This final project was realized within the framework of the Erasmus exchange programme between Universitat Politècnica de Catalunya, Vrije Universiteit Brussel, Politecnico di Torino and submitted in partial fulfillment of the requirements for the master degree in Computer Science Engineering.
<table>
<thead>
<tr>
<th>Acronym</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ASON</td>
<td>Automatically Switched Optical Network</td>
</tr>
<tr>
<td>ATM</td>
<td>Asynchronous Transfer Mode</td>
</tr>
<tr>
<td>CC</td>
<td>Connection Controller</td>
</tr>
<tr>
<td>CoS</td>
<td>Class of Service</td>
</tr>
<tr>
<td>DCN</td>
<td>Data Communication Network</td>
</tr>
<tr>
<td>FA</td>
<td>Forwarding Adjacency</td>
</tr>
<tr>
<td>FSC</td>
<td>Fiber Switching Capable</td>
</tr>
<tr>
<td>GLR</td>
<td>Generalized Label Request</td>
</tr>
<tr>
<td>GMPLS</td>
<td>Generalized Multi-Protocol Label Switching</td>
</tr>
<tr>
<td>GUI</td>
<td>Graphical User Interface</td>
</tr>
<tr>
<td>IETF</td>
<td>Internet Engineering Task Force</td>
</tr>
<tr>
<td>IGP</td>
<td>Interior Gateway Protocol</td>
</tr>
<tr>
<td>IP</td>
<td>Internet Protocol</td>
</tr>
<tr>
<td>IPoWDM</td>
<td>IP over WDM</td>
</tr>
<tr>
<td>IPTV</td>
<td>IP Television</td>
</tr>
<tr>
<td>ISO</td>
<td>International Standard Organization</td>
</tr>
<tr>
<td>ITU-T</td>
<td>International Telecommunication Union – Telecommunication Standardization Bureau</td>
</tr>
<tr>
<td>LIB</td>
<td>Label Information Base</td>
</tr>
<tr>
<td>LRM</td>
<td>Link Resource Manager</td>
</tr>
<tr>
<td>LSC</td>
<td>Lambda Switching Capable</td>
</tr>
<tr>
<td>LSP</td>
<td>Label Switched Path</td>
</tr>
<tr>
<td>MPλS</td>
<td>Multi-Protocol Lambda Switching</td>
</tr>
<tr>
<td>MPLS</td>
<td>Multi-Protocol Label Switching</td>
</tr>
<tr>
<td>MT</td>
<td>Multi-Topology</td>
</tr>
<tr>
<td>MTE</td>
<td>Multilayer Traffic Engineering</td>
</tr>
<tr>
<td>OADM</td>
<td>Optical Add-Drop Multiplexer</td>
</tr>
<tr>
<td>OAM&amp;P</td>
<td>Operations, Administration, Maintenance, and Provisioning</td>
</tr>
<tr>
<td>OC</td>
<td>Optical Carrier</td>
</tr>
<tr>
<td>OEO</td>
<td>Optical - Electronic – Optical</td>
</tr>
<tr>
<td>OIF</td>
<td>Optical Internetworking Foundation</td>
</tr>
<tr>
<td>OSI</td>
<td>Open Systems Interconnection</td>
</tr>
<tr>
<td>OSPF</td>
<td>Open Shortest Path First</td>
</tr>
<tr>
<td>OXC</td>
<td>Optical Cross Connect</td>
</tr>
<tr>
<td>PSC</td>
<td>Packet Switching Capable</td>
</tr>
<tr>
<td>QoS</td>
<td>Quality of Service</td>
</tr>
<tr>
<td>RSVP</td>
<td>Resource reSerVation Protocol</td>
</tr>
</tbody>
</table>
**RSVP-TE** – Resource ReSerVation Protocol – Traffic Engineering

**RWA** – Routing and Wavelength Assignment

**SDH** – Synchronous Digital Hierarchy

**SLA** – Service Level Agreement

**SLRC** – Single-Layered Route Computation

**SONET** – Synchronous Optical NETwork

**SPF** – Shortest Path First

**TDM** – Time Division Multiplexing

**TE** – Traffic Engineering

**TED** – Traffic Engineering Database

**TLRC** – Two-Layered Route Computation

**UNI** – User to Network Interface

**WADM** – Wavelength Add-Drop Multiplexer

**WDM** – Wavelength Division Multiplexing
Contents

1 Introduction .............................................................................................................................................. 1
  1.1 Organization of the Thesis ............................................................................................................. 6

2 ASON/GMPLS Overview ..................................................................................................................... 9
  2.1 IPoWDM .......................................................................................................................................... 10
    2.1.1 IPoWDM Network Architecture ............................................................................................... 12
    2.1.2 Control Plane Integration .......................................................................................................... 13
  2.2 ASON ............................................................................................................................................... 14
    2.2.1 ASON Control Plane .................................................................................................................. 16
  2.3 GMPLS .......................................................................................................................................... 17
  2.4 Traffic Engineering in IPoWDM Networks ...................................................................................... 20
    2.4.1 Multilayer TE with ASON/GMPLS routers (MTE) ................................................................. 21
    2.4.2 On Line Traffic Engineering ..................................................................................................... 24

3 Service Differentiation ........................................................................................................................ 27
  3.1 Service Differentiation: the evolution towards modern scenarios ................................................. 28
  3.2 Multi-Layer Traffic Engineering Policies (MTE) .......................................................................... 30
    3.2.1 Virtual Topology First (VT-F) Policy ......................................................................................... 30
    3.2.2 Physical Topology First (PTF) Policy ......................................................................................... 31
  3.3 Service Differentiation policies ....................................................................................................... 32
    3.3.1 Routing Policy Differentiation (RP-Diff) .................................................................................. 33
    3.3.2 Virtual Topology Hard Differentiated Service (VT-HardDiff) ................................................. 34
    3.3.3 Virtual Topology Soft Differentiated Service (VT-SoftDiff) .................................................... 36

4 GMPLS Protocols for Service Differentiation ..................................................................................... 39
  4.1 OSPF Overview .............................................................................................................................. 40
4.1.1 Areas............................................................................................................ 42
4.1.2 Type of Router ............................................................................................ 42
4.1.3 OSPF Functionalities ................................................................................ 43
4.1.4 OSPF Type of Packets ............................................................................. 43
4.1.5 The OSPF Protocols ............................................................................... 45
4.2 The Opaque LSA Option ........................................................................... 47
4.2.1 The Opaque LSA ..................................................................................... 47
4.2.2 Flooding Opaque LSAs .......................................................................... 48
4.3 Traffic Engineering Extensions to OSPF (OSPF-TE) ............................... 49
4.3.1 Traffic Engineering LSAs (TE-LSA) .................................................... 49
4.3.2 TE-LSA Header ...................................................................................... 50
4.3.3 TE-LSA Payload ................................................................................... 50
4.3.4 Sub-TLVs Details .................................................................................. 51
4.4 Multi-Topology Routing in OSPF (OSPF-MT) ........................................ 53
4.4.1 Multi-Topology Routing vs. TOS-Based Routing .................................... 54
4.4.2 Protocol Extensions for Multi-Topology .............................................. 54
4.4.3 Shortest Path Calculation ...................................................................... 55
4.4.4 Default Topology Link Exclusion Capability ........................................ 56
5 Working Environment: ASON/GMPLS Simulation Tool ......................... 59
5.1 OMNet++ ..................................................................................................... 60
5.1.1 Modelling Concepts .............................................................................. 61
5.1.2 Building and Running simulations ....................................................... 64
5.1.3 Analyzing Simulation Results .............................................................. 65
5.2 The INET Framework .............................................................................. 65
5.3 IP/MPLS over ASON/GMPLS Simulation Tool ........................................ 66
5.3.1 Statistics Collector ................................................................. 68
5.3.2 Connection Generator .......................................................... 68
5.3.3 Network .................................................................................. 68
5.3.4 GMPLS Node .......................................................................... 69
5.3.5 Setup Handler .......................................................................... 70
5.3.6 Physical Layer .......................................................................... 70
5.3.7 Control Layer ........................................................................... 71
5.3.8 Virtual Layer ............................................................................ 76

6 Simulator Extensions ......................................................................... 81

6.1 Chapter overview and Goal of this work ........................................... 82
6.2 Simulator Extensions ....................................................................... 83
  6.2.1 Link State Module .................................................................... 83
  6.2.2 Implementation Steps ................................................................. 83
6.3 OSPF Integration .......................................................................... 84
6.4 OSPF C++ Classes ........................................................................ 85
  6.4.1 Interface classes ....................................................................... 85
  6.4.2 IMessageHandler classes ........................................................ 87
  6.4.3 Neighbor classes ..................................................................... 89
  6.4.4 OSPFLSA Classes ................................................................. 92
6.5 OSPF-TE ................................................................................... 98
  6.5.1 Link TLVs ............................................................................. 98
  6.5.2 OSPF-TE Implementation ......................................................... 103
  6.5.3 The OSPFPacket class ........................................................... 103
  6.5.4 OSPFNeighbor class ............................................................... 106
  6.5.5 OSPFMessageHandler classes ................................................. 107
  6.5.6 Synchronizing TED and TE-LSA Database ............................... 108
Chapter 1 – Introduction
Abstract

Transport networks have moved towards a model called Automatically Switched Optical Network (ASON) controlled by the Generalized Multi-protocol Label Switching (GMPLS) control plane. An ASON/GMPLS network is an intelligent optical network providing the Internet Protocol (IP)/Multi Protocol Label Switching (MPLS) layer with dynamic and distributed connection provisioning.

In the IP/MPLS layer the service differentiation has become necessary to cope with the aggregation of various services, including performance-sensitive as well as best effort applications. The Differentiated Services (DiffServ) technology has become quite mature in IP/MPLS domains and the research world has recently made an effort to extend it within the ASON/GMPLS layer. In [14] the DiffServ is executed in a multilayer fashion by means of a technique called virtual topology differentiation. This technique allows different Classes of Service (CoS) to be accommodated over independent virtual topologies and with different multilayer routing policies. The authors demonstrate how different CoS are provided with different levels of QoS. However, the extension of the GMPLS control plane for the introduction of the DiffServ in IP/MPLS over ASON/GMPLS networks is not addressed.

This thesis addresses the problem of multilayer service differentiation in IP/MPLS over ASON/GMPLS network focusing on the GMPLS control plane. We start from the Multi-Topology (MT) routing technique used in the Open Shortest Path First (OSPF) protocol to address the MT problem in IP networks. We first make the suitable adaptation to employ this technique in the GMPLS control plane as well. The extended MT technique is then implemented in an IP/MPLS over ASON/GMPLS simulator. We finally use the extended simulator to simulate three strategies introduced in [14] in order to make an analysis of the multilayer service differentiation.
1 Introduction
Since its birth, the Internet has been continuously growing at an exceptional exponential rate. Many companies interested in this phenomenon have been studying for years the traffic growth of the global network to foresee and plan upgrades of the Internet devices, such as switches and routers, and transmission mediums.

IP networks today carry traffic of different types, characteristics, and performance requirements. According to [25], the total IP traffic will almost double every 2 years through 2011 with a yearly increasing rate of about 46%. In [25], the authors also illustrates how the different consumer applications will be influencing the growth of the Internet traffic in the next years.

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>By Sub-Segment (terabytes per month)</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Web, e-mail, file transfer</td>
<td>382,084</td>
<td>505,996</td>
<td>692,812</td>
<td>948,425</td>
<td>1,233,172</td>
<td>1,603,615</td>
<td>2,786,415</td>
</tr>
<tr>
<td>P2P</td>
<td>1,060,226</td>
<td>1,259,770</td>
<td>1,772,293</td>
<td>2,379,025</td>
<td>3,111,591</td>
<td>4,040,409</td>
<td>5,299,360</td>
</tr>
<tr>
<td>Gaming</td>
<td>66,844</td>
<td>91,943</td>
<td>133,357</td>
<td>185,090</td>
<td>250,574</td>
<td>316,212</td>
<td>386,322</td>
</tr>
<tr>
<td>Video Communications</td>
<td>11,629</td>
<td>15,575</td>
<td>24,932</td>
<td>36,608</td>
<td>47,173</td>
<td>66,101</td>
<td>92,453</td>
</tr>
<tr>
<td>VoIP</td>
<td>10,365</td>
<td>23,035</td>
<td>35,339</td>
<td>57,653</td>
<td>73,575</td>
<td>92,815</td>
<td>110,456</td>
</tr>
<tr>
<td>Internet Video to PC</td>
<td>53,674</td>
<td>174,427</td>
<td>484,027</td>
<td>835,184</td>
<td>1,232,461</td>
<td>1,726,114</td>
<td>2,331,908</td>
</tr>
<tr>
<td>Internet Video to TV</td>
<td>0</td>
<td>12,727</td>
<td>110,952</td>
<td>353,095</td>
<td>620,197</td>
<td>936,560</td>
<td>1,342,482</td>
</tr>
<tr>
<td><strong>By Geography (TB per month)</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>North America</td>
<td>534,256</td>
<td>619,705</td>
<td>917,365</td>
<td>1,287,206</td>
<td>1,698,700</td>
<td>2,242,841</td>
<td>2,861,772</td>
</tr>
<tr>
<td>Western Europe</td>
<td>334,600</td>
<td>505,329</td>
<td>614,015</td>
<td>821,041</td>
<td>1,066,310</td>
<td>1,651,070</td>
<td>2,348,721</td>
</tr>
<tr>
<td>Asia Pacific</td>
<td>505,782</td>
<td>819,072</td>
<td>1,281,277</td>
<td>1,742,834</td>
<td>2,315,755</td>
<td>3,049,994</td>
<td>4,653,774</td>
</tr>
<tr>
<td>Japan</td>
<td>60,680</td>
<td>95,747</td>
<td>147,733</td>
<td>223,120</td>
<td>319,788</td>
<td>436,057</td>
<td>558,631</td>
</tr>
<tr>
<td>Latin America</td>
<td>19,917</td>
<td>33,755</td>
<td>57,033</td>
<td>90,365</td>
<td>130,466</td>
<td>189,962</td>
<td>286,559</td>
</tr>
<tr>
<td>Central Eastern Europe</td>
<td>40,723</td>
<td>59,037</td>
<td>90,196</td>
<td>122,272</td>
<td>165,367</td>
<td>222,880</td>
<td>294,901</td>
</tr>
<tr>
<td>Middle East and Africa</td>
<td>9,435</td>
<td>18,708</td>
<td>33,904</td>
<td>54,613</td>
<td>84,537</td>
<td>127,688</td>
<td>185,549</td>
</tr>
<tr>
<td><strong>Total (TB per month)</strong></td>
<td>1,564,822</td>
<td>2,153,473</td>
<td>3,257,572</td>
<td>4,801,670</td>
<td>6,571,044</td>
<td>8,783,838</td>
<td>12,289,908</td>
</tr>
</tbody>
</table>

*Cisco, 2006*

**Figure 1.1 - IP Traffic Growth by subsegment**

**Definitions:**

Web, E-mail, and File Transfer – includes Web, e-mail, instant messaging, newsgroups, and file transfer (excluding P2P and commercial file sharing, such as iTunes)

P2P – includes peer-to-peer traffic from all recognized P2P systems, such as BitTorrent, eDonkey, etc.

Gaming – includes casual online gaming, networked console gaming, and multiplayer virtual world gaming

Video Communications – includes PC-based video calling, Web-based video conferencing, and Web-based video monitoring

VoIP – includes traffic from real-time voice services and P2P-based VoIP, but excludes wholesale voice transport

Internet Video to PC – free or pay TV or VoD viewed on a PC; excludes P2P video file downloads

Internet Video to TV – free or pay TV or VoD delivered via Internet but viewed on a TV screen using a STB or media gateway
As shown in Figure 1.1, over the last few years the main driving forces of the Internet traffic growth have been the commercial video service, or IP Television (IPTV), the Voice over IP service (VoIP) and the Internet video and multimedia applications (such as YouTube). Actual networks accommodate several types of applications, ranging from real-time and streaming, to bandwidth guaranteed, to traditional best effort applications. As result of a general convergence over the IP technology of most of the services offered by internet providers, IP networks are being actually used to deliver bundled services that include performance-sensitive applications, such as voice and video, as well as the so called “best effort” data services. Large enterprises also rely heavily on IP networks to transfer a mix of both critical (e.g., data center backup) and non-critical data for their business needs.

To support this traffic aggregation, service differentiation is necessary, driving the network evolution towards transport infrastructures enabling the provisioning of connections with certain Quality of Service (QoS) guarantees, such as high bandwidth, low end-to-end delay, low delay jitter and minimal losses. For instance, Real-time applications require the utilization of real-time channels, which must be set up with specific traffic characteristics and QoS satisfying requirements. Thus, the time required to set up an end-to-end real-time channel is one of the fundamental metrics to be taken into consideration in real-time applications.

Transport networks are moving towards a model of high performance Internet Protocol/Multi-Protocol Label Switching (IP/MPLS) routers interconnected through intelligent backbones, which directly provide an infrastructure for new IP services that are compatible with existing IP services [26]. In order to optimize the routing for the expected traffic demand, network operators often employ traffic engineering (TE) algorithms to determine the paths, which are then implemented using shortest-path routing protocols, such as OSPF or IS-IS. MPLS improves the IP layer Traffic Engineering by means of the fast label switching technology that creates connection oriented Label Switching Paths (LSPs).

Below the IP/MPLS layer, optical fiber using Wavelength Division Multiplexing (WDM) technique is the most promising wire line technology, offering an enormous network capacity. An IP over WDM network is designated for the transmission of IP traffic in a
WDM-enabled optical network to leverage both IP universal connectivity and massive WDM bandwidth capacity [27]. There are different types of IPoWDM control architectures which differ in their dynamic capabilities, configurations and the interactions between the IP and the optical control plane.

An IPoWDM network supports emerging requirements such as dynamic and rapid provisioning of connections, automatic topology discovery, reactive Traffic Engineering and fast restoration. A key issue to achieve these functionalities is the definition of the optical control plane responsible for the routing and signaling processes. A first model of optical network with automatic switching capability has recently been standardized by the International Telecommunication Union, Telecommunication Standardization Sector (ITU-T): the Automatic Switched Optical Network (ASON) [3]. While current optical networks only provide statically allocated transport capacity, the ASON dynamically sets up and tears down optical channels. To achieve this functionality the architecture of a control plane, which is responsible for the routing and signaling process, is defined. The Generalized Multi-protocol Label Switching (GMPLS) set of protocols [1] has been widely recognized as the paradigm implementing the future ASON control plane. GMPLS is a networking specification standardized by Internet Engineering Task Force (IETF) [6]. GMPLS is a technology that provides enhancements to MPLS to support network switching for time, wavelength, and space switching as well as for packet switching.

The Service Differentiation techniques and Traffic Engineering policies that provide load balancing for IP/MPLS networks can be extended and integrated with the ASON/GMPLS layer. This results in Multilayer Traffic Engineering (MTE) paradigm and integrated routing policies. These techniques allow the operator to accommodate service requests depending on their Class of Service (CoS), either in the IP/MPLS domain or in the optical domain, by aggregating traffic on the existing capacity or setting up new optical connections, respectively [14].

The choice of the Service Differentiation policies can influence the network performances from the operator’s perspective (e.g., optimization of the network resources) and from the user’s perspective (e.g., QoS offered). In [14] the DiffServ is executed in a
multilayer fashion by means of a technique called virtual topology differentiation. This technique allows different Classes of Service (CoS) to be accommodated over independent virtual topologies and with different multilayer routing policies.

- **Routing Policy Differentiation (RP-Diff) scheme.**
  This approach consists of a simple algorithm where the MTE routing policy is established on the base of the CoS the service request belongs to.

- **Virtual Topology Hard Differentiation (VT-HardDiff) scheme.**
  Traffic belonging to different CoS is accommodated over separated lightpaths (OLSPs). Therefore, different virtual topologies will be formed by different set of OLSPs accommodating different CoS. The virtual topologies are independent from each other, thus different routing policies can be used to accommodate traffic, without sharing any resource.

- **Virtual Topology Soft Differentiation (VT-SoftDiff) scheme.**
  This approach follows the same schema of VT-HardDiff. However, a certain degree of resource sharing is introduced. This allows virtual topologies to share part of the established OLSPs.

This thesis addresses the problem of multilayer service differentiation in IP/MPLS over ASON/GMPLS network focusing on the GMPLS control plane. We start from the Multi-Topology (MT) routing technique used in the Open Shortest Path First (OSPF) protocol to address the MT problem in IP networks. We first make the suitable adaptation to employ this technique in the GMPS control plane as well. The extended MT technique is then implemented in an IP/MPLS over ASON/GMPLS simulator previously developed at the Vrije Universiteit Brussel [7].

The simulation framework used by the IP/MPLS over ASON/GMPLS simulator is OMNeT++. This engine is commonly used for network simulations because of the availability of the INET Framework [21] that is a collection of OMNeT++ modules [22], ready to be used, implementing different protocols of the Open Systems Interconnection (OSI) stack. In particular, the INET’s OSPF module has been suitably extended for the implementation of the Multi-Topology capability and the Service Differentiation policies listed above.

After elaborating on the design and implementation of the simulator, the thesis presents an extensive set of experiments. The experiments focus on two of the key metrics used
Chapter 1 - Introduction

to evaluate the QoS provided by networks to “time-sensitive” applications: blocking probability and packet delay.

1.1 Organization of the Thesis

Some theoretical basics are needed to properly understand the development of this work. For this reason, the first sections of this thesis explain basic concepts about the topic we are talking about. Chapter 2 presents a description of the modern optical networks, introducing the internet-working model used in the development of the ASON/GMPLS router simulator. ASON control plane is introduced together with its natural paradigm GMPLS.

Chapter 3 introduces the Service Differentiation, which is the main topic of this work. First, a brief overview about the state of the art is presented, followed by a description of two commonly used MTE routing policies: VT-F and PT-F. The second part of this section presents finally three different algorithms that will be implemented in our simulator, in order to test their performances under the Service Differentiation perspective: Routing Policy Differentiation, Virtual Topology Hard Differentiation and Virtual Topology Hard Differentiation, designed in [14].

Chapter 4 gives an overview over the preliminary study over the protocols needed to properly implement the Multi-Topology routing and the Service Differentiation algorithms introduced before. Traffic Engineering and Multi-Topology extensions to the OSPF are defined. This is an important part of the thesis, giving the protocol basics to implement the MT routing within a realistic IP/MPLS over ASON/GMPLS simulator. Specifications given in this chapter will be then followed during the implementation stage of this work.

In chapter 5, the OMNeT++ simulation engine is described to let the reader understand how a network model is implemented and how it behaves during simulations. In the second part of the chapter the INET Framework is introduced and, in particular, we focus on the description of the available simulation tool that represents the starting point of our work.
Chapter 1 - Introduction

Chapter 6 gives implementation details over the simulator extensions. Firstly the goal of the thesis is shown and the simulator is described from the point of view of the OM-NeT++ modules. Configuration files and interaction between the modules are also described. In the second part of this chapter our IP/MPLS over ASON/GMPLS simulator extensions are explained from a lower abstraction point of view. The main steps covered during the implementation stage are described, together with the main implemented C++ classes, their methods and variables. When needed, the chapter focuses on how the two layers inside each router are synchronized to enable Multi-Layer Traffic Engineering and Service Differentiation when setting up a connection.

Chapter 7 presents an extensive set of experiments carried out using our final simulator in order to achieve some important goals:

- Evaluate the contribution to the Service Differentiation given by the MT routing approach proposed in [14], applied to a realistic IP/MPLS over ASON/GMPLS environment.
- Compare the results obtained simulating the three Service Differentiation algorithms.

Also, a brief description of the traffic model implemented, some further OMNET++ modules added to the simulator and their implementation is given.

Finally, Chapter 8 gives some conclusions on the experiments results and the simulator development. Future enhancements and developments of the simulator are also suggested.
2
ASON/GMPLS Overview
Chapter 2 – ASON/GMPLS Overview

This chapter firstly presents some basics about IP-over-WDM technology. In particular it gives a brief introduction on the optical networks evolution and advantages. Then we will describe the architecture and the internetworking model adopted for the creation of an ASON/GMPLS router model that is the one used in this work. Afterwards we will focus on the optical network’s control plane integration and description for an Automatically Switched Optical Network (ASON), going into depth in describing the ITU-T G.8080 official architecture for it. Subsequently GMPLS protocols suite will be introduced to use it as ASON interfaces described in the ITU standard. In the last part of this chapter Traffic Engineering on Multi-Layer networks is described and in particular online algorithms and grooming for traffic engineering are introduced.

2.1 IPoWDM

It is widely believed that IP provides the only convergence layer in the global and ubiquitous Internet. IP, a layer 3 protocol, is designed to address network level interoperability and routing over different subnets with different layer 2 technologies. Above the IP layer, there are a great variety of IP-based services and appliances that are still evolving. An example is IP-based home networking interconnecting a wide range of electronic devices. Hence, the inevitable dominance of IP traffic makes apparent the engineering practices that the network infrastructure should be optimized for IP. Below the IP layer, optical fiber using WDM is the most promising wireline technology, offering an enormous network capacity able to sustain the continuous Internet growth.

WDM technology will become more attractive as the price of WDM systems lowers. In addition, WDM optical networks are no longer just point-to-point pipes providing physical link service, but blend well with any new level of network flexibility requirements.

The control plane is responsible for transporting control messages to exchange reachability and availability information and computing and setting up the data forwarding paths. The data plane is responsible for the transmission of user and application traffic such as packet buffering and forwarding. IP does not separate the data plane from the control plane, and this in turn requires QoS mechanisms at routers to differentiate control messages from data packets.
Combining IP and WDM means, in the data plane, one can assign WDM optical network resources to forward IP traffic efficiently, and in the control plane, one can construct a unified control plane, presumably IP-centric, across IP and WDM networks. IP over WDM will also address all levels of interoperability issues on intra and inter-WDM optical networks and IP networks.

The motivation behind IP over WDM can be summarized as follows:

- WDM optical networks can address the continuous growth of the Internet traffic by exploiting the existing fiber infrastructure. The use of WDM technology can significantly increase the use of the fiber bandwidth.
- Most of the data traffic across networks is IP. Nearly all the end-user data application uses IP. Conventional voice traffic can also be packetized with Voice-over-IP (VoIP) techniques.
- IPoWDM inherits the flexibility and the adaptability offered in the IP control protocols.
- IPoWDM aims to achieve dynamic on-demand bandwidth allocation (or real-time provisioning) in optical networks.
- IPoWDM hopes to address WDM or optical Network Elements (NE) vendor interoperability with the help of IP protocols.
- IPoWDM can achieve dynamic restoration by leveraging the distributed control mechanisms implemented in the network.
- From a service point of view, IP over WDM networks can take advantage of the QoS frameworks, models, policies, and mechanisms proposed for and developed in the IP network.
- Given the lessons learned from IP and ATM integration, IP and WDM need a closer integration for efficiency and flexibility. For example, classical IP over ATM is static and complex, and IP to ATM address resolution is mandatory to translate between IP addresses and ATM addresses.

IP over WDM integration will eventually translate into an efficient optical network transport reducing the cost of IP traffic and increasing the utilization of the optical network.
2.1.1 IPoWDM Network Architecture

There are different types of IPoWDM architectures which differ in their dynamic capabilities, configurations and the interactions between the IP and the optical control plane. For our work we decided to refer exclusively to the IP over reconfigurable WDM architecture, which is described below, for its reconfiguration capabilities and natural support for online multi-layer traffic engineering.

Under an IP over reconfigurable WDM architecture, the router interfaces from the IP routers are connected to the client interfaces of the WDM network. Figure 2.1 illustrates such an IP over Re-configurable WDM network. In this architecture, the WDM crossconnects and add/drop interfaces are themselves interconnected into a WDM network with multi-wavelength fiber links. Therefore, the WDM network itself has a physical topology and a lightpath topology.

The WDM physical topology is composed of NEs interconnected by fibers; the WDM lightpath topology is formed by wavelength channel connections. Reconfigurable WDM is a circuit switching technology so the wavelength channel setup and tear-down are conducted in preserved phases. It is important to point out that IP traffic switching and
wavelength switching never work in the same layer in an IP over reconfigurable networks. This can be translated into an overlay network.

Lightpaths in the WDM network are designed to conform to the IP topology. By appropriately configuring the WDM cross-connects, a given router interface can be connected to any other router interface at any other router. As a result, the neighbouring router for a given router interface is configurable under this architecture. This infers that the physical networks can support a number of virtual topologies subject to the same network resource constraints. Lightpaths are set up, managed and torn down by the optical Control Plane using the Data Control Network (DCN) to exchange messages between the nodes and set all the switching devices. A brief description of the Control Plane and its different implementation models is given below.

2.1.2 Control Plane Integration

Control Plane (CP) is used in the literature to refer to the set of real-time mechanisms and algorithms needed for call or connection control. It deals mainly with the signaling to set up, supervise and release calls and connections. Although a detailed decomposition of the control plane and a description of each component is not the purpose of this work, we can safely assume that the signaling protocols for connection setup, the routing protocols supporting network discovery, and the protection/recovery mechanisms are the most significant features of the control plane. In this way, it is easier to track all of the recent control plane advances and proposals about the integration of multiple layers such as IP, ATM, SDH and WDM.

A significant element in the IP and optical integration is the corresponding business model proposed by each framework. The three basic business models, along with the requirements and issues that they impose, are summarized next:

- **The overlay model.** The routing algorithm, topology distribution and connection setup signaling protocols of the IP and the WDM networks are independent. The overlay model is the one that allows an easy migration from the existing situation to the deployment of ONEs for the transport of the IP directly over WDM. However, the implementation complexity of this model is burden, and it does not promote the integration of the control plane of the IP and the WDM net-
works. Only a formal request is passed from the IP client layer to the optical server layer.

- **The peer model.** The IP network has full topological view of the optical network and just a single routing algorithm instance is running in both the IP and the WDM networks. This model promotes the integration of the control plane of the IP and the WDM networks and is simpler in implementation, but its operation is far more complex than the overlay. In addition, this model can work only in cases where there is a single entity operating and managing the IP and the optical administrative domains.

- **The augmented model.** This is a combination of the previous two models. Each layer has its own protocols; however, routing information exchange is allowed between the two layers. This model can be seen as the golden mean, combining the advantages of the peer and overlay model and minimizing their disadvantages at the same time.

The development of this work, taking into account the previous considerations and the available simulation tools, has been based on the augmented model, regarding the protocols’ behavior between IP layer and optical layer.

### 2.2 ASON

The existing transport networks provide SONET/SDH and WDM services whose connections are provisioned via network management protocols. This process is both slow (weeks to months) relative to the switching speed and costly to the network providers.

An automatic switched optical network (ASON) is an optical/transport network that has dynamic connection capability. It encompasses SONET/SDH, wavelength, and potentially fiber connection services in both OEO and all-optical networks. There are a number of added values related to such a capability:

- Traffic engineering of optical channels: Where bandwidth assignment is based on actual demand patterns.
- Mesh network topologies and restoration: Mesh network topologies can in general be engineered for better utilization for a given demand matrix. Ring topologies might not be as efficient due to the asymmetry of traffic patterns.
- Managed bandwidth to core IP network connectivity: A switched optical network can provide bandwidth and connectivity to an IP network in a dynamic manner compared to the relatively static service available today.
Introduction of new optical services: The availability of switched optical networks will facilitate the introduction of new services at the optical layer. Those services include bandwidth on demand and optical virtual private networks (OVPN).

ASON/ASTN architecture belongs to client-server models or the overlay network models as defined in [2]. The salient feature of this model is the existence of well-recognized boundaries between client networks and provider domains. Client/provider separation is a direct recognition of today's networking realities where ownership of layer 3 and layer 1 equipment belongs to different organizations. This client/provider domain separation entails the running of different routing instances at each domain. Thus there is no need to share topology information between carriers and their clients.

As applied to optical networks, the ASON network interfaces are shown in figure 2.2. In this image all the components that compose ASON are shown. The architecture shown is intended to allow switching of optical network connections within the optical transport network under control of ASON signaling network.

There are three separate planes involved in the ASON network:

- A transport plane (TP)
- A control plane (CP)
- A management plane (MP)
The transport plane contains the transport network elements (switches and links) that carry the entity that is switched, i.e. optical connections. The transport plane is linked to the client network with a Physical Interface (PI) and to the Management Plane through a Network Management Interface for the Transport network (NMI-T). End-to-end connections are setup within the transport plane under the control of the ASON control plane (CP). The Control Plane contains Optical Connection Controller (OCC) linked together by the Internal Node to Node Interface (I-NNI). ASON CP is also linked to the client network using the User to Network Interface (UNI) and to the management plane using the Network to Management Interface for the ASON control plane (NMI-A). OCCs are also linked to their controlled transport plane devices by the Connection Control Interface (CCI).

In the next section we will focus on the Control Plane part of the ASON architecture and its related protocols are introduced afterwards in the GMPLS paragraph.

### 2.2.1 ASON Control Plane

A well-designed control plane architecture should give service providers better control of their network, while providing faster and improved accuracy of circuit set-up.

In summary, the control plane architecture should:

- Be applicable to a variety of transport network technologies (e.g., SONET/SDH, OTN, PXC). In order to achieve this goal, it is essential that the architecture isolates technology dependent aspects from technology independent aspects, and address them separately.
- Be sufficiently flexible to accommodate a range of different network scenarios. This goal may be achieved by partitioning the control plane into distinct components. This, allows vendors and service providers to decide the location of these components, and also allows the service provider to decide the security and policy control of these components.

The I-NNI defines the interface between adjacent optical connection controls (OCC) in the same network. There are two main aspects of I-NNI. Those are signaling and routing.

Path selection and setup through the optical network requires a signaling protocol. To facilitate the automation of the optical connection setup, nodes in the optical network must have an updated view of its adjacencies and of the utilization levels at the
various links of the network. This updated view is sometime referred to as state information. State information dissemination is defined as the manner in which local physical resource information is disseminated throughout the network. This information can then be distributed to the different nodes in the network using the control plane transport network IGP.

ASON I-NNI could be based on two key protocols, IP and MPLS. Since MPLS employs the principle of separation between the control and the forward planes, its extension to support I-NNI signaling is feasible. Generalized MPLS [1] defines MPLS extensions to suit types of label switching other than the in-packet label. Those other types include, time slot switching, wavelength and waveband switching, and position switching between fibers. Both CR-LDP and RSVP-TE, have been extended to allow for the request and the binding of generalized labels. With generalized MPLS, a label switched path (LSP) is established with the appropriate encoding type (e.g. SONET, wavelength, etc.). LSP establishment takes into account specific characteristics that belong to a particular technology.

GMPLS traffic engineering (TE) requires the availability of routing protocols that are capable of summarizing link state information in their databases. Extensions to IP routing protocols, OSPF and IS-IS, in support of link state information for generalized MPLS are usually used and are part of the purpose of this work.

### 2.3 GMPLS

Within the past years, the International Engineering Task Force (IETF) has extended the MPLS suite of protocols to include devices that switch in time, wavelength, (e.g., DWDM) and space domains (e.g., OXC) via GMPLS. This allows GMPLS–based networks to find and provision an optimal path based on user traffic requirements for a flow that potentially starts on an IP network, is then transported by SONET, and then is switched through a specific wavelength on a specific physical fiber.

Table 2.1 gives a summary of the GMPLS framework.
Table 2.1 - GMPLS domain of applicability

<table>
<thead>
<tr>
<th>Switching Domain</th>
<th>Traffic Type</th>
<th>Forwarding Scheme</th>
<th>Example of Device</th>
<th>Nomenclature</th>
</tr>
</thead>
<tbody>
<tr>
<td>Packet, cell</td>
<td>IP, Asynchronous Transfer Mode (ATM)</td>
<td>Label as shim header, virtual channel connection (VCC)</td>
<td>IP router, ATM switch</td>
<td>Packet Switch capable (PSC)</td>
</tr>
<tr>
<td>Time</td>
<td>TDM/SONET</td>
<td>Time slot in repeating cycle</td>
<td>Digital cross-connect system (DCS), ADM</td>
<td>TDM capable</td>
</tr>
<tr>
<td>Wavelength</td>
<td>Transparent</td>
<td>Lambda</td>
<td>DWDM</td>
<td>Lambda switch capable (LSC)</td>
</tr>
<tr>
<td>Physical space</td>
<td>Transparent</td>
<td>Fiber, line</td>
<td>OXC</td>
<td>Fiber switch capable (FSC)</td>
</tr>
</tbody>
</table>

GMPLS was introduced to generalize the MPLS architecture to also consider non packet-based control planes in addition to the conventional packet networks.

The conventional MPLS label is extended to a generalised label to allow the representation of not only data packets labels, but also labels that identify timeslots, wavelengths, and fibers. Waveband switching is introduced to switch a set of contiguous wavelengths together to a new waveband. Waveband switching can reduce the distortion on the individual wavelengths and thus allow tighter separation of the individual wavelengths.

The spirit of GMPLS is to propose a common IP-based control plane for all types of networks including optical circuit switched networks.

- GMPLS peering can be applied to addressing, signalling, and routing. A global addressing scheme is the foundation of network peering.
- GMPLS proposes IP addressing for all networks. Signalling represents the agreed message formats for automatic circuit setup and teardown.
- GMPLS suggests the use of RSVP (Reservation Protocol) and LDP (Label Distribution Protocol) as the standard signalling mechanism. Routing provides the intelligence into the network through routing tables (so that signalling can provide its best performance as a dynamic, distributed process).
- GMPLS proposes the use of OSPF (Open Shorter Path First) and BGP (Border Gateway Protocol) for routing reachability and availability information sharing.
The evolution of MPLS into GMPLS has extended the signaling (RSVP–TE, CR–LDP) and routing protocols (OSPF–TE, IS–IS–TE). The extensions accommodate the characteristics of TDM/SONET and optical networks. A new protocol, link-management protocol (LMP), has been introduced to manage and maintain the health of the control and data planes between two neighboring nodes. LMP is an IP-based protocol that includes extensions to RSVP–TE and CR–LDP.

Table 2.2 summarizes these protocols and the extensions for GMPLS.

<table>
<thead>
<tr>
<th>Protocols</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Routing</strong></td>
<td><strong>Description</strong></td>
</tr>
<tr>
<td>OSPF-TE</td>
<td>Routing protocols for the auto-discovery of network topology, advertise resource availability (e.g., bandwidth or protection type). The major enhancements are as follows:</td>
</tr>
<tr>
<td>IS-IS-TE</td>
<td>- Advertising of link-protection type (1+1, 1:1, unprotected, extra traffic)</td>
</tr>
<tr>
<td></td>
<td>- Implementing derived links (forwarding adjacency) for improved scalability</td>
</tr>
<tr>
<td></td>
<td>- Accepting and advertising links with no IP address—link ID</td>
</tr>
<tr>
<td></td>
<td>- Incoming and outgoing interface ID</td>
</tr>
<tr>
<td></td>
<td>- Route discovery for back-up that is different from the primary path (shared-risk link group)</td>
</tr>
<tr>
<td><strong>Signaling</strong></td>
<td><strong>Description</strong></td>
</tr>
<tr>
<td>RSVP-TE</td>
<td>Signaling protocols for the establishment of traffic-engineered LSPs. The major enhancements are as follows:</td>
</tr>
<tr>
<td>CR-LDP</td>
<td>- Label exchange to include non-packet networks (generalized labels)</td>
</tr>
<tr>
<td></td>
<td>- Establishment of bidirectional LSPs</td>
</tr>
<tr>
<td></td>
<td>- Signaling for the establishment of a back-up path (protection Information)</td>
</tr>
<tr>
<td></td>
<td>- Expediting label assignment via suggested label</td>
</tr>
<tr>
<td></td>
<td>- Waveband switching support—set of contiguous wavelengths switched together</td>
</tr>
<tr>
<td><strong>Link Management</strong></td>
<td><strong>Description</strong></td>
</tr>
<tr>
<td>LMP</td>
<td>Control-Channel Management: Established by negotiating link parameters (e.g., frequency in sending keep-alive messages) and ensuring the health of a link (hello protocol)</td>
</tr>
<tr>
<td></td>
<td>Link-Connectivity Verification: Ensures the physical connectivity of the link between the neighboring nodes using a PING–like test message</td>
</tr>
</tbody>
</table>

GMPLS protocols, listed in Table 2.2, offer a set of well-thought-out building blocks for the control plane for transport networks, thus it makes sense to try to apply them to the ASON architecture. Unfortunately today the IETF, ITU-T and the Optical Interworking Fo-
rum (OIF) are all proposing different ways to utilize GMPLS to meet the requirements of the ASON architecture; thus there is not a best solution but only multiple solutions ratified as standards.

### 2.4 Traffic Engineering in IPoWDM Networks

The optical Control Plane, described in the previous paragraphs, makes also available IP over WDM Traffic Engineering (TE) that aims at utilizing IPoWDM resources (for example, IP routers and electrical buffers, WDM switches, fibers and wavelengths) efficiently and effectively, to transport IP flows and packets. IPoWDM traffic engineering includes IP/MPLS traffic engineering and WDM traffic engineering as shown below in Figure 2.3.

![Figure 2.3 - IPoWDM Traffic Engineering (TE)](image)

Traffic Engineering can be performed in two ways, off-line traffic engineering and on-line traffic engineering.

The first deals with an off-line lightpaths set-up and tear-down, in the sense that the virtual topology of the network comes from a graph optimization and changes only according to statistics measurement on the traffic load that are supposed to be known.
The second provide an on-line lightpath creation on demand and aims to achieve load balancing on the network by using heuristics grooming algorithms [2] This work will refer only to the online TE algorithms, as this work focus on dynamic grooming policies.

2.4.1 Multilayer TE with ASON/GMPLS routers (MTE)

With an ASON being the underlying transport infrastructure, the traffic engineering can be executed in a multilayer fashion. The traditional load balancing on the IP/MPLS layer is combined with the on-demand bandwidth provisioning on the ASON layer. This technique is called Multi-Layer Traffic Engineering (MTE).

Multilayer traffic engineering is performed in a distributed way based on GMPLS protocols exchange of messages.

We consider three layers: fiber, lambda and packet. Packet LSPs are accommodated in lightpaths, Lightpaths are accommodated in fibers. The structure of the photonic MPLS router, that is a router including IP and optical layers controlled using GMPLS protocols, is shown in figure 2.4 [7].

![Figure 2.4 - Multilayer Traffic Engineering in Photonic MPLS router](image-url)
It consists of a packet-switching fabric, lambda-switching fabric, and photonic MPLS router manager. In the photonic MPLS router manager, the GMPLS controller distributes its own IP and photonic link states, and collects the link states of other photonic MPLS routers with the routing protocol of Open Shortest Path First (OSPF) extensions. Based on link-state information, PCE (Path Computation Element) finds an appropriate multi-layer route, and the signalling protocol of the Resource Reservation Protocol with traffic engineering (RSVP-TE) extensions module sets up each layer’s LSPs. PCE provides the functions of traffic engineering, including LSP routes and optimal virtual network topology reconfiguration control, and judges whether a new lightpath should be established or not when a packet LSP is requested.

The analysis of the Traffic Engineering in multi-layer networks such as IPoWDM networks requires special focus on the Photonic MPLS router, whose architecture is described more in depth below.

Figure 2.5 shows the model of a Photonic MPLS router, based on the architecture above-mentioned. The IP packet and lambda switching fabrics are connected by internal PSC ports.
The above architecture allows a node to have either or both of the following functionalities:

- **Source/destination point or switching point of an OLSP.** Let us suppose that the node is a switching point of an LSP. The node receives the traffic from an input OLSP, therefore the traffic being in optical form. The light beam is converted into electronic format and given to the IP/MPLS router to execute the label swapping operation. The packets are then reconverted into optical format and placed onto the output OLSP.

- **Switching point of an OLSP.** In this case, the traffic being carried over an optical signal with a certain wavelength enters the node from an input fiber. The node bypasses the IP/MPLS router and performs the label swapping like operations only into the optical domain. More precisely, according to the decision previously taken by the control protocols, the signal is translated to another wavelength and placed onto an output fiber.

When two IP/MPLS routers are connected through an OLSP, they can be virtually considered adjacent, no matter how many optical fibers the OLSP spans.

MTE has a more significant impact when the operator is owner of the integrated model described in Figure 2.5, since the routing algorithm can take into consideration both layers' state in accommodating traffic and consequently improving resource optimization. Figure 2.6, shows the framework of a dynamic multilayer routing. If a new lightpath must be set up to support packet LSP routing, a lightpath setup request is invoked and lightpath routing is performed. The lightpath routing result is returned to the packet LSP routing procedure for confirmation of its acceptability. This process is iterated until the desired result is obtained. If successful, the multilayer routing procedure notifies its acceptance of the packet LSP setup request.
The photonic MPLS router uses the RSVP-TE signalling protocol (resource reservation protocol with traffic engineering) extensions to establish packet LSPs and lightpaths in multi-layer networks. An upper-layer LSP setup request can trigger lower-layer LSP setup if needed. If there is no lower-layer LSP between adjacent nodes a lower-layer LSP is set up before the upper-layer LSP.

2.4.2 On Line Traffic Engineering

Wavelength paths, called Optical Label Switched Paths (OLSP) or lightpaths are set up and released in a distributed manner based on GMPLS protocols. Since the photonic MPLS router has both types of switching capabilities and can handle GMPLS, it enables us to create the optimum network configuration with regard to IP and optical network resources. Multilayer on-line traffic engineering, which yields the dynamic cooperation of IP/MPLS and optical layer is required to provide IP service cost effectively.

Consider the case in which source and destination IP routers request packet-LSPs with specified bandwidth. Packet-LSPs are routed on the optical network as Optical-LSPs. If the specified packet-LSP bandwidth is much smaller than the OLSP bandwidth, the one-hop OLSP between the source and destination IP routers is not fully utilized. In order to better utilize network resources, low-speed packet LSPs should be efficiently merged at some transit nodes into high-speed OLSP. This agglomeration is called traffic grooming.
In this work, it has been considered an on-line approach in which connection requests with different bandwidths arrive randomly. In this way the routes have to establish lightpath in a real-time manner within the limits of the network resources. Since is difficult to predict traffic demands precisely, the on-line approach is realistic and useful in utilizing network resources more fully and maximizing revenue from the given resources.

In this chapter it has been introduced the IPoWDM architecture and the inter-networking model on the base of which this work has been developed. Later, it is shown the ASON official architecture and the GMPLS suite of protocols. Multi-layer traffic engineering is also explained and online traffic policies to set up connections are introduced.
IP networks today carry different types of traffic with different performance requirements. For instance, many ISP’s use IP networks to deliver bundled services that include performance-sensitive applications, such as voice and video, as well as the best effort data services. In order to provide QoS in such multi-service environment, service differentiation is necessary.

Service differentiation is widely available in modern routers, e.g., in the form of scheduling and buffer management mechanisms that resolves the resource contention. In IP networks, this is largely controlled by routing that determines how network resources (link bandwidth) are assigned to carry different traffic flows, in order to ensure the best performance for high-priority traffic, while still maintaining reasonable performance for low-priority traffic.

In a multi-service network, the differentiation of the offered Quality of Service (QoS) is highly desirable because it allows the operator to sell added value services which increase the Return on Investment (ROI).

### 3.1 Service Differentiation: the evolution towards modern scenarios

One of the most deployed technologies over IP networks for QoS is the Differentiated Service (DiffServ). DiffServ operates on the principle of traffic classification, where each data packet is placed into a limited number of traffic classes. Each router on the network is configured to differentiate traffic based on its class. Each traffic class can be managed differently, ensuring preferential treatment for higher-priority traffic on the network. DiffServ-aware routers implement Per-Hop Behaviors (PHBs), which define the packet forwarding properties associated with a class of traffic. Different PHBs may be defined to offer guarantees in terms of, for example, low-loss, low-latency forwarding or best-effort forwarding properties, jitter, etc. All the traffic flowing through a router that belongs to the same class is referred to as a Behavior Aggregate (BA).

In recent years DiffServ has been also integrated with the Multi Protocol Label Switching (MPLS) technology. The MPLS is mostly used to improve the IP layer Traffic Engineering
creating connection oriented Label Switching Paths (LSPs). By Integrating DiffServ over MPLS we can satisfy the two main requirements of QoS:

- End to End network resources allocation (MPLS)
- Different policies for traffic treatments, in terms of delay, loss, jitter etc., depending on its Class of Service (DiffServ)

The way towards the Multi-Layer Traffic Engineering
As already introduced in Chapter 2, the optical technologies on which operators are nowadays largely investing permits to bypass the electronic packet switching at intermediate nodes and improve the communication performances. Hence, the traditional Traffic Engineering schemes deployed combining DiffServ and MPLS technologies need to be extended and integrated with the ASON/GMPLS layer. This new scenario allows the operator to accommodate services either in IP/MPLS domain, or in the optical domain, applying QoS techniques over a multi-layer environment. Both operator and final user can take advantage from integrated Multi-layer Traffic Engineering. The former achieves a better optimization of network resources, the latter gain benefits in terms of percept QoS.

Chapter overview
This chapter presents the basic concepts representing the starting point of this work. First, two of the most used Multi-Layer TE routing policies will be introduced, together with their advantages and drawbacks. Then, the explained policies will be combined together in order to obtain three different Service Differentiation schemes, as defined in [14].

The Service Differentiation algorithms are mainly based on the multi-topology routing (MT Routing) approach, where different routes are computed for different service classes, allowing greater flexibility in exploiting available network resources to meet their service goals.

The MT Routing has been recently proposed (June 2007) in [5], thus:

- Three Service Differentiation schemes exploiting this recent concept have been defined [14]
• We can implement them over an IP/MPLS over ASON/GMPLS network simulator and simulate their behavior
• We can evaluate their contribution to the Service Differentiation supplied by the network.

These points group together the ideas and the concepts from which this work will be developed.

3.2 Multi-Layer Traffic Engineering Policies (MTE)

All the OLSPs created out of a pool of physical resources in the optical network can be considered as a virtual topology decoupled from the physical topology. This allows operators to consolidate high capacity WDM transport with digital bandwidth management to enable reconfigurable service mapping and MTE.

From the operator’s perspective, integrated MTE achieves better optimized resource utilization and increases the network flexibility in accommodating new services. New services can be routed either on the existing virtual topology or on newly established OLSPs. This has also impact on the QoS experienced from the user’s point of view.

The two basic multi-layer routing policies are described in the 2 following sub-sections.

3.2.1 Virtual Topology First (VT-F) Policy

A system implementing the VT-F policy first attempts to aggregate a new service request over the existing virtual topology.

The steps of VT-F are summarized in Figure 3.1.

![VTF Flow Chart](image)

**Figure 3.1 - VTF Flow Chart**

**Step 1.** Check if there is any series of existing lightpaths with available resources that connect source and destination nodes using one or more hops (pref-
ereference is given to the shortest path) If found, go to step 3. Otherwise, go to step 2.

**Step 2.** Check if a new direct lightpath can be set up between the source and destination nodes. If yes, go to step 3. Otherwise, go to step 4.

**Step 3.** Accept the connection request and start its creation with protocol messages’ exchange.

**Step 4.** Reject the connection request.

### 3.2.2 Physical Topology First (PTF) Policy

A system implementing the PTF policy, first attempts to establish a new direct OLSP between source and destination nodes.

The steps of PT-F are summarized in Figure 3.2.

![PTF Flow Chart](image)

**Step 1.** Check if a new direct lightpath can be set up between the source and destination nodes. If yes, go to step 3. Otherwise, go to step 2.

**Step 2.** Check if there is any series of existing lightpaths with available resources that connect source and destination nodes using one or more hops (preference is given to the shortest path) If found, go to step 3. Otherwise, go to step 4.

**Step 3.** Accept the connection request and start its creation with protocol messages’ exchange.

**Step 4.** Reject the connection request.

The main difference between PTF and VTF can be summarized into these two main points:

- **Resource Optimization.** The VTF policy achieves high capacity utilization while saving physical resources as a consequence of a lower number of OLSP estab-
lishments. The PTF policy, instead, implies a high consumption of wavelengths and ports and less optimized capacity optimization, due to less aggregation of sub-wavelength services [14].

- **Offered QoS.** The PTF policy facilitates the traffic transmission over direct OLSPs while VTF aggregates traffic on the existing virtual topology and as a result LSPs are more likely accommodated on the paths containing more than one OLSP. The higher amount of provided capacity implies that a PTF based network is more flexible to react to an increase of the traffic demand and therefore achieves lower blocking probability. Moreover, direct OLSPs avoid the possible bottleneck caused by electronic switching at intermediate routers, thus PTF improves the end-to-end packet delay and packet loss [14].

### 3.3 Service Differentiation policies

Currently one of the main challenges in IP/MPLS over ASON/GMPLS networks is the definition of MTE strategies by which achieve a good trade off between the users’ needs (i.e. QoS) and the operators’ perspective (i.e. resource optimization). Moreover, the differentiation of the offered QoS is highly desirable by operators because of the possibility to sell added value services which increase the Return On Investment (ROI) index [14].

The comparison between the two multilayer routing policies illustrated in the last section highlights that VTF is an optimal policy from the operator's perspective, due to the better resource optimization achieved. On the other hand, PTF is the scheme that better meets the user’s needs reaching better results in terms of QoS.

In [14], different schemes are defined in order to achieve a good trade off between user and operator’s needs. This goal can be reached by means of two different approaches, defined in sections 3.2.2 and 3.2.3 by combining the main characteristics of VTF and PTF policies and based on the transmission of different Class Of Services (CoS) onto different virtual topologies. These two approaches are in opposition with the scheme described in section 3.2.1, that merely differentiates the routing policy on the base of the traffic priority.

In [14], two different CoS are introduced: High Priority (HP) traffic and Low Priority (LP) traffic. For HP traffic we mean traffic needing certain Quality of Service (QoS) guarantees, such as high bandwidth, low end-to-end delay, low delay jitter and minimal losses (e.g,
real time traffic, critical data applications, etc.). Best effort traffic (web surfing, etc.) can be considered as LP traffic, instead.

### 3.3.1 Routing Policy Differentiation (RP-Diff)

The RP-Diff scheme is based on a simple algorithm that decides on the multilayer routing policy according to the CoS the service request belongs to. More precisely, the system accommodates HP services by means of the PTF policy while LP traffic is routed using the VTF policy. Figure 3.3 gives a representation of the RP-Diff behaviour, reporting the scheme algorithm.

![Figure 3.3 – the RP-Diff Algorithm](image)

The aim of the RP-Diff algorithm is to provide HP services with a high level of QoS. This is achieved by using the PTF policy for HP requests, as illustrated in figure above. PTF is the policy resulting in the best performance in terms of blocking probability, end-to-end delay and packet loss as a consequence of the high amount of capacity it provides. However, such high amount of provided capacity makes PTF not efficient with respect to the resource utilization and therefore it decreases the operator’s revenue. This lack of resource optimization can be compensated by the use of the VTF policy when accommodating LP traffic [14].

As we can observe in figure 3.4 the RP-Diff scheme accommodates new connection requests originating a single virtual topology, where network resources are completely shared among different CoSs.
3.3.2 Virtual Topology Hard Differentiated Service (VT-HardDiff)

As explained in last section RP-Diff applies different routing policies for different CoS over the same virtual topology. The VT-HardDiff approach introduces a stronger approach in terms of diversification between HP and LP traffic: it builds several virtual topologies, each of which is dedicated to the transmission of one CoS. Consequently, such different virtual topologies obtained can be considered as independent each other and use different routing policies in order to accommodate traffic without sharing resources.

The VT-HardDiff scheme transmits traffic belonging to different CoS onto different OLSPs. Each newly created OLSP is marked with the identifier of the CoS having triggered the OLSP installation. During its entire lifecycle the OLSP can only be used to aggregate traffic belonging to the CoS that has triggered the lightpath setup. Therefore, the system operates an OLSP differentiation that does not allow services belonging to different CoS to share resources on the virtual topology. Figure 3.5 shows the VT-HardDiff approach.
VT-HardDiff allows an operator to map different CoS onto different virtual topologies originates from the same physical infrastructure. Figure 3.5 highlights how a newly created OLSP is marked as HP-OLSP (HPO) or LP-OLSP (LPO) and will be used to aggregate only HP or LP traffic, respectively.

We obtain two different virtual topologies: the HP Topology (HPT), composed by the set of HPOs, while the LPOs are grouped into the LP Topology (LPT). In figure 3.6 the red arrows in the physical topology form an HPO. That HPO is part of the HPT and considered by the routing procedure only to accommodate HP services. The inverse procedure is adopted for LPOs, which aggregates LP traffic and form the LP Topology (blue arrows).

The high level of QoS for HP traffic are achieved by using the PTF routing policy in HPT, while the use of the VTF policy in the LPT allow the operator to obtain a good resource optimization.
However, in [14], the authors point out how this approach could result sometimes not feasible, reaching the desired QoS differentiation at the expense of some performance degradation of both the Classes of Service. This is the main reason why the VT-SoftDiff scheme is going to be introduced.

3.3.3 Virtual Topology Soft Differentiated Service (VT-SoftDiff)

The VT-SoftDiff scheme is based on the same approach as the VT-HardDiff. However, in the VT-SoftDiff scheme HPT and LPT are allowed to share a limited amount of resources. This is achieved by slightly modifying the VTF policy in the LPT. The flow chart of the VT-SoftDiff scheme is reported below while an example is illustrated in figure 3.8.

![Figure 3.7 - A flow chart representing the SoftDiff policy](image)

Figure 3.7 represents the main modification to the HardDiff scheme. When executing the VTF policy, the LPT is allowed to use the resources allocated to the HPT but only in case of direct OLSp connecting source and destination. Thus, to route an LP LSP from source s to destination d, the virtual topology graph considered also takes into account existing OLSPs directly connecting s and d.

Two main questions could come out observing the behaviour of the SoftDiff scheme.

- Why the SoftDiff algorithm considers HP lightpaths to accommodate LP requests and not vice versa?
- Why the SoftDiff algorithm considers only direct HP lightpaths between source and destination?

The reason of this choice consists on the fact that the VTF routing policy accommodates LP traffic mostly onto multi-OLSP paths. By one hand, using the LP virtual topology to
transmit HP traffic is not feasible because the QoS requirements could not be guaranteed: for instance, the LPT could experience more instability and consequently it can be reconfigured with a higher frequency compared to the HPT. On the other hand it is reasonable to allow only direct HP lightpaths sharing in order to limit the capacity utilization of the HP virtual topology and prevent service degradation.

![Virtual Topologies](image)

**Figure 3.8 – An example of Soft-Diff Virtual Topologies**

**Chapter summary**

This chapter presented the Service Differentiation. Section 3.1 introduced two MTE routing policies: VT-F, which offers the best resource optimization by preferring traffic grooming instead of new OLSP establishment; and PT-F, where a higher level of QoS is guaranteed but implying high consumption of resources.

The second part introduces three different approaches used to implement the Service Differentiation in IP/MPLS over ASON/GMPLS networks, combining the MTE routing policies previously described.

VT-RPDiff which merely uses different routing policies to accommodate different connection requests with different priorities.

VT-HardDiff achieves a *stronger* differentiation of the offered QoS. The differentiation is obtained by accommodating traffic with different priorities onto different virtual topologies independent each other.

VT-SoftDiff introduces a certain degree of resource sharing. It is based on the same vir-
virtual topology differentiation approach as VT-HardDiff but allows the virtual topologies to share some of the OLSPs set up.

In Chapter 3 we introduced the whole set of concepts giving rise to the development of this thesis. They can be summarized as follow:

- We have three Service Differentiation schemes already defined in [14].
- We want to extend with these schemes a previous developed IP/MPLS over ASON/GMPLS network simulator
- We want to evaluate and compare the contribute these schemes give to the overall Service Differentiation supplied by the network

How can we integrate this capability within the simulator environment? Is there any available research on the Service Differentiation protocols implications? Is it reasonable to implement new protocols from scratch or extending some of the existent simulator modules is possible?

The development of the concepts listed above implies some preliminary research: next chapter gives us the answer to these questions.
4

GMPLS Protocols for Service Differentiation
In this chapter we are going to analyze the main protocols adopted to add the Service Differentiation capability to the simulator. Some of them are not available and need to be introduced from scratch; some others are already available within the INET framework and need to be extended to fulfill the GMPLS requirements. Chapter 4 gives some theoretical basics useful to better understand how the protocols involved in Service Differentiation will be implemented.

The whole network has to share the same information about TE properties and virtual Multi-Topologies each link belongs to. In other words, we want each node to spread the information on the network links and to keep it synchronized with all the other nodes. Using a Link State Protocol such as OSPF can be considered as the natural way to face this problem. Moreover, in [6] the OSPF with its Traffic Engineering extensions (OSPF-TE) is proposed as link state protocol within the GMPLS framework.

The first part of this chapter presents a brief overview over the Open Shortest Path First protocol (OSPF), focusing only on procedures and characteristics which are going to be useful for a better understanding over the development of this work. The second part of the chapter deals with the Traffic Engineering and Multi-Topology Routing extensions to OSPF, needed in order to comply with the GMPLS framework and to implement the Traffic Differentiation capability. More detailed information about the OSPF and its protocols can be found on [17]. Each protocol will be analyzed more in depth under the implementation perspective later in chapter 6.

4.1 OSPF Overview

OSPF is a link-state (LS) routing protocol. A link state protocol uses an algorithm in which all the entire network topology and the state of the links are known by each router and shared with the others by means of message flooding. Each node then independently calculates the best next hop for every possible destination in the network. (It does this using only its local copy of the network map, and without communicating in any other way with any other node.) The collection of best next hops forms the routing table for the node. This contrasts with distance-vector routing protocols, which work by having each node sharing its routing table with its neighbors. In a link-state protocol, the only
information passed between the nodes is information used to construct the connectivity maps.

Each router running OSPF maintains a database describing the topology, called Link-State database (LS-database). The local state is then distributed throughout the network by means of flooding. Two routers that have interfaces to a common network are called neighboring routers. Neighboring routers form relationships between them (called adjacencies) in order to exchange routing information.

Each router spread the information about the state of its links by flooding the so called Link-State Advertisement over the whole network. During an OSPF process every router floods LSA’s and uses the LSAs received from other routers to build its LS-database. An example of this process can be seen below: on the left we can see a simple topology; on the right the LS-Database built by router A is represented as matrix.

![Diagram of network topology and LS-Database]

Figure 4.1- An example of LS-Database for node A

When the database is done, it builds a tree of shortest path to other routers, with itself as the root of the tree, using the information taken from the database. The Shortest Path Tree obtained generates the routing table used to forward packets to any destination. When the topology changes in someplace, the routers affected by this change flood the new topology information using LSAs. These advertisements are received by other routers and they recalculate their own LS-databases and flood new LSAs to their neighbors.
4.1.1 Areas
The size of the LS-Database, the routing algorithm complexity and the amount of exchanged messages increase for wide topologies, leading to scalability problems. Hierarchical routing is the answer to this problem: the network is partitioned in “independent sectors” (Areas) connected among them by a “backbone” (Backbone Area).

The backbone Area is the special OSPF Area 0. It contains all ABRs and is responsible for distributing routing information between non-backbone areas.

When routing a packet between two non-backbone areas the backbone is used. The path that the packet will travel can be broken up into three contiguous pieces: an intra-area path from the source to an ABR, a backbone path between the source and destination areas, and another intra-area path to the destination.

4.1.2 Type of Router
OSPF recognizes three types of routers:

*Internal Router*. A router with all directly connected networks belonging to the same area.

*Area Border Router (ABR)*. A router that attaches to multiple areas. These routers run multiple copies of the basic algorithm, one copy for each attached area. *Backbone Routers*: A router that has an interface to the backbone area.

*AS Boundary Router (ASBR)*. A router that exchanges routing information with routers belonging to other Autonomous Systems (AS). A network running its own autonomous routing protocol is called Autonomous System. OSPF specification provides support to the management of multi-AS systems.

An LS-database is built for each different area the router belongs to, using flooding information obtained by listening to other routers. This database will be exactly the same for all internal routers belonging to the same area. Each area border router will have one of this databases for each area to which the router belongs to.
4.1.3 OSPF Functionalities

As previously said, a separate copy of OSPF’s routing algorithm runs in each area. Routers having interfaces to multiple areas run multiple copies of the algorithm.

The routing can be executed in two different ways:

**Intra-area routing:** When a router starts, it first waits for indication from the lower-level protocols that its interfaces are functional. Being the interfaces functional, the router startst to build adjacencies with its neighbors (the Hello protocol, described later in section 4.1.5).

Router’s adjacencies are reflected in its Link-State Advertisements (LSAs). A router periodically advertises its link-state, flooding LSAs throughout the area. Link-state is also advertised when a router’s state changes. Thereby, relationship between adjacencies and link-state allows the protocol to detect dead routers in a timely fashion.

**Inter-area routing:** In order to be able to route to destinations outside of the area, the area border routers (ABRs) inject additional routing information into the area. This information is a distillation of the rest of the AS’s topology. Each ABR is connected to the backbone; each of them summarizes the topology of its attached non-backbone areas for transmission on the backbone and hence to all other ABRs. Each ABR then has a complete topological information concerning the backbone and the area summaries from each of the other ABRs. From this information the router calculates paths to all inter-area destinations. The router then advertises these paths into its attached areas, enabling the internal routers to pick the best exit route when forwarding to inter-area destinations.

This work does not consider multi-area networks; thereby the attention will be focused only to the Intra Area Routing procedures. However, more information is available in [19], where Inter-area routing procedures are described in depth.

4.1.4 OSPF Type of Packets

All OSPF packets share a common 24 byte protocol header. This header contains all the information necessary to determine whether the packet should be accepted for further
processing.

The next figure shows the fields contained in the OSPF header:

```
0   8   16   31
Version  Type  Packet Length
         Router ID
         Area ID
         Checksum  Authentication Type
         Authentication
```

Figure 4.2 - OSPF Packet Header

The OSPF packet types are:

<table>
<thead>
<tr>
<th>Type</th>
<th>Packet name</th>
<th>Protocol function</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Hello</td>
<td>Discover/maintain neighbors</td>
</tr>
<tr>
<td>2</td>
<td>Database Description</td>
<td>Summarize database contents</td>
</tr>
<tr>
<td>3</td>
<td>Link State Request</td>
<td>Database download</td>
</tr>
<tr>
<td>4</td>
<td>Link State Update</td>
<td>Database update</td>
</tr>
<tr>
<td>5</td>
<td>Link State Ack</td>
<td>Flooding acknowledgment</td>
</tr>
</tbody>
</table>

Table 4.1 - OSPF Packet types

OSPF uses Hello packets to discover and maintain neighbor relationships.

Database Description (DDP) and Link State Request packets are used in the forming of adjacencies.

OSPF’s reliable update mechanism is implemented by Link State Update and Link State Ack packets. Each Link State Update packet carries a set of new Link State Advertisements (LSAs) one hop further away from their point of origination.

The different types of OSPF LSAs are listed below:

<table>
<thead>
<tr>
<th>LSA Type</th>
<th>LSA name</th>
<th>LSA description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Router LSAs</td>
<td>Originated by all routers. This LSA describes the collected states of the router’s interfaces to an area. Flooded throughout a single area only.</td>
</tr>
</tbody>
</table>
Table 4.2 - OSPF LSA Types

<table>
<thead>
<tr>
<th>Type</th>
<th>OSPF LSA Types</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>Network LSAs</td>
<td>Originated for broadcast networks by the Designated Router. This LSA contains the list of routers connected to the network. Flooded throughout a single area only.</td>
</tr>
<tr>
<td>3,4</td>
<td>Summary LSAs</td>
<td>Originated by area border routers, and flooded throughout the LSA’s associated area. Each summary-LSA describes a route to a destination outside the area, yet still inside the AS. Type 3 summary-LSAs describe routers to networks. Type 4 summary-LSAs describe routes to AS boundary routers.</td>
</tr>
<tr>
<td>5</td>
<td>AS External LSAs</td>
<td>Originated by AS boundary routers, and flooded throughout the AS. Each AS-external-LSA describes a route to a destination in another Autonomous System. Default routes for the AS can also be described by AS-external-LSAs.</td>
</tr>
</tbody>
</table>

OSPF routing packets (except Hello) are sent only over adjacencies. The IP source address is one end of a router adjacency and the IP destination address can be either the other end of the adjacency or an IP multicast address.

4.1.5 The OSPF Protocols

The Hello Protocol

OSPF creates adjacencies between neighboring routers for the purpose of exchanging routing information.

The Hello Protocol is responsible for establishing and maintaining neighbor relationships. It also ensures that communication between neighbors is bidirectional. Each router advertises itself by periodically multicasting Hello packets. This allows neighbors to be discovered dynamically: bidirectional communication is indicated when the router sees itself listed in the neighbor’s Hello packet. Figure 4.2’s sequence diagram summarizes the Hello Protocol procedure.

If an adjacency is formed, the first step is to synchronize the neighbor’s LS-databases, by the Database Exchange Protocol.

The Database Exchange Protocol

In a link-state routing protocol, it is very important for all router’s LS-databases to stay
synchronized. Each router describes its database by sending a sequence of Database Description (DD) packets to its neighbors. Each DD packet describes a set of LSAs belonging to the router's database. When the neighbor sees an LSA that is more recent than its own database copy, it makes a note that this newer LSA should be requested. This sending and receiving of DD packets is called the "Database Exchange Process", whose procedure is showed by figure 4.2. During this process the two routers form a master/slave relationship: Each DD packet sent by the master is acked by the slave, echoing the sequence number. Each DD packet with the M-bit set indicates that there are more packets to follow. The Database Exchange Process is over when a router has received and sent a DD packet with the M-bit off. During and after the Database Exchange Process, each router has a list of those LSAs for which the neighbor has more up-to-date instances. These LSAs are requested in Link State Request (LSR) packets. When the Database Exchange Process has been completed and all LSRs have been satisfied, the databases are synchronized and the routers are marked as fully adjacent. At this time the adjacency is fully-functional and is advertised in the two router's router-LSAs.

Figure 4.3 - Hello protocol and DBExchange protocol sequence diagram
Over the last several years the OSPF routing protocol has been widely deployed throughout the Internet. However, the original version of the OSPF needs to be extended in order to comply with the evolution of networking technology. Traffic Engineering and Multi-Topology routing are only some examples of the capabilities to develop.

Due to the OSPF fixed packet format, it is not so easy to introduce any backward-compatible extensions. This is the main reason why it is extremely important the Opaque LSA Option, described in the following section.

### 4.2 The Opaque LSA Option

The whole network has to share the same information about TE link properties. In other words, we want each node to spread the information on the network links and to keep it synchronized with all the other nodes. This is exactly what the OSPF protocol does, as explained in last section.

How can we represent and flood TE information throughout the network? RFC 3630 (September 2003) specifies Traffic Engineering extensions to the OSPF protocol. Moreover, in [6], the OSPF with its Traffic Engineering extensions (OSPF-TE) is proposed as link state protocol within the GMPLS framework.

Due to the fixed nature of the OSPF protocol, it is not possible to modify the structure of the original specification [10]. This section describes enhancements to the OSPF protocol to support a new class of link-state advertisements (LSA) called Opaque LSAs.

Opaque LSAs provide a generalized mechanism to allow for the future extensibility of OSPF. The information contained in Opaque LSAs may be used directly by OSPF or indirectly by some application wishing to distribute information throughout the OSPF domains.

#### 4.2.1 The Opaque LSA

Opaque LSAs consist of a standard LSA header followed by a 32-bit aligned application-specific information field.
Like any other LSA, the Opaque LSA is used to flood topology information throughout the network. It has a flooding scope associated with it, depending on the Opaque LSA type. Based on the flooding three types of Opaque LSAs are defined:

- **Link-state type 9** denotes a link-local scope. Type-9 Opaque LSAs are not flooded beyond the local (sub)network.

- **Link-state type 10** denotes an area-local scope. Type-10 Opaque LSAs are not flooded beyond the borders of their associated area.

- **Link-state type 11** denotes that the LSA is flooded throughout the Autonomous System (AS).

### 4.2.2 Flooding Opaque LSAs

The flooding of Opaque LSAs must follow the rules of flooding Scope. We call *opaque-capable* all the routers able manage the information carried by Opaque LSAs. When opaque-capable routers and non-opaque-capable OSPF routers are mixed together in a routing domain, the Opaque LSAs are not flooded to the non-opaque-capable routers.

As a general design principle, optional OSPF advertisements are only flooded to those routers that understand them.

An opaque-capable router learns of its neighbor's opaque capability at the beginning of the "Database Exchange Process" (see Section 4.1.5). A neighbor is opaque-capable if it sets the *O-bit* in the Options field of its Database Description packets. Thereby, Opaque LSAs are included in the Database summary list that is sent to the neighbor only if the neighbor is opaque capable.

![Figure 4.4 - The OSPF Packet Options Field](image)

In case of non-opaque-capable router receiving Opaque LSAs, they will be simply discarded.

The OSPF flooding procedure modifications for Opaque LSAs are exhaustively described in section 3.1 and 3.2 of [10].
4.3 Traffic Engineering Extensions to OSPF (OSPF-TE)

OSPF has been elected for MPLS and GMPLS to build Traffic engineering functions using the OSPF opaque LSA option introduced in [10]. In this section we are going to explain the basics of the Traffic Engineering extensions to OSPF which have been defined by RFC 3630 [4].

Traffic engineering extensions can be used to build an extended link state database, just as router LSAs are used to build a “regular” link state database. The difference is that the extended link state database (Traffic Engineering database) has additional link attributes.

Traffic Engineering Database (let’s call it TED) can be used for:

- Monitoring the extended link attributes
- Local constraint-based source routing
- Global traffic engineering

Local constraint-based source routing means that a router R can compute a path from a source node A to a destination node B. Typically, A is R itself, and B is specified by a “router address”. This path may be subject to various constraints on the attributes of the links and nodes that the path traverses.

For Global traffic engineering, a device can build a Traffic Engineering database, input an optimization function and thus compute optimal or near-optimal routing for the entire network. The device can subsequently monitor the traffic engineering topology and react to changes by re-computing the optimal routes.

4.3.1 Traffic Engineering LSAs (TE-LSA)

TE-related advertisements are encapsulated into OSPF opaque LSAs and presented for distribution to OSPF, which acts as transport mechanism.
One new LSA is defined, the Traffic Engineering LSA (TE LSA). This LSA describes routers, point-to-point links and connections to multi-access networks (it is similar to a Router LSA, see section 4.1).

The TE LSAs are type 10, which has an area flooding scope (see section 4.2). Thus they are flooded only within the LSA originator’s OSPF area, and their payload is delivered to the TE layer of every OSPF-TE speaker within the area. The TE layer is responsible for building and managing the local TED. It also produces network TE graph with data switches represented as graph vertices and synchronized TE links as edges. It is this network graph that is used to compute paths through the network for data services (LSPs and OLSPs).

4.3.2 TE-LSA Header
As we can see in figure 4.5 the Traffic Engineering LSA starts with the standard LSA header.

![Figure 4.5 - The LSA header](image)

Figure above underlines the LSA ID of the Opaque LSA [10], which is defined as having 8 bits of type data and 24 bits of type-specific data. The Traffic Engineering LSA uses type 1.

4.3.3 TE-LSA Payload
The TE LSA payload is structured as a set of Type-Length-Value blocks (TLVs). The format of each TLV is described in figure 4.6
Top-level TLVs may nest other TLVs (sub-TLVs) within themselves. Two types of TE top-level TLVs are currently defined:

1. Router Address
2. TE Link

Only one top-level TLV should be contained in each opaque LSA to facilitate small updates when one of the TLVs changes.

The Router Address TLV is used so that the advertising controller can specify one of its IP addresses that is always routable (for example, a loopback address). The router address TLV is type 1, has a length of 4 octets and a value that is the four octet IP address.

The TE Link TLV is used to advertise attributes of one TE link. Each TE link attribute is encoded as a separate sub-TLV and there are no ordering requirements for the sub-TLVs. The Link TLV is type 2, and the length is variable.

### 4.3.4 Sub-TLVs Details

The data structures used in the ASON/GMPLS layer are slightly different from IP/MPLS layer ones because they deal with a different environment. Thus, it is clear that optical links are represented by a different set of TE parameters from the IP/MPLS links.

- Fiber parameters representation is needed
- Some of the sub-TLVs implemented are common to both the IP/MPLS and ASON/GMPLS layers
- New specific sub-TLVs need to be defined to represent peculiar parameters for optical links and IP/MPLS links.

The most important Sub-TLVs defined for OSPF-TE are described in this section (see [4] for more details on TE link attributes).

**Link Type.** The Link Type sub-TLV defines the type of the link:
Chapter 1 – Introduction

1. Point-to-point
2. Multi-access

**Link ID.** The Link ID sub-TLV identifies the other end of the link. For point-to-point links, this is the Router ID of the neighbor. The Link ID is identical to the contents of the Link ID field in the Router LSA for these link types.

**Local Interface IP Address.** The Local Interface IP Address sub-TLV specifies the IP address of the interface corresponding to this link.

**Remote Interface IP Address.** The Remote Interface IP Address sub-TLV specifies the IP address of the neighbor’s interface corresponding to this link.

**Maximum Bandwidth.** The Maximum Bandwidth sub-TLV specifies the maximum bandwidth that can be used on this link, in this direction.

**Unreserved Bandwidth.** The Unreserved Bandwidth sub-TLV specifies the amount of bandwidth not yet reserved at each of the eight priority levels in IEEE floating point format. The values correspond to the bandwidth that can be reserved with a setup priority of 0 through 7, arranged in increasing order.

The whole set of TLVs represents the TE information stored by each router into the Traffic Engineering Database. This is a very important data structure, involved in both the MTE routing policies and the Service Differentiation schemes processes. Here below we can see a screenshot of a router TE Database. We can easily observe how the database fields represents the same information carried by the sub-TLVs described above.

![Figure 4.7 - An example of TE Database](image-url)
4.4 Multi-Topology Routing in OSPF (OSPF-MT)

In last sections we described the OSPF protocol which will be integrated within the simulation tool and its Traffic Engineering extensions. These represent the starting point of the implementation part of this work, described in Chapter 6.

The actual simulator can implement only the first Service Differentiation scheme described in Chapter 3, the RP-Diff scheme. The Multi-Topology capability is necessary to implement SoftDiff and HardDiff schemes.

We can summarize in 3 main points the Multi-Topology requirements:

- Each link belongs now to one or more virtual topologies. Thus, new information about the topology it belongs has to be advertised over the network.
- The way each router calculates the shortest path to destination has to be modified. Now the process has to take into account new constraints about the topology.

Newly created lightpaths need to be marked with the topology they belong to. Observing the list above, it can be easily deduced that using OSPF to spread the MT information. Nevertheless, we already know that OSPF uses a fixed packet format; therefore it is not easy to introduce any backward-compatible extensions.

Two approaches can be followed to face this problem:

- **The Opaque LSA approach.** A Multi-Topology field can be defined and added to the TE-LSAs, creating a new “MT-subTLV”. This is the same approach followed for OSPF-TE.

- **OSPF packet field redefinition.** It is not possible to modify the original OSPF packets data structure; nevertheless some of the fields originally defined are nowadays deprecated. Can they be redefined?

In June 2007 the RFC 4915 was published in order to specify a way to implement the Multi-Topology capability within the OSPF protocol. This document can be adapted to our purposes, addressing our choices on the second approach listed above.

The original OSPF specification [10] introduced a Type of Service (TOS) metric in order to announce a different link cost based on TOS. TOS-based routing was never deployed
and was subsequently deprecated. The idea is to reuse the TOS-based metric fields, which will be redefined and used to advertise different topologies.

### 4.4.1 Multi-Topology Routing vs. TOS-Based Routing

Multi-Topology (MT) routing offers some particular capabilities which make it different from the traditional TOS-based routing:

- With TOS routing, traffic that is unreachable in the routing table associated with the corresponding TOS will revert to the `default` routing table, which includes the whole network without taking into account any TOS constraint. With Multi-Topology routing, this is optional.

- With TOS routing, individual links or prefixes could not be excluded from a topology: all links or prefixes were either advertised explicitly or defaulted to the TOS 0 metric. With MT routing, links or prefixes that are not advertised for a specific topology do not exist in that topology.

### 4.4.2 Protocol Extensions for Multi-Topology

The OSPF fixed packet format requires the redefinition of the TOS-based metric fields contained into the Link State Advertisement packets. In this section, we are going to explain how to redefine the TOS fields and their new function in implementing the Multi-Topology capability.

![Figure 4.8 - Router LSA packet format](image)

Each link is described by a set of fields like the one represented in figure 4.8.
Each link is typed according to the kind of attached network. It is also labeled with its 
**Link ID**. This Link ID gives a name to the entity that is on the other end of the link. In ad-
ddition, the **Link Data** field is specified for each link. This field gives 32 bits of extra in-
formation for the link. The **#TOS** field defines the number of different Type Of Service con-
cerning a particular link. Finally, the **TOS fields** describe the metric cost of the link for 
each corresponding Type Of Service. If a link is not part of a particular TOS, the corre-
spanding line is omitted from the list.

Figure 4.8 shows the Router LSA packet’s fields to be redefined:

- The TOS field is redefined as **MT-ID** in the payload of Router, Summary, and 
  Type-5 and Type-7 AS-external-LSAs (described in section 4.1). 
  Each MT has its own MT-ID metric field. When a link is not part of a given MT, 
  the corresponding MT-ID metric is excluded from the LSA.
- The TOS metric field is redefined as **MT-ID metric** and reused to advertise topol-
  ogy specific metric for links and prefixes belonging to that topology. MT-ID me-
  trics in LSAs should be in ascending order of MT-ID. If an MT-ID exists in an LSA 
  or router link multiple times, the metric in the first MT-ID instance MUST be 
  used.
- When a router establishes a FULL adjacency over a link that belongs to a set of 
  MTs, it advertises the corresponding cost for each MT-ID.
- By default, all links are included in the default topology and all advertised prefi-
  xes belonging to the default topology will use the TOS 0 metric as in [10].

The Network-LSA does not contain any MT information since the Designated Router (DR) 
is shared by all MTs. Hence, there is no change to the Network-LSA.

**4.4.3 Shortest Path Calculation**

Shortest Path calculation follows almost the same set of procedures already described in 
section 16.1 of [19], with some differences from the original OSPF definition:

- During the Shortest Path Calculation (SPF) for a given MT-ID, only the links and 
  metrics for that MT-ID are considered.
By considering MT-ID metrics in the LSAs, OSPF computes multiple topologies and finds paths to IP prefixes for each MT independently. A separate SPF will be computed for each MT-ID to find independent paths to IP prefixes.

Here again, the MT information needs to be stored by each router within the TE Database. The MT field is involved in the Service Differentiation schemes presented in Chapter 3, as it represents a constraint to the Shortest Path calculation from source to destination. Figure below shows an example of TE Database, where we can find the MT field together with the other TE fields defined in last sections.

![Figure 4.9 - An example of TE Database, with MT field](image)

### 4.4.4 Default Topology Link Exclusion Capability

By default, the Multi-Topologies imply that all the routers participate in the default topology. However, it can be useful to exclude some links from the default topology and reserve them for some specific classes of traffic.

A new parameter is defined: the `DefaultExclusionCapability` parameter. This configurable parameter ensures that all routers in an area have this capability enabled before the default topology can be disabled on a router link in the area without causing backward-compatibility problems.

DefaultExclusionCapability parameter is represented redefining the unused T-bit as the MT-bit in the option field.

![Figure 4.10 - The OSPF Packet options field](image)
Figure 4.10 represents the LSA options field and the MT-bit. If Default Exclusion Capability is enabled, the bit is set in Hello packets and in Database Description packets, in order to ensure that a Multi-Topology link-excluding capable router will only form an adjacency with another similarly configured router.

If DefaultExclusionCapability is disabled:

- The MT-bit must be cleared in Hello packets and Database Description packets.
- If a link participates in a non-default topology, it is automatically included in the default topology to support backward compatibility between MT and non-MT routers.

If DefaultExclusionCapability is enabled:

- The MT-bit MUST be set in Hello packets and SHOULD be set in Database Description packets.
- The router will only accept a Hello packet if the MT-bit is set.

When DefaultExclusionCapability is set to enabled, a router is said to be operating in DefaultExclusionCapability mode.

It is outside of the scope of this work to specify the interoperability between MT-capable routers and non-MT-capable routers, which is fully explained in section 5 and 6 of [5].

This chapter introduced one of the main protocols implied in Service Differentiation which is OSPF, together with the most important protocols involved in the OSPF process. Then, some extension have been defined, following the RFCs specifications over Opaque LSAs, Traffic Engineering and MultiTopology routing [10][4][5].

We could say that, as chapter 3 introduced the Service Differentiation basics, so this chapter introduced the theoretical basics to start our work of implementing the Service Differentiation capability.

Next chapter analyzes the last step before passing to the implementation stage, the simulation environments used in this work that requires some preliminary study before starting with the implementation.
Chapter 5 – Working Environment: ASON/GMPLS Simulation Tool

5

Working Environment:
ASON/GMPLS Simulation Tool
The basic idea of this work can be considered as the third step of a project which is being carried on in recent years by the department of Electronics and Informatics at the Vrije Universiteit Brussel. First, an IP over WDM network simulator was developed. Then, the simulation tool was enhanced to develop a realistic IP/MPLS over ASON/GMPLS system. Our contribution to this process consists on adding the Service Differentiation capability to the simulator. This implies the integration of a realistic link state protocol (OSPF and its TE extension) and the implementation of the Multi-Topology capability. The Service Differentiation schemes described in Chapter 3 can be finally implemented and evaluated according on the simulations results.

This chapter gives an overview of the tools used to implement the service differentiation in IP/MPLS over ASON/GMPLS network. In paragraph 5.1 the OMNeT++ simulation environment is be described. In paragraph 5.2 we show the INET Framework for OMNeT++. Chapter 5.3 gives a detailed description of the simulation package used as starting point for the simulation of this work.

5.1 OMNet++

OMNeT++ is an object-oriented modular discrete event network simulator. The simulator can be used for:

- traffic modeling of telecommunication networks
- protocol modeling
- modeling queuing networks
- modeling multiprocessors and other distributed hardware systems
- validating hardware architectures
- evaluating performance aspects of complex software systems
- modeling any other system where the discrete event approach is suitable

OMNEST is the commercially supported version of OMNeT++. OMNeT++ is only free for academic and non-profit use. For commercial purposes one needs to obtain OMNEST licenses from Omnest Global, Inc [22].
5.1.1 Modelling Concepts

OMNeT++ provides efficient tools for the user to describe the structure of the actual system. Some of the main features are [22]:

- hierarchically nested modules
- modules are instances of module types
- modules communicate with messages through channels
- flexible module parameters
- topology description language

An OMNeT++ model consists of hierarchically nested modules. The depth of module nesting is not limited, which allows the user to reflect the logical structure of the actual system in the model structure. Modules communicate through message passing. Messages can contain arbitrarily complex data structures. Modules can send messages either directly to their destination or along a predefined path, through gates and connections. Modules can have their own parameters. Parameters can be used to customize module behavior and to parameterize the model’s topology.

Modules at the lowest level of the module hierarchy encapsulate behavior. These modules are termed simple modules, and they are programmed in C++ using the simulation library.

OMNeT++ models are often referred to as networks. The top level module is the system module. The system module contains submodules, which can also contain submodules themselves. The depth of module nesting is not limited; this allows the user to reflect the logical structure of the actual system in the model structure.

![Figure 5.1 - OMNet++ simple and compound modules](image-url)
Modules that contain submodules are termed **compound modules**, as opposed to **simple modules** which are at the lowest level of the module hierarchy. Simple modules contain the algorithms in the model. The user implements the simple modules in C++, using the OMNeT++ simulation class library.

Each module has a structure that is described in its OMNeT++’s Network Element Descriptor (NED) file; samples of NED files for modules built for our purpose are shown in chapter 6.

Both simple and compound modules are instances of **module types**. While describing the model, the user defines module types; instances of these module types serve as components for more complex module types. Finally, the user creates the system module as an instance of a previously defined module type; all modules of the network are instantiated as submodules and sub-submodules of the system module.

Modules communicate by exchanging **messages**. In an actual simulation, messages can represent frames or packets in a computer network, jobs or customers in a queuing network or other types of mobile entities. Messages can contain arbitrarily complex data structures. Simple modules can send messages either directly to their destination or along a predefined path, through gates and connections.

The “local simulation time” of a module advances when the module receives a message. The message can arrive from another module or from the same module *(self-messages are used to implement timers)*.

**Gates** are the input and output interfaces of modules; messages are sent out through output gates and arrive through input gates.

Each **connection** (also called link) is created within a single level of the module hierarchy: within a compound module, one can connect the corresponding gates of two submodules, or a gate of one submodule and a gate of the compound module.

Due to the hierarchical structure of the model, messages typically travel through a series of connections, to start and arrive in simple modules. In [22] there are examples of message exchange possibilities.
For each connection can be assigned three parameters, which facilitate the modeling of communication networks, but can be useful in other models too: propagation delay, bit error rate and data rate, all three being optional. One can specify link parameters individually for each connection, or define link types and use them throughout the whole model.

Modules can have parameters. Parameters can be assigned either in the NED files or the configuration file omnetpp.ini. Parameters may be used to customize simple module behaviour, and for parameterizing the model topology.

The simple modules of a model contain algorithms as C++ functions. The full flexibility and power of the programming language can be used, supported by the OMNeT++ simulation class library. The simulation programmer can choose between event-driven and process-style description, and can freely use object-oriented concepts (inheritance, polymorphism etc) and design patterns to extend the functionality of the simulator.

Simulation objects (messages, modules, queues etc.) are represented by C++ classes. They have been designed to work together efficiently, creating a powerful simulation programming framework. The following classes are part of the simulation class library:

- modules, gates, connections etc.
- parameters
- messages
- container classes (e.g. queue, array)
- data collection classes
- statistic and distribution estimation classes
transient detection and result accuracy detection classes

The classes are also specially instrumented, allowing one to traverse objects of a running simulation and display information about them such as name, class name, state variables or contents. This feature has made it possible to create a simulation GUI where all internals of the simulation are visible.

5.1.2 Building and Running simulations

An OMNeT++ model consists of the following parts:

- **NED language topology description(s):** (.ned files) which describe the module structure with parameters, gates etc. NED files can be written using any text editor or the GNED graphical editor.
- **Message definitions:** (.msg files). You can define various message types and add data fields to them. OMNeT++ will translate message definitions into full-fledged C++ classes.
- **Simple modules sources:** They are normal C++ files containing classes that specifies simple modules' behaviour (.h/.cc suffix.)

The simulation system provides the following components:

- **Simulation kernel:** This contains the code that manages the simulation and the simulation class library. It is written in C++, compiled and put together to form a library (a file with .a or .lib extension)
- **User interfaces:** OMNeT++ user interfaces are used in simulation execution, to facilitate debugging, demonstration, or batch execution of simulations. There are several user interfaces, written in C++, compiled and put together into libraries (.a or .lib files)

Simulation programs are built from the above-mentioned components. First, .msg files are translated into C++ code using the opp_msgc. program. Then all C++ sources are compiled, and linked with the simulation kernel and a user interface library to form a simulation executable. NED files can either be also translated into C++ (using nedtool)
and linked in, or loaded dynamically in their original text forms when the simulation program starts.

The simulation executable is a standalone program, thus it can be run on other machines without OMNeT++ or the model files being present. When the program is started, it reads a configuration file (usually called omnetpp.ini). This file contains settings that control how the simulation is executed, values for model parameters, etc.

The configuration file can also prescribe several simulation runs; in the simplest case, they will be executed by the simulation program one after another. Examples of configuration files and models are in [22].

5.1.3 Analyzing Simulation Results

The output of the simulation is written into data files: output vector files, output scalar files, and possibly the user’s own output files. OMNeT++ provides a GUI tool named Plove to view and plot the contents of output vector files. It is not expected that someone will process the result files using OMNeT++ alone: output files are text files in a format which can be read into math packages like Matlab or Octave, or imported into spreadsheets like OpenOffice Calc, Gnumeric or MS Excel. All these external programs provide rich functionality for statistical analysis and visualization, and it is outside the scope of OMNeT++ to duplicate their efforts.

Output scalar files can be visualized using the Scalars tool. It can draw bar charts, x-y plots (e.g. throughput vs. offered load), or export data via the clipboard for more detailed analysis into spreadsheets and other programs.

5.2 The INET Framework

INET Framework is a collection of OMNeT/OMNEST simulation modules implementing different layers of the ISO/OSI protocol stack, from the application layer to the Data Link layer. In figure 5.3 the OSI stack is shown.
Chapter 5 – Working Environment: ASON/GMPLS Simulation Tool

Figure 5.3 - ISO/OSI stack model

INET Framework contains IPv4, IPv6, TCP, UDP protocol implementations, and several application models. The framework also includes an MPLS model with RSVP-TE and LDP signaling. Data link models are PPP, Ethernet and 802.11x. Static routing can be set up using network auto-configurators, or one can use routing protocol implementations.

All the protocols are represented by simple or compound modules described by their NED file and their C++ classes and are immediately usable in every OMNeT++ simulation.

More information about the INET Framework can be found in [23].

5.3 IP/MPLS over ASON/GMPLS Simulation Tool

The simulator extended in this work is part of a previous project and has been implemented by the department of Electronics and Informatics at the Vrije Universiteit Brussel (VUB).

This section describes the simulator architecture [7], focusing on the modules which are mainly involved in our work and that have been modified.

For a deeper description of the Simulator architecture and implementation, see [7].

Each node in the simulator contains:

- An Optical CrossConnect (OXC) with fully convertible wavelength capabilities, this represents the physical layer of the router.
- An ASON agent, which controls and supervises the optical layer and the ports’ connections using GMPLS protocols adapted to the optical environment.
• An IP/MPLS router, connected to the optical layer through a certain number of ports, which can setup and teardown MPLS LSPs over existing Lightpaths cooperating with the ASON control plane.
• Ports, which are the optical-electronic and vice versa transducers that connect the optical and electronic layer of the routers.

Figure 5.4 shows the “external” view of the simulator, as an OMNET++ compound module in which we can see an example of network topology.

The main parameters of the simulation environment are:

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>interArrivalTime</td>
<td>the mean value of the stochastic distribution chosen to model the arrival of connection requests</td>
</tr>
<tr>
<td>holdingTime</td>
<td>the mean value of the stochastic distribution chosen to model the time a connection is active</td>
</tr>
<tr>
<td>numOfCalls</td>
<td>the number of connection requests we want to generate during a single simulation run</td>
</tr>
<tr>
<td>numOfNodes</td>
<td>the number of GMPLS routers contained in the network</td>
</tr>
<tr>
<td>numOfPorts</td>
<td>define the router capabilities in terms of wavelengths and ports as explained afterwards</td>
</tr>
<tr>
<td>numOfWaves</td>
<td>define the router capabilities in terms of wavelengths and ports as explained afterwards</td>
</tr>
<tr>
<td>lightpathCapacity</td>
<td>the capacity of a single lightpath/wavelength in terms of Gbps</td>
</tr>
</tbody>
</table>

All the above parameters can be changed for each simulation run, and their values are defined by the omnetpp.ini file.

All the modules seen in figure 5.4 will be described in the following paragraphs. Also, the NED files describing the module will be shown when containing useful information for a better understanding of the simulator architecture.
5.3.1 Statistics Collector

This simple module stores all the statistical information about the network’s behavior and saves it in a file at the end of the simulation. The saved results can then be analyzed using different tools such as Microsoft Excel®, Plove®, MATLAB® or others. Currently we can measure setup times and blocking probability for different priorities and policies, after the processing of each connection request to the network. StatisticsCollector module is also very easy to extend, if you want to measure other interesting things, by simply adding C++ variables and methods to the class. It supports both OMNeT++ scalar and vectorial outputs.

5.3.2 Connection Generator

This simple module generates connection requests for the network under analysis. Each connection has:

- A random source and a random destination chosen from the routers of the network using a uniform distribution
- A random priority for the connection that can be chosen with different distributions; currently only high priority and low priority are supported
- The bandwidth required to accommodate the connection properly; in our implementation is an integer randomly chosen between 1 and 2 with a 90% probability (45% each one) and 3 with 10% probability (in unity of Gbps)
- The holding time for the connection that is a parameter of the simulation

We will see in the next sections how the connection generator module will be modified and included inside each host module, in order to manage the exchange of IP traffic among hosts.

5.3.3 Network

Each module implements a router in the network under analysis. Each connection between two nodes, like in figure 5.4, represents an optical fiber with a given delay and capacity so that the network models the physical topology of an ASON/GMPLS system.

The network can be configured easily using the NED language from which OMNeT++ build the proper C++ file, enabling us to use different topologies easily.
As explained before, the ASON control plane and the optical data plane run parallel to each other and they are implemented using OMNeT++ channels as shown below.

```
//Channel used to model datarate of a single port
channel Wavelength
  datarate 10e9
endchannel

//Channel used to model optical Data Transmission delay
channel Fiber
  //given a light speed of 3e8 m/s we have 0.33ms delay every 100Km
  delay 0.33e-3
endchannel

//Channel used to model optical Control Transmission delay
channel ControlLink
  delay 0.33e-3
  datarate 10e9
endchannel
```

Parameters such as delay of each fiber or datarate of the control plane can then be easily changed modifying accordingly the NED file.

### 5.3.4 GMPLS Node

Each node of the network implements an ASON/GMPLS router as a compound module containing several simple modules, including INET ones.

In figure 5.5 the architecture of what is inside each ASON/GMPLS router is shown.

![Figure 5.5 – Network node view](image-url)
Each node contains different modules that cooperate to satisfy a connection request, using different policies, arrived from the generator. This architecture has been implemented trying to obtain a result close to the general architecture of an ASON/GMPLS router, as we can see in figure below.

**Figure 5.6 – ASON/GMPLS developed router vs. Theoretical model**

All the modules inside each router of the network are now being analyzed in the following pages.

### 5.3.5 Setup Handler

The SetupHandler is a simple module that receives the connection requests from the generator and tries to establish a connection between source and destination, applying different policies as explained later in chapter 6. Although it doesn’t implement any protocol, it is one of the most important modules, because it is the responsible of the synchronization and cooperation between the electronic IP/MPLS layer and the ASON controlled optical layer. It is also the one in charge for the statistics update, for example after the creation or a refusal of a connection.

### 5.3.6 Physical Layer

Implements an ASON-controlled, fully wavelength convertible, Optical Cross Connect (OXC) capable of setting up Lightpaths on demand using wavelength conversion and connections to the ports between the optical and the electrical layers.
This module is fully controlled by the Control module, explained afterward, that implements the ASON control plane of the router. Each electronic packet generated by the IP/MPLS layer travels through the ports and reaches the optical layer in this module where it is switched properly and delivered towards the destination. At the same way when it arrives at the destination’s physical layer it is converted back to the electronic world and reach the IP/MPLS destination through the proper port.

Two important parameters are defined:

- **numOfWaves**: defines the number of wavelengths wherein each fiber is partitioned
- **numOfPorts**: defines the number of ports between the optical and the electronic IP/MPLS layer

These parameters are very important because they are used to choose the optical capabilities of each router of the network. Thus, the choice of the numOfWaves and numOfPorts values can influence the simulation results. They can be easily configured using the `omnetpp.ini` configuration file.

```
nsf.numOfWaves = 24
nsf.numOfPorts = 36
```

### 5.3.7 Control Layer

This module implements the ASON control plane of the ASON/GMPLS router using modules adapted and derived from the INET Framework together with brand new purpose-built OMNeT++ modules.

The Control block is a compound module containing several specialized submodules that cooperate using both function calls and message exchanges to provide optical connection’s service. In Figure 5.7 the inside view of the Control module is shown and afterward a complete explanation of each module is given.
The typical IP/MPLS architecture arises from the above figure, in which we can easily find the signaling and routing layer, followed by the Network and MPLS layers and finally the data link layer. Figure 5.8 shows the IP/MPLS over ASON router protocol stack.

An exhaustive explanation of all the sub-modules inside the Control module can be found in [7]. In the following pages we will just give a brief description, trying to focus on the characteristics which are important to this work.

**CtrlLinkStateRouting – Link state routing protocol for ASON**

This module implements a very basic routing protocol for ASON and it is a direct evolu-
tion of the LinkState protocol used in the INET Framework for the RSVP Router simulator. As we can see in the following sections this module will be replaced by a more realistic OSPF protocol implementation.

**CtrlRSVP – Resource reSerVation Protocol for ASON**

This module implements the Resource reSerVation Protocol (RSVP) used in ASON control plane to setup, manage and tear down optical connections (Lightpath). It is a simple OMNeT++ module providing GMPLS support from ASON routers, including the ability to setup, manage and tear down optical lightpaths, neighbor discovery and failure detection.

**CtrlTED – Traffic Engineering Database for ASON**

This module implements the Traffic Engineering Database (TED) of the ASON control plane. Despite its name, this module is not merely a container of information for the router but it is used in many situations such as finding available optical routes whereupon set up new lightpaths. All the information inside the TED are updated by the LinkState protocol while the RSVP uses it to check the availability of optical resources when setting up a lightpath to fulfill a connection request.

The CtrlTED stores optical resources information for each link, such as the total number of wavelengths and ports and the available number of them.

In figure 5.9 an example of what is inside the ASON’s TED is given.
As said before for each fiber attached to the router we save in the Traffic Engineering Database the number of ports, wavelengths and the state of the link. All this information is updated by RSVP, shared with the other routers of the network using the LinkState protocol and used to search for available optical paths by the Link Resource Manager module.

**CtrlLibTable**

This module implements the Label Information Base Table (LIB Table) of the ASON control plane, in which the RSVP protocol stores the label operations (swap, pop and push) for all the lightpaths that starts from it, end in it or that passes through it.

The LIB Table module is keep up to date by the RSVP ASON module after the processing of RSVP messages.

Below a snapshot of what is inside the LIB Table is shown in Figure 5.10. Note that all the labels contained in the Control module's LIB Table are representing wavelengths.
class std::vector<struct LIBEntry> {  
  struct LIBEntry lib[] = {  
    inLabel1 interface ppp0 outLabel{POP}
    outInterface{io0} color:0
  
    struct LIBEntry lib[1] = inLabel2 interface any outLabel{PUSH 2}
    outInterface{ppp0} color:0
  
    struct LIBEntry lib[2] = inLabel3 interface ppp1 outLabel{POP}
    outInterface{io0} color:0
  
    struct LIBEntry lib[3] = inLabel4 interface any outLabel{PUSH 1}
    outInterface{ppp2} color:0
  
    struct LIBEntry lib[4] = inLabel5 interface any outLabel{PUSH 4}
    outInterface{ppp2} color:0
  
    struct LIBEntry lib[5] = inLabel6 interface ppp1 outLabel{POP}
    outInterface{io0} color:0
  
    struct LIBEntry lib[6] = inLabel7 interface ppp0 outLabel{SWAP 7}
    outInterface{ppp2} color:0
  
    struct LIBEntry lib[7] = inLabel8 interface ppp2 outLabel{POP}
    outInterface{io0} color:0
  
    struct LIBEntry lib[8] = inLabel9 interface any outLabel{PUSH 12}
    outInterface{ppp1} color:0
  
    struct LIBEntry lib[9] = inLabel10 interface ppp1 outLabel{POP}
    outInterface{io0} color:0
  
    struct LIBEntry lib[10] = inLabel11 interface ppp1 outLabel{POP}
    outInterface{io0} color:0
  
    struct LIBEntry lib[11] = inLabel12 interface any outLabel{PUSH 10}
    outInterface{ppp0} color:0
  
  
  
Figure 5.10 – Example of ASON LIB Table’s content

NetworkLayer
This compound module implements the IPv4 layer of the ASON control router; it is an untouched version of the INET one. There are no changes neither in the C++ classes, nor in the NED files of all its submodules.

RoutingTable
This module implements the IPv4 routing table used by the IP layer to forward packets to the other routers of the ASON control network.

In our system the routing table is created automatically at the beginning of the simulation and it is updated at runtime by the OSPF module, for example after a change in the topology occurs.

NotificationBoard
This module implements the Notification Board of INET Framework and it is unchanged in our system. It works in a publisher-subscriber mode and it is used to broadcast the state change of a module to all the subscribed modules. An example of this is in the Linkstate flooding: when there is a change in TED content, the changed entry index is sent to all the subscribers for the event NB_TED_CHANGED, including Linkstate module.
LinkResourceManager
This is probably the most important module inside the ASON control layer; it implements the Link Resource Manager (LRM) of an ASON network, which is responsible for the allocation and freeing of resources of transport links and also for providing status information about them [2], for instance when it has to find a route for a new lightpath for the Setup module.

5.3.8 Virtual Layer
This module implements the IP/MPLS electronic layer of the ASON/GMPLS router using modules that we adapted and derived from the INET Framework together with brand new purpose-built OMNeT++ modules.

The Virtual block is a compound module containing several specialized submodules that cooperate using both function calls and message exchanges to provide IP/MPLS connection’s service. In Figure 5.11 the inside view of the Virtual module is shown and afterward a complete explanation of each module is given.

Figure 5.11 – Inside view of Virtual module
We can easily observe from Figure 5.11, that the structure of the Virtual module is the same of the Control module. This is due to our code reusing policy that allowed us to implement the two blocks without coding too much.

An exhaustive explanation of all the sub-modules inside the Control module can be found in [7]. In the following pages we will just give a brief description, trying to focus on the characteristics which are important to this work.

**GLinkStateRouting – Link state routing protocol for IP/MPLS**

This module implements the routing protocol for the IP/MPLS layer and it is a direct evolution from the LinkState protocol used in the INET Framework for the RSVP Router simulator. As we can see in the following sections this module will be replaced by a more realistic OSPF protocol implementation.

**GRSVP – Resource reSerVation Protocol for IP/MPLS**

This module implements the Resource reSerVation Protocol (RSVP) used in IP/MPLS layer to setup, manage and tear down Label Switched Paths (LSP) over existing Lightpath allowing connections to groom. It is a simple OMNET++ module providing GMPLS support for IP/MPLS dynamic routers; this includes the ability to setup, manage and tear down LSPs, neighbor discovery and failure detection all adapted to the new dynamic environment.

**GTED – Traffic Engineering Database for IP/MPLS**

This module implements the Traffic Engineering Database (TED) of the IP/MPLS layer. Despite its name, this module is not merely a container of information for the router but it is used in many situations such as finding available lightpaths whereupon set up LSPs. All the information inside the TED are updated by the LinkState protocol while the RSVP uses it to check the availability of electronic resources, bandwidth in our case, when setting up a LSP to fulfill a connection request.

Below an example of what is inside the Virtual layer TED is given in Figure.
As said before for each link, that in the case of IP/MPLS layer is a lightpath, attached to the router we save in the Traffic Engineering Database the bandwidth and the state of the link. All this information is updated by RSVP, shared with the other routers of the network using the LinkState protocol and utilized to search for available lightpaths by the Connection Controller module.

We will see how the GTED module will be slightly modified during our work, in order to store more information over the multi-topology the node belongs to.

**GLibTable**

This module implements the Label Information Base Table (LIB Table) of the IP/MPLS layer, in which the RSVP protocol stores the label operations (swap, pop and push) for all the LSPs that starts from it, end in it or that passes through it.

**Classifier**

This module implements a database called Classifier used by MPLS to sort incoming and outgoing packets as MPLS or normal IP ones. This module is directly derived from the old INET one and it is not used because the original version of the Simulation Tool doesn’t implement MPLS traffic on the Virtual layer yet; only control packets are ex-
changed and they are forwarded using the classic IP routing. Thus, this module will be modified in order to properly implement the MPLS traffic.

**NetworkLayer**
This compound module implements the IPv4 layer of the IP/MPLS Virtual layer; it is an untouched version of the INET one. There are no changes in the C++ classes nor in the NED files of all its submodules.

**RoutingTable**
This module implements the IPv4 routing table used by the IP layer to forward packets to the other routers of the IP/MPLS virtual network. Like the NetworkLayer the Routing-Table module used in our work is an untouched version of the INET’s one except for its initialization: the routing table is created automatically at the beginning of the simulation and it is updated at runtime by the LinkState module, for example after a change in the topology occurs.

**NotificationBoard**
This module implements the Notification Board of INET Framework and it is unchanged in our system. It works in a publisher-subscriber mode and it is used to broadcast the state change of a module to all the subscribed modules. An example of this is in the Linkstate flooding: when there is a change in TED content, the changed entry index is sent to all the subscribers for the event NB_TED_CHANGED, including Linkstate module.

**ConnectionController**
This is probably the most important module inside the IP/MPLS Virtual layer; it implements the Connection Controller (CC) of an ASON network, which is responsible for the management of connection’s setup, release and modification using Link Resource Manager, Setup and Virtual layer’s modules.

The rest of this thesis elaborates on the design of the ASON/GMPLS simulator extension for Service Differentiation and its integration with the INET Framework. Special emphasis will be put on describing the modules and the protocols mainly involved during our implementation.
Chapter 5 – Working Environment: ASON/GMPLS Simulation Tool
Chapter 6 – Simulation Extensions

6

Simulator Extensions
6.1 Chapter overview and Goal of this work

In the previous sections the state of the art and the theoretical basics of this project have been explained in depth. This is the starting point of our work:

- An ASON/GMPLS network simulator, with MTE capability
- The PTF Routing policy, which maximizes the physical topology usage
- The VTF Routing policy, which maximizes the virtual topology usage

In [14], authors define three Service Differentiation schemes, using the routing policies mentioned above and defining two different CoS: High Priority (HP) and Low Priority (LP). Aim of this work is to implement and evaluate the behavior of these schemes, obtained following two main approaches [14]:

- Different routing policies for different CoS (RP-Diff)
- Different virtual topologies for different CoS

The former use different routing policies over the same virtual topology, the latter creates different virtual topologies for each CoS defined over the network. Moreover, the topologies created can be either completely independent from each other (this is the case of the Hard-Diff strategy), or can share parts of the network resources (this is the case of the Soft-Diff strategy).

This section describes how the simulation tool has been extended in order to:

- Improve the GMPLS control plane of the IP/MPLS over ASON/GMPLS network simulator. (OSPF-TE implementation)
- Implement the MT-capability in order to introduce the Service Differentiation in IP/MPLS over ASON/GMPLS networks (OSPF-MT implementation)
- Analyze the performance provided by the Service Differentiation policies designed in [14].
6.2 Simulator Extensions

6.2.1 Link State Module

The simulation tool, following the Augmented Model schema, presents a Link State module within its Control Layer and Virtual layer, as previously explained in section 5.3. These modules implement the routing protocol for the IP/MPLS and ASON layers; they are a direct evolution from a link state routing protocol used in the INET Framework for the RSVP Router simulator.

The link state module implements a minimalist version of a real link state protocol as the one described in section 4. Result of this solution is a very simple implementation, easy to understand, whose main advantage is speeding up the simulation time. On the other hand, there are also some important drawbacks.

- the link state protocol just collects and floods information over the entire network topology
- Neighbors, areas and AS management is not implemented
- In order to make our simulator as compliant as possible to the GMPLS framework requirements, a more complex and complete version of the link state protocol should be implemented
- A more realistic link state protocol permits to reach better simulation results in terms of information accuracy.

6.2.2 Implementation Steps

For the reasons listed above the Simulation tool will be extended and partially modified following three main steps:

- OSPF Implementation
- TE Extension for OSPF
- MT Capability for OSPF

The following sections describe each of these steps in depth and the modules directly or partially involved in their realization, using class diagrams and UML diagrams and focusing on the implementation details from the C++ point of view.
6.3 OSPF Integration

Part of this work consists on the implementation of the OSPF protocol that will be used by the simulation tool. The simulator’s link state module is a direct evolution from the OSPFv2 module already available from the INET Framework.

As we can see in the following sections the OSPF integration in the simulation environment is not complicated; nevertheless the main issues we had to face in this first implementation step concern on interfacing the new OSPF module to the simulator environment, as we can see in figure 6.1.

6.1a – internal view of the Control Layer

As highlighted in figures above, each layer of a single router has its own routing module, following the Augmented model schema described in the previous chapters. Thus, the Control and Virtual layer NED files include the CtrlLinkStateModule and the GLinkStateModule, respectively (an example is given below).

```plaintext
module GMPLS_Control
parameters:
  .
submodules:
  .
  linkStateRouting: CtrlLinkStateRouting;
```

```plaintext
module GMPLS_LSR
parameters:
```

6.1b – internal view of the Virtual Layer
Chapter 6 – Simulator Extensions

The integration of the OSPF inside the simulator architecture does not imply particular differences between the Control and the Virtual layer from the C++ classes point of view. Thus, next section describes the main OSPF classes without taking into account the prefixes used during the implementation (\textit{G} in case of Virtual Layer and \textit{Ctrl} in case of Control Layer).

6.4 OSPF C++ Classes

Giving an exhaustive explanation of all the classes belonging to the OSPF implementation is not easy due to their strict relations, for this reason they are now being illustrated divided into groups based on their primary function.

6.4.1 Interface classes

An OSPF interface is the connection between a router and a network. Each router advises the state of its interfaces and their neighbors by router LSAs (see chapter 4).

Each interface is defined by a set of items collected altogether into the \textit{interface data structure}. Below there is a list of the main items.

<table>
<thead>
<tr>
<th>Interface Type</th>
<th>The OSPF interface type is either point-to-point, broadcast, NBMA, Point-to-MultiPoint or virtual link.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface State</td>
<td>The functional level of the interface.</td>
</tr>
<tr>
<td>Interface IP Address</td>
<td>The IP address associated with the interface. This appears as the IP source address in all routing protocol packets originated over this interface.</td>
</tr>
<tr>
<td>Interface IP Mask</td>
<td>Also referred to as the subnet mask, this indicates the portion of the IP interface address that identifies the attached network. Masking the IP interface address with the IP interface mask yields the IP network number of the attached network.</td>
</tr>
<tr>
<td>Area ID</td>
<td>The Area ID of the area to which the attached</td>
</tr>
</tbody>
</table>
network belongs. All routing protocol packets originating from the interface are labelled with this Area ID.

**Hello Interval**
The length of time, in seconds, between the Hello packets that the router sends on the interface.

**Router Dead Interval**
The number of seconds before the router's neighbors will declare it down, when they stop hearing the router's Hello Packets.

**Hello Timer**
An interval timer that causes the interface to send a Hello packet. This timer fires every HelloInterval seconds.

**Neighbor routers list**
The other routers attached to this network. This list is formed by the Hello Protocol.

### Interface states
The various states that router interfaces may attain are now being listed in order of progressing functionality. For example, the inoperative state is listed first, followed by a list of intermediate states before the final, fully functional state is achieved. Below, the inheritance diagram the interface state classes.

![Figure 6.2 - hierarchy diagram of the InterfaceState classes](image)

#### InterfaceStateDown
This is the initial interface state. In this state, the lower-level protocols have indicated that the interface is unusable. No protocol traffic at all will be sent or received on such a interface.

#### InterfaceStatePoint2Point
In this state, the interface is operational, and connects either to a physical point-to-point network or to a virtual link. Upon entering this state, the router attempts to form an ad-
Chapter 1 - Introduction

Adjacency with the neighboring router. Hello Packets are sent to the neighbor every HelloInterval seconds.
For more information over the other interface states, refer to [19].

6.4.2 IMessageHandler classes

The Message Handler and its subclasses manage all the OSPF messages the router receives. Depending on the message received, different actions can be summoned. Below, the inheritance diagram of the IMessageHandler class.

![Inheritance Diagram](image)

HelloHandler

Each router advertises itself by periodically multicasting Hello packets, which are handled by the HelloHandler class. This allows neighbors to be discovered dynamically: bidirectional communication is indicated when the router sees itself listed in the neighbor’s Hello packet.

Below, the UML diagram of the main methods of the class HelloHandler.

```
+GHelloHandler(in containingRouter : Router*)
+updateLSAs(in neighbor : Neighbor*)
```

![UML Diagram](image)

DatabaseDescriptionHandler

If an adjacency is formed, the first step is to synchronize the neighbor’s LS-databases, by the Database Exchange Protocol.

In a link-state routing protocol, it is very important for all router’s LS-databases to stay
synchronized. The DatabaseDescriptionHandler performs this function together with the neighbor classes, described later in this chapter.

Figure 6.3 shows the DatabaseDescriptionHandler UML class diagram.

Each router describes its database by sending a sequence of Database Description (DD) packets to its neighbors. Each DD packet describes a set of LSAs belonging to the router's database. This sending and receiving of DD packets is called the "Database Exchange Process", defined in section 4.5.1 and whose procedure is showed in figure 6.4.

+GDatabaseDescriptionHandle(in containingRouter : Router*)

**Figure 6.5 - The DataBase Description Handler UML Class diagram**

**Figure 6.6 - The Hello process and the Database Exchange process.**

**LinkStateRequestHandler**

During the Database Exchange Procedure, neighbors synchronize their databases ex-
Changing link state messages. When a router finds a new LSA description on a received DD packet a new Link State Request in sent to the neighbor, who manages it through the LinkStateRequestHandlerClass.

<table>
<thead>
<tr>
<th>GLinkStateRequestHandler</th>
</tr>
</thead>
<tbody>
<tr>
<td>+GLinkStateRequestHandler(in containingRouter : Router*)</td>
</tr>
</tbody>
</table>

**Figure 6.7 - The LinkStateRequestHandler UML Class Diagram**

The LinkStateRequestHandler receives the message and check the list of LSA requested by the neighbor. Then, a link state update message is created and sent back.

**LinkStateUpdateHandler**

When a link state update message is received, the linkStateUpdateHandler processes the new LSAs and update its database. System calls are largely used in order to communicate with the other router’s modules. The router’s TED is then updated and the knowledge learnt about the topology is flooded out.

Figure 6.8 shows up the class attributes and the methods involved in the database updating process.

<table>
<thead>
<tr>
<th>GLinkStateUpdateHandler</th>
</tr>
</thead>
<tbody>
<tr>
<td>+validateLSChecksum(in isa : GOSPFLSA*) : bool</td>
</tr>
<tr>
<td>+acknowledgeLSA(in isaHeader : GOSPFLSAHeader &amp; , in intf : Interface*, in acknowledgementFlags : AcknowledgementFlags, in isSource : Router*)</td>
</tr>
<tr>
<td>+GLinkStateUpdateHandler(in containingRouter : Router*)</td>
</tr>
</tbody>
</table>

**Figure 6.8 - LinkStateUpdateHandler UML Class**

**6.4.3 Neighbor classes**

The neighbor management schema is similar to the interface one described in section 6.4.1. A neighbor data structure is maintained by the OSPF module, which also make use of a the neighbor state machine to rule the actual state of its neighbors.
The neighbor data structure contains all information pertinent to the forming or formed adjacency between two neighbors.

<table>
<thead>
<tr>
<th>State</th>
<th>The functional level of the neighbor conversation.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Inactivity Timer</td>
<td>A single shot timer whose firing indicates that no Hello Packet has been seen from this neighbor recently. The length of the timer is RouterDeadInterval seconds.</td>
</tr>
<tr>
<td>Master/Slave</td>
<td>When the two neighbors are exchanging databases, they form a master/slave relationship. The master sends the first Database Description Packet, and is the only part that is allowed to retransmit. The slave can only respond to the master's Database Description Packets. The master/slave relationship is negotiated in state ExStart.</td>
</tr>
<tr>
<td>DD Sequence Number</td>
<td>The DD Sequence number of the Database Description packet that is currently being sent to the neighbor.</td>
</tr>
<tr>
<td>Neighbor ID</td>
<td>The OSPF Router ID of the neighboring router. The Neighbor ID is learned when Hello packets are received from the neighbor</td>
</tr>
<tr>
<td>Neighbor IP Address</td>
<td>The IP address of the neighboring router's interface to the attached network. Used as the Destination IP address when protocol packets are sent as unicasts along this adjacency.</td>
</tr>
<tr>
<td>Database summary list</td>
<td>The complete list of LSAs that make up the area link-state database, at the moment the neighbor goes into Database Exchange state. This list is sent to the neighbor in Database Description packets.</td>
</tr>
<tr>
<td>Link state request List</td>
<td>The list of LSAs that need to be received from this neighbor in order to synchronize the two neighbors' link-state databases. This list is created as Database Description packets are received, and is then sent to the neighbor in Link State Request packets. The list is depleted as appropriate Link State Update packets are received.</td>
</tr>
</tbody>
</table>

**Neighbor States**

Below, the hierarchy diagram of neighbor states is listed in order of progressive functionality. For example, the inoperative state is listed first, followed by a list of intermediate states before the final, fully functional state is achieved.
We are going now to describe briefly each neighbor state. For a more detailed description of neighbor state changes, together with the additional actions involved in each change, see Section 10.3 of [19].

**Down State**
This is the initial state of a neighbor conversation. It indicates that there has been no recent information received from the neighbor.

**Init State**
In this state, a Hello packet has recently been seen from the neighbor. However, bidirectional communication has not yet been established with the neighbor (i.e., the router itself did not appear in the neighbor’s Hello packet). All neighbors in this state (or higher) are listed in the Hello packets sent from the associated interface.

**2-Way State**
In this state, communication between the two routers is bidirectional. This has been assured by the operation of the Hello Protocol.

**Exchange Start State**
This is the first step in creating an adjacency between the two neighboring routers. The goal of this step is to decide which router is the master, and to decide upon the initial
DD sequence number. Neighbor conversations in this state or greater are called adjacencies.

**Exchange State**
In this state the router is describing its entire link state database by sending Database Description packets to the neighbor. Each Database Description Packet has a DD sequence number, and is explicitly acknowledged.

**Loading**
In this state, Link State Request packets are sent to the neighbor asking for the more recent LSAs that have been discovered (but not yet received) in the Exchange state.

**Full State**
In this state, the neighboring routers are fully adjacent. These adjacencies will now appear in router-LSAs and network-LSAs.

Here below, a simple flow chart describing how the neighbor state machine evolves.

![Figure 6.10 – Neighbor state machine diagram](image)

6.4.4 **OSPF LSA Classes**
Each router in the Autonomous System originates one or more link state advertisements (LSAs). This section defines five distinct types of LSAs, which are described below. The collection of LSAs forms the link-state database. Each separate type of LSA has a separate function. Router-LSAs and network-LSAs describe how an area's routers and net-
works are interconnected. Summary-LSAs provide a way of condensing an area's routing information. AS-external-LSAs provide a way of transparently advertising externally derived routing information throughout the Autonomous System.

As the context of this work is a single area Network, a general description of all the LSA types is being given, focusing only on the Router-LSA Type.

**RouterLSA**
Originated by all routers. This LSA describes the collected states of the router's interfaces to an area. Flooded throughout a single area only.

**NetworkLSA**
Originated for broadcast and NBMA networks by the Designated Router. This LSA contains the list of routers connected to the network. Flooded throughout a single area only.

**SummaryLSA**
Originated by area border routers, and flooded throughout the LSA’s associated area. Each summary-LSA describes a route to a destination outside the area, yet still inside the AS (i.e., an inter-area route).

**ASExternalLSA**
Originated by AS boundary routers, and flooded throughout the AS. Each AS-external-LSA describes a route to a destination in another Autonomous System. Default routes for the AS can also be described by AS-external-LSAs.
More information over Network, Summary and ASExternal LSAs are given in depth in [19]. The rest of this section focuses on the RouterLSA class, useful to better understand the Traffic Engineering and MultiTopology extensions. Figure 6.12 shows part of the Router-LSA packet format.

A router originates a router-LSA for each area that it belongs to. Such an LSA describes the collected states of the router’s links to the area. The LSA is flooded throughout the particular area, and no further.

As each OSPF packet, the first 20 bytes of the RouterLSA consist of the generic LSA header, not represented in this case.

The router-LSA describes the router's working connections (i.e., interfaces or links) to the area; each link is described by a set of fields like the one represented in figure 6.12.

- Each link is typed according to the kind of attached network. Each link is also labelled with its Link ID. This Link ID gives a name to the entity that is on the other end of the link.
- In addition, the Link Data field is specified for each link. This field gives 32 bits of extra information for the link.
- The #TOS field defines the number of different Type Of Service concerning a particular link
- The TOS fields describe the metric cost of the link for each corresponding Type Of Service. If a link is not part of a particular TOS, the corresponding line is omitted from the list.

Figure 6.12 – an example of the RouterLSA packet
A general summary of the OSPF process between the routers in the network simulated can be useful before describing the following steps of this work: OSPF-TE and OSPF-MT. As previously mentioned, the first main stage in the algorithm is to give a map of the network to every node and to store it in its database. This is done with several simple subsidiary steps [7].

**Determining the neighbors of each node**

First, each node needs to determine what other ports it is connected to, over fully-working links; it does this by the Hello Protocol process. After this, the neighbor state reach the value of **EX-Start**, so the neighbors can start the exchange process.

**Synchronizing the topology databases**

Once a new adjacency is established, by the DataBase Exchange Process two neighbors keep their topology database synchronized. Then, the neighbors are fully adjacent and the state machine value is set to **Full**.

**Distributing the information for the map**

Next, each node periodically makes up a set of short messages, the link-state advertisements described in section 6.4.4, which:

- Identify the node which is producing it.
- Identify all the other nodes to which it is directly connected.
- Include a sequence number, which increases every time the source node makes up a new version of the message. This is implemented using an OMNeT++ timestamp.
- Include information about the state of each link of interest.

This messages are then flooded throughout the network. As a necessary precursor, each node in the network remembers, for every other node in the network, the sequence number of the last link-state message which it received from that node.

Starting with the node which originally produced the message, it sends a copy to all of its neighbors. When a link-state advertisement is received at a node, the node looks up the sequence number it has stored for the source of that link-state message. If this message is newer (i.e. has a higher sequence number), it is saved, and a copy is sent in turn to each of that node's neighbors. This procedure rapidly gets a copy of the latest version of
each node's link-state advertisement to every node in the network. A scheme of message flooding over a simple network is shown in Figure 6.13.

![Figure 6.13 - The Flooding Procedure](image)

The messages are exchanged in three cases:

- During the neighbor discovery process, by the DatabaseExchange Protocol
- When a change in the state of the links or the topology is detected (i.e., every time the TED changes an event of the type NB_TE_CHANGED is raised by the NotificationBoard and the OSPFRouting module, that is subscribed to receive notifications for this kind of event, floods a new LinkState Advertisement to all the peers.
- In order to keep the adjacencies alive and the databases up to date, a set of timers are set up and trigger the exchange of Link State Advertisement packets among routers.

**Creating the map**

Finally, with the complete set of link-state advertisements (one from each node in the network) in hand, it is obviously easy to produce the graph for the map of the network. The OSPF LSAs database is kept up to date and the TED is built. The LSA database and the TED store in different ways the same kind of data. So they could be considered (and they are, indeed) as a redundancy within the system. However, this choice was the less
expensive in terms of coding during the integration of the OSPF module into the simulator.

The second main stage in the link-state algorithm is to produce routing tables and stores them in RoutingTable modules, by inspecting its LSA database. This is again done with several steps.

Calculating the shortest paths
Each node independently runs an algorithm over the map to determine the shortest path from itself to every other node in the network; this is achieved by the OSPFRouting module using Dijkstra’s algorithm for weighted shortest paths on all the link state entries.

Basically, a node maintains two data structures: a tree containing nodes which are "done", and a list of candidates. The algorithm starts with both structures empty; it then adds to the first one the node itself. The algorithm then repetitively:

• Adds to the second (candidate) list all nodes which are connected to the node just added to the tree (excepting of course any nodes which are already in either the tree or the candidate list).

• Of the nodes in the candidate list, moves to the tree (attaching it to the appropriate neighbor node already there) the one which is the closest to any of the nodes already in the tree.

• Repeat as long as there are any nodes left in the candidate list. (When there are none, all the nodes in the network will have been added to the tree.)

This procedure ends with the tree containing all the nodes in the network, with the node on which the algorithm is running as the root of the tree. The shortest path from that node to any other node is indicated by the list of nodes one traverses to get from the root of the tree, to the desired node in the tree.

Filling the routing table
With the shortest paths in hand, filling in the routing table is again obviously easy. For any given destination node, the best next hop for that destination is the node which is
the first step from the root node, down the branch in the shortest-path tree which leads
toward the desired destination node. To create the routing table, it is only necessary to
walk the tree, remembering the identity of the node at the head of each branch, and fill-
ing in the routing table entry for each node one comes across with that identity.

6.5 OSPF-TE

The Traffic Engineering Extension to OSPF has to deal with the fixed and not flexible na-
ture of the OSPF standard. How can a router flood TE data in the same way the link state
information is advertised by LSAs? The answer can be found in RFC 2370, which defines
the OSPF Opaque LSA Option [10].

The Opaque LSA option defines enhancements to the OSPF protocol to support a new
class of LSA; consist of a standard LSA header followed by application-specific informa-
tion. In this case, Opaque LSAs can be used to represent Traffic Engineering information.

As already described in section 4, Traffic Engineering extensions are implemented fol-
lowing the RFC 3630 standard [4]. Three types of Opaque LSAs exist, each of which has a
different flooding scope. This work uses only Type 10 LSAs, which have an area flooding
scope.

A new LSA is implemented, in order to represent Traffic Engineering data: the TE-LSA.
The Traffic Engineering LSA starts with the standard LSA header, followed by a payload
consisting of one or more nested Type/Length/Value (TLV) triplets for extensibility. Each
link connecting the router to the network is so described by a specific Link TLV and its
sub-TLVs, identified by type. For more information over the TE-LSA header and others
types of TLVs, refer to section 4.

6.5.1 Link TLVs

The Link TLV describes a single link and it is constructed of a set of sub-TLVs, listed with-
out following any particular order.

The data structures used by protocols contained in the Virtual layer are slightly different
from Control layer ones because they deal with a different environment. Thus, it is clear
that optical links are represented by a different set of TE parameters from the IP/MPLS links.

- Fiber parameters representation is needed
- Some of the sub-TLVs implemented are common to both the IP/MPLS and ASON/GMPLS layers
- New specific sub-TLVs need to be defined to represent peculiar parameters for optical links and IP/MPLS links.

Firstly, common sub-TLVs are being defined. Each sub-TLV is defined by Type, Length and Value fields.

**Link Type Sub-TLV**

The Link Type sub-TLV defines the type of the link:

1 - Point-to-point
2 - Multi-access

The Link Type sub-TLV is TLV type 1, and is one octet in length.

<table>
<thead>
<tr>
<th>Type = 1</th>
<th>Length = 1 octect</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Defines the type of the link (PointToPoint, MultiAccess..)</td>
</tr>
</tbody>
</table>

The simulation environment presents mostly Point To Point links.

**Link ID Sub-TLV**

The Link ID sub-TLV identifies the other end of the link. For point-to-point links, this is the Router ID of the neighbor. For multi-access links, this is the interface address of the designated router. The Link ID is identical to the contents of the Link ID field in the Router LSA for these link types.

The Link ID sub-TLV is TLV type 2, and is four octets in length.
Local Interface IP Address sub-TLV

The Local Interface IP Address sub-TLV specifies the IP address(es) of the interface corresponding to this link. If there are multiple local addresses on the link, they are all listed in this sub-TLV. The Local Interface IP Address sub-TLV is TLV type 3, and is 4N octets in length, where N is the number of local addresses.

Remote Interface IP Address sub-TLV

The Remote Interface IP Address sub-TLV specifies the IP address(es) of the neighbor’s interface corresponding to this link. This and the local address are used to discern multiple parallel links between systems. The Remote Interface IP Address sub-TLV is TLV type 4, and is 4N octets in length, where N is the number of neighbor addresses.

The following group list the sub-TLVs defined appositely for the IP/MPLS environment. In this case, they represent information about the maximum and the available bandwidth that can be used on each link.
**Maximum Bandwidth sub-TLV**

The Maximum Bandwidth sub-TLV specifies the maximum bandwidth that can be used on this link, in this direction (from the system originating the LSA to its neighbor), in IEEE floating point format. The maximum bandwidth value is expressed in Gbps.

The Maximum Bandwidth sub-TLV is TLV type 6, and is four octets in length.

<table>
<thead>
<tr>
<th>Type = 6</th>
<th>Length = 4 octets</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Specifies the maximum bandwidth that can be used on this link, from the system originating the LSA to its neighbor</td>
</tr>
</tbody>
</table>

**Unreserved Bandwidth sub-TLV**

The Unreserved Bandwidth sub-TLV specifies the amount of bandwidth not yet reserved at each of the eight priority levels in IEEE floating point format. The values correspond to the bandwidth that can be reserved with a setup priority of 0 through 7, arranged in increasing order with priority 0 occurring at the start of the sub-TLV, and priority 7 at the end of the sub-TLV. The initial values (before any bandwidth is reserved) are all set to the Maximum Reservable Bandwidth. Each value will be less than or equal to the Maximum Reservable Bandwidth. As in the previous case, the units are expressed in Gbps.

<table>
<thead>
<tr>
<th>Type = 8</th>
<th>Length = 8 x 4 octets</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Correspond to the bandwidth that can be reserved with a setup priority of 0 through 7</td>
</tr>
</tbody>
</table>

Actually, the capability to reserve a different amount of bandwidth on the base of the setup priority is not implemented; so by default the priority 7 is used for each link.

Finally, the last group of newly created sub-TLVs is presented below. In the case of the Control Layer, sub-TLVs have to deal to the optical environment, representing information about ports and wavelengths.
**Number of Ports sub-TLV**

The number of Ports sub-TLV represents in a Photonic MPLS router the number of internal PSC ports connecting IP packet and lambda switching fabrics.

<table>
<thead>
<tr>
<th>Type = 10</th>
<th>Length = 1 octet</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>Define the number of ports</td>
<td></td>
</tr>
</tbody>
</table>

Number of Ports sub-TLV is type 10 and is 1 octet in length.

**Number of Wavelengths sub-TLV**

The number of wavelengths sub-TLV specifies the maximum amount of the fiber wavelengths. The number of wavelengths sub-TLV value is expressed in natural numbers, is type 11 and 1 octet in length.

<table>
<thead>
<tr>
<th>Type = 11</th>
<th>Length = 1 octet</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>Define the number of different fiber’s wavelengths</td>
<td></td>
</tr>
</tbody>
</table>

**Available Wavelengths sub-TLV**

The value of the available wavelengths sub-TLV corresponds to the amount of wavelengths still not reserved to set up a new lightpath. Its value is expressed in natural numbers, is type 12 and 1 octet in length.

<table>
<thead>
<tr>
<th>Type = 12</th>
<th>Length = 1 octet</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>Define the number of available fiber’s wavelengths</td>
<td></td>
</tr>
</tbody>
</table>
6.5.2 OSPF-TE Implementation

The most important modifications to the original OSPF module are now being presented under the C++ perspective. The following sections present the UML Diagrams with the most interesting attributes and methods included within the newly created C++ classes.

For a better understanding, classes’ description is ordered following the main steps of the OSPF process; first, the OSPFPacket .msg file has to be modified, then the message handling process needs to be extended to the new defined TE-LSA. Finally, the TE-LSA management itself has to be implemented, including the process of message exchanging and synchronization between peers.

6.5.3 The OSPFPacket class

Referring to [10] and [4], the OSPFPacket .msg file needs to be extended, including the Opaque LSA capability and defining the TE-LSA Type/Length/Value data structure.

The following lines show part of the OSPFPacket description file, highlighting the newly implemented capabilities.

```cpp
class GOSPFLSAHeader extends cObject
{
    fields:
        unsigned short lsAge = 0;
        GOSFPOptions lsOptions;
        char lsType enum (LSAType) = RouterLSAType;
        unsigned long linkStateID;
        IPAddress advertisingRouter;
        long lsSequenceNumber = 0;
        short lsChecksum = 0;
}
```
unsigned short length = 0;

..
..

// OSPF-TE Extension: Opaque LSA Type 10 Definition
//
enum TLVType {
    TLVRouterAddressType = 1; // not implemented
    TLVLinkType = 2;
}

enum subTLVType {
    LinkTypeType = 1;
    LinkIDType = 2;
    LocalInterfaceIPType = 3;
    RemoteInterfaceIPType = 4;
    TrafficEngineeringMetricType = 5;
    MaximumBWType = 6;
    MaximumReservableBWType = 7;
    UnreservedBWType = 8;
    AdminGroupType = 9;
    NumPortsType = 10;
    NumWavesType = 11;
    AvailableWavesType = 12;
};

//
// classes GTLVData and GTLVData_avBW SUB_TLV - represents a sub-TLV
// in case of GLink TLV
//
class GTLVData {
    fields:
        unsigned char type enum (subTLVType) = LinkTypeType;
        unsigned char length;
        double value[];
};

//
// class GTLVLink extends cObject - represents the payload of a OSPF-TE Opaque LSA (Only Type2 TLV: GLink)
//
class GTLVLink {
    fields:
        unsigned char type enum (TLVType) = TLVLinkType;
First, the Opaque LSA type 10 capability has been added, followed by the GOSPFTE_LSA implementation. The new link state advertisement extends the generic GOSPFLSA definition, from which derives the LSA header. The payload follows the Type/Length/Value schema and is defined by the GTLVLink class.

All the Traffic Engineering information describing interfaces, bandwidths, ports and wavelengths are finally defined by the GTLV class and the subTLVType enum statement.

Note that the given code represents the GOSPF TE-LSA implementation. The CtrlOSPF TE-LSA definition is almost the same, so there is no need to examine it separately.
### 6.5.4 OSPFNeighbor class

As previously explained, the OSPFNeighbor classes group is important to maintain the adjacency between peers and manage the neighbor state machine values.

```c
struct OSPFNeighbor {
    NeighborState *state;
    NeighborState *previousState;
    GOSPFTimer *ddRetransmissionTimer;
    GOSPFTimer *updateRetransmissionTimer;
    bool updateRetransmissionTimerActive;
    GOSPFTimer *requestRetransmissionTimerActive;
    DatabaseExchangeRelationshipType databaseExchangeRelationship;
    unsigned long ddSequenceNumber;
    RouterID neighborID;
    bool opaqueLSACapability;
    unsigned char neighborPriority;
    IPv4Address neighborIPAddress;
    GOSPFOptions neighborOptions;
    DesignatedRouterID neighborsDesignatedRouter;
    DesignatedRouterID neighborsBackupDesignatedRouter;
    bool designatedRoutersSetUp;
    short neighborsRouterDeadInterval;
    list databaseSummaryList;
    list linkStateRequestList;
    ...
    ...
};
```

Both the GOSPFNeighbor and CtrlOSPFNeighbor classes (the latter is not shown as its UML diagram is close to the former) have been extended to the Opaque LSA capability. The **OpaqueLSACapability attribute** and its corresponding getter and setter methods are defined in order to guarantee some compatibility between Opaque LSA capable and not-capable routers.

#### CreateDatabaseSummary

During the DataBaseExchange Process, two peers exchange their LSA Database Summary lists. This operation is performed by the **CreateDatabaseSummary method**. The OSPF...
module creates an LSA Summary list, including RouterLSAs and the corresponding TELSAs.

Once received and elaborated the summary list, a router sends a request to synchronize its database with the peer. The LSA request list is created and sent by the `AddToRequestList` and `SendLinkStateRequestPacket` methods.

### 6.5.5 OSPFMessageHandler classes

All the routers involved to the OSPF process keep their topology database by sending and receiving Link State Advertisement. Therefore, it is clear that the OSPF module message handler classes need to be extended to manage TE information. In particular, this section focus on two of the five classes described in section 6.4.2: `LinkStateRequestHandler` and `LinkStateUpdateHandler`.

**LinkStateRequestHandler class**

Once received and elaborated the summary list, each router sends a Link State Request message to the peer.

The `LinkStateRequestHandler` class (UML class diagram is shown in figure 6.5) manages this kind of message by the `ProcessPacket` method. First, the neighbor state machine value and the LSA request list are checked in order to avoid errors. Then, a Link State Update message is created, containing the requested LSAs.

**LinkStateUpdateHandler class**

As in the previous case, the `LinkStateUpdateHandler` class manages the received update messages by the `ProcessPacket method`, which processes this kind of message in case of

- adjacency initialization
- update received, due to a network topology change

First, the neighbor state machine value and the received LSA list format are checked, in order to avoid errors. Then, for each LSA received:
• if not already stored in the local database, the update is flooded out to the other peers and installed in the database.
• If already present in the local database, the received LSA timestamp is compared to the local one: if newer, the update is flooded out to the other peers and installed in the database.
• Otherwise, the current LSA is ignored.

The process of installing new TE-LSAs is important to synchronize the local LSA database with the local TED, as we are going to see in the next section.

A more detailed description of the update message management can be found in section 13 of [19].

6.5.6 Synchronizing TED and TE-LSA Database

TED and OSPF TE-LSA database store, in different ways, almost the same kind of data. The reason of this redundancy is mainly due to the choice of integrating the OSPF module within the simulator system, without spending too much time in coding and dedicating most of the effort in the TE and MT capabilities implementation.

Synchronization between TED and TE-LSA DB is bidirectional and is triggered by two main events.

• A new connection has been set up. TED is changed and TE-LSA database needs to be updated (here again, we are generally talking about TED instead of CtrlTED and GTED).
• A new link state update message has been received and installed in the TE-LSA database. Consequently, TED also needs to be updated.
Figure 6.15 and 6.16 show part of the attributes and methods belonging to the OSPFArea and OSPFRouter classes.

<table>
<thead>
<tr>
<th>Area</th>
</tr>
</thead>
<tbody>
<tr>
<td>- arealID : AreaID</td>
</tr>
<tr>
<td>- advertiseAddressRanges : map&lt;IPv4AddressRange, bool, IPv4AddressRange_Less&gt;</td>
</tr>
<tr>
<td>- associatedInterfaces : vector&lt;Interface*&gt;</td>
</tr>
<tr>
<td>- routerLSAs : vector&lt;GRouterLSA*&gt;</td>
</tr>
<tr>
<td>- teLSAs : vector&lt;GTE_LSA*&gt;</td>
</tr>
<tr>
<td>- networkLSAs : vector&lt;GNetworkLSA*&gt;</td>
</tr>
<tr>
<td>- summaryLSAs : vector&lt;GSummaryLSA*&gt;</td>
</tr>
<tr>
<td>- spfTreeRoot : GRouterLSA *</td>
</tr>
</tbody>
</table>

```
+Area(in id : AreaID = BackboneAreaID)
+GetRouterLSA(in i : unsigned long) : GRouterLSA *
+GetTeLSA(in i : unsigned long) : const GTE_LSA *
+AddInterface(in intf : Interface*)
+GetInterface(in address : IPv4Address) : Interface *
+InstallRouterLSA(in lsa : GOSPFRouterLSA*) : bool
+InstallTeLSA(in lsa : GOSPFTE_LSA*, in lsaRouterID : IPAddress) : bool
+FindTeLSA(in linkStateID : LinkStateID) : GTE_LSA *
+FindTeLSA(in linkStateID : LinkStateID) : const GTE_LSA *
+findSubTLV(in t : subTLVType, in lsa : GTE_LSA*, in bwVec[] : unsigned long, in localIP : IPAddress) : bool
+AgeDatabase()
+FloodLSA(in lsa : GOSPFLSA*, in intf : Interface* = 0, in neighbor : Neighbor* = 0) : bool
+OriginateRouterLSA() : GRouterLSA *
+OriginateTeLSA(in neighbor : Neighbor*) : GTE_LSA *
+updateRouterLSA(in entry : GTELinkStateInfo) : GRouterLSA *
+deleteLSAs(in routerID : IPAddress)
+CalculateShortestPathTree(in newRoutingTable : vector<RoutingTableEntry*> &)
+...() |
```

**Figure 6.15 – OSPFArea UML Class diagram**

From the example above arise some of the attributes and methods implemented for the OSPF-TE.

- The **teLSAs vector** attribute, is the TE-LSA database.
- Once error checking test is passed and the state machine value is correctly verified, the TE-LSA update message processing involves the **FindTeLSA, GetTeLSA, findSubTLV** methods.
- If the received LSA is newer than the local one (if exists), finally the update is installed by **InstallTeLSA** and flooded out by **FloodLSA**.
The InstallTeLSA method synchronizes the TED with the new TE-LSA database information by the SyncGTED system call, defined in figure 6.16.

<table>
<thead>
<tr>
<th>Router</th>
</tr>
</thead>
<tbody>
<tr>
<td>+routerID : RouterID</td>
</tr>
<tr>
<td>+areasByID : map</td>
</tr>
<tr>
<td>+areas : vector&lt;Area*&gt;</td>
</tr>
<tr>
<td>+ageTimer : OSPFTimer *</td>
</tr>
<tr>
<td>+routingTable : vector&lt;RoutingTableEntry*&gt;</td>
</tr>
<tr>
<td>+messageHandler : GMMessageHandler *</td>
</tr>
<tr>
<td>+opaqueLSACapability : bool</td>
</tr>
<tr>
<td>+routingMod : cSimpleModule *</td>
</tr>
<tr>
<td>+...</td>
</tr>
<tr>
<td>+Router(id : RouterID in containingModule : cSimpleModule*)</td>
</tr>
<tr>
<td>+GetRouterID(id : RouterID)</td>
</tr>
<tr>
<td>+GetRouterID() : RouterID</td>
</tr>
<tr>
<td>+GetOpaqueLSACapability(capability : bool)</td>
</tr>
<tr>
<td>+GetOpaqueLSACapability() : bool</td>
</tr>
<tr>
<td>+GetMessageHandler() : GMMessageHandler *</td>
</tr>
<tr>
<td>+AddRoutingTableEntry(in entry : RoutingTableEntry*)</td>
</tr>
<tr>
<td>+SyncTeDB(in d : TEDChangelinfo*)</td>
</tr>
<tr>
<td>+InstallLSA(in ls : GOSPFLSA*, in neighbor : IPAdress, in areaID : AreaID, in backboneAreaID : AreaID) : bool</td>
</tr>
<tr>
<td>+FindLSA(in ls : LSA*, in neighbor : IPAdress, in areaID : AreaID) : bool</td>
</tr>
<tr>
<td>+RebuildRoutingTable()</td>
</tr>
<tr>
<td>+SyncGTED(in ls : GOSPFLSA*, in backboneAreaID = backboneAreaID, in intf : InterfaceID) : bool</td>
</tr>
<tr>
<td>+SyncGTED(in ls : GOSPFLSA*, in intf : InterfaceID) : bool</td>
</tr>
<tr>
<td>+ProcessTeDBEntry(in newEntry : GTELinkStateInfo) : bool</td>
</tr>
<tr>
<td>+...()</td>
</tr>
<tr>
<td>+...()</td>
</tr>
</tbody>
</table>

**Figure 6.16 – Part of the GOSPFRouter UML Class Diagram**

Figure above highlights many important elements involved in the OSPF process.

As the GOSPFRouter class is directly correlated to the OMNET++ OSPF module, we can see how several attributes and methods are implemented to manage the Area data structure, the routing table and the message handling. As usual, the OpaqueLSACapability flag is set during the OSPF module initialization.

As the syncGTED module is used to synchronize the TED with the TE-LSA DB, so the TE-LSA DB is synchronized with the TED by the syncTeDB method. It is important to keep both the data structures perfectly synchronized, because the TED is used during the connection setup to calculate the best path to destination; on the other hand, TED information is advertised by TE-LSAs flooding to keep the topology knowledge synchronized among all the routers.

Here below, we can see the difference between GTED and CtrlTED data. Moreover, it is possible to see the fields representing all the sub-TLVs defined before.
6.6 OSPF-MT

Until now, the Service Differentiation has been implemented changing the way the system try to accommodate new connection requests, depending on the request priority. In order to implement the Virtual Topology Differentiation (VT-Diff) schemes described in chapter 3, we need to implement the Multi-Topology capability. This section explains how the Multi-Topology capability has been developed and the main C++ classes which have been renewed or newly implemented.

As described in section 3 and 4.3, the OSPF-MT implementation follows the RFC 4915 [5] approach and can be divided in 4 main steps:
• Each link belongs now to one or more virtual topologies. Thus, a new field about the topology will be added to the flooded Link State Advertisement.
• The topology field can be added either to TE-LSAs (following the Opaque LSA approach) or redefining some OSPF packet fields defined in earlier specifications and nowadays deprecated.
• The way the Virtual TED (GTED) calculates the shortest path to destination has to be modified. Now the process has to take into account new constraints about the topology.
• Every newly created lightpath needs to be marked with the topology they belong to. As we already know, we choose topology IDs “1” and “2” for HP requests and LP requests, respectively.

6.6.1 The RFC 4915 approach

As we know OSPF uses a fixed packet format. It is not easy to introduce any backward-compatible extensions. Thus, the MT capability can be implemented following two alternative approaches:

• **Opaque LSA approach.** It is the approach previously chose for OSPF-TE. A new sub-TLV can be implemented and added to the TE-LSAs, describing the virtual topologies a link belongs to.
• **Packet fields reusing approach.** The original OSPF specification [19] introduced *Type of Service* (TOS) metric in order to announce a different link cost based on TOS. However TOS-based routing was never deployed and was subsequently deprecated, so we can reuse the TOS-based metric fields.

The second approach is the one suggested by the RFC 4915, published in June 2007, whose specifications will be followed to implement the Multi-Topology capability for OSPF.

*We propose to reuse the TOS-based metric fields. They have been redefined and are used to advertise different topologies by advertising separate metrics for each of them* [5].

Figures below show how the TOS fields are being redefined and reused in Router Link State Advertisement.
Figure 6.18a – Part of the RouterLSA as originally defined by the OSPF specification.

Figure 6.18a shows the data structure used to describe a single link of the list contained inside the RouterLSAs flooded out by each OSPF router, as originally described by the OSPF standard. Deprecated fields are marked with a red cross:

- The #TOS field defines the number of different Type Of Service concerning a particular link.
- TOS fields describe the metric cost of the link for each corresponding Type Of Service. If a link is not part of a particular TOS, the corresponding line is omitted from the list.

The fields described above are going to be redefined as follows:

Figure 6.18b – Part of the RouterLSA a redefined by RFC 4915
The TOS metric field is reused to advertise topology specific metric for links and prefixes belonging to that topology. The TOS field is redefined as MT-ID in the payload of Router, Summary, and Type-5 and Type-7 AS-external-LSAs.

When a router establishes a full adjacency over a link that belongs to a set of MTs, it advertises the corresponding cost for each MT-ID. By default, all links are included in the default topology and all advertised prefixes belonging to the default topology will use the TOS 0 metric.

Each MT has its own MT-ID metric field. When a link is not part of a given MT, the corresponding MT-ID metric is excluded from the LSA. Here below, part of the modified OSPFPacket.msg file.

```plaintext
enum MT_Type{
    DefaultTopology = 0; //(default priority)
    LPTopology = 1;
    HPTopology = 2;
};

struct GMTData {
    fields:
        int            mt;
        unsigned int   mtMetric[3];
};
```

// Represents an OSPF Router LSA
//
class GOSPFRouterLSA extends GOSPFLSA
{
6.6.2 Connection Request accommodation

New connection requests are managed and accommodated by the **Setup module**. The message handling has been completely renewed from the original one, choosing the routing policy to apply on the base of the TE strategy configured.

The following figure shows the new version of the Setup class which has been implemented.

![SetupHandler UML Class Diagram](image)

**Figure 6.19 – SetupHandler UML Class Diagram**

As we can see in figure 6.19 SetupHandler is one of the most important modules. The C++ class implements the Service Differentiation, managing new connection requests and applying the service differentiation policies depending on the request priority.
New connection requests are accommodated by two main methods:

- In case of RP-Diff strategy, by the `defaultStrategy` method.
- In case of Soft-Diff and Hard-Diff strategies, by the `serviceDiff_Strategy` method.

`PolicyIpLayerFirst` and `policyOpticalLayerFirst` implement the VT-F and PT-F policies, respectively. In case a new lightpath needs to be created, the `setupOlsp` method is summoned, while the `setupLsp` method is called in case of grooming over an existing lightpath. The establishment of a new connection between source and destination is almost the same as the original simulator’s one. However, as we have seen, the way the `setupHandler` manages new requests has been changed, depending now on the request priority.

### 6.6.3 Calculating the Shortest Path

When trying to accommodate a new connection request, the GTED has the main role in calculating the best path to destination, using the `CalculateShortestPath` method. Some modifications have been applied, in order to properly implement the TE strategies defined previously in this work. For a better understand on how the Service Differentiation algorithms have been implemented, both Soft-Diff and Hard-Diff strategies are represented below by their algorithm in natural language and in C++ source code.

---

**Figure 6.20a — The Soft-Diff algorithm in natural words.**

```plaintext
L_{IP} <- 0
L_{OLP} <- 0
L_{OLP}(s,d) <- 0
For each new connection request R_i
    If (R_i is HP Request) then
        Route <- PTF (L_{OLP})
        If (route <> 0) then
            If (route is NEW LIGHTPATH) then
                New_lp <- route
                L_{OLP} <- L_{OLP} U {new_lp}
                L_{OLP}(s,d) <- L_{OLP}(s,d) U {new_lp}
                Accept R_i
            Else
                Reject R_i
        Else
            Route <- VTF (L_{IP} U L_{OLP}(s,d))
            If (route <> 0) then
                If (route is NEW LIGHTPATH) then
                    New_lp <- route
                    L_{IP} <- L_{IP} U {new_lp}
                    Accept R_i
                Else
                    Reject R_i
            End if
        End if
    Else
        Route <- VTF (L_{IP} U L_{OLP}(s,d))
        If (route <> 0) then
            If (route is NEW LIGHTPATH) then
                New_lp <- route
                L_{IP} <- L_{IP} U {new_lp}
                Accept R_i
            Else
                Reject R_i
        End if
    End if
End for
```

---

**Figure 6.20b — The Hard-Diff algorithm in natural words.**

```plaintext
L_{IP} <- 0
L_{OLP} <- 0
L_{OLP}(s,d) <- 0
For each new connection request R_i
    If (R_i is HP Request) then
        Route <- PTF (L_{OLP})
        If (route <> 0) then
            If (route is NEW LIGHTPATH) then
                New_lp <- route
                L_{OLP} <- L_{OLP} U {new_lp}
                Accept R_i
            Else
                Reject R_i
        Else
            Route <- VTF (L_{IP} U L_{OLP}(s,d))
            If (route <> 0) then
                If (route is NEW LIGHTPATH) then
                    New_lp <- route
                    L_{IP} <- L_{IP} U {new_lp}
                    Accept R_i
                Else
                    Reject R_i
            End if
        End if
    Else
        Route <- VTF (L_{IP} U L_{OLP}(s,d))
        If (route <> 0) then
            If (route is NEW LIGHTPATH) then
                New_lp <- route
                L_{IP} <- L_{IP} U {new_lp}
                Accept R_i
            Else
                Reject R_i
        End if
    End if
End for
```
// select edges that have enough bandwidth left, and store them into edges[].
// meanwhile, collect vertices in vectices[].
if (teStrategy == 0) //Default Strategy RP-Diff
{
    for (unsigned int i = 0; i < topology.size(); i++)
    {
        if(!topology[i].state)
            continue;
        if(topology[i].UnResvBandwidth[7] < req_bandwidth)
            continue;
        edge_t edge;
        edge.src = assignIndex(vertices, topology[i].advrouter);
        edge.dest = assignIndex(vertices, topology[i].linkid);
        edge.metric = topology[i].metric;
        edges.push_back(edge);
    }
}
else if (teStrategy == 1) //HardDiff strategy
{
    for (unsigned int i = 0; i < topology.size(); i++)
    {
        if(!topology[i].state)
            continue;
        if(topology[i].UnResvBandwidth[7] < req_bandwidth)
            continue;

        if (mt == HPTopology) //so discard LP links
            continue;

        else if (mt == LPTopology) //so considers only LP links and direct HP links
        {
            if (topology[i].mt != 0){
                if (topology[i].mt != 2){
                    continue;
                }
            }
            else //MT info still not received about this link: discard
                continue;
        }
        else //MT info still not received about this link: discard
            continue;
    }
}
edge_t edge;
edge.src = assignIndex(vertices, topology[i].advrouter);
edge.dest = assignIndex(vertices, topology[i].linkid);
edge.metric = topology[i].metric;
edges.push_back(edge);
}
}
else //SoftDiff Strategy
{
    for (unsigned int i = 0; i < topology.size(); i++)
    {
        if(!topology[i].state)
            continue;
        if(topology[i].UnResvBandwidth[7] < req_bandwidth)
            continue;
        if (mt == HPTopology) //so discard LP links
        {
            if (topology[i].mt != 0)
            {
                if (topology[i].mt != 2)
                {
                    continue;
                }
            }
            else{
                if (topology[i].linkid.getInt () != IPAddress(127, 0, 0, 1).getInt ())
                    //MT info still not received about this link: discard
                    continue;
            }
        }
        else if (mt == LPTopology) //so considers only LP links and
direct HP links
        {
            if (topology[i].mt != 0)
            {
                if (topology[i].mt == 2)
                {
                    if (topology[i].advrouter != this->routerId ||
topology[i].linkid != destID.getInt ()){
                        continue;
                    }
                }
            }
            else{
                if (topology[i].linkid.getInt () != IPAddress(127, 0, 0, 1).getInt ()){
                    //MT info still not received about this link: discard
                    continue;
                }
            }
        }
        else{
        }
6.6.4 Virtual TED updating

Once a new connection between source and destination nodes is correctly established, the virtual topology change needs to be advertised over the whole network. This is achieved in our simulation by filling the MT-fields on RouterLSAs as defined before and flooding the new information over the network (see the `UpdateRouterLSATopology` method listed among the OSPFArea methods).

When a new Link State Update is received (see the `LinkStateUpdateHandler` implementation previously explained), the Virtual TED is consequently modified updating the `mt field` corresponding to the newly established link. Here below an example of the GTED content for a better understanding.
Figure 6.21 – An example of the GTED content

From figure above arise the information stored in GTED to implement the MT capability. The newly MT fields added to TED entries are very important during the path computation, because only properly synchronized information about the virtual topologies permits the system to calculate the correct path between source and destination and guarantee the Service Differentiation.
Chapter 7 – Simulation Study

Simulation Study
In this chapter several experiments will be executed using the extended ASON/GMPLS simulator described in Chapter 5 and Chapter 6. This simulation study has been executed to analyze the behavior, in terms of service differentiation, of the three MTE policies introduced in chapters 3 and 4. Each MTE policy will be analyzed evaluating its performances under two main perspectives:

- Service Differentiation in terms of Blocking Probability
- Service Differentiation in terms of traffic delay

Section 7.1 presents the network topology used for our experiments and the common simulation settings to both the perspectives listed above. Then, in section 7.2 presents the traffic model and some further implementation issues over the traffic simulation. Finally, section 7.3 evaluates the simulation results in terms of blocking probability and packet end-to-end delay. Results are discussed in each section together with the parameters used to achieve them.

7.1 System model and simulation settings

In this section the system model implemented with our simulator and used for all the simulations is introduced. In the first paragraph the network topology chosen for the simulations is described. In the second paragraph we present the traffic model used by the simulator, in terms of inter-arrival time and service time. In the third and last paragraph of this section, the value of the parameters in the configuration files of our simulator are shown and described.

7.1.1 NetworkTopology

To evaluate the performance of the proposed ASON/GMPLS simulator an extensive set of experiments has been executed. Some simulation tests have been firstly performed considering a small network, in order to test mostly the correct behaviour of the implementation.

The topology used in our test simulations is the “Test” network. It consists of 7 nodes and 11 bidirectional optical links and it is shown in Figure 7.7.1.
The second topology, used to run our definitive simulations and evaluate the MTE policies’ performances is the (New York – San Francisco) NSF WDM-Network. NSF (National Science Foundation) is the US Research and Education Network (NREN). Although there are several representations of the NFS net, here we consider the one described in [27] (shown in the figure below), composed by 14 network nodes connected through 21 bidirectional optical links.
Next sections will afterwards present the experiments’ output only using the NSF topology.

### 7.1.2 Traffic Model

The Generator module generates connection requests between a random source and destination, according to a Poisson stochastic process. The average inter-arrival time of connection requests is determined by the parameter $T_{ia}$. The parameter $T_{hold}$ represents the average holding time of each accepted request. Both inter-arrival time and holding time follow a negative exponential distribution with mean $T_{ia}$ and $T_{hold}$. If we call $\lambda = \frac{1}{T_{ia}}$ the inter-arrival frequency for each node, and $\mu = \frac{1}{T_{hold}}$ the frequency of service, the load for each node (expressed in Erlang) can be determined by the following equation:

$$Offered\ load = \frac{\lambda}{\mu} \cdot \frac{B_a}{C} = \frac{T_{hold}}{T_{ia}} \cdot \frac{B_a}{C}$$

Where $C$ is the capacity of a single lightpath and $B_a$ is the average bandwidth requested by the connections. Thus $\frac{B_a}{C}$ represents the fraction of lightpath capacity requested by each connection.

To increase the network load, we increase the $T_{hold}$ parameter while we fix $T_{ia}$ to a constant value.

Settings above are common to both the simulation scenarios listed at the beginning of this chapter. However, some other settings are peculiar to the second set of simulations, whose results are analyzed to evaluate the network from the traffic exchange performances point of view.

In literature, it has been clearly demonstrated as the Internet traffic is better modelled by Pareto or exponentially distributed statistics. Moreover, in principle the multiplexing of several sources of Pareto traffic generates self-similar or long-range-dependent (LRD) network traffic.
For this reason, we are going to investigate the performances of the simulated MTE policies using a heavy-tailed truncated Pareto distribution, which is considered by many researchers to be a good model for burst traffic in real networks. [12] [13].

In our model, packets arrive in bursts (ON periods), which are separated by idle periods (OFF periods). To generate a Pareto-distributed sequence of ON periods, one can generate a Pareto-distributed sequence of burst (packet train) sizes, followed by Pareto-distributed idle periods. The minimum burst size is 1, corresponding to a single packet arrival. The formula to generate a Pareto distribution is

$$X_{Pareto} = \frac{b}{x^a}$$

where $x$ is a uniformly distributed value in the range $(0, 1]$, $b$ is the minimum nonzero value of $XPARETO$, denoted $b_{on}$ and $b_{off}$ for the packet train and the idle period, respectively, and $a$ is the tail index or shape parameter of the Pareto distribution.

However, computer simulations using the above formula generate a truncated Pareto distribution because of the discreet $x$ value. In contrast, any true Pareto distribution of sufficiently great length will have values that exceed the range generated by computer simulations.

The model above is the one adopted in our simulation. Nevertheless, in reality, traffic in the core network depends on many externalities and is still an open research issue.
7.1.3 Simulation Settings

Table 7.1 shows the common parameters used for all the experiments.

<table>
<thead>
<tr>
<th></th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Total number of requests</td>
<td>50000</td>
</tr>
<tr>
<td>Lightpath capacity (C)</td>
<td>15 Gbps</td>
</tr>
<tr>
<td>Average connection bandwidth</td>
<td></td>
</tr>
<tr>
<td>requested (B&lt;sub&gt;a&lt;/sub&gt;)</td>
<td></td>
</tr>
<tr>
<td>Distribution</td>
<td></td>
</tr>
<tr>
<td>2 Gbps</td>
<td></td>
</tr>
<tr>
<td>33.3%</td>
<td>1 Gbps</td>
</tr>
<tr>
<td>33.3%</td>
<td>2 Gbps</td>
</tr>
<tr>
<td>33.3%</td>
<td>3 Gbps</td>
</tr>
<tr>
<td>Fiber propagation delay</td>
<td>0.33ms</td>
</tr>
<tr>
<td>corresponding to 100 Km for</td>
<td>each optical</td>
</tr>
<tr>
<td>each connection request</td>
<td>link</td>
</tr>
<tr>
<td>IP processing delay</td>
<td>10μs</td>
</tr>
</tbody>
</table>

The duration of each simulation run is determined by the parameter callCounter described in chapter 5. The number of connection requests, in Table 7.1, has been chosen as a compromise between the stability of the results in terms of convergence of the blocking probability and the duration of each simulation time. The simulation terminates when all the requests have been processed. The duration of each simulation run in real-time depends on the size of the network topology used and on the number of ports of each ASON/GMPLS router.

The capacity of each lightpath is set to 15 Gbps and each connection request requires from 1 Gbps to 3 Gbps using the discrete statistical distribution in Table 7.1. A connection request of 1 Gbps corresponds to an OC-24 optical line while the 2 Gbps request corresponds to an OC-48 optical line.

The propagation delay used in the simulations is set to the value in Table 7.1 because it corresponds to an average length for the optical fibres of 100 Km; assuming the speed of light as the propagation speed for the optical signals we have a delay of 0.33ms every 100 Km.
The IP processing delay, that is the time that each packet needs to be processed by the IP layer, is set to the value in Table 7.1 because it is the same used in all the INET simulation samples and is thought to be the average of today’s devices.

### 7.1.4 OSPF Timers and protocol blocking probability issues

Results from previous research [7] make us aware of some issues deriving from the blocking probability study. Under certain condition the connection request rejection is not due to the lack of network resources, but can be caused by protocol convergence problems, instead.

The reason of this problem is in the highly dynamic behavior of the virtual (IP/MPLS) topology; while VT-First, preferring the grooming operation as first step, generates only few changes in the IP topology and its link state protocol reach the convergence in time; PT-First, preferring the lightpath creation as first step, generates several changes in the IP topology and the routing protocol not always reach the convergence causing, for many connection requests, a “false positive”. “False positive” means that the policy starts the creation of a Lightpath over an available optical route but the route is actually full because of resources being reserved at the same time.

The fact that the virtual topology is more dynamic when using the PT-First policy causes also another phenomenon. The IP/MPLS control and data packets share the same links (in-band signaling) and due to the quick changes in the network there is a high probability of having a not connected network graph, meaning that nodes in one connected component cannot transmit to the node of others connected components; these nodes will not be able to update the link state information until a new lightpath connects the two components. This results in a increment of the overall blocking probability. The same phenomenon is not present at the ASON layer because the control links are separated from the data links (out-of-band signaling) and the topology is fixed during all the simulation run; thus every router can always reach the others. Therefore, while in the ASON control plane the only blocking probability extra-contribution is given by the slow convergence of the link state protocol, in the IP/MPLS layer there is also an extra-contribution given by the graph connection problem discussed above.
The problems mentioned above are typical of the distributed model implemented in our system. Using protocol messages exchange each router of the network knows the state of all the links and can autonomously decide to accept or reject a connection request and calculate the route that the Lightpath or the Label Switched Path must follow. This is actually the approach adopted for real network; nevertheless it presents some drawbacks as the ones described above.

These problems were mostly experienced in the original version of the simulator, where only few basic functions were implemented on the local routing protocol module (see [7] for more information).

Using advanced features of OSPF-TE, such as timeout and advanced link management, the performance slightly improves. In particular, some timers to manage the protocol convergence are defined by the original OSPF specification as follows:

- **Min Hello Interval.** Triggers the creation of new Hello packets to send to adjacencies.
- **Min LS interval.** The minimum time between distinct generations of any particular LSA. The value of MinLSInterval is set to 5 seconds.
- **Min LS arrival.** For any particular LSA, the minimum time that must elapse between receptions of new LSA instances during flooding. LSA instances received at higher frequencies are discarded. The value of MinLSArrival is set to 1 second.

Several studies have been made in literature on how to modify the value of some OSPF timers [8] [9]. Delays may slow down the network convergence time, especially in case of highly-dynamic networks like the one in use: this issue can be addressed substantially improving the OSPF network operations by using more aggressive configurations and decreasing the temporal gap defined by the timers described above.

Nevertheless, finding optimal values to guarantee fast convergence in modern network is not easy. By one hand, OSPF delays are essentials in order to guarantee stable operation of the network, avoiding an excess of control traffic flooding or phenomenon like network routes flapping. On the other hand, reducing delays can lead to better performances in terms of convergence and blocking ratio.

MinLSInterval should be non-zero, value 50ms was used instead of default 5 seconds in this work. A good rule is keeping MinLSArrival < MinLSInterval [9].
7.2 TCP/UDP Traffic Simulation

In this section the system model implemented with our simulator and used for all the simulations with traffic exchange is introduced. First we describe the traffic model, and some modification to the system used in last section. Then, peculiar values of the parameters regarding the traffic simulation are given.

7.2.1 Network Model

For this set of simulation the model defined in last section has been slightly modified, as the traffic simulation requires some further effort:

- We need to define several hosts linked to the network described before
- For each host, we have to implement a generic application able to exchange UDP/TCP traffic with a randomly chosen destination host.

The following figure shows an example of the simulator environment modified to implement the traffic exchange between nodes.

![Network Model Diagram](image)

Figure 7.3 – An example of the modified Network model

The example above shows two new modules added to the network; the most important is the connectionManager module, placed on top-left of figure 7.3 instead of the connectionGenerator module.

The connectionManager choose a source according to a uniform distribution among the
hosts connected to the network. Then, it triggers a new connection request by sending a message to the chosen source host, whose model is described in the following part.

### 7.2.2 Standard Host implementation

During the implementation of the host module, a previous INET version has been reused, modified and enhanced to support traffic exchange. Moreover as clearly shown below, the previously defined connectionGenerator module has been integrated inside each host. The connGenerator module is now in charge of handling messages from the connectionManager, generating connection requests and managing the BasicApp module implemented to exchange TCP/UDP traffic with others hosts.

![Figure 7.4 – An internal view of the host module](image)

Figure 7.4 highlights two newly implemented elements added to the INET StandardHost module. The former is the connGenerator module explained before, the latter is the BasicApp module, directly derived from the genericApp class we are going to describe here below.

**BasicApp module**

The INET framework presents several implementations of TCP or UDP application, on the base of which it is possible to derive different traffic models. However, none of these application can manage both TCP and UDP traffic; this is the main reason why we de-
chided to implement a brand new application module, enhancing or sometimes completely renewing the code already available on INET.

Here below, the UML diagram of the new module, with a brief description of its main components.

![UML Diagram](image)

**Figure 7.5 - BasicApp UML Class Diagram**

The BasicApp module is in charge of managing the TCP connections and the UDP traffic exchange between the local hosts and the other hosts of the network. In particular, we can briefly see how the information concerning TCP and UDP traffic is stored within two data structures, `connectionsMap` and `connectionsMapUDP` (of course, in case of UDP we cannot talk about a real “connection”).
Most of the methods listed above have been implemented for the management of UDP packets and TCP flows. In particular, `generateNewONTime` and `generateNewOFFTime` methods are in charge of generating a value for ON end OFF timers scheduled during the exchange of data bursts between source and destination.

### 7.2.3 Simulation Settings

Here below, the parameters settings used for this set of simulations, in adding to the common settings listed in section 7.1.3.

<table>
<thead>
<tr>
<th>pareto_shape</th>
<th>1.1</th>
</tr>
</thead>
<tbody>
<tr>
<td>n_hosts</td>
<td>70</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Traffic Percentage</th>
<th>UDP</th>
<th>TCP</th>
</tr>
</thead>
<tbody>
<tr>
<td>30%</td>
<td></td>
<td></td>
</tr>
<tr>
<td>70%</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Packet Length (Bytes)</th>
<th>UDP</th>
<th>TCP</th>
</tr>
</thead>
<tbody>
<tr>
<td>Truncnormal (100, 10)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Truncnormal (350, 20)</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

UDP traffic has been chosen as High Priority traffic, while Low Priority has been assigned to TCP traffic. This choice has been driven considering the traffic delay and delay jitter as critical parameters for most of the real time applications (VoIP, Video over IP, etc.) using UDP as preferred transport protocol. On the other hand, we considered as LP traffic the tradition best effort traffic, such as web surfing and e-mail exchange (which is mostly TCP traffic).

The pareto_shape parameter’s value is 1.1 and is a common value set by researchers for realistic traffic simulations. However, the pareto_shape parameter depends on the traffic type we want to simulate. To find a reasonable pareto_shape parameter, one should sample real traffic parameters during transmissions (or borrow the results from other research), and compute the mean value.
7.3 Simulations Analysis

The next set of experiments aims at studying the service differentiation offered by RP-Diff, Soft-Diff and Hard-Diff in terms of blocking probability and packet delay under different traffic load conditions.

For each experiment, each router has been configured with an amount of optical resources equal to 24 wavelengths and 16 ports. The traffic load offered by the generator module has been set to different values keeping the inter-arrival time $T_{ia}$ fixed to a constant value while varying the holding time $T_{hold}$.

**RP-Diff policy evaluation**

Here below we can see the blocking probability under different traffic load conditions using RP-Diff with an inter-arrival time $T_{ia}$ =0.6s.

![Figure 7.6 - Blocking Probability for HP and LP traffic in the RP-Diff policy](image)

As shown in figure 7.6, the service differentiation merely based on the use of different multilayer routing policies seems to be not enough to achieve performance differentiation in terms of blocking probability. The blocking ratio of HP and LP traffic results nearly the same both in case of low traffic load and high traffic load.

The reason of these results can be easily explained. Despite the two different MTE routing policies applied, the network’s resources are completely shared between HP traffic end LP traffic. The gap between the two curves is not significant and sometimes it
can overlap, for traffic load greater than 120 Erlang.

For the reasons explained above, it is likely to expect similar results from the simulations concerning the packet delay, showed by figure 7.4.

![Figure 7.7 - Packet delay for HP traffic and LP traffic in the RP-Diff policy](image)

Even though we reach a certain degree of differentiation between HP traffic and LP traffic, we can see that the values of packet delay gap tend to decrease as the traffic load rises until 125 Erlang, where the two curves overlap.

One of the reasons for this behavior can be found analyzing the average length of the LSPs set up for LP and HP traffic. Using RP-Diff there is complete sharing of the network resources, so we expect only a slight difference between HP and LP LSPs.

![Figure 7.8 - The average LSP length in the RP-Diff strategy](image)
Figure above confirms that some differentiation policies to manage the network resource sharing are needed too. Using RP-Diff, there is no Service Differentiation at all in terms of blocking probability or cannot be guaranteed, anyway, in terms of packet delay as we can see in figure 7.8. To observe a differentiation in terms of LSP length it could be necessary run some simulations over a bigger network topology, as NSF give almost the same results for HP and LP LSP lengths.

**Hard-Diff policy evaluation**

As previously introduced in chapter 3, using the Hard-Diff policy we aim at the service differentiation by introducing a stronger discrimination between HP and LP traffic. Connection requests are now accommodated originating different virtual topologies for different request priorities.

Figure 7.9 shows the blocking probability for the Hard-Diff policy, using the same settings and the same traffic load conditions as the previous experiment.

![Figure 7.9 – Blocking Probability for HP traffic and LP traffic in the Hard-Diff policy](image)

As we can see above, it is clear how the Hard-Diff approach achieves a substantial diversification between the QoS offered to HP and LP traffic. Figure 7.9 highlights the behavior of the two curves representing the Blocking Probability: we can easily see that they never overlap. The same behavior has been observed analyzing the results obtained simulating traffic exchange among the network’s nodes, as showed in figure 7.10.
Figure 7.10 - Packet delay for HP traffic and LP traffic in the Hard-Diff policy

The Hard-Diff strategy demonstrates how differentiating the service offered in terms of resource sharing can be a good way to achieve better results than RP-Diff strategy. On the other hand, the gap between HP and LP blocking ratio seems to be more than what is desirable; some further analysis is needed, comparing RP-Diff and Hard-Diff results as in figure 7.11.

Figure 7.11a - Blocking ratio for HP and LP traffic in Hard-Diff and RP-Diff
Figures above permit to analyze the behavior of both the RP-Diff and Hard-Diff strategies. The blocking probability is higher for the Hard-Diff strategy for Low Priority. Hence, we can say that a certain degree of service differentiation has been achieved, but the performances degradation is such that RP-Diff still results more convenient than HardDiff. Once created, an OLSP can accommodate only traffic belonging to one CoS. This implies that each CoS has a lower amount of capacity available for grooming compared to the RP-Diff policy where HP and LP traffic are mixed over a unique virtual topology. The excessive degradation of the network performances for LP traffic in terms of blocking probability makes the Hard-Diff strategy still not feasible. The problem can be solved trying to allow a certain degree of resource sharing between the two virtual topologies, as explained here below by the Soft-Diff strategy.

**Soft-Diff policy evaluation**

The Soft-Diff scheme follows the same approach as the one introduced by Hard-Diff, implementing an enhanced algorithm similar to the previous one which takes into account a certain degree of resource sharing between HPT and LPT.

HP and LP traffic are still transmitted over two different virtual topologies, but LPT now can use a limited amount of HPT’s resources to overcome the limitation experienced during the Hard-Diff tests. When the system tries to accommodate new LP requests, resources allocation is performed taking into account also direct HP lightpaths between
source and destination (an example can be found in chapter 3, where the algorithm is explained in depth).

Figure 7.12 shows the Blocking Probability in the Soft-Diff strategy under the same traffic conditions as the previous experiments. In this case, we chase to represent both the RP-Diff and Soft-Diff curves, allowing to compare the two performances at first glance.

Figure 7.12 - Blocking ratio of HP and LP traffic for RP-Diff and Soft-Diff policies

In this case the network improves its performances from the blocking ratio point of view. This confirms that a certain degree of resource sharing leads to better results, as we can see in figure above.

As illustrated above, for low loaded networks RP-Diff is still the more convenient policy. This is due to the very low blocking ratio achieved by both HP and LP traffic in the RP-Diff approach.

On the other hand, once the traffic load increases, the Soft-Diff scheme reaches better results when the blocking ratio in the RP-Diff increases both for LP and HP traffic. This phenomenon does not affect the Soft-Diff scheme where the HP traffic blocking ratio remains negligible, while the LP blocking ratio increases. The advantage of the Soft-Diff policy is further highlighted when the traffic load rise over 120 Erlang and the HP and LP blocking ratio reach values comparable with the ones assumed by the curve representing the LP blocking probability of the Soft-Diff strategy.

This is a very important goal the Soft-Diff policy reached:
- **Service Differentiation** has been achieved. By one hand, RP-Diff offers nearly the same blocking % for both HP and LP traffic. On the other hand, Soft-Diff keeps the blocking % for HP traffic to negligible values, even when the traffic load increase.

- A good **trade-off** between Service Differentiation and LP performances has been reached. It is clear that the virtual topologies differentiation implies a certain degree of performances degradation for Low Priority traffic. Contrary to the results observed before using the Hard-Diff strategy, now the LP blocking ratio can be considered comparable with the one assumed by RP-Diff.

What we said before is further confirmed by the **packet delay** analysis.

In this case we decided to compare the service differentiation provided by Soft-Diff and RP-Diff more in depth:

- first, presenting the average packet delay under different traffic load conditions for both the policies;
- then, choosing a particular traffic load to record the delay suffered by each packet exchanged over the LP and HP Topologies.

Figure above shows the RP-Diff curve already seen before, and the behavior observed for the Soft-Diff strategy. As we can easily see, for low traffic conditions RP-Diff remains the best policy; nevertheless after 110 Erlang both LPT and HPT seem to suffer less than the RP-Diff topology the traffic load increasing.
In figure 7.13 it is already possible to observe how the differentiation offered by Soft-Diff is slightly better than the RP-Diff’s one. However, here below we can see some screen-shots from OMNeT++ representing the delay (in seconds) suffered by each packet exchanged over the network in case of 125 Erlang of traffic load.

Figures above show clearly what we expected: the service differentiation appears more evident than as we observe in figure 7.13, where only the average value is represented. This is because, as we can see in figure 7.15c, LP traffic can suffer both high and very low delays. Thus, it is a good idea to make some experiments in order to verify that the dis-
tance between each recorded packet delay and the average delay is reasonable. This is an important parameter, giving us some preliminary indications on the variation in the time between packets arriving delay and average packet delay during the traffic exchange (jitter).

In order to evaluate this parameter we analyze the variance of the packet delay; the result is showed here below for both RP-Diff and Soft-Diff policies.

![Figure 7.16 – traffic delay variance for RP-Diff and Soft-Diff policies](image)

As said before, of course the Soft-Diff approach implies a slight degradation of the performances offered for LP traffic. RP-Diff still remains more feasible for low-medium traffic load conditions, while, from 120 Erlang on, the curve reaches values comparable to the Soft-Diff Low Priority. Evaluating the variance is important to have a first impression about certain critical parameters: for some real-time applications, average packet delay is not enough as guarantee of QoS; maximum packet delay and delay jitter are important parameters as well (e.g, VoIP, Video over IP, etc.).

In figure 7.16 packet delay over LP Topology can be considered less “stable” than the HP Topology’s one. Nevertheless, comparing Soft-Diff behavior with the RP-Diff’s one, we can see how the differentiation between HPT and LPT remains reasonable.

**Some final considerations**

In this chapter we presented a set of experiments in order to evaluate the Service Differentiation offered by three MTE policies. The first one, RP-Diff is based on a simple algorithm that decides on the multilayer routing policy according to the CoS the service re-
quest belongs to. The second one, Hard-Diff, transmits traffic belonging to different CoS onto different lightpaths. In practice, the system operates a resource differentiation which does not allow services belonging to one CoS to share network resources on the virtual topology. However, a certain degree of resource sharing (the Soft-Diff policy) needs to be introduced in order to not deteriorate the performance with respect to a system that transmits traffic on a single virtual topology. As already explained in chapter 3, a further clarification is needed: why the LPT is allowed to use direct OLSPs of the HPT but not vice versa? This decision has 2 main reasons:

- The LPT can undergo more instability and consequently it can be reconfigured with a higher frequency compared to HPT. This means that HP services cannot be accommodated on the LPT otherwise the provided QoS could not be guaranteed.
- The limited capacity available experienced in the Hard-Diff scheme is more significant for the LP traffic due to the use of VTF in the LPT.

By forcing the system to consider only OLSPs directly connecting source and destination, we limit the capacity utilization of the HPT and prevent service degradation [14]. Figure 7.17 and 7.18 summarizes the behavior of the three strategies in terms of blocking probability and packet delay.

![Figure 7.17 - Blocking ratio comparison for HardDiff, SoftDiff and RPDiff](image-url)
Figure 7.18 - Packet Delay comparison for HardDiff, SoftDiff and RPDiff

Considering the results obtained from the whole set of simulations explained in this chapter and the resuming charts showed in figures above, we can see how accommodating different CoS over different virtual Topologies (in this case Low Priority and High Priority Topology) improves the Service Differentiation supplied by the network.

This approach suffers some limitation as well: in case of a large amount of different Class of Services, further research is needed in order to find the best trade off among the number of CoS, the network topology, and the less capacity available to allocate resources for each virtual topology.
8 Conclusions
This thesis addressed the problem of multilayer service differentiation in IP/MPLS over ASON/GMPLS network focusing on the GMPLS control plane. In [14], authors propose three different Service Differentiation algorithms: VT-RPDiff, VT-HardDiff and VT-SoftDiff. The first one implements a mere routing policy differentiation, while the second and the third one use the Multi-Topology (MT) routing techniques to accommodate traffic belonging to different CoS onto dedicated virtual topologies (HPT and LPT).

We started from the Multi-Topology routing technique used in the Open Shortest Path First (OSPF) protocol to address the MT problem in IP networks. We first made the suitable adaptation to employ this technique in the GMPS control plane as well. The extended MT technique has been then implemented in a previously developed IP/MPLS over ASON/GMPLS simulator [7].

An extensive set of experiments has been executed to compare the behaviour of the implemented Service Differentiation policies. In case of medium/high traffic load conditions Soft-Diff gives the best contribution to the Service Differentiation supplied by the simulated network: the HP Topology keeps the blocking ratio and the packet delay to lower values than RP-Diff, whose performances are comparable with the LP Topology ones.

On the other hand, RP-Diff doesn’t provide the desired QoS differentiation but guarantees better performances in terms of blocking probability and end-to-end packet delay in case of low traffic load. The Hard-Diff approach resulted too much “aggressive”, reaching the desired QoS differentiation at the expense of some performance degradation of both the Classes of Service.

The simulation results confirm that a scheme that merely differentiates the routing policy is not sufficient to achieve the desired QoS diversification, as affirmed in [14]. In modern networks the differentiation of the QoS is highly desirable not only under the user perspective, but also from the provider point of view. Different virtual topology re-configuration techniques can be applied, for example reconfiguring the HPT with a lower frequency, avoiding some service degradation for critical services. Operators can use different multilayer routing policies to accommodate the traffic, selling added value services which increase the Return On Investment [14].
While the current implementation is oriented towards supporting experiments in terms of blocking probability and end-to-end packet delay, it also provides the basics to improve its functionalities towards more sophisticated experiments.

- Additional MTE routing policies and Service Differentiation algorithms can be easily added, evaluated and compared using the current simulation engine.
- Link failures management at the optical and virtual layer to study restoration and reconfiguration algorithms using GMPLS protocols.
- Support for multi-Area networks, developing the multi-area and multi-AS OSPF capabilities.
- Implementation of more sophisticated and realistic data traffic exchange to study end to end transmitting delay and QoS algorithms related to packet management.
- Implementation of different control model architecture, instead of the “Augmented Model” developed in [7].

Changes and improvements listed above can be easily implemented; other interesting features and scenarios could be supported with additional effort. Moreover, in this work we used the NSF network topology, with limited network resources capacity as a good trade-off between results accuracy and simulation time. Defining new scenarios, with bigger network topologies and higher capacity could be useful for a more evident differentiation among the simulated Service Differentiation policies. Nevertheless, this functionality has to face the simulation time and system resources consumption issues.

Our implementation experience suggests that extending OMNeT++ to support additional technologies, protocols and mechanisms hinges on:

- Understanding the architecture and programming model of the OMNeT++ and INET Framework simulation engine
- Gaining insight into the IP/MPLS and ASON/GMPLS technologies and protocols
- Training in depth on the simulator architecture and implementation issues.
References


