength availability is proposed (see figure 5.4). In this sub-TLV, the first body field expresses the base wavelength or frequency in a grid of wavelengths. The base wavelength or frequency is, here, expressed in the label format presented earlier (see section 4.1.3). Hence, this first field can hold either a wavelength (in the case of a CWDM grid), or a frequency (in the case of a DWDM grid). The second field expresses bandwidth per wavelength in bytes per second (in floating point number representation, see section 4.1.1). Then, the third field expresses a variable length bitmask (zero-padded so that the defined sub-TLV will always contain an even set of 4-octet words). For experimenting with this link sub-TLV, a type value of 32768 has been used.

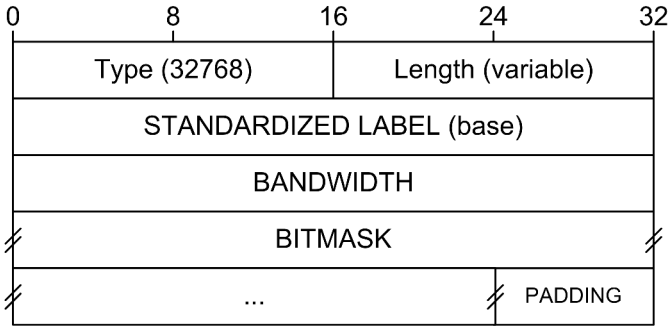

Figure 5.4: The link sub-TLV defined for wavelength availability.

Utilizing this link sub-TLV, a grid of wavelengths can be unambiguously advertised in the GMPLS routing process. More specifically, the base wavelength (or frequency) specifies the first entry in a grid. If the base wavelength is unreserved (i.e. available for use), the first bit in the bitmask is set to one. If the same wavelength is reserved (i.e. not available for use) the first bit in the bitmask is instead set to zero. Uncompressing the grid of wavelengths expressed in this sub-TLV, the consecutive bits in the bitmask then indicate the availability of subsequent wavelengths (given the grid channel spacing).

Because the label format defined for the experimental sub-TLV specifies the channel spacing of an expressed grid of wavelengths, all wavelengths in the grid are deterministically defined. However, if this information is not available in the utilized label format, then wavelengths must be parsed in some other way (e.g. by treating the channel spacing as being uniform for all links in the network).

Hence, disseminating this sub-TLV, a compressed grid of wavelengths can be distributed in the GMPLS control plane. Consequently, control plane traffic is reduced at the cost of additional computation overhead (i.e. the compressed grid of wavelengths needs to be uncompressed before it can be interpreted). This way, the defined link sub-TLV offers additional functionality for service providers requiring wavelength availability to be dynamically updated. Nevertheless, this sub-TLV will (as it is envisioned) be optional in a GMPLS control plane implementation.

5.2.2 Interface selectivity

Another new link sub-TLV is proposed for interface selectivity (see figure 5.5). The body of this sub-TLV expresses a variable length list of unselectable interfaces. To experiment with this link sub-TLV, a type value of 32769 has been used to indicate that IPv4 addresses are carried and a type value of 32770 to instead indicate 32-bit interface identifiers.

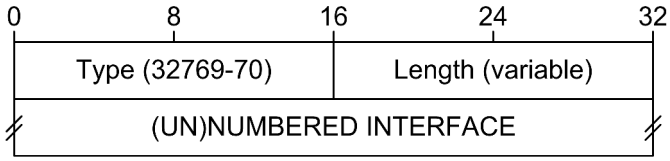

Figure 5.5: The link sub-TLV defined for interface selectivity.

Implementing this link sub-TLV, unselectable interfaces can be declared for data plane links advertised in the GMPLS control plane. For experimenting with this link sub-TLV, unselectable interfaces have been specified to be the interfaces to which a declaring interface (i.e. data plane link) is not locally connected. This way, the interface connectivity of a network element with limited internal connectivity could be specified and distributed.

Making use of this link sub-TLV, the blocking switch network architecture of OADMs and ROADMs (see section 4.2) can be reflected in the GMPLS control plane. Distributing the limited internal connectivity for these types of devices, a path computation process could detect illegal link sequences and discard candidate paths violating the interface selectivity constraint. Bypassing the limited interface selectivity possibly existing in an optical network segment, this sub-TLV provides service providers with a means to make use of a blocking switch architecture. This link sub-TLV is thought to be optional, but useful for dealing with blocking switch network architectures.

5.2.3 User-defined constraints

In order to enable user-defined constraints to be associated with links in a provisioned GMPLS network, a third new sub-TLV (see figure 5.6) is proposed. This sub-TLV has been defined to distribute code-value pairs representing arbitrary constraints. The body of this sub-TLV holds a variable number of 4-octet words (in each 4-octet word, the first octet specifies a value code followed by the value associated with that code). For experimenting with this link sub-TLV, a type value of 32771 has been used.

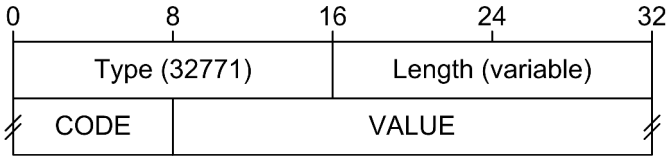

Figure 5.6: The link sub-TLV defined for user-defined constraints.

For experimental purposes, one value code (0) has been reserved. This reserved code value has been used to indicate the wavelength conversion capability of an advertising interface (meaning that conversion capability has been indicated for outgoing links). Applying user-defined constraints on links in a provisioned network domain, the remaining set of code-value pairs are available for other uses. Potential use would be information describing optical impairments and constraints (see section 4.3), however the link sub-TLV is generic enough to also express other types of constraints (e.g. end-to-end delay or a set of ordered metrics). Envisioned to be optional, this sub-TLV is useful to provide service providers with a mechanism for distributing arbitrary (that is, user-defined) link constraints in a GMPLS control plane.

5.2.4 Candidate CSPF algorithm

In order to harness the information that could be distributed in the link sub-TLVs earlier defined (see sections 5.2.1, 5.2.2, and 5.2.3), a candidate CSPF algorithm (see section 3.2) has also been implemented. This candidate CSPF algorithm searches for *all loop-less* paths in an optical network segment, at the same time evaluating wavelength availability, interface selectivity, and user-defined constraints. Growing candidate paths, this algorithm extends the abstract operation of the Breadth First Search (BFS) algorithm [1]. Mimicking this well-known SPF algorithm (see figure 5.7), the candidate CSPF algorithm is expected to find *all* paths. In addition, BFS algorithms usually have excellent asymptotic run-time (when operating on tree structures, see section 3.2.2).

```
add start node s to FIFO queue q
  while q is not empty
    pop first element f from q
    for all neighbours n of f
      if n not visited
        update path to n
        mark n as visited
        insert n into q
      end if
    end for
  end while
done
```

Figure 5.7: The abstract operation of the BFS algorithm.

Initially, the candidate algorithm creates an initial path containing only the source node (it is from this path that all other paths will be grown). When this initial path has been created, it is added to a First-In-First-Out (FIFO) queue. Adding *paths* to this queue deviates from the conventional operation of the BFS algorithm (however, this is necessary to continuously keep track of the grown paths). Proceeding, the algorithm starts to operate on the candidate path queue by iteratively removing and examining paths from the queue, until the queue becomes empty. Within each such iteration, the end node of an examined path is first extracted. The neighbors of this extracted node, not forming a loop with the examined path, are then considered. By not considering neighbors that form loops, the candidate algorithm is able to safely operate on graph structures (i.e. tree structures with loops, note that this is necessary to avoid eliminating *any* potential paths). Examining a neighbor, the currently examined path is then copied and expanded with this neighbor node. After this, the expanded path is checked for viability by running a suitable path-evaluation function. If this path-evaluation function returns false, the expanded path will be discarded. Otherwise, when the evaluation function returns true, the expanded path will be either further examined by future iterations (i.e. added to the FIFO queue), or added to a set of discovered paths (when an examined neighbor is the destination node). When all neighbors have been examined, a new path is extracted from the queue (and a new iteration is started). Because expanded paths are always checked for viability, this candidate algorithm will only accept viable paths. The abstract operation of the candidate algorithm is presented in the figure below (see figure 5.8).

```
add start path sp to FIFO queue q
  while q is not empty
    pop first element fp from q
    extract end node e from fp
    for all neighbours n of e
      if n does not form loop with fp
        add n to new path np := fp + n
        if np is a viable path
          if n is the destination node d
            add np to found paths
          else
            insert np into q
          end if
        else
          discard np  // not a viable path, hence discard
        end if
      end if
    end for
  end while
done
```

Figure 5.8: The abstract operation of the candidate CSPF algorithm.

When candidate paths are checked for viablity, a path-evaluation function is to be called. Depending on the properties of this function, different meanings may apply to the resulting path-evaluation. The path-evaluation function used together with the candidate CSPF algorithm has therefore, here, been carefully defined. This path-evaluation function should return false if (1) any accumulated user-defined constraint exceeds a requested maximum value, (2) there is no wavelength left for use on a path (i.e. it is not possible to preserve wavelength continuity), and (3) an added link violates the semantics of interface selectivity (i.e. is not possible due to a blocking switch network architecture). If none of the above conditions occurs, then the path-evaluation function is to return true.

Closing this section, with regard to the candidate algorithm, paths can be easily grown in orders other than breadth first (that is, not using a BFS algorithm). Changing the type of queue (and in some cases adding *minor* modifications), candidate paths can be differently grown. This would have semantic meaning by causing the candidate algorithm to compute only a limited number of paths (that is, not *all* paths). For example, replacing the FIFO queue by a Last-In-First-Out (LIFO) queue (and not always popping examined paths from this LIFO queue) would result in growing paths depth first. In the same way, a minimum priority queue would result in growing paths according to the priority of this queue (this corresponds to the operation of the Dijkstra algorithm).

5.3 Software implementation

This section presents the implemented software extensions. Here, the modification done to an open source software suite to abstract the above defined functions (see section 5.2) are detailed (here the focus is on the extended functionality provided).

5.3.1 Open source software suite

In order to implement the proposed functional extensions, the DRAGON project[1] open source software suite has been used as a base. Built on software components from other open source software projects, this project seeks to enable service provisioning in multiple layers and domains via the GMPLS framework. The other built-in projects are (1) the KOM-RSVP engine[2] (extended to support RSVP-TE) for GMPLS signaling, and (2) the Zebra OSPF daemon[3] for GMPLS routing (extended to support OSPF-TE). However, in order to abstract a suitable PCE, the DRAGON project has developed a new software component. This component currently consists of two separate parts, the NARB and RCE (see section 5.1.1). Of these two components, the RCE is the one responsible for collecting information about a network domain and computing network paths in it. Thus, as we are only interested in implementing the path computation extensions, only the RCE will be extended.

5.3.2 Implemented Zebra extensions

Extending the GMPLS routing process, the Zebra OSPF daemon has been extended with the experimental link sub-TLVs and additional Virtual TeletYpe (VTY) commands. VTY commands enable adding, removing, or changing information in the link state database maintained by a Zebra OSPF daemon. Thus, to enable direct modification of this database, syntax and semantics for a (1) *wavegrid* command (managing wavelength availability on links), (2) *unselectable* command (managing unselectable interfaces for links), and (3) *constraint* command (managing user-defined constraints on links) have been specified and implemented. With these VTY commands, the previously described link sub-TLVs can be added to or removed from the set of sub-TLVs distributed by the Zebra OSPF daemon.

The implemented VTY command scope has been set to apply to the *OSPF_TE_IF_NODE* command node. This means that the VTY commands are issuable in the context of the Zebra OSPF daemon configuration files. Consequently, the experimental link sub-TLVs can only be added to the GMPLS control plane when the OSPF daemon is started (i.e. when the configuration files are parsed). By extending the VTY command scope, these commands could be issuable from other command locations (e.g. the Zebra OSPF daemon TELNET interface) by associating them with other command nodes. This part of the software extensions was jointly developed together with S. Reinhold [20]. A complete specification of the added OSPF-TE VTY commands is given in the appendixes (see Appendix B).

1 See http://dragon.maxgigapop.net or http://dragon.east.isi.edu for more information
2 See http://www.kom.tu-darmstadt.de/en/downloads/software/kom-rsvp-engine for more information
3 See http://www.zebra.org for more information

5.3.3 Implemented RCE extensions

Implementing the path computation extensions, the RCE component has been extended with the experimental sub-TLVs and the candidate CSPF algorithm. More specifically, the added extensions enable the experimental link sub-TLVs to be stored in the RCE TED and defines a new path computation event. Additionally, the RCE has also been extended with a new format for returning computed paths in the form of an ERO signaling object including label ERO sub-objects (containing wavelength labels).

The new path computation event abstracts the candidate CSPF algorithm and defines additional processing needed to fully support it. When this path computation event is requested from the RCE, six conceptual steps are sequentially executed. The first two steps are no different from those of the algorithms already implemented in the RCE. This includes verifying the client request and building a logical topology from the TED. Once a logical topology has been built, a third step (unique to the implemented path computation event) prunes the built topology. When pruning this topology, all links not meeting the requested (1) switching type, (2) encoding type, and (3) minimum requested bandwidth are removed. Note that handling link-type constraints by pre-processing the constructed graph is, at this stage, required because the path-evaluation function will later only consider path-type constraints. The fourth step is to run the candidate algorithm on the now pruned topology. While running the algorithm (i.e. computing paths), all information needed to extract paths with wavelength precision is accumulated and a path vector (containing discovered paths between the source and destination nodes) is continuously grown. Once this path computation finishes, a fifth step post-processes the returned result by sorting the grown path vector (if not empty) by user preference. In keeping with the abstract operation of the BFS algorithm, the default user preference (when no user preference has been specified) is to sort paths by size (i.e. number of links in a path). Finally, a last step (i.e. the sixth step) extracts a single path from the sorted path vector. For this path, an ERO including upstream and downstream label ERO sub-objects (containing wavelength labels) for each interface is built and returned (to a requesting client).

In order to simplify the implementation of this algorithm, some limitations have been accepted. To begin, only user-defined constraints that are positive additive (i.e. summable constraints which can only take on positive values) are supported. Adding support for negative additive constraints (i.e. constraints that can be compensated for) could be added by defining a set of constraints which are only evaluated when a complete path has been discovered. In addition, the implementation of the candidate algorithm presumes that all links stored in the TED are bidirectional, and that a wavelength can be used in two directions. In the future, reverse links could also be looked up and examined. A complete specification of the implemented Zebra and RCE software extensions is given in the appendixes (see Appendix C).

6 Verification and analysis

This chapter provides a functional verification of the virtual test-bed and the software implementation. Here, a performace indication of the implemented software extensions is also given.

6.1 Test-bed verification

To verify the functionality of the virtual test-bed, the virtual machines, their network configuration (i.e. ”host-only” network), and their software configuration (i.e. operating system and compilers) have been inspected. Here, the inspected virtual test-bed is the same as that described in section 5.1.1. Verifying the virtual machines, their installation and hardware properties have first been checked (using the host computer console). Secondly, the ”host-only” network connectivity has been checked (using a ping utility). Further, the compilation and executability of software in the virtual test-bed have been checked (by loading unmodified software onto the virtual machines and executing it). This has led to the following observations:

1. All virtual machines are *correctly* configured and installed. Here, *correctly* refers to the expected set of emulated hardware and operation of the virtual machines.

2. All virtual machines are connected to each other in the defined ”host-only” network, and the GRE tunnels appropriately connect the virtual machines.

3. The software suite can be installed and executed on all virtual machines. When executing the software, the expected behavior occurs.

Given these observations, the functional testing of the virtual test-bed implementation is considered complete. Further verification has not been performed.

6.2 Software implementation verification

To verify the software implementation functionality, the extended software components have been deployed onto the virtual machines. Enabling these software components to be functionally verified, the data plane topology (see section 5.1.3) has then been associated with more information. Associating the data plane links with additional constraints (and attributes, see table 6.1), different test-cases have then been defined and executed. For this, the aim has been to ensure that no relevant test-case would be excluded (and all relevant test-cases included). In the same way as suggested in section 5.1.3, VLSR2 and VLSR6 here mimic data plane ROADMs. Additionally, the wavelengths associated with all data plane links appertain to a DWDM grid with a 12.5 GHz channel spacing and 40 Gbit/s bandwidth per wavelength. Note here that a user-defined constraint code of 0 declares interface conversion capability (whereas a value of 1 indicates full conversion capability, see section 5.2.3).

data plane link	available wavelengths	unselectable interfaces	user-defined constraints
data1 (VLSR1)	190000 - 190112.5 GHz	-	<0,0>, <1,1>, <2,5>
data1 (VLSR2)	190000 - 190112.5 GHz	10.0.4.1	<0,0>, <1,1>, <2,5>
data2 (VLSR2)	190000 - 190112.5 GHz	10.0.3.1	<0,0>, <1,10>, <2,5>
data2 (VLSR6)	190000 - 190112.5 GHz	10.0.8.1	<0,0>, <1,10>, <2,5>
data3 (VLSR6)	190000 - 190112.5 GHz	10.0.7.1	<0,0>, <1,1>, <2,5>
data3 (VLSR7)	190000 - 190112.5 GHz	-	<0,0>, <1,1>, <2,5>
data4 (VLSR2)	190000 - 190112.5 GHz	10.0.4.1, 10.0.1.1	<0,0>, <1,1>, <2,5>
data4 (VLSR3)	190000 - 190112.5 GHz	-	<0,0>, <1,1>, <2,5>
data5 (VLSR2)	190000 - 190112.5 GHz	10.0.0.2, 10.0.3.1	<0,0>, <1,1>, <2,5>
data5 (VLSR3)	190000 - 190112.5 GHz	-	<0,0>, <1,1>, <2,5>
data6 (VLSR3)	190000 - 190112.5 GHz	-	<0,0>, <1,1>, <2,5>
data6 (VLSR4)	190000 - 190112.5 GHz	-	<0,1>, <1,1>, <2,5>
data7 (VLSR4)	190050 - 190162.5 GHz	-	<0,1>, <1,1>, <2,5>
data7 (VLSR5)	190050 - 190162.5 GHz	-	<0,0>, <1,1>, <2,5>
data8 (VLSR5)	190000 - 190112.5 GHz	-	<0,0>, <1,1>, <2,5>
data8 (VLSR6)	190000 - 190112.5 GHz	10.0.2.1, 10.0.8.1	<0,0>, <1,1>, <2,5>
data9 (VLSR5)	190000 - 190112.5 GHz	-	<0,0>, <1,1>, <2,5>
data9 (VLSR6)	190000 - 190112.5 GHz	10.0.1.2, 10.0.7.1	<0,0>, <1,1>, <2,5>

Table 6.1: Constraints associated with the data plane links. Symbols in parentheses indicate the link ends advertised in the control plane.

When inspecting the table above, we see that (1) VLSR2 and VLSR6 have been configured with unselectable interfaces, (2) the outgoing interfaces on VLSR4 are conversion capable, and (3) three user-defined constraints have been placed on all data plane links (in addition, we see that *data7* has been offset by four wavelengths). To then verify the GMPLS routing extensions, the link state database of a Zebra OSPF daemon was examined using a TELNET client; leading to the following observation: Connecting to a Zebra OSPF daemon in the network domain and instructing it to show its link state database shows the *correct* data plane topology (here, *correct* refers to the *entire* topology as defined by all Zebra OSPF daemon configuration files).

Once the GMPLS routing process was verified, the RCE was in turn verified using a software testing utility (included in the DRAGON software suite). This testing utility is essentially an RCE software client (compare PCC) and was, before verification, extended to support client requests including algorithm selection, path preference, and user-defined constraints. For reference, the only two paths that exist (in the data plane) between VLSR1 and VSLR7 have been defined $path_1$: VLSR1-VLSR2-VLSR6-VLSR7, and $path_2$: VLSR1-VLSR2-VLSR3-VLSR4-VLSR5-VLSR6-VLSR7. By issuing specific client requests with this testing utility, the following observations were then made:

1. Issuing a client request to the RCE defining requested switching capability as non-LSC, encoding as non-lambda, or bandwidth as more than 40 Gbit/s results in no path being returned. This is correct because of the incorrect traffic specifications.

2. Issuing a client request to the RCE defining requested switching capability as LSC, encoding as lambda, and bandwidth as 1 kB/s results in $path_1$ being returned. This is correct because $path_1$ is the shortest path (with respect to the number of hops).

3. Issuing the client request defined in 2., while also setting path preference to be the user-defined constraint with code 1, results in $path_2$ being returned. This is correct because $path_2$ now is the shortest path (with respect to this user-defined constraint).

4. Issuing the client request defined in 3., while also requiring the user-defined constraint with code 2 to be less than 20, results in that $path_1$ is returned. This is correct because $path_2$ violates the specified constraint (i.e. with a value of 30).

5. Issuing the client request defined in 4., while also requiring the user-defined constraint with code 1 to be less than 10, results in no path being returned. This is correct because $path_1$ violates the specified constraint (i.e. with a value of 12).

6. For *all* returned paths, wavelength continuity, interface selectivity, or user-defined constraints are never violated and wavelength labels always correctly extracted.

Given these observations, the functional verification of the software implementation is considered complete. The performance evaluation in section 6.3 somewhat contributes to verification, but no further verification has been performed.

6.3 Software implementation performance

In this section, a performance evaluation of the software implementation is given. Here, the same constraints and attributes as in section 6.2 have been associated with the data plane links.

6.3.1 Theoretical network overhead

In theory, dissemination of additional link sub-TLVs in the GMPLS control plane results in that more information about networking resources is made available. However, this comes at the cost of an increased network overhead. Typically, this overhead (N_o) will equal the sum of all bytes added to LSAs (N_b) times the number of links on which an LSA will be distributed (N_d). N_b will in turn equal the sum of all appended link sub-TLVs specific to the candidate solution (see section 5.2.1, 5.2.2, and 5.2.3). Furthermore, given that an LSA is distributed on all links in an OSPF area, N_d will approximate the number of control plane links. This theoretical network overhead is given table 6.2. In this table, a few theoretical distribution times given different link bandwidths are also presented (note that the actual data exchange between VLSRs has not been monitored).

N_b	N_d	$N_o = N_b \times N_d$	link bandwidth	distribution time
18 x 32 bytes + 4 x 20 bytes = 576 bytes + 80 bytes = 656 bytes	11	656 x 11 bytes = 7216 bytes = 7.216 kbytes = 57.728 kbits	10 Mbit/s	5.7728 ms
			100 Mbit/s	577.28 µs
			1000 Mbit/s	57.728 µs

Table 6.2: Theoretical network overhead. Distribution time excludes LSA processing.

Of course, including (or excluding) the newly defined link sub-TLVs from a GMPLS routing process will involve a trade-off between service efficiency and reliability. By increasing the number of links and attributes (or bytes, N_b) distributed in a GMPLS control plane, the number of links these are distributed on (N_d) need not be affected. This is because (in GMPLS) the data plane topology can be kept separate from the control plane topology. Keeping N_d constant over time, additional data plane links can be added without necessarily resulting in a non-linear growth of overhead bytes. Thus, a resulting network overhead (N_o) could actually grow linearly instead of quadratically when additional links or attributes are added to a data plane configuration.

6.3.2 Time efficiency

To evaluate time efficiency, the path computation process implemented by a RCE has been divided into separate time intervals. These time intervals will serve as reference points, indicating where a RCE process spends time. The time intervals are specified below:

T1: time spent verifying a client path computation request
T2: time spent building a logical topology from the TED
T3: time spent pruning the built logical topology
T4: time spent searching and sorting found paths
T5: time spent extracting labels and returning an ERO
T6: total time spent (T1+T2+T3+T4+T5).

Enabling time spent in these time intervals to be observed, the RCE software component has been extended to read the time at specific time instants. Here, time is read by querying the *CLOCK_PROCESS_CPUTIME_ID* clock (a high resolution per-process timer maintained by the host computer CPU) each time a specified time interval is either entered or exited. Once this clock has been queried twice, the time difference between entering and exiting any specific time interval is computed. This way, a set of time population samples have been produced, after which time efficiency has been evaluated based on these population samples. Here, the time population samples have been generated by sequentially issuing 1000 requests (for a path between VLSR1 and VLSR7, each request requiring four user-defined constraints to be evaluated) using the testing utility described in section 6.2.

Due to the volatile nature of the virtual machines (running as processes on the host computer), some values in the time population samples have been removed. More specifically, all values deviating from a time population sample median with more than 50% have been discarded. These population sample adjustments are necessary to eliminate incorrect observations, rendering only meaningful time values to be analyzed. In addition, all population samples have been proved normally distributed by dividing the observed time values (in a specific sample) into intervals and then count the value frequency in these intervals. Given that we now have a set of normally distributed population samples with observed means and standard deviations, 95% confidence intervals for these population samples have then been calculated using the t-method (see table 6.3 and figure 6.1).

T1 [µs]	T2 [µs]	T3 [µs]	T4 [µs]	T5 [µs]	T6 [µs]
4.87±1.26	37.16±5.92	2.35±0.59	98.72±11.02	3.28±0.6	146.38±16.73

Table 6.3: Observed processing time in each time interval. Observed time is given as 95% confidence intervals for the population samples.

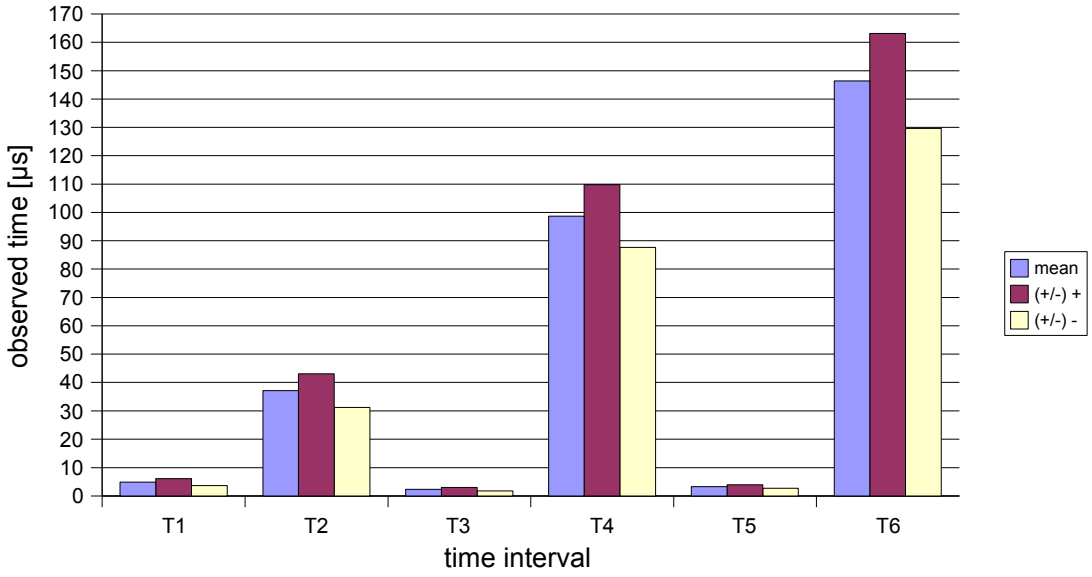

Figure 6.1: Observed processing time in specified time intervals. This corresponds to the confidence intervals given in the table above (see table 6.3).

In figure 6.1, we see that the RCE implementation spends most of the time building a logical topology, searching for, and sorting network paths. Interesting to see is that the time spent searching for and sorting *all loop-less* network paths (T4) is only slightly more than double the time spent building a logical topology (T2). Noting the total observed time spent by a path computation event (T6), the RCE also seems quite responsive (given the operational circumstances). Handling an arrived client request, we see that less than a few hundred microseconds is spent on a single path computation event. Nevertheless, observing performance in the virtual test-bed generated results that were hard to interpret. Therefore, it would be useful to compare the observed time relative to other observed time (e.g. comparing the time efficiency of several different candidate solutions).

6.3.3 Space efficiency

To evaluate space efficiency, the candidate path queue has first been analyzed. This has been done by prompting a RCE to show the queue contents at specific points in time. For this, three different queue elements have been defined: (1) physical paths (equal to fiber paths), (2) physical links (equal to fiber links), and (3) implicit wavelength paths (equal to lambda paths). Here, physical links correspond to the links in a data plane configuration. In turn, a physical path is a sequence of physical links. Finally, a lambda path is an implicit wavelength path that can be extracted, with wavelength precision, from a physical path.

Extracting the candidate path queue contents in each *outer* search iteration (when a client request was processed), the following results were obtained (see table 6.4 and figure 6.2). Note that an *outer* search iteration corresponds to examining a single candidate path.

queue elements	**search iteration**												
	0	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**
fiber paths	0	1	1	2	2	2	2	2	3	3	2	1	0
fiber links	0	0	1	4	5	6	7	8	14	15	10	5	0
lambda paths	0	0	10	20	20	20	110	106	126	180	120	60	0

Table 6.4: Candidate path queue elements. Elements are given for each outer search iteration performed by the candidate CSPF algorithm.

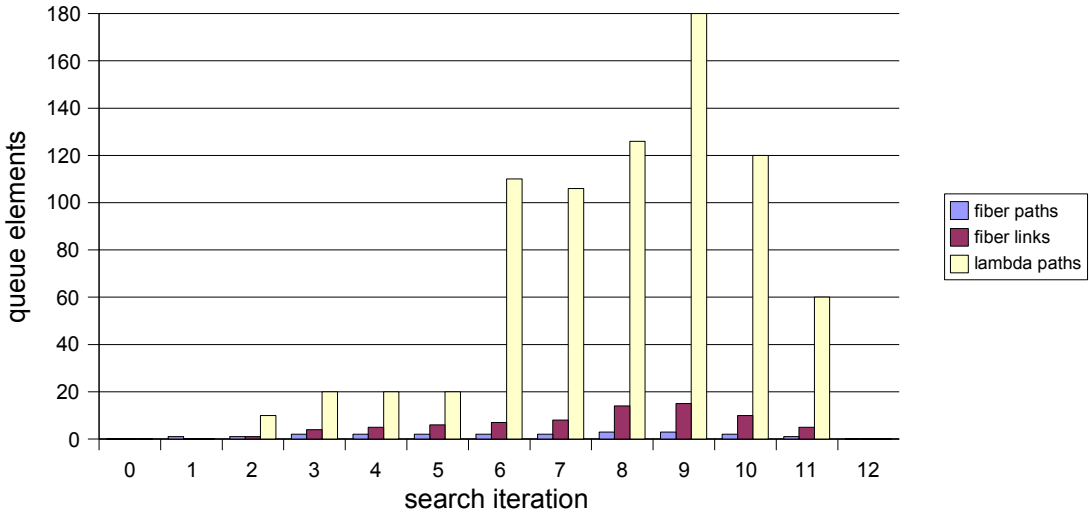

Figure 6.2: Candidate path queue elements. These correspond to those given in the table above (see table 6.4).

In figure 6.2, we see that the number of fiber paths in the candidate path queue remains low throughout the entire search. This is very likely because the candidate solution implementation only explicitly maintains a set of fiber paths, from which other information is extracted when needed. Given the number of implicitly held lambda paths, both time and space is thus saved by not abstracting candidate paths with this resolution. In addition, when analyzing how the number of explicit and implicit queue elements propagate, the candidate algorithm search pattern could be derived. For example, the instant increase in lambda paths in search iteration 6 and 9 indicates that a wavelength conversion capable VLSR has previously been processed (generating a lot of potential lambda paths). In turn, drops in the number of fiber paths in the candidate queue indicates that a path has been (in a previous search iteration) either discarded or discovered.

In addition to this, the RCE stack and heap usage have also been profiled. This has been accomplished by using the Massif heap profiler tool built-in to the Valgrind software suite[1]. This profile was generated by loading the RCE program into Massif and having it process 1000 requests (for a path between VLSR1 and VLSR7, using the testing utility described in section 6.1). Then, once the RCE finished processing all issued client requests, Massif was prompted to terminate. Upon terminating, Massif in turn generated a textual summary of the location and amount of consumed spacetime together with the complementary graph given in the appendixes (see Appendix D). Note (when analyzing this complementary graph) that profiling with Massif runs the RCE at about one twentieth the speed of normal execution.

The stack and heap profile generated by Massif has resulted in some observations. First of all, the amount of memory allocated by the stack is nearly constant over time (equal to about 5% of the totally consumed spacetime). This is expected as the RCE iteratively examines candidate paths with no recursive function calls. Secondly, very little spacetime is consumed during an entire path computation event. In fact, the amount of spacetime consumed for maintaining the candidate path queue is only about 1% of the total amount of consumed spacetime. In addition, this amount never seems to exceed 5 kB.

Ignoring the unrelated functions (especially the message API functions) that leak memory, a very small percentage of process memory is allocated when searching for *all loop-less* paths. Partially, this is because the RCE process runs in a single thread (i.e. it is not threaded, resulting in sequential handling of the issued requests). Moreover, by immediately discarding candidate paths that violate constraints, the size of the candidate path queue is kept small. Because of this, more complex data plane topologies are not expected to result in exponential increases in consumed spacetime. Additionally, the RCE implementation seems to utilize memory very efficiently (given that many potential paths are intermediately computed).

1 See http://www.valgrind.org for more information

7 Conclusion and future work

In conclusion, we have seen that handling additional routing constraints in GMPLS is feasible. By distributing optical constraints (and attributes) in a generic format via OSPF-TE, efficient traffic engineering in optical network segments is possible. By also extending CSPF algorithms with support for optical network parameters it is possible to deal with wavelength continuity, limited connectivity, and optical impairments. Additionally, this means that a PCE can be deployed in more complex network regions (that is, optical network regions). Returning paths computed in such network segments in the form of RSVP-TE ERO signaling objects, network paths are ready to be immediately signaled (requiring no additional processing) when returned.

Implementing this functionality, very little processing overhead was observed. Although the software implementation was run in a volatile environment (i.e. on virtual machines), the implemented RCE (compare PCE) was quite responsive to client requests. Enhancing this RCE to be stateful, instantaneous client access of pre-computed paths will also be possible. Based upon the theoretical network overhead in bytes, the flooding effect introduced by OSPF-TE could, in some cases, become an issue. On the contrary, as the performance of networking components will most likely continue to improve, this effect will decrease. In the future, the need for efficient and reliable end-to-end service provisioning in multi-layered networks will also very likely increase. Embracing new technology, to provide service provisioning with GMPLS appears to be an efficient way of accomplishing this.

7.1 Future work

Some logical follow-up work related to the work seen in this thesis include (1) further analysis of the extended GMPLS routing process, (2) a comparison of CSPF algorithms in a PCE, and (3) a physical network implementation of the candidate solution. Analyzing the extended GMPLS routing process, placement and amount of network information could be evaluated. In a comparison of CSPF algorithms, other algorithms performing the same amount of work, or less, could be implemented and analyzed. Implementing the candidate solution in a physical network, the GMPLS routing process must be interfaced to physical networking components. In addition, the GMPLS *signaling* process could be extended to support the proposed standardization of the wavelength label format (see section 4.1.3).

Even further, the introduced simplifications could be addressed as future work. This would include examining reverse links (for bidirectionality), extending the VTY command scopes (to that of other command nodes), and adding support for negative additive constraints (the constraints that can be compensated for). In addition, another element of future work will be to request Internet Assigned Numbers Authority (IANA) assignment of sub-TLV type numbers and specific field values (that is, officially publish these sub-TLVs).

References

[1] Adrian Farrel and Igor Bryskin, *GMPLS: Architecture and Applications*, Morgan Kaufmann (Elsevier), San Franscisco, 2005, ISBN: 0-12-088422-4.

[2] D. Awduche, J. Malcolm, J. Agogbua, M. O'Dell, and J. McManus, *Requirements for Traffic Engineering over MPLS*, IETF Request for Comments 2702, September 1999. [www]: <http://www.ietf.org/rfc/rfc2702.txt>. Last access on 070830.

[3] E. Rosen, A. Viswanathan, and R. Callon, *Multiprotocol Label Switching Architecture*, IETF Request for Comments 3031, January 2001. [www]: <http://www.ietf.org/rfc/rfc3031.txt>. Last access on 070830.

[4] E. Mannie, *Generalized Multi-Protocol Label Switching (GMPLS) Architecture*, IETF Request for Comments 3945, October 2004. [www]: <http://www.ietf.org/rfc/rfc3945.txt>. Last access on 070830.

[5] J. Moy, *OSPF Version 2*, IETF Request for Comments 2328, April 1998. [www]: <http://www.ietf.org/rfc/rfc2328.txt>. Last access on 070830.

[6] D. Katz, K. Kompella, and D. Yeung, *Traffic Engineering (TE) Extensions to OSPF Version 2*, IETF Request for Comments 3630, September 2003. [www]: <http://www.ietf.org/rfc/rfc3630.txt>. Last access on 070830.

[7] R. Braden, L. Zhang, S. Berson, S. Herzog, and S. Jamin, *Resource ReSerVation Protocol (RSVP) - Version 1 Functional Specification*, IETF Request for Comments 2205, September 1997. [www]: <http://www.ietf.org/rfc/rfc2205.txt>. Last access on 070830.

[8] D. Awduche, L. Berger, D. Gan, T. Li, V. Srinivasan, and G. Swallow, *RSVP-TE: Extensions to RSVP for LSP Tunnels*, IETF Request for Comments 3209, December 2001. [www]: <http://www.ietf.org/rfc/rfc3209.txt>. Last access on 070830.

[9] K. Kompella and Y. Rekhter, *Routing Extensions in Support of Generalized Multi-Protocol Label Switching (GMPLS)*, IETF Request for Comments 4202, October 2005. [www]: <http://www.ietf.org/rfc/rfc4202.txt>. Last access on 070830.

[10] K. Kompella and Y. Rekhter, *OSPF Extensions in Support of Generalized Multi-Protocol Label Switching (GMPLS)*, IETF Request for Comments 4203, October 2005. [www]: <http://www.ietf.org/rfc/rfc4203.txt>. Last access on 070830.

[11] L. Berger, *Generalized Multi-Protocol Label Switching (GMPLS) Signaling Functional Description*, IETF Request for Comments 3471, January 2003. [www]: <http://www.ietf.org/rfc/rfc3471.txt>. Last access on 070830.

[12] L. Berger, *Generalized Multi-Protocol Label Switching (GMPLS) Signaling Resource ReserVation Protocol-Traffic Engineering (RSVP-TE) Extensions*, IETF Request for Comments 3473, January 2003. [www]: <http://www.ietf.org/rfc/rfc3473.txt>. Last access on 070830.

[13] A. Farrel, J.-P. Vasseur, and J. Ash, *A Path Computation Element (PCE)-Based Architecture*, IETF Request for Comments 4655, August 2006. [www]: <http://www.ietf.org/rfc/rfc4655.txt>. Last access on 070830.

[14] G. Bernstein and Y. Lee, *Applicability of GMPLS and PCE to Wavelength Switched Optical Networks*, IETF Internet Draft, June 25, 2007. [www]: <http://tools.ietf.org/wg/ccamp/draft-bernstein-ccamp-wavelength-switched-00.txt>. Last access on 070830.

[15] T. Otani, H. Guo, K. Miyazaki, and D. Caviglia, *Generalized Labels of Lambda-Switching Capable Label Switching Routers (LSR)*, IETF Internet Draft, June 2007. [www]: <http://tools.ietf.org/wg/ccamp/draft-otani-ccamp-gmpls-lambda-labels-00.txt>. Last access on 070830.

[16] ITU-T Recommendation G.694.1, *Spectral grids for WDM applications: DWDM frequency grid*, June 2002. [www]: <http://www.itu.int/rec/T-REC-G.694.1-200206-I/en>. Last access on 071120.

[17] ITU-T Recommendation G.694.2, *Spectral grids for WDM applications: CWDM wavelength grid*, December 2003. [www]: <http://www.itu.int/rec/T-REC-G.694.2-200312-I/en>. Last access on 071120.

[18] W. Imajuku, Y. Sone, I. Nishioka, and S. Seno, *Routing Extensions to Support Network Elements with Switching Constraint*, IETF Internet Draft, July 2007. [www]: <http://tools.ietf.org/wg/ccamp/draft-imajuku-ccamp-rtg-switching-constraint-02.txt>. Last access on 070830.

[19] J. Strand and A. Chiu, *Impairments and Other Constraints on Optical Layer Routing*, IETF Request for Comments 4054, May 2005. [www]: <http://www.ietf.org/rfc/rfc4054.txt>. Last access on 070830.

[20] S. Reinhold, *GMPLS for optical network segments (working title)*, work in progress (M.Sc. thesis), 2007, Department of Communication Systems (COS), Royal Institute of Technology, Stockholm, Sweden.

[21] D. Farinacci, T. Li, S. Hanks, D. Meyer, and P. Traina, *Generic Routing Encapsulation (GRE)*, IETF Request for Comments 2784, March 2000. [www]: <http://www.ietf.org/rfc/rfc2784.txt>. Last access on 071006.

Appendix A: Abbreviations and acronyms

AS	Autonomous System
ASE	Amplifier Spontaneous Emission
BER	Bit Error Rate
BFS	Breadth First Search
CSPF	Constrained Shortest Path First
CR-LDP	Constraint-based Routing-Label Distribution Protocol
CWDM	Coarse Wavelength Division Multiplexing
DWDM	Dense Wavelength Division Multiplexing
ERO	Explicit Route Object
FEC	Forwarding Equivalence Class
FIFO	First-In-First-Out
FSC	Fiber Switching Capable
G-PID	Generalized Payload-ID
GLSR	Generalized Label Switching Router
GMPLS	Generalized Multi-Protocol Label Switching
GRE	Generic Routing Encapsulation
IANA	Internet Assigned Numbers Authority
IETF	Internet Engineering Task Force
IS-IS	Intermediate System to Intermediate System
IP	Internet Protocol
KSP	K Shortest Paths
L2SC	Layer-2 Switching Capable
LIFO	Last-In-First-Out
LMP	Link Management Protocol
LMP-WDM	Link Management Protocol-Wavelength Division Multiplexing
LSC	Lambda Switching Capable
LSA	Link State Advertisement
LSP	Label Switched Path
LSR	Label Switching Router
MLCP	Multi-Layer Control Plane
MPLS	Multi-Protocol Label Switching
NARB	Network Aware Resource Broker
OADM	Optical Add-Drop Multiplexer
OEO	Optical-Electronic-Optical
OOO	Optical-Optical-Optical
OSNR	Optical Signal-to-Noise Ratio
OSPF	Open Shortest Path First
OSPF-TE	Open Shortest Path First-Traffic Engineering
PCC	Path Computation Client
PCE	Path Computation Element
PMD	Polarization Mode Dispersion
PSC	Packet Switching Capable
QoS	Quality of Service
RCE	Resource Computation Engine

RFC	Request For Comments
ROADM	Reconfigurable Optical Add-Drop Multiplexer
RSVP	Resource ReSerVation Protocol
RSVP-TE	Resource ReSerVation Protocol-Traffic Engineering
SPF	Shortest Path First
TED	Traffic Engineering Database
TLV	Type-Length-Value
TDM	Time-Division Multiplexing
TE	Traffic Engineering
VLSR	Virtual Label Switching Router
VTY	Virtual TeletYpe
WDM	Wavelength Division Multiplexing

Appendix B: Table of OSPF-TE VTY commands

Table B.1 holds the OSPF-TE VTY commands extending the set of Zebra OSPF daemon commands. These VTY commands can be given in the *OSPF_TE_IF_NODE* command node scope, therefore they can be used in an OSPF daemon configuration file.

Command	Arg. 1	Arg. 2	Arg. 3	Arg. 4	Arg. 5	Arg. 6	Resulting action in DB
wavegrid	new	dwdm	cs [GHz]	freq. [GHz]	grid size	bw [B/s]	creates a dwdm wavegrid
		cwdm	cs [nm]	λ [nm]	grid size	bw [B/s]	creates a cwdm wavegrid
	delete	dwdm	-	-	-	-	deletes a dwdm wavegrid
		cwdm	-	-	-	-	deletes a cwdm wavegrid
	set	dwdm	cs [GHz]	freq. [GHz]	-	-	sets a frequency in wavegrid
		cwdm	cs [nm]	λ [nm]	-	-	sets a wavelength in wavegrid
	unset	dwdm	cs [GHz]	freq. [GHz]	-	-	unsets a frequency in wavegrid
		cwdm	cs [nm]	λ [nm]	-	-	unsets a wavelength in wavegrid
unselectable	add	IPv4 address	-	-	-	-	adds unselectable interface
	delete	IPv4 address	-	-	-	-	deletes unselectable interface
constraint	add	8-bit code	24-bit value	-	-	-	adds user-defined constraint
	delete	8-bit code	24-bit value	-	-	-	deletes user-defined constraint

Table B.1: OSPF-TE VTY commands.

Appendix C: Table of DRAGON software changes

Table C.1 holds the DRAGON (i.e. Zebra OSPF daemon and RCE) software changes. For each created or modified file, the changes and their importance is indicated.

File	Changes	Importance
$OSPF/zebra/ospfd/ospf_te_lsa.h	added support for experimental sub-TLVs	critical
$OSPF/zebra/ospfd/ospf_te_lsa.c	added support for experimental sub-TLVs	critical
$OSPF/zebra/ospfd/ospf_te.h	defined structures for experimental sub-TLVs	critical
$OSPF/zebra/ospfd/ospf_te.c	added support for experimental sub-TLVs, defined and installed VTY commands	critical
$NARB/rce/rce_lsa.hh	added defines for experimental sub-TLVs and user-defined constraints	moderate
$NARB/rce/rce_lsa.cc	added support for adding information carried by experimental sub-TLVs to links in the TED	critical
$NARB/rce/rce_lsp.hh	added define for path computation option	moderate
$NARB/rce/rce_lsp.cc	added support for loading path requirements and triggering path computation event	moderate
$NARB/rce/resource.hh	added support for adding information carried by experimental sub-TLVs to links in the TED	critical
$NARB/rce/resource.cc	added support for adding information carried by experimental sub-TLVs to links in the TED	critical
$NARB/rce/path_req.hh	defined a class representing path requirements appended to path computation request, new file	critical
$NARB/rce/path_req.cc	defined a class representing path requirements appended to path computation request, new file	critical
$NARB/rce/pcen_bfs.hh	defined classes abstracting a path computation event, added custom sort class, new file	critical
$NARB/rce/pcen_bfs.cc	defined classes abstracting a path computation event, added custom sort class, new file	critical
$NARB/rce/rce_test.cc	added support for requesting path computation events and parsing responses from the RCE	moderate

Table C.1: DRAGON software changes. Here, $OSPF maps to the location of the dragon-sw package while $NARB maps to the location of the narb-sw package.

Appendix D: A RCE stack and heap profile

Figure D.1 shows a RCE stack and heap profile. This profile contains the total amount of spacetime consumed by a RCE process which has sequentially processed and returned 1000 client requests (for a path between VLSR1 and VLSR7).

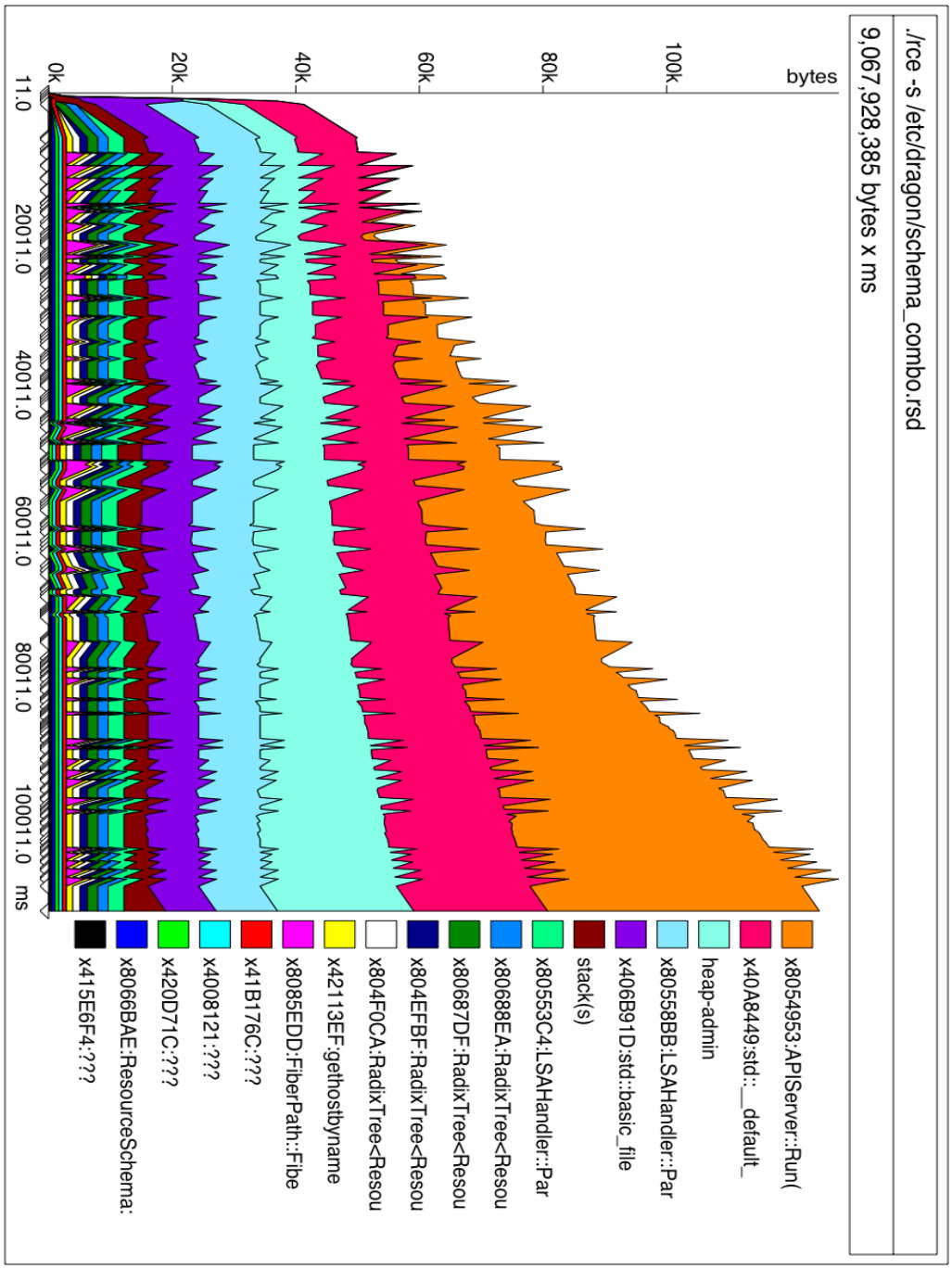

Figure D.1: The RCE stack and heap profile. Stack spacetime is shown in brown, heap spacetime allocated for candidate paths is shown in purple.

COS/CCS 2007-25

www.kth.se